무료 전자 책

배우기

C# Language

Free unaffiliated eBook created from
Stack Overflow contributors.

#c#

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: csharp-language

It is an unofficial and free C# Language ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official C# Language.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

1: C

C # Microsoft C . C # Windows, Mac OS X Linux CIL .

1.0, 2.0 5.0 ECMA (ECMA-334) C # .

1.0	2002-01-01
1.2	2003-04-01
2.0	2005-09-01
3.0	2007-08-01
4.0	2010-04-01
5.0	2013-06-01
6.0	2015-07-01
7.0	2017-03-07

Examples

(Visual Studio)

1. Visual Studio
2. → .
3.
4. `Program.cs`
5. `Main()` .

```
public class Program
{
    public static void Main()
    {
        // Prints a message to the console.
        System.Console.WriteLine("Hello, World!");

        /* Wait for the user to press a key. This is a common
           way to prevent the console window from terminating
           and disappearing before the programmer can see the contents
           of the window, when the application is run via Start from within VS. */
        System.Console.ReadKey();
    }
}
```

6. -> **F5** **Ctrl + F5** ( ) .

- `class Program` . `Program Program` . . . `Program Main` .
- `static void Main()` C # `Main` . `Main` . `Main` .
- `System.Console.WriteLine("Hello, world!");` (`Hello, world!`) .
- `System.Console.ReadKey()` . . . `main()` .

Microsoft Build Tools `MSBuild csc.exe` *(C #)* .

`HelloWorld.cs` .

```
%WINDIR%\Microsoft.NET\Framework64\v4.0.30319\csc.exe HelloWorld.cs
```

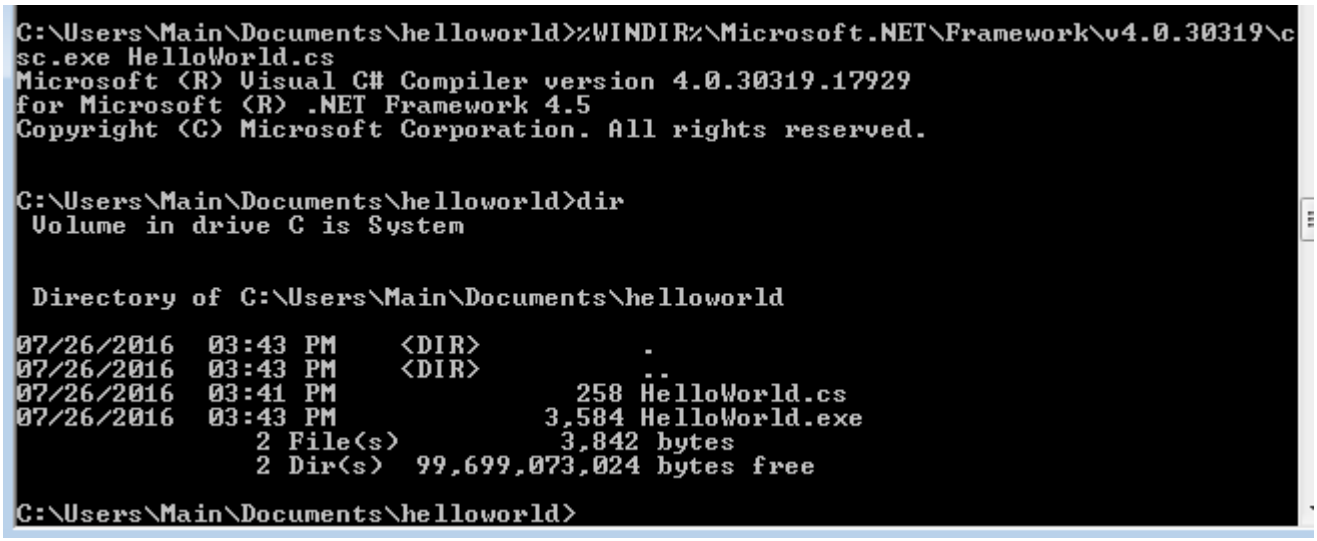
. (`ClassA` HelloWorld `HelloWorld.cs` main)

```
%WINDIR%\Microsoft.NET\Framework64\v4.0.30319\csc.exe HelloWorld.cs /main:HelloWorld.ClassA
```

HelloWorld .

*: **.NET Framework v4.0** . .NET . 32 .NET Framework , **framework64** . Windows csc.exe Framework (32).*

```
dir %WINDIR%\Microsoft.NET\Framework\csc.exe /s/b
dir %WINDIR%\Microsoft.NET\Framework64\csc.exe /s/b
```



`HelloWorld.exe` . Enter .

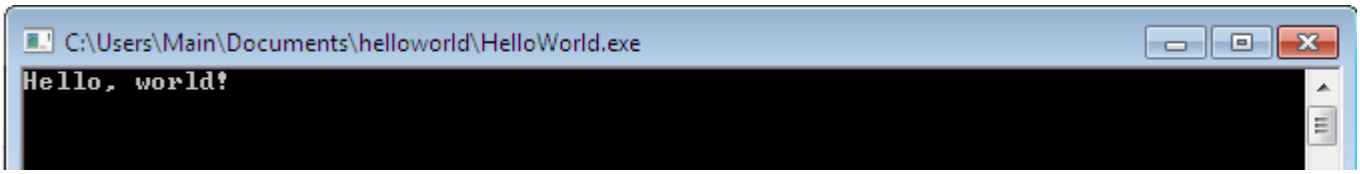
```
HelloWorld.exe
```

.

, !

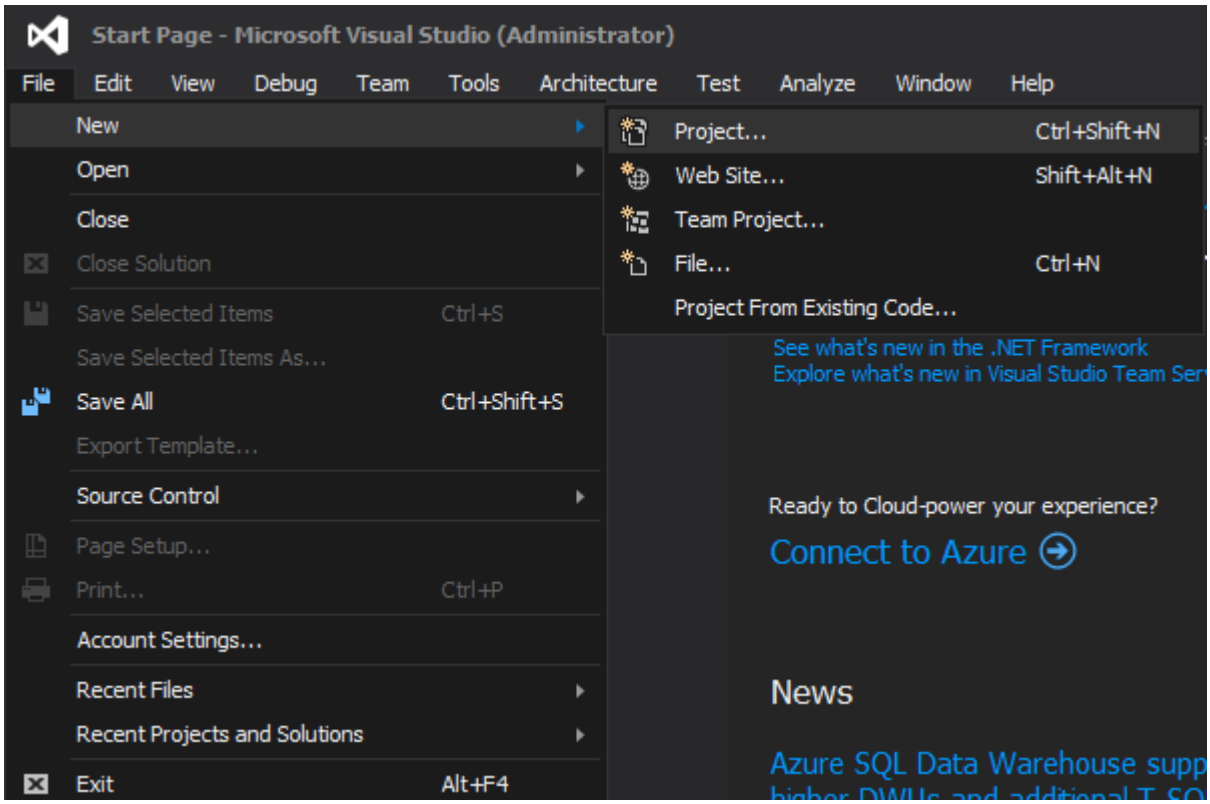
```
C:\Users\Main\Documents\helloworld>HelloWorld
Hello, world!
```

" **Hello, world!** " **.**

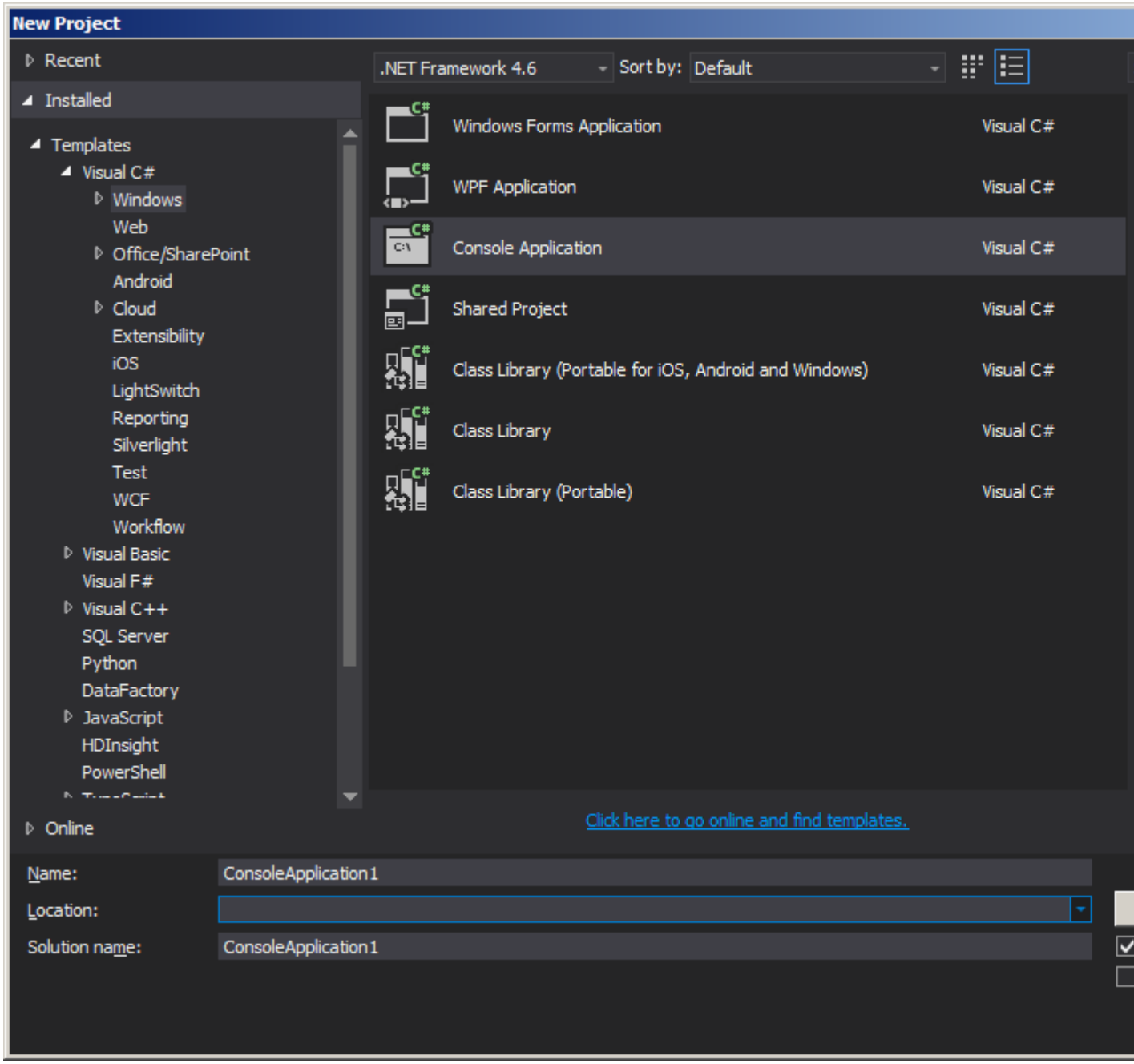


Visual Studio ( )

1. **Visual Studio** . Visual Studio VisualStudio.com . , .
2. **Visual Studio** **.**
3. **.** → → .

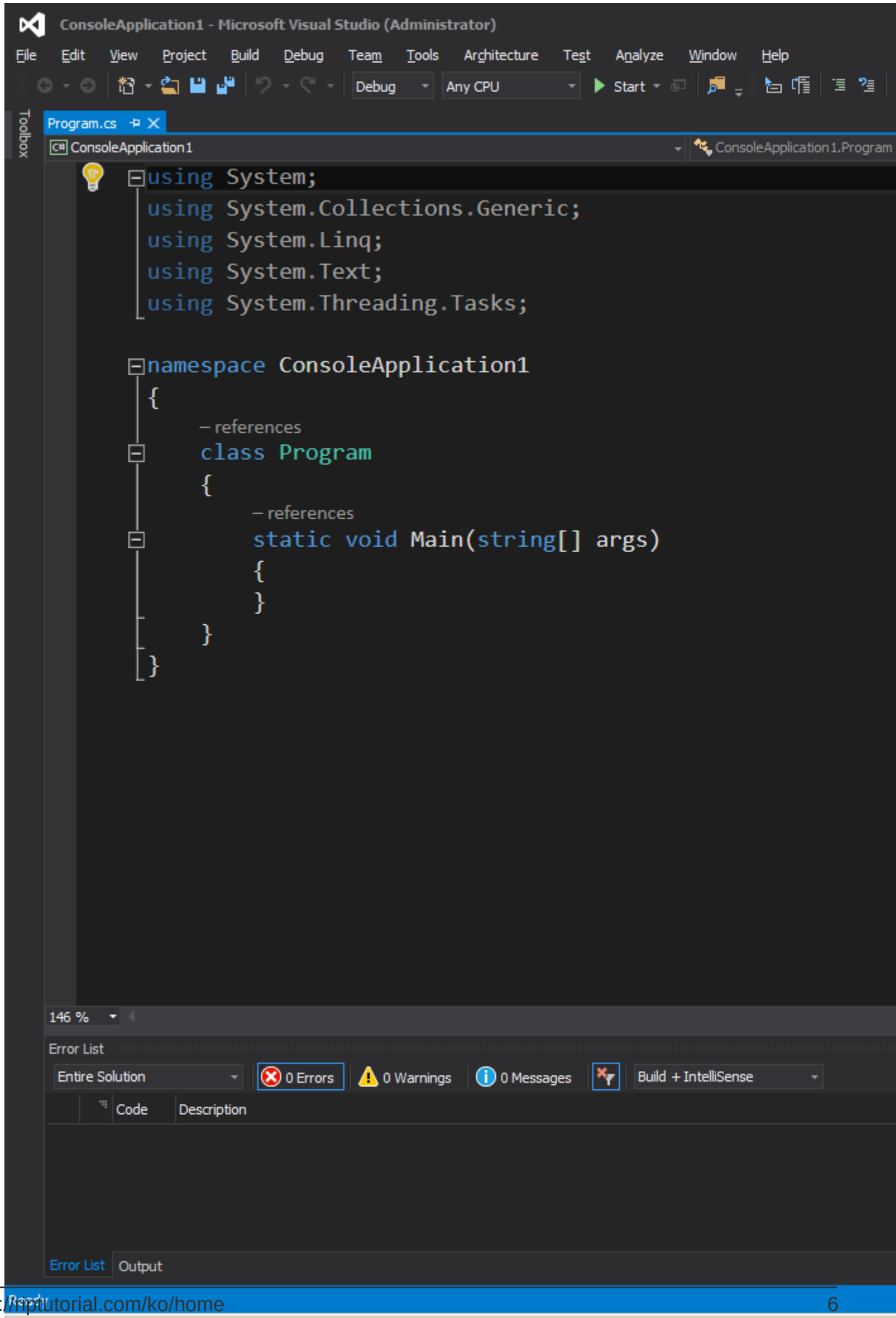


4. → **Visual C #** → **.**


New Project
Recent
Installed
Templates
Visual C#
Windows
Web
Office/SharePoint
Android
Cloud
Extensibility
iOS
LightSwitch
Reporting
Silverlight
Test
WCF
Workflow
Visual Basic
Visual F#
Visual C++
SQL Server
Python
DataFactory
JavaScript
HDInsight
PowerShell
Online
.NET Framework 4.6
Sort by:
Default
Windows Forms Application
Visual C#
WPF Application
Visual C#
Console Application
Visual C#
Shared Project
Visual C#
Class Library (Portable for iOS, Android and Windows)
Visual C#
Class Library
Visual C#
Class Library (Portable)
Visual C#
Click here to go online and find templates.
Name:
ConsoleApplication1
Location:
Solution name:
ConsoleApplication1

5. □□□ □□□□ . .

6. . .


ConsoleApplication1 - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug Team Tools Architecture Test Analyze Window Help
Debug
Any CPU
Start
Toolbox
Program.cs
ConsoleApplication1
ConsoleApplication1.Program
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace ConsoleApplication1
{
0 references
class Program
{
0 references
static void Main(string[] args)
{
}
}
}
146 %
Error List
Entire Solution
0 Errors
0 Warnings
0 Messages
Build + IntelliSense
Code
Description
Error List
Output
Ready

(. .)

7. . "Hello world!" `Program.cs` .

```
using System;

namespace ConsoleApplication1
{
    public class Program
    {
        public static void Main(string[] args)
        {
        }
    }
}
```

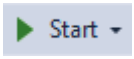
`Program.cs public static void Main(string[] args)` ( ).

```
Console.WriteLine("Hello world!");
Console.Read();
```

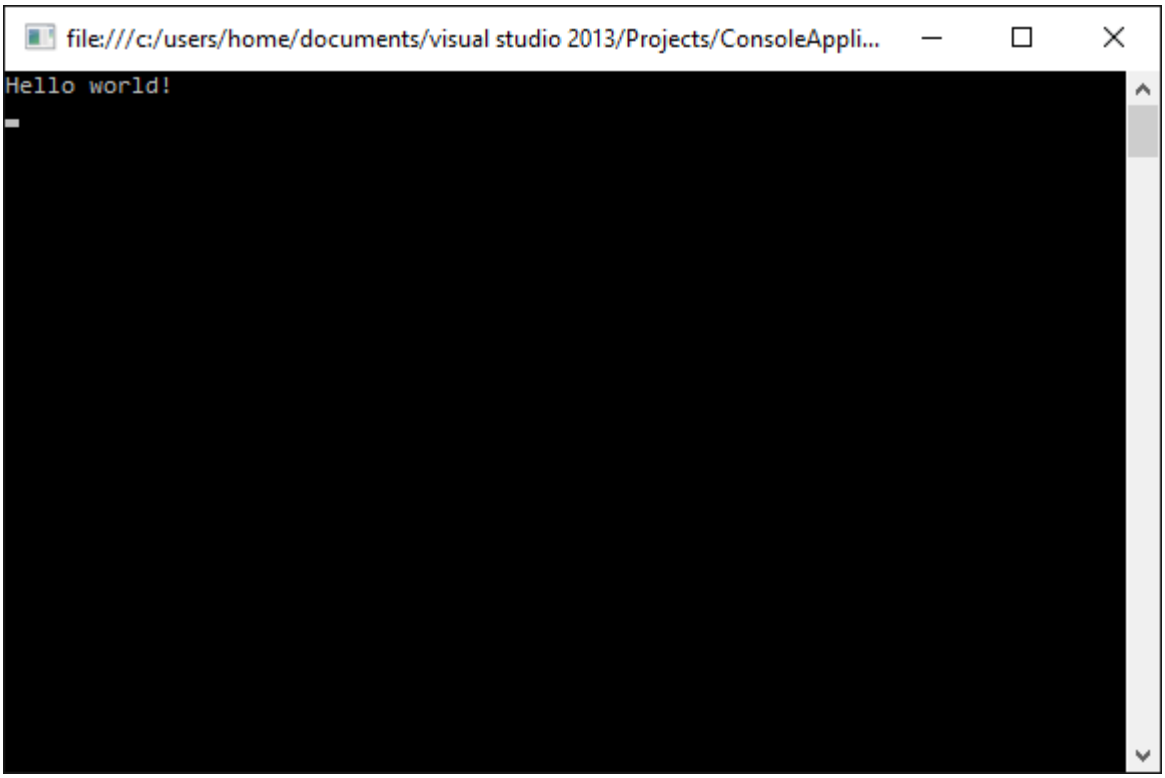
`Console.Read()` **?** "Hello world!" . . . Without `Console.Read();` , "Hello world!" . . .

. .

```
using System;

namespace ConsoleApplication1
{
    public class Program
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Hello world!");
            Console.Read();
        }
    }
}
```

8. . . 

F5 . → . . .



9. . . Console.Read() . .

Mono

Mono .

Mono Mac OS X, Windows Linux .

HelloWorld.cs .

```
public class Program
{
    public static void Main()
    {
        System.Console.WriteLine("Hello, world!");
        System.Console.WriteLine("Press any key to exit..");
        System.Console.Read();
    }
}
```

Windows Mono Mono . Mac Linux .

HelloWorld.cs .

```
mcs -out:HelloWorld.exe HelloWorld.cs
```

HelloWorld.exe :

```
mono HelloWorld.exe
```

.

```
Hello, world!
Press any key to exit..
```

.NET Core

.NET Core SDK .

- Windows
- OSX
-
-

.

1. `mkdir hello_world` `cd hello_world` .
2. `dotnet new console` .
   .
   - **hello_world.csproj**

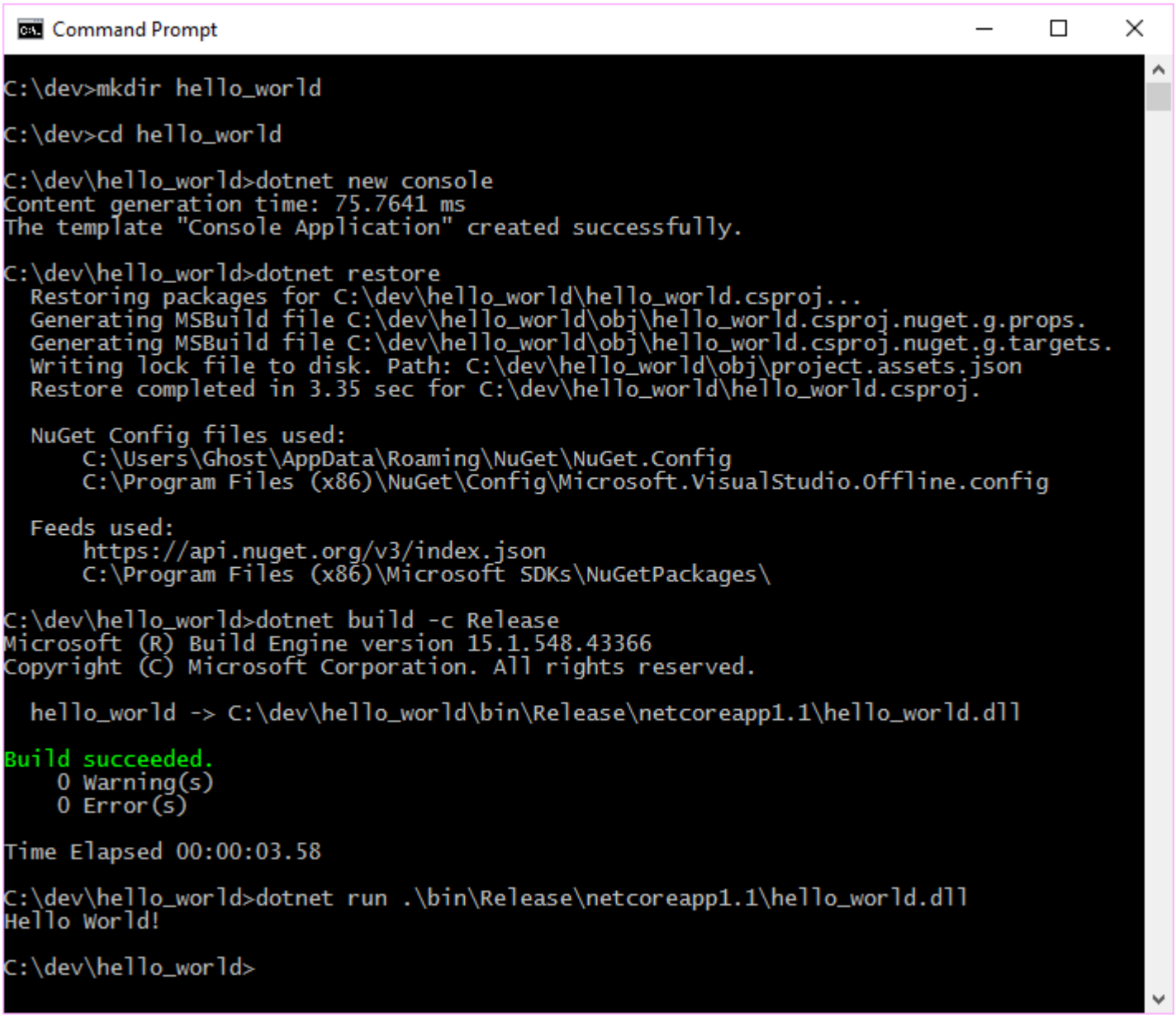
     ```
     <Project Sdk="Microsoft.NET.Sdk">

       <PropertyGroup>
         <OutputType>Exe</OutputType>
         <TargetFramework>netcoreapp1.1</TargetFramework>
       </PropertyGroup>

     </Project>
     ```

   - **Program.cs**

     ```
     using System;

     namespace hello_world
     {
         class Program
         {
             static void Main(string[] args)
             {
                 Console.WriteLine("Hello World!");
             }
         }
     }
     ```

3. `dotnet restore` `dotnet restore` .
4. Debug `dotnet build -c Release` `dotnet build` . `dotnet run` throw.
5. Debug `dotnet run .\bin\Release\netcoreapp1.1\hello_world.dll` `dotnet run` `dotnet run .\bin\Release\netcoreapp1.1\hello_world.dll` `dotnet run`

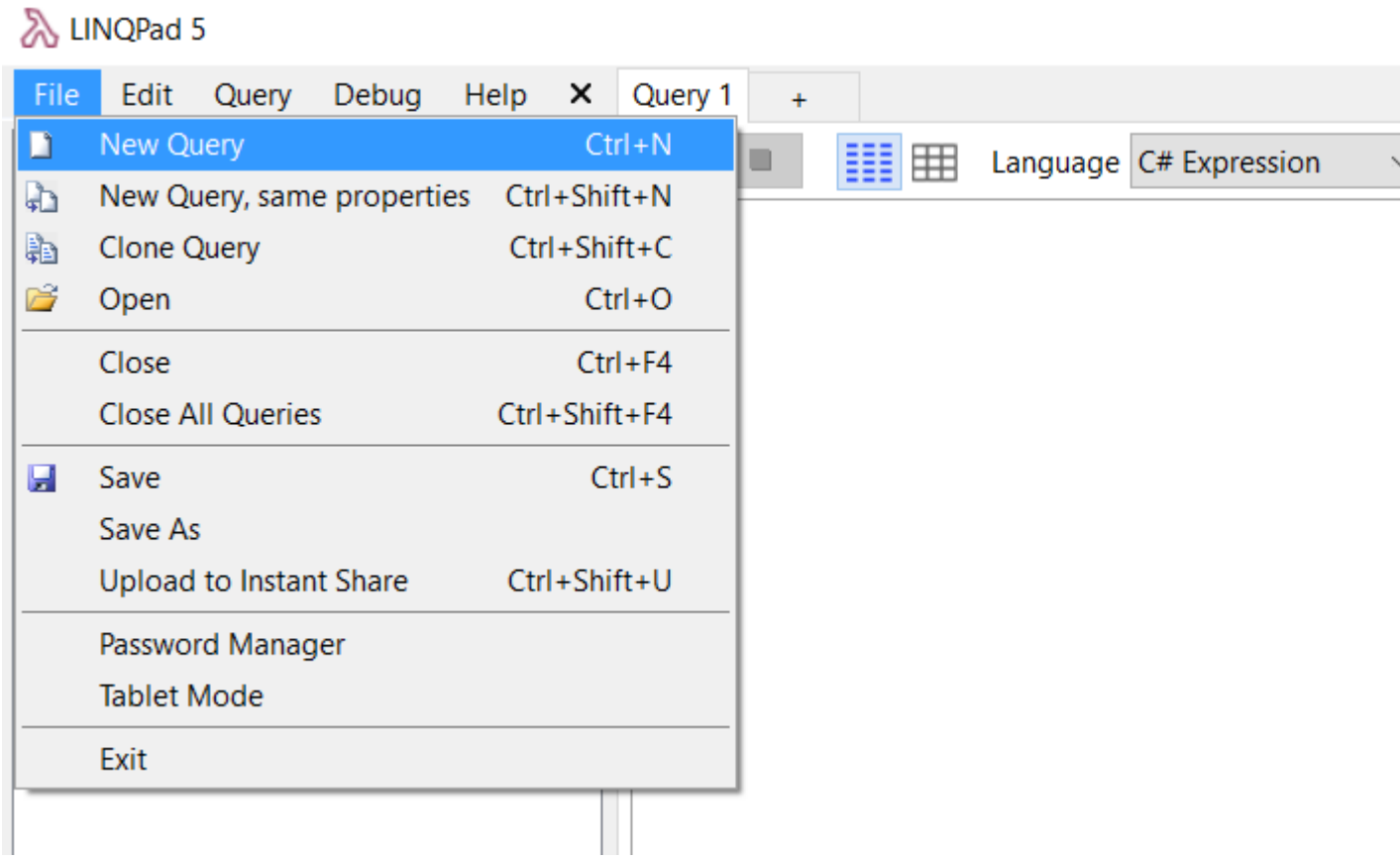
`.\bin\Release\netcoreapp1.1\hello_world.dll` for Release.



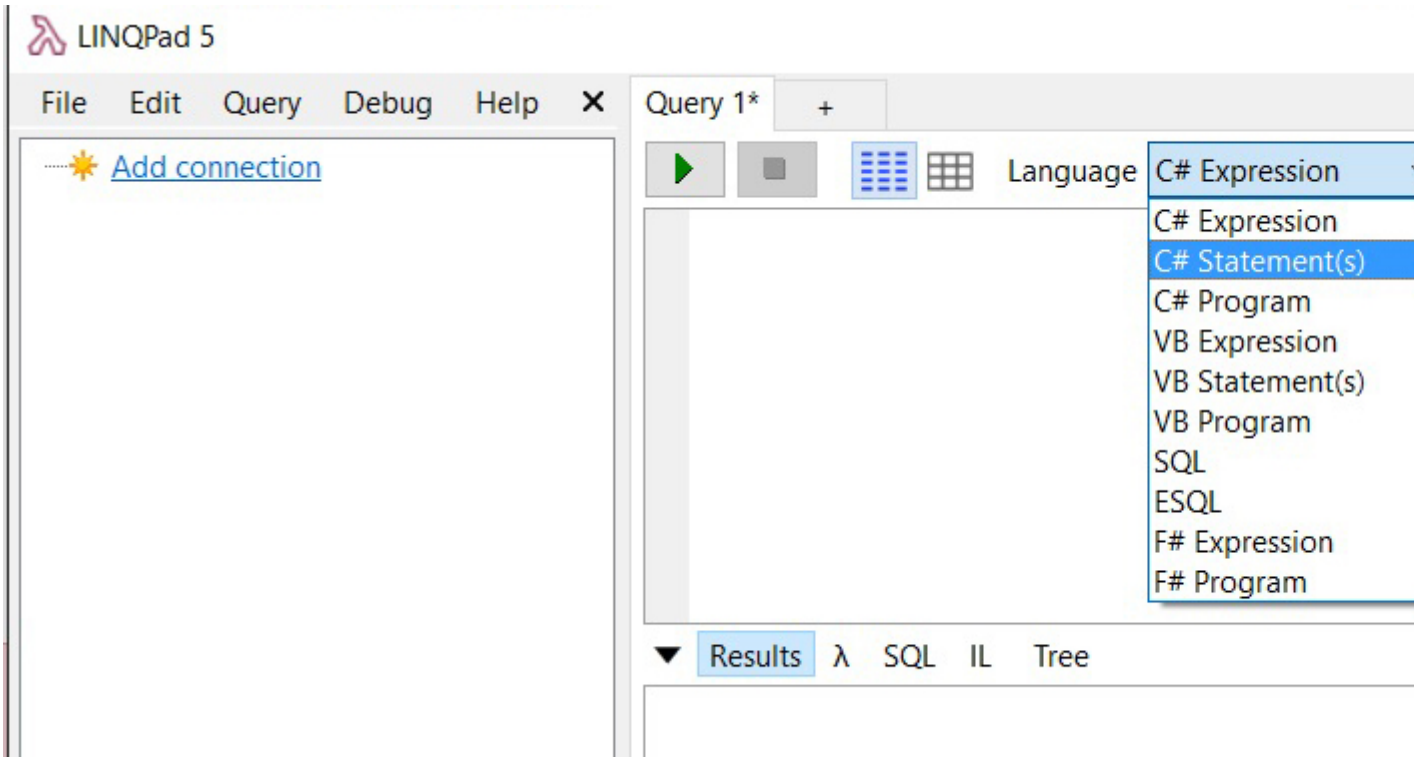
LinqPad

LinqPad .Net (C #, F # VB.Net) .

1. LinqPad
2. (`Ctrl` + `N`)

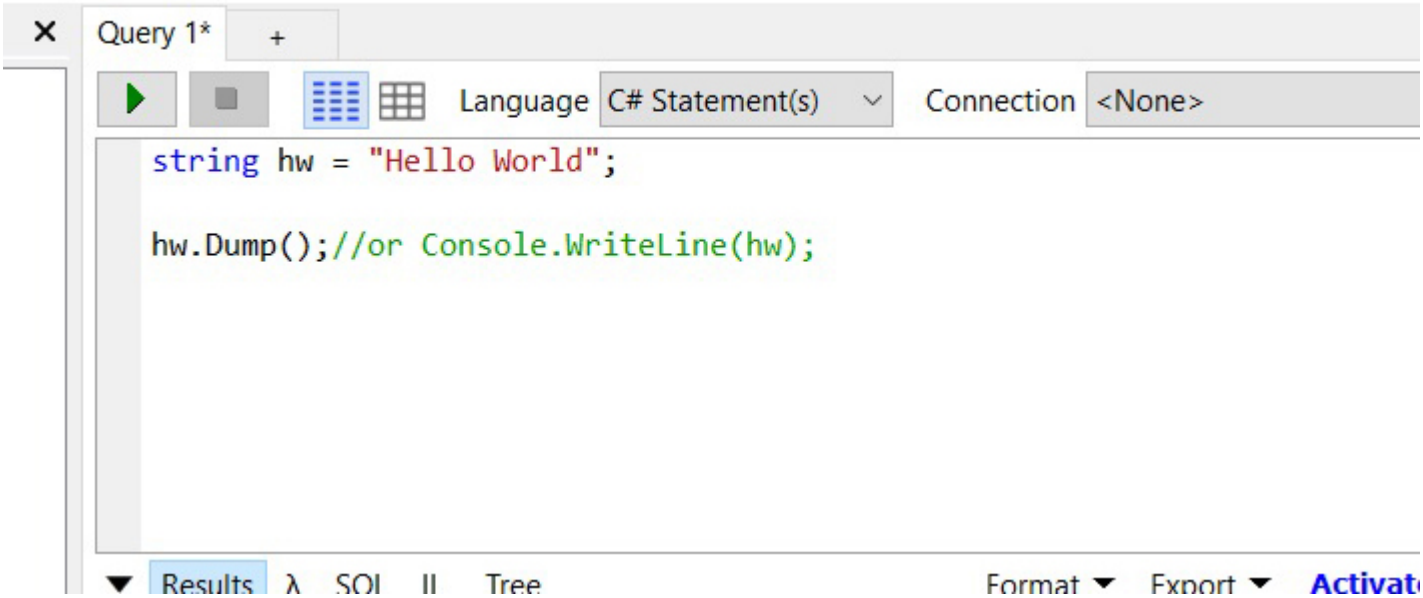


3. "C # " .

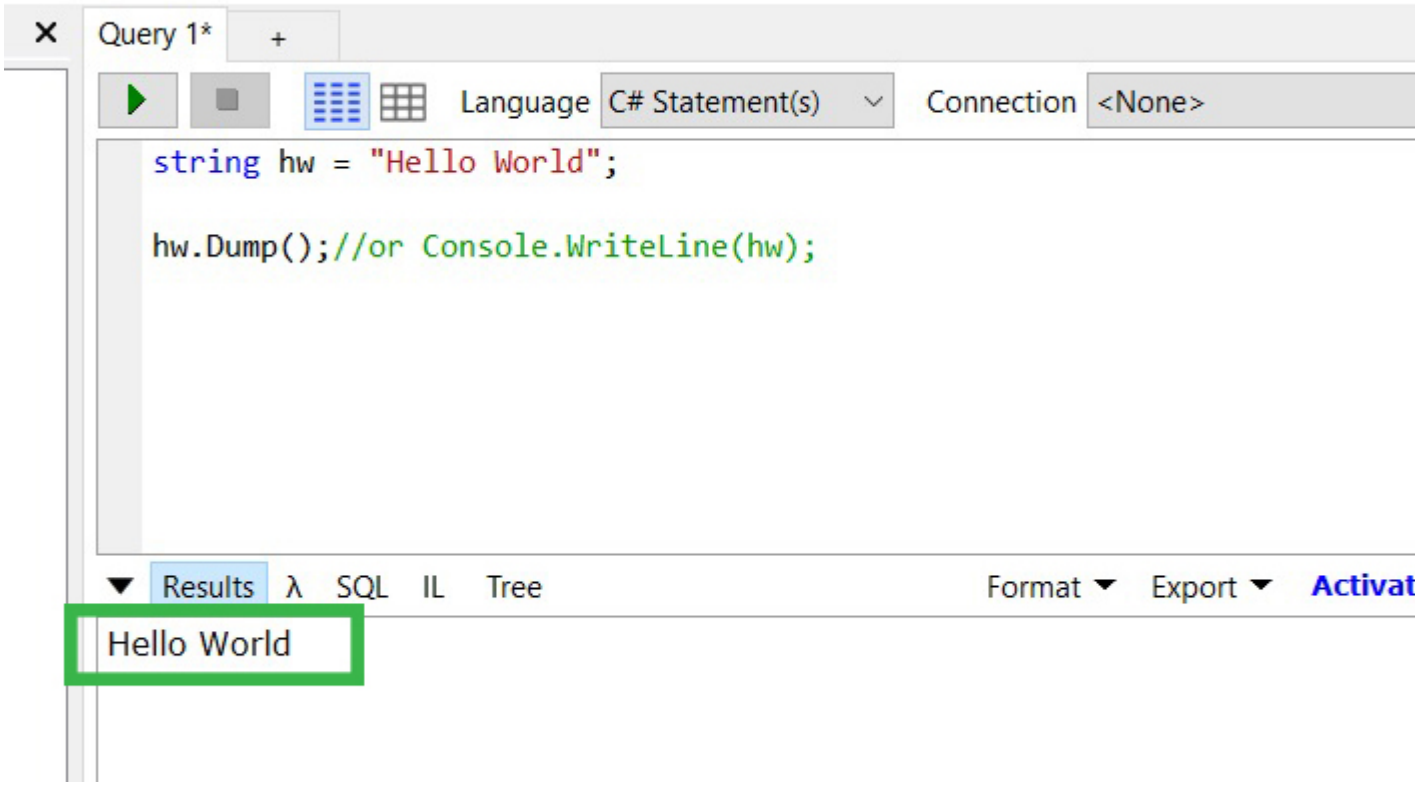


4. (F5) .

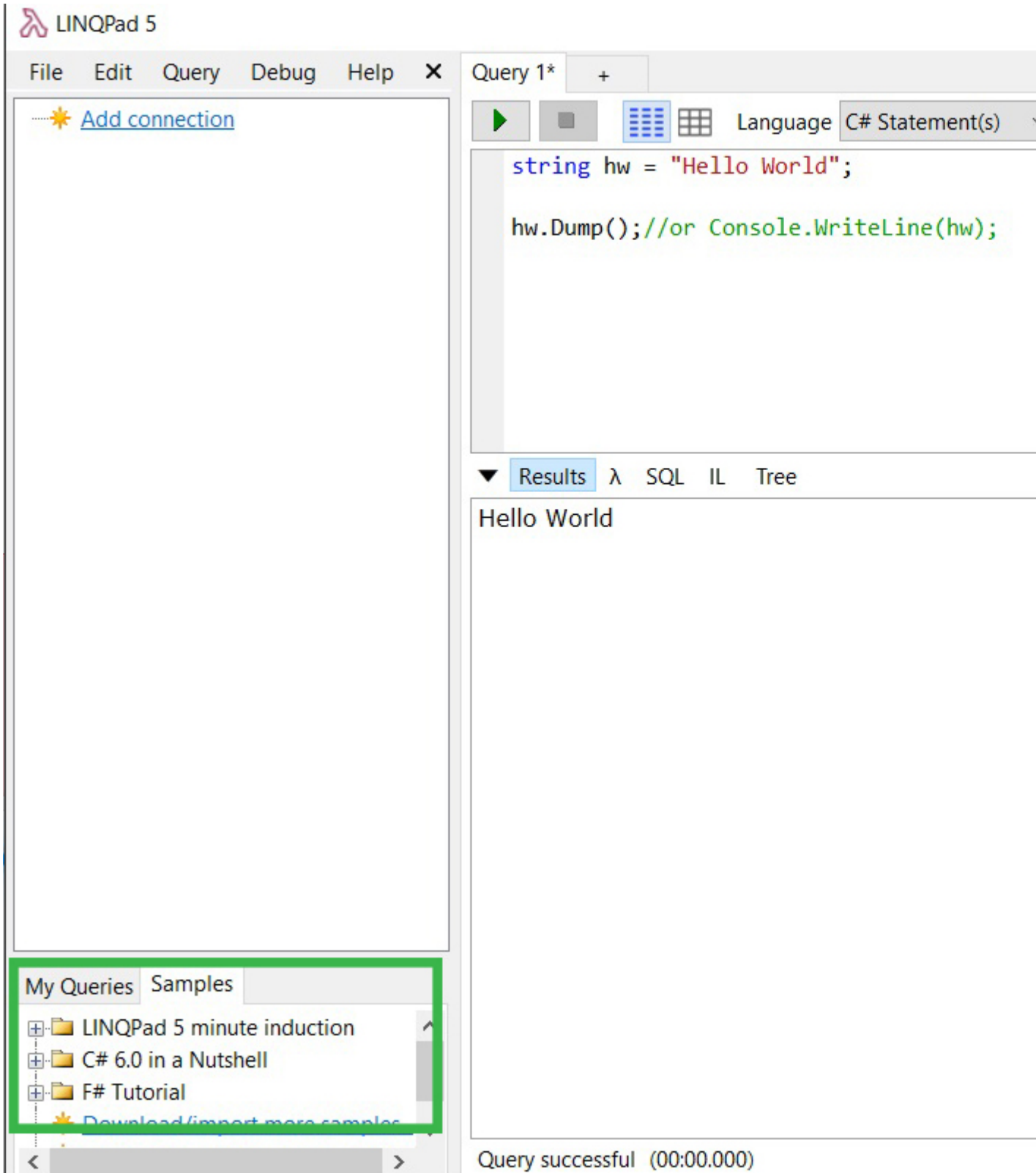
```
string hw = "Hello World";

hw.Dump(); //or Console.WriteLine(hw);
```



5. "Hello World" .

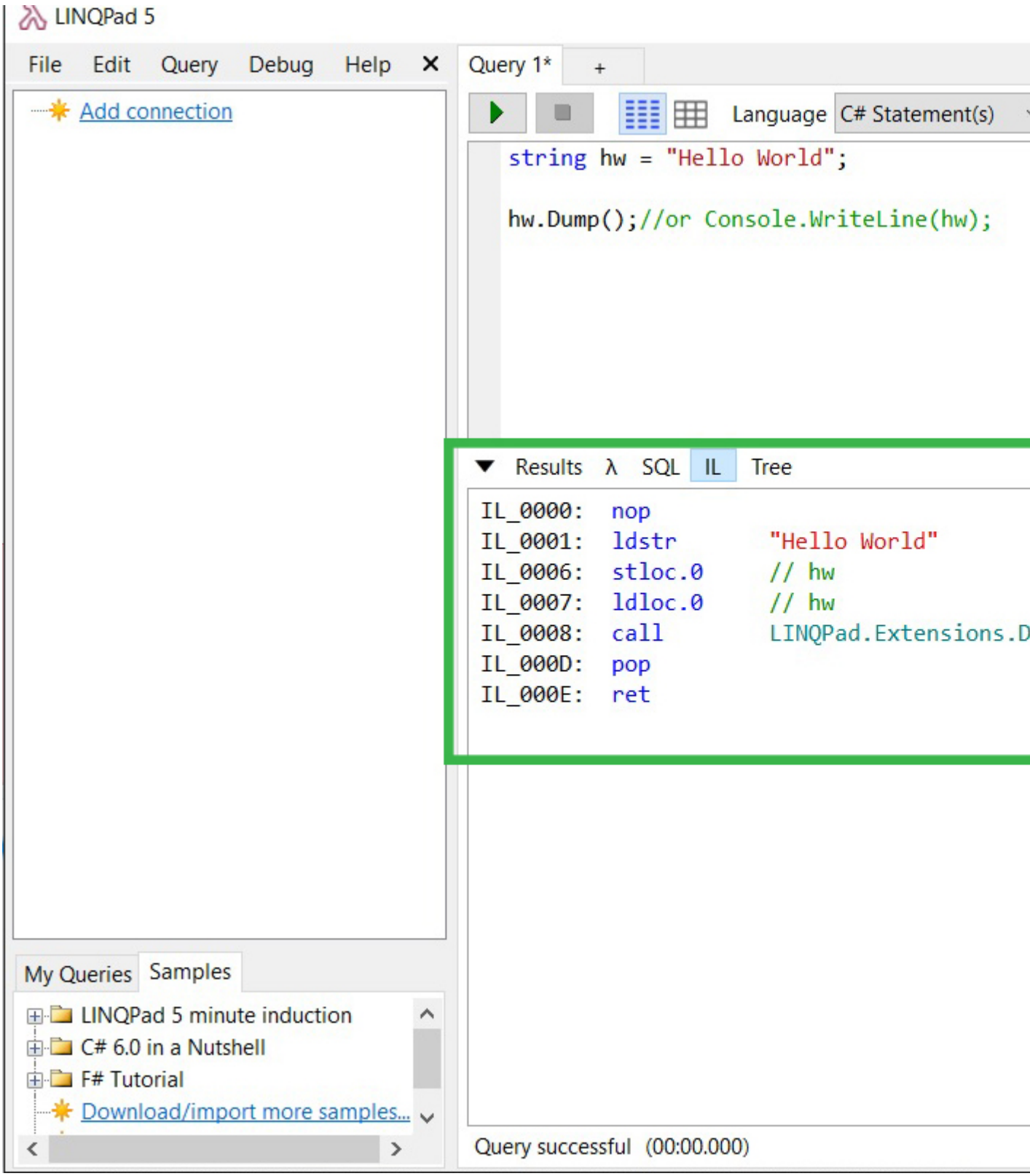


6. .Net "" LinqPad . .Net .


LINQPad 5
File
Edit
Query
Debug
Help
Add connection
My Queries
Samples
LINQPad 5 minute induction
C# 6.0 in a Nutshell
F# Tutorial
Query 1*
Language
C# Statement(s)
string hw = "Hello World";
hw.Dump();//or Console.WriteLine(hw);
Results
λ
SQL
IL
Tree
Hello World
Query successful (00:00.000)

:

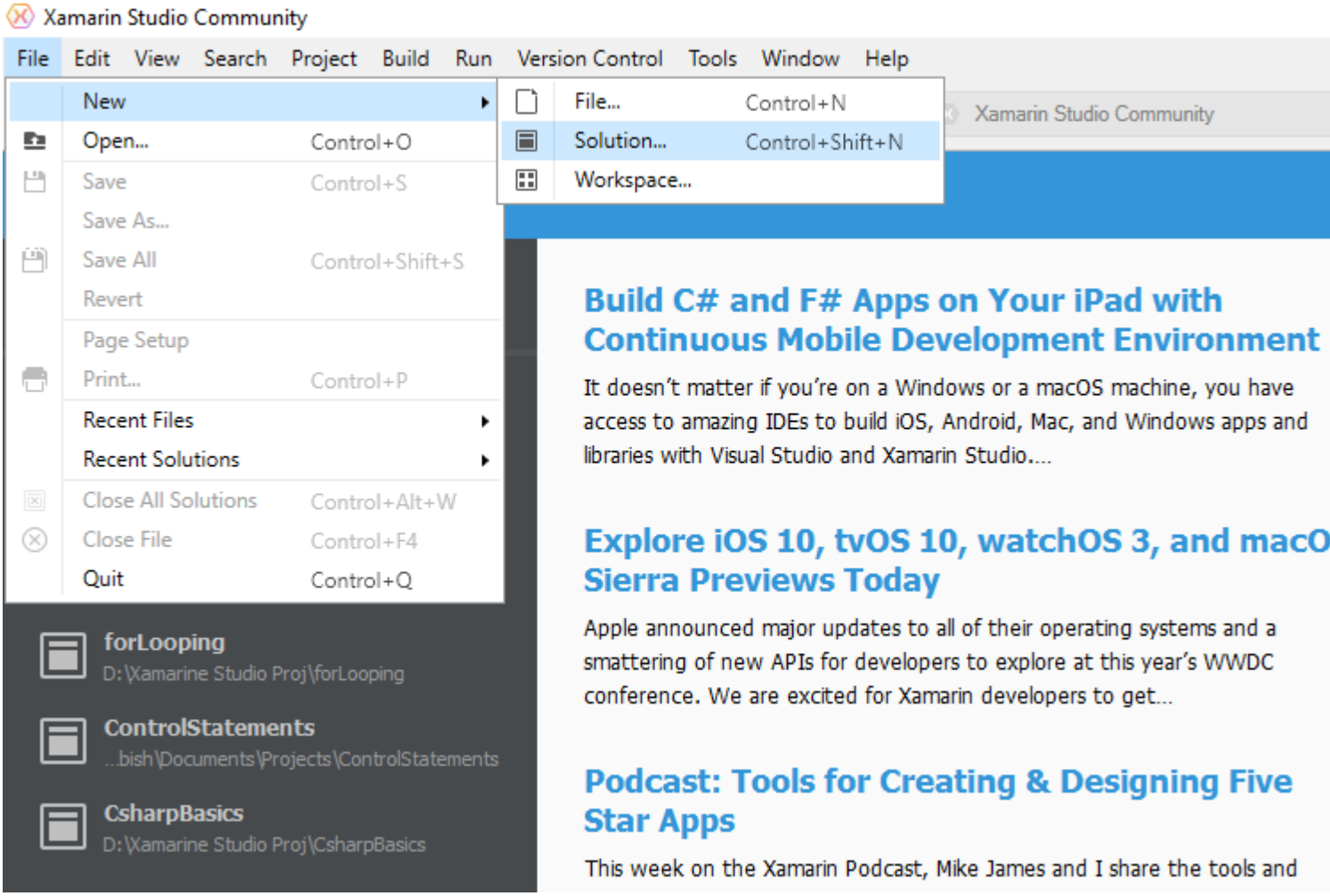
1. "" .net . .



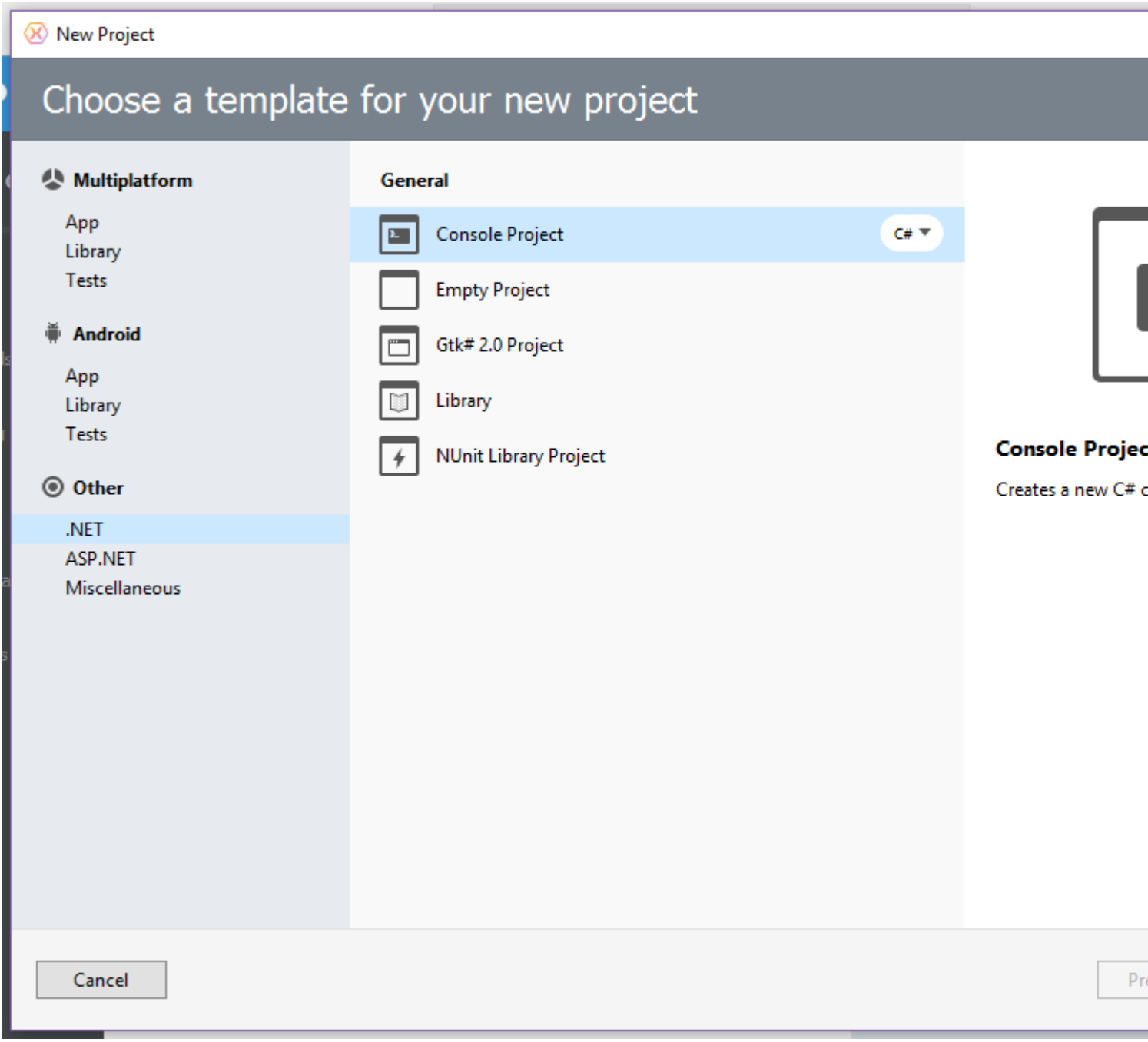
2. `LINQ to SQL` `Linq to Entities` SQL LINQ .

Xamarin Studio

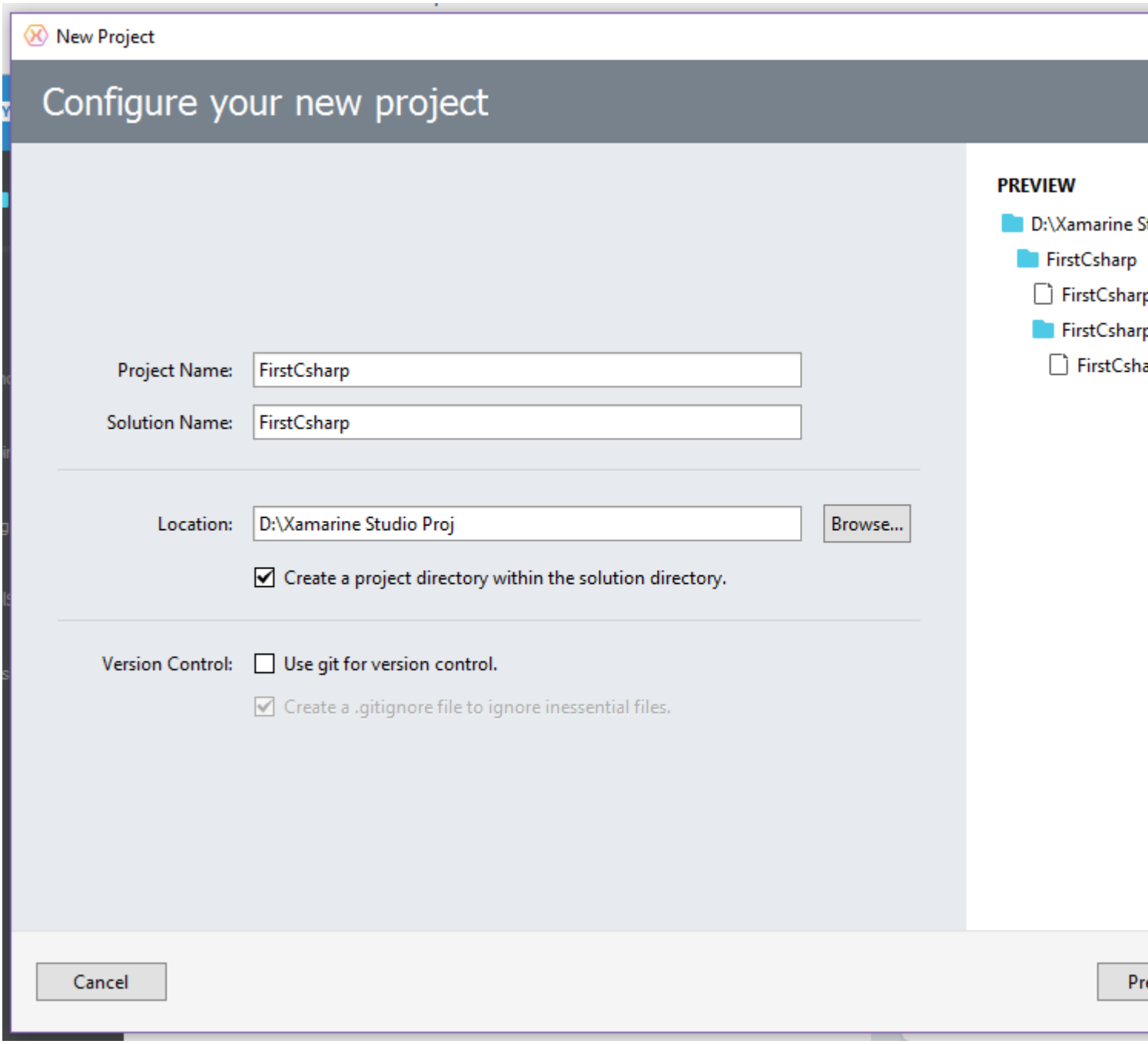
1. Xamarin Studio Community .
2. Xamarin Studio .
3. → → .


Xamarin Studio Community
File Edit View Search Project Build Run Version Control Tools Window Help
New
Open... Control+O
Save Control+S
Save As...
Save All Control+Shift+S
Revert
Page Setup
Print... Control+P
Recent Files
Recent Solutions
Close All Solutions Control+Alt+W
Close File Control+F4
Quit Control+Q
File... Control+N
Solution... Control+Shift+N
Workspace...
Xamarin Studio Community
forLooping
D:\Xamarine Studio Proj\forLooping
ControlStatements
...bish\Documents\Projects\ControlStatements
CsharpBasics
D:\Xamarine Studio Proj\CsharpBasics
Build C# and F# Apps on Your iPad with Continuous Mobile Development Environment
It doesn't matter if you're on a Windows or a macOS machine, you have access to amazing IDEs to build iOS, Android, Mac, and Windows apps and libraries with Visual Studio and Xamarin Studio....
Explore iOS 10, tvOS 10, watchOS 3, and macO Sierra Previews Today
Apple announced major updates to all of their operating systems and a smattering of new APIs for developers to explore at this year's WWDC conference. We are excited for Xamarin developers to get...
Podcast: Tools for Creating & Designing Five Star Apps
This week on the Xamarin Podcast, Mike James and I share the tools and

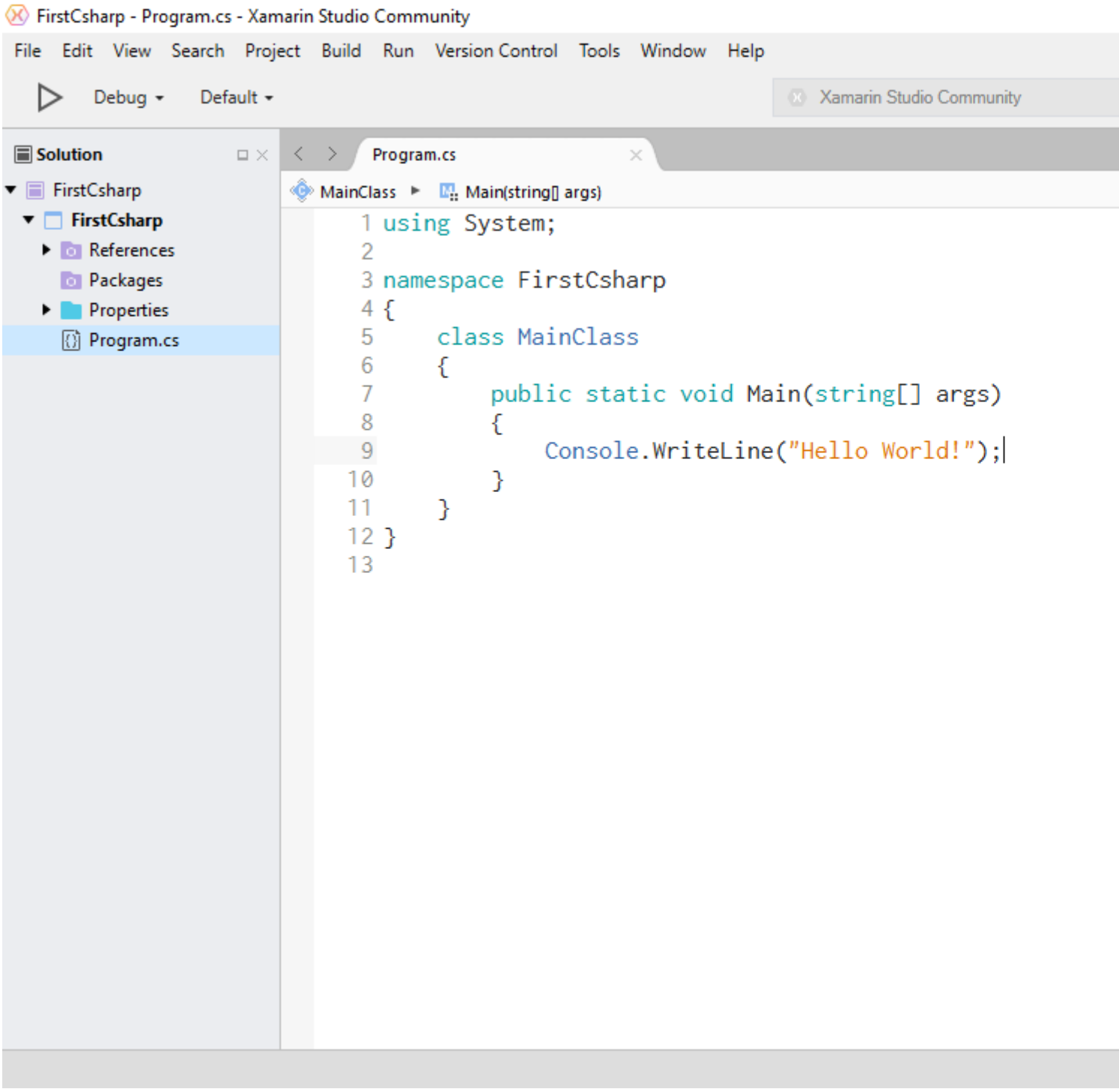
4. **.NET** → **C #** .
5. □□ .


New Project
Choose a template for your new project
Multiplatform
App
Library
Tests
Android
App
Library
Tests
Other
.NET
ASP.NET
Miscellaneous
General
Console Project
C#
Empty Project
Gtk# 2.0 Project
Library
NUnit Library Project
Console Projec
Creates a new C#
Cancel

6. □□□□ . . . □□□ .


New Project
Configure your new project
PREVIEW
D:\Xamarine St
FirstCsharp
FirstCsharp
FirstCsharp
FirstCsha
Project Name:
FirstCsharp
Solution Name:
FirstCsharp
Location:
D:\Xamarine Studio Proj
Browse...
Create a project directory within the solution directory.
Version Control:
Use git for version control.
Create a .gitignore file to ignore inessential files.
Cancel
Pr

7. .

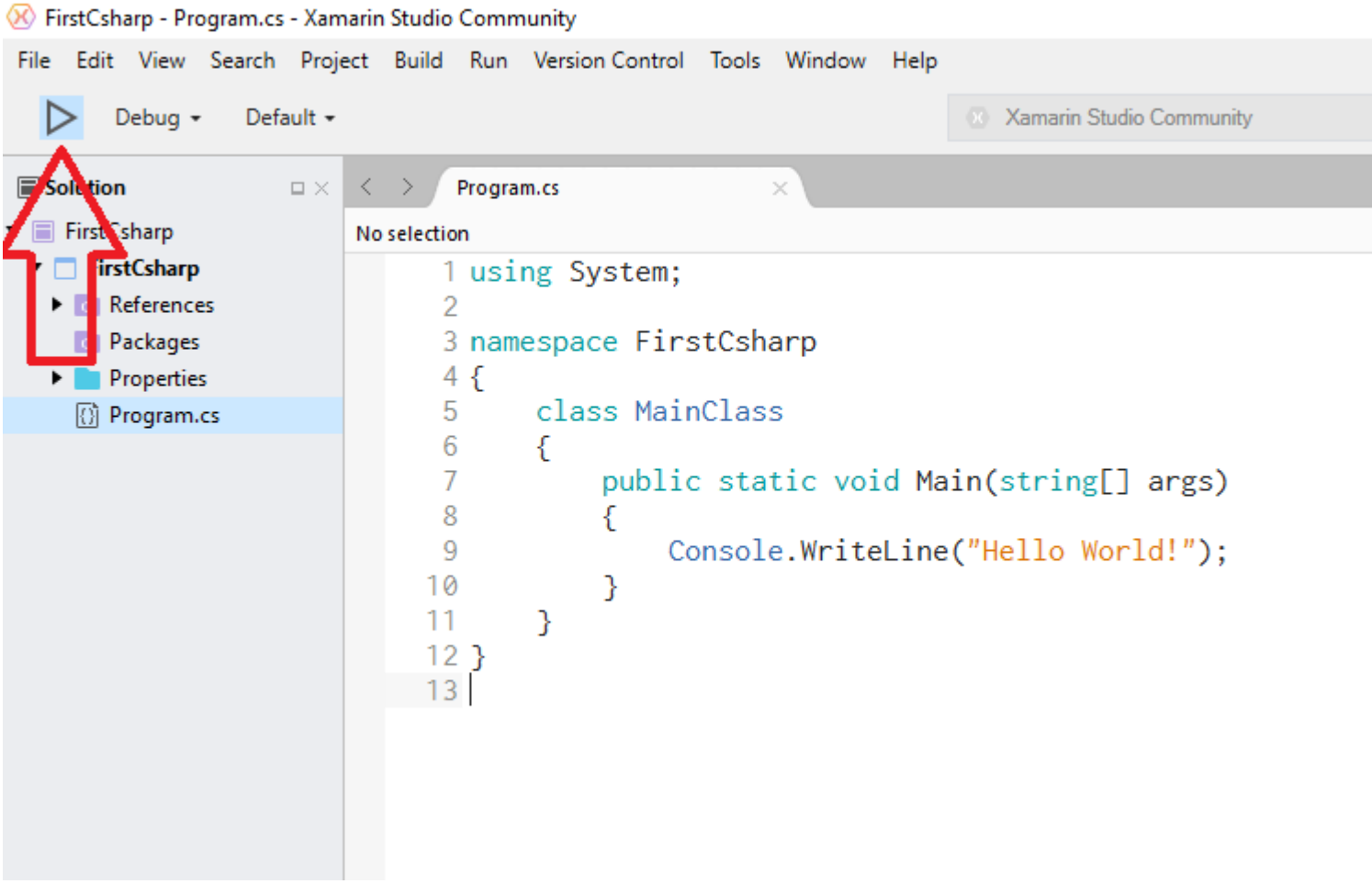


8. .

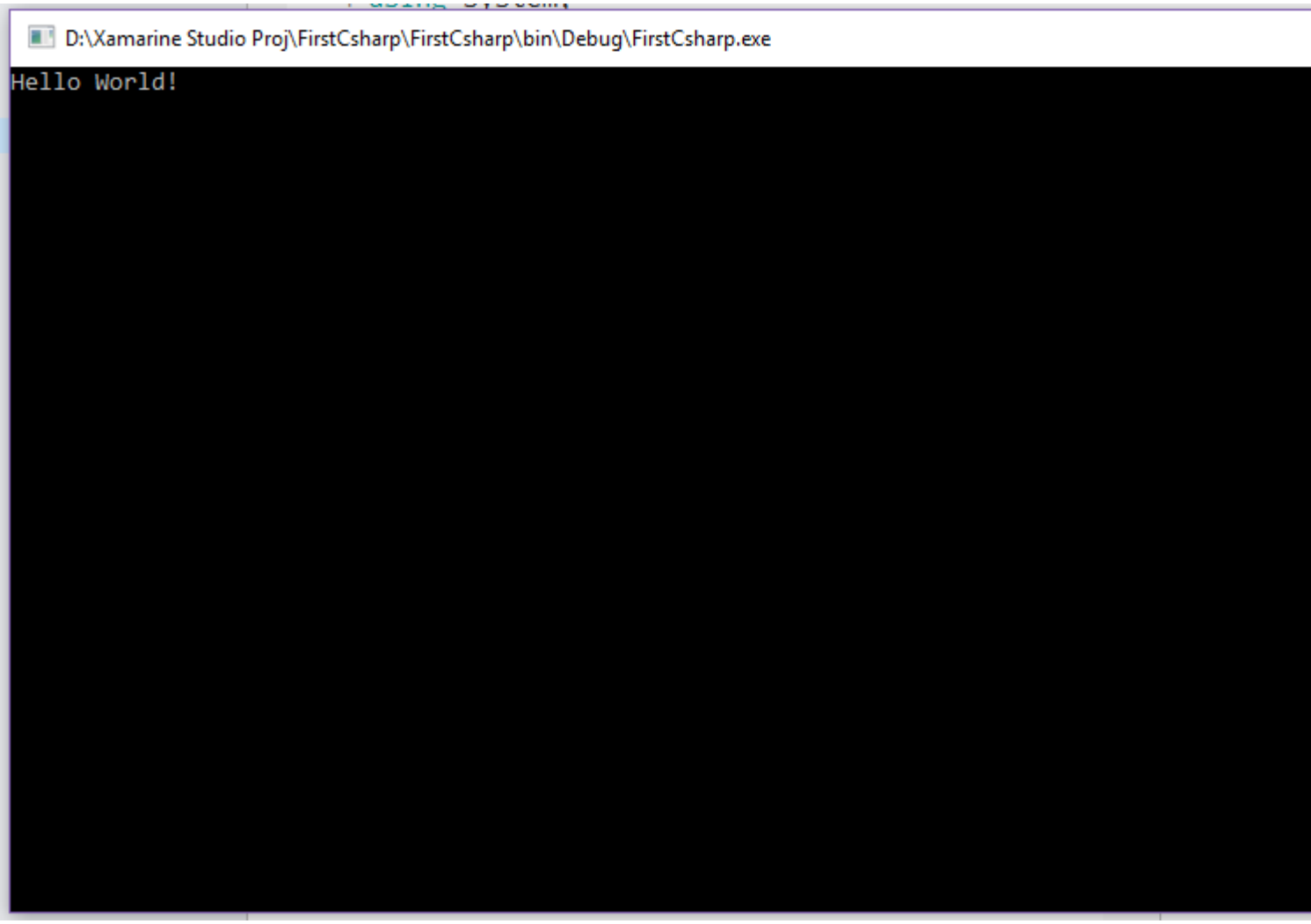
```
using System;

namespace FirstCsharp
{
    public class MainClass
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
            Console.ReadLine();
        }
    }
```

```
}
```

9. F5 □□ .


FirstCsharp - Program.cs - Xamarin Studio Community
File Edit View Search Project Build Run Version Control Tools Window Help
Debug
Default
Xamarin Studio Community
Solution
FirstCsharp
References
Packages
Properties
Program.cs
Program.cs
No selection
1 using System;
2
3 namespace FirstCsharp
4 {
5 class MainClass
6 {
7 public static void Main(string[] args)
8 {
9 Console.WriteLine("Hello World!");
10 }
11 }
12 }
13

10. .



C # : https://riptutorial.com/ko/csharp/topic/15/c-sharp--

2: .NET (Roslyn)

Examples

MSBuild

`Microsoft.CodeAnalysis.CSharp.Workspaces` Nuget .

```
var workspace = Microsoft.CodeAnalysis.MSBuild.MSBuildWorkspace.Create();
var project = await workspace.OpenProjectAsync(projectFilePath);
var compilation = await project.GetCompilationAsync();

foreach (var diagnostic in compilation.GetDiagnostics()
    .Where(d => d.Severity == Microsoft.CodeAnalysis.DiagnosticSeverity.Error))
{
    Console.WriteLine(diagnostic);
}
```

. . .

(Syntax Tree) , ( ).

`compilation Microsoft.CodeAnalysis.Compilation` . . (`DescendantNodes`) Linq .

```
foreach (var syntaxTree in compilation.SyntaxTrees)
{
    var root = await syntaxTree.GetRootAsync();
    var declaredIdentifiers = root.DescendantNodes()
        .Where(an => an is VariableDeclaratorSyntax)
        .Cast<VariableDeclaratorSyntax>()
        .Select(vd => vd.Identifier);

    foreach (var di in declaredIdentifiers)
    {
        Console.WriteLine(di);
    }
}
```

C # . Visual Studio `Syntax Visualizer` . Roslyn .

(Semantic Model) . . ( ).

```
var workspace = Microsoft.CodeAnalysis.MSBuild.MSBuildWorkspace.Create();
var sln = await workspace.OpenSolutionAsync(solutionFilePath);
var project = sln.Projects.First();
var compilation = await project.GetCompilationAsync();

foreach (var syntaxTree in compilation.SyntaxTrees)
{
    var root = await syntaxTree.GetRootAsync();

    var declaredIdentifiers = root.DescendantNodes()
```

```
        .Where(an => an is VariableDeclaratorSyntax)
        .Cast<VariableDeclaratorSyntax>();

    foreach (var di in declaredIdentifiers)
    {
        Console.WriteLine(di.Identifier);
        // => "root"

        var variableSymbol = compilation
            .GetSemanticModel(syntaxTree)
            .GetDeclaredSymbol(di) as ILocalSymbol;

        Console.WriteLine(variableSymbol.Type);
        // => "Microsoft.CodeAnalysis.SyntaxNode"

        var references = await SymbolFinder.FindReferencesAsync(variableSymbol, sln);
        foreach (var reference in references)
        {
            foreach (var loc in reference.Locations)
            {
                Console.WriteLine(loc.Location.SourceSpan);
                // => "[1375..1379)"
            }
        }
    }
}
```

. .

.NET (Roslyn) : https://riptutorial.com/ko/csharp/topic/4886/-net----roslyn-

3: .NET

Examples

Large Object Heap Object Heap `OutOfMemoryException` .

.NET 4.5.1 Large Object Heap .

```
GCSettings.LargeObjectHeapCompactionMode = GCLargeObjectHeapCompactionMode.CompactOnce;
GC.Collect();
```

(CLR) `GC` .

.NET GC . ( ) GC . .

GC . . . weak .

:

```
WeakReference reference = new WeakReference(new object(), false);

GC.Collect();

object target = reference.Target;
if (target != null)
  DoSomething(target);
```

. .

. .

.

```
Source.Event += new EventHandler(Handler)
```

. .NET . . .

.

:

```
public static class WeakEventManager
    {
    public static void SetHandler<S, TArgs>(
    Action<EventHandler<TArgs>> add,
    Action<EventHandler<TArgs>> remove,
    S subscriber,
    Action<S, TArgs> action)
    where TArgs : EventArgs
    where S : class
```

```
        {
            var subscrWeakRef = new WeakReference(subscriber);
            EventHandler<TArgs> handler = null;

            handler = (s, e) =>
            {
                var subscrStrongRef = subscrWeakRef.Target as S;
                if (subscrStrongRef != null)
                {
                    action(subscrStrongRef, e);
                }
                else
                {
                    remove(handler);
                    handler = null;
                }
            };

            add(handler);
        }
    }
```

:

```
 EventSource s = new EventSource();
 Subscriber subscriber = new Subscriber();
 WeakEventManager.SetHandler<Subscriber, SomeEventArgs>(a => s.Event += a, r => s.Event -= r,
subscriber, (s,e) => { s.HandleEvent(e); });
```

.

```
public event EventHandler<SomeEventArgs> Event;
```

MSDN :

- weak .
- .
- . .

.NET : https://riptutorial.com/ko/csharp/topic/1287/-net--

4: .NET

- .Net `unsafe` => " "
- (`unsafe`).

for .

```
for (int i = 0; i < array.Length; i++)
{
    array[i] = 0;
}
```

.NET Framework `IndexOutOfRangeException` .

.

```
unsafe
{
    fixed (int* ptr = array)
    {
        for (int i = 0; i <= array.Length; i++)
        {
            *(ptr+i) = 0;
        }
    }
}
```

Examples

```
void Main()
{
    unsafe
    {
        int[] a = {1, 2, 3};
        fixed(int* b = a)
        {
            Console.WriteLine(b[4]);
        }
    }
}
```

3 (4) . `1910457872` .

`unsafe` throw .

`IndexOutOfRangeException` . .

:

```
class Program
{
```

```
    static void Main(string[] args)
    {
        unsafe
        {
            int[] array = new int[1000];
            fixed (int* ptr = array)
            {
                for (int i = 0; i < array.Length; i++)
                {
                    *(ptr+i) = i; //assigning the value with the pointer
                }
            }
        }
    }
}
```

:

```
class Program
{
    static void Main(string[] args)
    {
        int[] array = new int[1000];

        for (int i = 0; i < array.Length; i++)
        {
            array[i] = i;
        }
    }
}
```

. , .

```
var s = "Hello";       // The string referenced by variable 's' is normally immutable, but
                       // since it is memory, we could change it if we can access it in an
                       // unsafe way.

unsafe                 // allows writing to memory; methods on System.String don't allow this
{
  fixed (char* c = s) // get pointer to string originally stored in read only memory
    for (int i = 0; i < s.Length; i++)
      c[i] = 'a';      // change data in memory allocated for original string "Hello"
}
Console.WriteLine(s); // The variable 's' still refers to the same System.String
                       // value in memory, but the contents at that location were
                       // changed by the unsafe write above.
                       // Displays: "aaaaa"
```

.NET : https://riptutorial.com/ko/csharp/topic/81/-net---

5: .zip

1. ZipArchive OpenRead (string archiveFileName)

archiveFileName , . .	

Examples

zip

.zip .

```
System.IO.Compression
System.IO.Compression.FileSystem

using (FileStream zipToOpen = new FileStream(@"C:\temp", FileMode.Open))
{
    using (ZipArchive archive = new ZipArchive(zipToOpen, ZipArchiveMode.Update))
    {
        ZipArchiveEntry readmeEntry = archive.CreateEntry("Readme.txt");
        using (StreamWriter writer = new StreamWriter(readmeEntry.Open()))
        {
            writer.WriteLine("Information about this package.");
            writer.WriteLine("========================");
        }
    }
}
```

Zip

byte[] .

```
public static byte[] ZipFiles(Dictionary<string, byte[]> files)
{
    using (MemoryStream ms = new MemoryStream())
    {
        using (ZipArchive archive = new ZipArchive(ms, ZipArchiveMode.Update))
        {
            foreach (var file in files)
            {
                ZipArchiveEntry orderEntry = archive.CreateEntry(file.Key); //create a file
with this name
                using (BinaryWriter writer = new BinaryWriter(orderEntry.Open()))
                {
                    writer.Write(file.Value); //write the binary data
                }
            }
        }
        //ZipArchive must be disposed before the MemoryStream has data
        return ms.ToArray();
    }
```

```
}
```

Zip

zip .

```
public static Dictionary<string, byte[]> GetFiles(byte[] zippedFile)
{
    using (MemoryStream ms = new MemoryStream(zippedFile))
    using (ZipArchive archive = new ZipArchive(ms, ZipArchiveMode.Read))
    {
        return archive.Entries.ToDictionary(x => x.FullName, x => ReadStream(x.Open()));
    }
}

private static byte[] ReadStream(Stream stream)
{
    using (var ms = new MemoryStream())
    {
        stream.CopyTo(ms);
        return ms.ToArray();
    }
}
```

zip .txt

```
using System;
using System.IO;
using System.IO.Compression;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            string zipPath = @"c:\example\start.zip";
            string extractPath = @"c:\example\extract";

            using (ZipArchive archive = ZipFile.OpenRead(zipPath))
            {
                foreach (ZipArchiveEntry entry in archive.Entries)
                {
                    if (entry.FullName.EndsWith(".txt", StringComparison.OrdinalIgnoreCase))
                    {
                        entry.ExtractToFile(Path.Combine(extractPath, entry.FullName));
                    }
                }
            }
        }
    }
}
```

.zip : https://riptutorial.com/ko/csharp/topic/6709/-zip----

6: ASP.NET ID

, , asp.net ID .

Examples

asp.net ID

1. MyClasses .

```
public class GmailEmailService:SmtpClient
{
    // Gmail user-name
    public string UserName { get; set; }

    public GmailEmailService() :
        base(ConfigurationManager.AppSettings["GmailHost"],
Int32.Parse(ConfigurationManager.AppSettings["GmailPort"]))
    {
        //Get values from web.config file:
        this.UserName = ConfigurationManager.AppSettings["GmailUserName"];
        this.EnableSsl = Boolean.Parse(ConfigurationManager.AppSettings["GmailSsl"]);
        this.UseDefaultCredentials = false;
        this.Credentials = new System.Net.NetworkCredential(this.UserName,
ConfigurationManager.AppSettings["GmailPassword"]);
    }
}
```

2. ID

```
public async Task SendAsync(IdentityMessage message)
{
    MailMessage email = new MailMessage(new MailAddress("youremailadress@domain.com",
"(any subject here)"),
    new MailAddress(message.Destination));
    email.Subject = message.Subject;
    email.Body = message.Body;

    email.IsBodyHtml = true;

    GmailEmailService mailClient = new GmailEmailService();
    await mailClient.SendMailAsync(email);
}
```

3. web.config . Gmail Gmail .

```
<add key="GmailUserName" value="youremail@yourdomain.com"/>
<add key="GmailPassword" value="yourPassword"/>
<add key="GmailHost" value="yourServer"/>
<add key="GmailPort" value="yourPort"/>
<add key="GmailSsl" value="chooseTrueOrFalse"/>
<!--Smptp Server (confirmations emails)-->
```

4.

. .

```
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.EntityFramework;
using Microsoft.AspNet.Identity.Owin;
using Microsoft.Owin.Security;
using Microsoft.Owin.Security.DataProtection;
using System;
using System.Linq;
using System.Threading.Tasks;
using System.Web;
using System.Web.Mvc;
using MYPROJECT.Models;

namespace MYPROJECT.Controllers
{
    [Authorize]
    public class AccountController : Controller
    {
        private ApplicationSignInManager _signInManager;
        private ApplicationUserManager _userManager;
        ApplicationDbContext _context;
        DpapiDataProtectionProvider provider = new DpapiDataProtectionProvider("MYPROJECT");
        EmailService EmailService = new EmailService();


        public AccountController()
            : this(new UserManager<ApplicationUser>(new UserStore<ApplicationUser>(new ApplicationDbContext
        {
            _context = new ApplicationDbContext();
        }

        public AccountController(UserManager<ApplicationUser> userManager, ApplicationSignInManager signIn
        {
            UserManager = userManager;
            SignInManager = signInManager;
            UserManager.UserTokenProvider = new DataProtectorTokenProvider<ApplicationUser>(
                    provider.Create("EmailConfirmation"));
        }

        private AccountController(UserManager<ApplicationUser> userManager)
        {
            UserManager = userManager;
            UserManager.UserTokenProvider = new DataProtectorTokenProvider<ApplicationUser>(
                    provider.Create("EmailConfirmation"));
        }

        public UserManager<ApplicationUser> UserManager { get; private set; }

        public ApplicationSignInManager SignInManager
        {
            get
            {
                return _signInManager ?? HttpContext.GetOwinContext().Get<ApplicationSignInManager>();
            }
            private set
            {
                _signInManager = value;
            }
        }
```

```
//
// GET: /Account/ForgotPassword
[AllowAnonymous]
public ActionResult ForgotPassword()
{
    return View();
}

//
// POST: /Account/ForgotPassword
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public async Task<ActionResult> ForgotPassword(ForgotPasswordViewModel model)
{
    if (ModelState.IsValid)
    {

        var user = await UserManager.FindByNameAsync(model.UserName);

        if (user == null || !(await UserManager.IsEmailConfirmedAsync(user.Id)))
        {
            // Don't reveal that the user does not exist or is not confirmed
            return View("ForgotPasswordConfirmation");
        }

        // For more information on how to enable account confirmation and password reset please visit http
        // Send an email with this link
        string code = await UserManager.GeneratePasswordResetTokenAsync(user.Id);
        code = HttpUtility.UrlEncode(code);
        var callbackUrl = Url.Action("ResetPassword", "Account", new { userId = user.Id, code = HttpUtility
        await EmailService.SendAsync(new IdentityMessage
        {
            Body = "Please reset your password by clicking <a href=\"" + callbackUrl + "\">here</a>",
            Destination = user.Email,
            Subject = "Reset Password"
        });
        //await UserManager.SendEmailAsync(user.Id, "Reset Password", "Please reset your password by clicki
        return RedirectToAction("ForgotPasswordConfirmation", "Account");
    }

    // If we got this far, something failed, redisplay form
    return View(model);
}
```

. !

ASP.NET ID : https://riptutorial.com/ko/csharp/topic/9577/asp-net-id

7: AssemblyInfo.cs

AssemblyInfo.cs .

Examples

[AssemblyTitle]

.

```
[assembly: AssemblyTitle("MyProduct")]
```

[ ]

. , .

```
[assembly: AssemblyProduct("MyProduct")]
```

AssemblyInfo

DRYness AssemblyInfo.cs . .

GlobalAssemblyInfo.cs

```
using System.Reflection;
using System.Runtime.InteropServices;
//using Stackoverflow domain as a made up example

// It is common, and mostly good, to use one GlobalAssemblyInfo.cs that is added
// as a link to many projects of the same product, details below
// Change these attribute values in local assembly info to modify the information.
[assembly: AssemblyProduct("Stackoverflow Q&A")]
[assembly: AssemblyCompany("Stackoverflow")]
[assembly: AssemblyCopyright("Copyright © Stackoverflow 2016")]

// The following GUID is for the ID of the typelib if this project is exposed to COM
[assembly: Guid("4e4f2d33-aaab-48ea-a63d-1f0a8e3c935f")]
[assembly: ComVisible(false)] //not going to expose ;)

// Version information for an assembly consists of the following four values:
// roughly translated from I reckon it is for SO, note that they most likely
// dynamically generate this file
//      Major Version  - Year 6 being 2016
//      Minor Version  - The month
//      Day Number     - Day of month
//      Revision       - Build number
// You can specify all the values or you can default the Build and Revision Numbers
// by using the '*' as shown below: [assembly: AssemblyVersion("year.month.day.*")]
[assembly: AssemblyVersion("2016.7.00.00")]
[assembly: AssemblyFileVersion("2016.7.27.3839")]
```

AssemblyInfo.cs -

```
//then the following might be put into a separate Assembly file per project, e.g.
[assembly: AssemblyTitle("Stackoveflow.Redis")]
```

GlobalAssemblyInfo.cs .

1. Add / Existing Item ... .
2. GlobalAssemblyInfo.cs .
3. Add-Button .
4. " " .

[AssemblyVersion]

.

```
[assembly: AssemblyVersion("1.0.*")]
```

`*` ("")

.NET API . , `this` title .

```
using System.Linq;
using System.Reflection;

...

Assembly assembly = typeof(this).Assembly;
var titleAttribute = assembly.GetCustomAttributes<AssemblyTitleAttribute>().FirstOrDefault();

Console.WriteLine($"This assembly title is {titleAttribute?.Title}");
```

(SVN ids Git SHA1) (Git) . AssemblyInfo.cs AssemblyInfo.cs .

GitVersionTask SemVer.Git.Fody NuGet . GitVersionTask AssemblyInfo.cs `Assembly*Version` . GitVersionTask .

Semantic Versioning SemVer .

AssemblyInfo . (Windows 10) .

.

- AssemblyTitle - (*:* MyCompany.MySolution.MyProject).
- AssemblyCompany -
- AssemblyProduct -
- AssemblyCopyright - .

'AssemblyTitle' DLL Properties Details 'File description'.

[AssemblyConfiguration]

AssemblyConfiguration : AssemblyConfiguration . . . .

```
#if (DEBUG)

[assembly: AssemblyConfiguration("Debug")]

#else

[assembly: AssemblyConfiguration("Release")]

#endif
```

[InternalsVisibleTo]

`internal InternalsVisibleTo .`

`MyAssembly.UnitTests internal MyAssembly .`

```
[assembly: InternalsVisibleTo("MyAssembly.UnitTests")]
```

`public .`

[AssemblyKeyFile]

GAC . . `.snk` .

1. VS2015 ( )
2. cd C : \ Directory_Name Enter .
3. sn -k KeyFileName.snk Enter .

keyFileName.snk . `AssemblyKeyFileAttribute snk` .

-> AssemblyInfo.cs

```
[assembly: AssemblyKeyFile(@"c:\Directory_Name\KeyFileName.snk")]
```

Thi . GAC .

:)

AssemblyInfo.cs : https://riptutorial.com/ko/csharp/topic/4264/assemblyinfo-cs-

8: BackgroundWorker

- `bgWorker.CancellationPending //returns whether the bgWorker was cancelled during its operation`
- `bgWorker.IsBusy //returns true if the bgWorker is in the middle of an operation`
- `bgWorker.ReportProgress(int x) //Reports a change in progress. Raises the "ProgressChanged" event`
- `bgWorker.RunWorkerAsync() //Starts the BackgroundWorker by raising the "DoWork" event`
- `bgWorker.CancelAsync() //instructs the BackgroundWorker to stop after the completion of a task.`

UI . . UI .

BackgroundWorker UI Control.Invoke UI . .

BackgroundWorker Windows Forms . WPF (/). UI INotifyPropertyChanged UI Dispatcher .

Examples

BackgroundWorker

BackgroundWorker .

```
    /* This is the backgroundworker's "DoWork" event handler. This
       method is what will contain all the work you
       wish to have your program perform without blocking the UI. */

    bgWorker.DoWork += bgWorker_DoWork;


    /*This is how the DoWork event method signature looks like:*/
    private void bgWorker_DoWork(object sender, DoWorkEventArgs e)
    {
        // Work to be done here
        // ...
        // To get a reference to the current Backgroundworker:
        BackgroundWorker worker = sender as BackgroundWorker;
        // The reference to the BackgroundWorker is often used to report progress
        worker.ReportProgress(...);
    }

    /*This is the method that will be run once the BackgroundWorker has completed its tasks */

    bgWorker.RunWorkerCompleted += bgWorker_CompletedWork;

    /*This is how the RunWorkerCompletedEvent event method signature looks like:*/
    private void bgWorker_CompletedWork(object sender, RunWorkerCompletedEventArgs e)
    {
        // Things to be done after the backgroundworker has finished
    }
```

```
   /* When you wish to have something occur when a change in progress
     occurs, (like the completion of a specific task) the "ProgressChanged"
     event handler is used. Note that ProgressChanged events may be invoked
     by calls to bgWorker.ReportProgress(...) only if bgWorker.WorkerReportsProgress
     is set to true.  */

     bgWorker.ProgressChanged += bgWorker_ProgressChanged;

    /*This is how the ProgressChanged event method signature looks like:*/
    private void bgWorker_ProgressChanged(object sender, ProgressChangedEventArgs e)
    {
        // Things to be done when a progress change has been reported

        /* The ProgressChangedEventArgs gives access to a percentage,
         allowing for easy reporting of how far along a process is*/
        int progress = e.ProgressPercentage;
    }
```

BackgroundWorker

BackgroundWorker .

```
bgWorker.WorkerSupportsCancellation = true;
```

...

```
bgWorker.WorkerReportsProgress = true;

//this must also be used in conjunction with the ProgressChanged event
```

BackgroundWorker

BackgroundWorker UI .

```
// BackgroundWorker is part of the ComponentModel namespace.
using System.ComponentModel;

namespace BGWorkerExample
{
     public partial class ExampleForm : Form
     {

      // the following creates an instance of the BackgroundWorker named "bgWorker"
      BackgroundWorker bgWorker = new BackgroundWorker();

      public ExampleForm() { ...
```

BackgroundWorker .

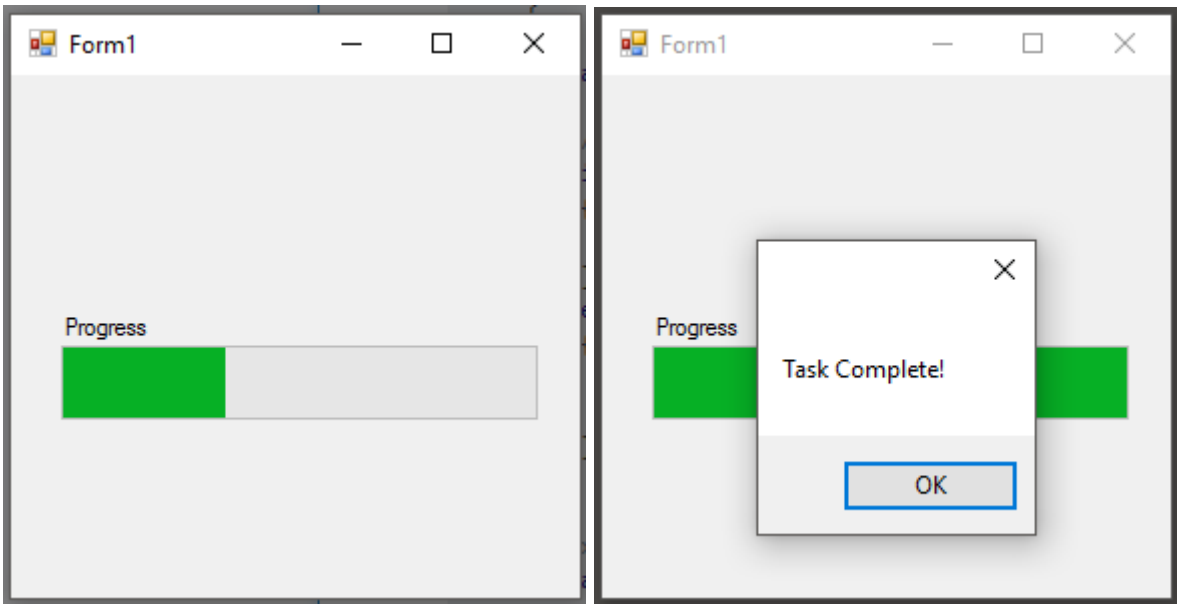
BackgroundWorker WinForms ProgressBar . backgroundWorker UI UI .

```
namespace BgWorkerExample
```

```
{
    public partial class Form1 : Form
{

    //a new instance of a backgroundWorker is created.
    BackgroundWorker bgWorker = new BackgroundWorker();

    public Form1()
    {
        InitializeComponent();

        prgProgressBar.Step = 1;

        //this assigns event handlers for the backgroundWorker
        bgWorker.DoWork += bgWorker_DoWork;
        bgWorker.RunWorkerCompleted += bgWorker_WorkComplete;

        //tell the backgroundWorker to raise the "DoWork" event, thus starting it.
        //Check to make sure the background worker is not already running.
        if(!bgWorker.IsBusy)
            bgWorker.RunWorkerAsync();

    }

    private void bgWorker_DoWork(object sender, DoWorkEventArgs e)
    {
        //this is the method that the backgroundworker will perform on in the background
thread.
        /* One thing to note! A try catch is not necessary as any exceptions will terminate
the backgroundWorker and report
          the error to the "RunWorkerCompleted" event */
        CountToY();
    }

    private void bgWorker_WorkComplete(object sender, RunWorkerCompletedEventArgs e)
    {
        //e.Error will contain any exceptions caught by the backgroundWorker
        if (e.Error != null)
        {
            MessageBox.Show(e.Error.Message);
        }
        else
        {
            MessageBox.Show("Task Complete!");
            prgProgressBar.Value = 0;
        }
    }

    // example method to perform a "long" running task.
    private void CountToY()
    {
        int x = 0;

        int maxProgress = 100;
        prgProgressBar.Maximum = maxProgress;


        while (x < maxProgress)
        {
            System.Threading.Thread.Sleep(50);
            Invoke(new Action(() => { prgProgressBar.PerformStep(); }));
```

```
                x += 1;
            }
        }


}
```

...



BackgroundWorker : https://riptutorial.com/ko/csharp/topic/1588/backgroundworker

9: BigInteger

?

`BigInteger` RAM . , , .

, , ., `BigInteger` RAM .

`Byte[]` . .

Examples

1000

`using System.Numerics` `System.Numerics` .

```
using System;
using System.Numerics;

namespace Euler_25
{
    class Program
    {
        static void Main(string[] args)
        {
            BigInteger l1 = 1;
            BigInteger l2 = 1;
            BigInteger current = l1 + l2;
            while (current.ToString().Length < 1000)
            {
                l2 = l1;
                l1 = current;
                current = l1 + l2;
            }
            Console.WriteLine(current);
        }
    }
}
```

1000 . `ulong` .

`BigInteger` RAM .

: `BigInteger` .NET 4.0 .

BigInteger : https://riptutorial.com/ko/csharp/topic/5654/biginteger

10: C # 3.0

C # 3.0 .Net 3.5 . LINQ (Language INtegrated Queries) .

:

- LINQ
-
-
-
-
-
-
-

Examples

(var)

var . var .

```
var squaredNumber = 10 * 10;
var squaredNumberDouble = 10.0 * 10.0;
var builder = new StringBuilder();
var anonymousObject = new
{
    One = SquaredNumber,
    Two = SquaredNumberDouble,
    Three = Builder
}
```

int , double , StringBuilder .

var . SquaredNumber = Builder int StringBuilder SquaredNumber = Builder .

(LINQ)

```
//Example 1
int[] array = { 1, 5, 2, 10, 7 };

// Select squares of all odd numbers in the array sorted in descending order
IEnumerable<int> query = from x in array
                         where x % 2 == 1
                         orderby x descending
                         select x * x;
// Result: 49, 25, 1
```

C # 3.0, LINQ

1 SQL .

```
//Example 2
IEnumerable<int> query = array.Where(x => x % 2 == 1)
    .OrderByDescending(x => x)
    .Select(x => x * x);
// Result: 49, 25, 1 using 'array' as defined in previous example
```

C # 3.0, LINQ

2 1 .

C # LINQ LINQ . . . MSDN - " , ."

Lambda Expresions . .

```
using System;
using System.Collections.Generic;
using System.Linq;

public class Program
{
    public static void Main()
    {
        var numberList = new List<int> {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
        var sumOfSquares = numberList.Select( number => number * number )
            .Aggregate( (int first, int second) => { return first + second; } );
        Console.WriteLine( sumOfSquares );
    }
}
```

1 10 .

. . .

```
.Select( (number) => number * number);
```

.

```
.Select( (int number) => number * number);
```

. . lambdas .

```
.Select( number => { return number * number; } );
```

select IEnumerable .

select . . . .

```
.Aggregate( (first, second) => { return first + second; } );
```

:

```
.Aggregate( (int first, int second) => first + second );
```

. . .

new ({) . .

```
var v = new { Amount = 108, Message = "Hello" };
```

. .

```
var a = new[] {
    new {
        Fruit = "Apple",
        Color = "Red"
    },
    new {
        Fruit = "Banana",
        Color = "Yellow"
    }
};
```

LINQ .

```
var productQuery = from prod in products
                   select new { prod.Color, prod.Price };
```

C # 3.0 : https://riptutorial.com/ko/csharp/topic/3820/c-sharp-3-0-

11: C # 4.0

Examples

. . . a

1. .
2. enum struct .
3. default (valueType) .

.

:

```
public void ExampleMethod(int required, string optValue = "test", int optNum = 42)
{
    //...
}
```

MSDN ,

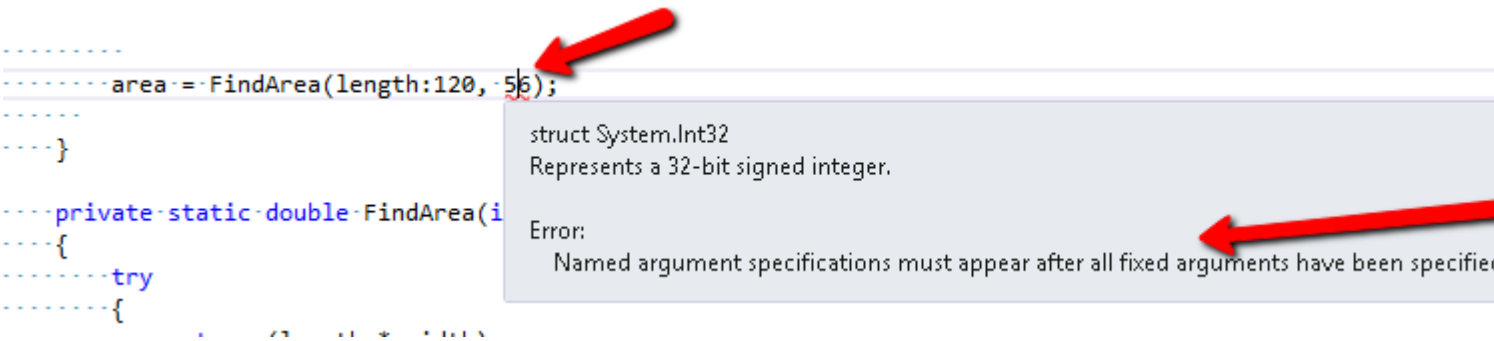
. . . .

:

```
// required = 3, optValue = "test", optNum = 4
ExampleMethod(3, optNum: 4);
// required = 2, optValue = "foo", optNum = 42
ExampleMethod(2, optValue: "foo");
// required = 6, optValue = "bar", optNum = 1
ExampleMethod(optNum: 1, optValue: "bar", required: 6);
```

.

.



.

out in (covariant) (contravariant) . , .

, IEnumerable<T> .

```
interface IEnumerable<out T>
{
    IEnumerator<T> GetEnumerator();
}
```

IComparer .

```
public interface IComparer<in T>
{
    int Compare(T x, T y);
}
```

COM ref

COM ref . COM .

```
void Increment(ref int x);
```

.

```
Increment(0); // no need for "ref" or a place holder variable any more
```

dynamic C# . System.Object System.Object (, ,) , . . :

```
// Returns the value of Length property or field of any object
int GetLength(dynamic obj)
{
    return obj.Length;
}

GetLength("Hello, world");        // a string has a Length property,
GetLength(new int[] { 1, 2, 3 }); // and so does an array,
GetLength(42);                    // but not an integer - an exception will be thrown
                                  // in GetLength method at run-time
```

Reflection . .

.

```
// Initialize the engine and execute a file
var runtime = ScriptRuntime.CreateFromConfiguration();
dynamic globals = runtime.Globals;
runtime.ExecuteFile("Calc.rb");

// Use Calc type from Ruby
dynamic calc = globals.Calc.@new();
calc.valueA = 1337;
calc.valueB = 666;
dynamic answer = calc.Calculate();
```

. Visitor .

C # 4.0 : https://riptutorial.com/ko/csharp/topic/3093/c-sharp-4-0-

12: C # 5.0

-
- MyTask **Async** () {doSomething (); }

  MyTaskAsync () .
- `<string>` MyStringTask **Async** () {return getSomeString (); }

  string MyString = MyStringTaskAsync () .
-
- MyCallerAttributes (MyMessage,

  [CallerMemberName] MemberName = "",

  [CallerFilePath] string SourceFilePath = "",

  [CallerLineNumber] int LineNumber = 0)
- Trace.WriteLine (" :"+ MyMessage);

  Trace.WriteLine ("Member :"+ MemberName);

  Trace.WriteLine ("   :"+ SourceFilePath);

  Trace.WriteLine ("Line Number :"+ LineNumber);

/	
Type<T> T .	

C # 5.0 Visual Studio .NET 2012 .

Examples

`async await` .

.

```
//This method is async because:
//1. It has async and Task or Task<T> as modifiers
//2. It ends in "Async"
async Task<int> GetStringLengthAsync(string URL){
    HttpClient client = new HttpClient();
    //Sends a GET request and returns the response body as a string
    Task<string> getString = client.GetStringAsync(URL);
    //Waits for getString to complete before returning its length
```

```
    string contents = await getString;
    return contents.Length;
}

private async void doProcess(){
    int length = await GetStringLengthAsync("http://example.com/");
    //Waits for all the above to finish before printing the number
    Console.WriteLine(length);
}
```

( ).

1 :

```
//This one using async event handlers, but not async coupled with await
private void DownloadAndUpdateAsync(string uri, string DownloadLocation){
    WebClient web = new WebClient();
    //Assign the event handler
    web.DownloadProgressChanged += new DownloadProgressChangedEventHandler(ProgressChanged);
    web.DownloadFileCompleted += new AsyncCompletedEventHandler(FileCompleted);
    //Download the file asynchronously
    web.DownloadFileAsync(new Uri(uri), DownloadLocation);
}

//event called for when download progress has changed
private void ProgressChanged(object sender, DownloadProgressChangedEventArgs e){
    //example code
    int i = 0;
    i++;
    doSomething();
}

//event called for when download has finished
private void FileCompleted(object sender, AsyncCompletedEventArgs e){
    Console.WriteLine("Completed!")
}
```

2 :

```
//however, this one does
//Refer to first example on why this method is async
private void DownloadAndUpdateAsync(string uri, string DownloadLocation){
    WebClient web = new WebClient();
    //Assign the event handler
    web.DownloadProgressChanged += new DownloadProgressChangedEventHandler(ProgressChanged);
    //Download the file async
    web.DownloadFileAsync(new Uri(uri), DownloadLocation);
    //Notice how there is no complete event, instead we're using techniques from the first
example
}
private void ProgressChanged(object sender, DownloadProgressChangedEventArgs e){
    int i = 0;
    i++;
    doSomething();
}
private void doProcess(){
    //Wait for the download to finish
    await DownloadAndUpdateAsync(new Uri("http://example.com/file"))
```

```
    doSomething();
}
```

CIA . 3 .

:

```
//This is the "calling method": the method that is calling the target method
public void doProcess()
{
    GetMessageCallerAttributes("Show my attributes.");
}
//This is the target method
//There are only 3 caller attributes
public void GetMessageCallerAttributes(string message,
    //gets the name of what is calling this method
    [System.Runtime.CompilerServices.CallerMemberName] string memberName = "",
    //gets the path of the file in which the "calling method" is in
    [System.Runtime.CompilerServices.CallerFilePath] string sourceFilePath = "",
    //gets the line number of the "calling method"
    [System.Runtime.CompilerServices.CallerLineNumber] int sourceLineNumber = 0)
{
    //Writes lines of all the attributes
    System.Diagnostics.Trace.WriteLine("Message: " + message);
    System.Diagnostics.Trace.WriteLine("Member: " + memberName);
    System.Diagnostics.Trace.WriteLine("Source File Path: " + sourceFilePath);
    System.Diagnostics.Trace.WriteLine("Line Number: " + sourceLineNumber);
}
```

:

```
//Message: Show my attributes.
//Member: doProcess
//Source File Path: c:\Path\To\The\File
//Line Number: 13
```

C # 5.0 : https://riptutorial.com/ko/csharp/topic/4584/c-sharp-5-0-

13: C # 6.0

C # Roslyn . .NET Framework 4.6 . C # 6 .NET 4.0 . .

C # Visual Studio 2015 .NET 4.6 2015 7 .

. `csc.exe` C ++ Win32 C # 6 C # .NET . "Roslyn" GitHub .

Examples

nameof string string . INotifyPropertyChanged .

```
public string SayHello(string greeted)
{
    if (greeted == null)
        throw new ArgumentNullException(nameof(greeted));

    Console.WriteLine("Hello, " + greeted);
}
```

nameof . . :

```
public static class Strings
{
    public const string Foo = nameof(Foo); // Rather than Foo = "Foo"
    public const string Bar = nameof(Bar); // Rather than Bar = "Bar"
}
```

nameof , case , switch .

Enum nameof . :

```
Console.WriteLine(Enum.One.ToString());
```

.

```
Console.WriteLine(nameof(Enum.One))
```

One .

---

nameof . :

```
string foo = "Foo";
string lengthName = nameof(foo.Length);
```

:

```
string lengthName = nameof(string.Length);
```

Length . .

---

nameof . , nameof .

```
public static int Main()
{
    Console.WriteLine(nameof(List<>)); // Compile-time error
    Console.WriteLine(nameof(Main())); // Compile-time error
}
```

.

```
Console.WriteLine(nameof(List<int>));  // "List"
Console.WriteLine(nameof(List<bool>)); // "List"
```

nameof .

( )

nameof 6.0 C # MemberExpression .

6.0

:

```
public static string NameOf<T>(Expression<Func<T>> propExp)
{
    var memberExpression = propExp.Body as MemberExpression;
    return memberExpression != null ? memberExpression.Member.Name : null;
}

public static string NameOf<TObj, T>(Expression<Func<TObj, T>> propExp)
{
    var memberExpression = propExp.Body as MemberExpression;
    return memberExpression != null ? memberExpression.Member.Name : null;
}
```

:

```
string variableName = NameOf(() => variable);
string propertyName = NameOf((Foo o) => o.Bar);
```

nameof nameof .

. .

, , .

```
public decimal TotalPrice => BasePrice + Taxes;
```

.

```
public decimal TotalPrice
{
    get
    {
        return BasePrice + Taxes;
    }
}
```

getter .

```
public object this[string key] => dictionary[key];
```

.

```
public object this[string key]
{
    get
    {
        return dictionary[key];
    }
}
```

```
static int Multiply(int a, int b) => a * b;
```

.

```
static int Multiply(int a, int b)
{
    return a * b;
}
```

void .

```
public void Dispose() => resource?.Dispose();
```

ToString Pair<T> .

```
public override string ToString() => $"{First}, {Second}";
```

override .

```
public class Foo
{
    public int Bar { get; }

    public string override ToString() => $"Bar: {Bar}";
}
```

:

```
public class Land
{
    public double Area { get; set; }

    public static Land operator +(Land first, Land second) =>
        new Land { Area = first.Area + second.Area };
}
```

. : `if` , `switch` , `for` , `foreach` , `while` , `do` , `try`

`if` . `for` `foreach` LINQ . .

```
IEnumerable<string> Digits
{
    get
    {
        for (int i = 0; i < 10; i++)
            yield return i.ToString();
    }
}
```

```
IEnumerable<string> Digits => Enumerable.Range(0, 10).Select(i => i.ToString());
```

.

`async` / `await` .

```
async Task<int> Foo() => await Bar();
```

:

```
Task<int> Foo() => Bar();
```

`boolean` catch `true` `catch` `true` .

. `catch` `if` throw . . . `throw` throw . .

CLR CL.NET 10 VB.NET F # . C # 6.0 C # .

`catch when` . `when bool` (, await). Exception `ex when` .

```
var SqlErrorToIgnore = 123;
try
{
    DoSQLOperations();
}
catch (SqlException ex) when (ex.Number != SqlErrorToIgnore)
{
    throw new Exception("An error occurred accessing the database", ex);
}
```

`when` `catch` . `true` `when` . `catch` `catch` (`when`). :

```
try
{ ... }
catch (Exception ex) when (someCondition) //If someCondition evaluates to true,
                                          //the rest of the catches are ignored.
{ ... }
catch (NotImplementedException ex) when (someMethod()) //someMethod() will only run if
                                                       //someCondition evaluates to false
{ ... }
catch(Exception ex) // If both when clauses evaluate to false
{ ... }
```

---

when

Exception when , Exception when false . when .

.

```
public static void Main()
{
    int a = 7;
    int b = 0;
    try
    {
        DoSomethingThatMightFail();
    }
    catch (Exception ex) when (a / b == 0)
    {
        // This block is never reached because a / b throws an ignored
        // DivideByZeroException which is treated as false.
    }
    catch (Exception ex)
    {
        // This block is reached since the DivideByZeroException in the
        // previous when clause is ignored.
    }
}

public static void DoSomethingThatMightFail()
{
    // This will always throw an ArgumentNullException.
    Type.GetType(null);
```

```
}
```

`throw` . 3 6.

```
1. int a = 0, b = 0;
2. try {
3.     int c = a / b;
4. }
5. catch (DivideByZeroException) {
6.     throw;
7. }
```

6 `throw` throw 6.

.

```
1. int a = 0, b = 0;
2. try {
3.     int c = a / b;
4. }
5. catch (DivideByZeroException) when (a != 0) {
6.     throw;
7. }
```

`a` 0 `catch` 3 . . , (5) `a 0` (6) (6) .

---

catch . `false Log` . throw .

3 . (**`when(...)`**).

```
try
{
    DoSomethingThatMightFail(s);
}
catch (Exception ex) when (Log(ex, "An error occurred"))
{
    // This catch block will never be reached
}

// ...

static bool Log(Exception ex, string message, params object[] args)
{
    Debug.Print(message, args);
    return false;
}
```

C # throw .

6.0

```
try
{
    DoSomethingThatMightFail(s);
```

```
}
catch (Exception ex)
{
     Log(ex, "An error occurred");
     throw;
}

// ...

static void Log(Exception ex, string message, params object[] args)
{
    Debug.Print(message, args);
}
```

---

finally

`finally` throw . `when` `finally` . (: culture) `finally` .

: finally

```
private static bool Flag = false;

static void Main(string[] args)
{
    Console.WriteLine("Start");
    try
    {
        SomeOperation();
    }
    catch (Exception) when (EvaluatesTo())
    {
        Console.WriteLine("Catch");
    }
    finally
    {
        Console.WriteLine("Outer Finally");
    }
}

private static bool EvaluatesTo()
{
    Console.WriteLine($"EvaluatesTo: {Flag}");
    return true;
}

private static void SomeOperation()
{
    try
    {
        Flag = true;
        throw new Exception("Boom");
    }
    finally
    {
        Flag = false;
```

```
        Console.WriteLine("Inner Finally");
    }
}
```

:

EvaluatesTo : True

SomeOperation when "" catch . :

```
private static void SomeOperation()
{
    try
    {
        Flag = true;
        throw new Exception("Boom");
    }
    catch
    {
       Flag = false;
       throw;
    }
    finally
    {
        Flag = false;
        Console.WriteLine("Inner Finally");
    }
}
```

IDisposable.Dispose using IDisposable .

} = . Coordinate .

6.0

```
public class Coordinate
{
    public int X { get; set; } = 34; // get or set auto-property with initializer

    public int Y { get; } = 89;      // read-only auto-property with initializer
}
```

. .

```
    public string Name { get; protected set; } = "Cheeze";
```

internal , internal protected private .

. .

```
    public List<string> Ingredients { get; } =
        new List<string> { "dough", "sauce", "cheese" };
```

. .

(C # 6.0)

C # 6 . public .

6.0

```
public class Coordinate
{
    private int _x = 34;
    public int X { get { return _x; } set { _x = value; } }

    private readonly int _y = 89;
    public int Y { get { return _y; } }

    private readonly int _z;
    public int Z { get { return _z; } }

    public Coordinate()
    {
        _z = 42;
    }
}
```

***:** C # 6.0 (getter setter) .*

. . ( ) .

```
// public decimal X { get; set; } = InitMe();  // generates compiler error

decimal InitMe() { return 4m; }
```

.

```
public class Rectangle
{
    public double Length { get; set; } = 1;
    public double Width { get; set; } = 1;
    public double Area { get; set; } = CalculateArea(1, 1);

    public static double CalculateArea(double length, double width)
    {
        return length * width;
```

```
    }
}
```

.

```
public short Type { get; private set; } = 15;
```

. . FingerPrint .

```
public class FingerPrint
{
  public DateTime TimeStamp { get; } = DateTime.UtcNow;

  public string User { get; } =
    System.Security.Principal.WindowsPrincipal.Current.Identity.Name;

  public string Process { get; } =
    System.Diagnostics.Process.GetCurrentProcess().ProcessName;
}
```

=> = , { get; } .

, .

```
public class UserGroupDto
{
    // Read-only auto-property with initializer:
    public ICollection<UserDto> Users1 { get; } = new HashSet<UserDto>();

    // Read-write field with initializer:
    public ICollection<UserDto> Users2 = new HashSet<UserDto>();

    // Read-only auto-property with expression body:
    public ICollection<UserDto> Users3 => new HashSet<UserDto>();
}
```

{ get; } . Users1 / Users2 . > from => { get; } .

HashSet<UserDto> Users3 (=> =) Users3 C # . .

.

```
public class UserGroupDto
{
    // This is a property returning the same instance
    // which was created when the UserGroupDto was instantiated.
    private ICollection<UserDto> _users1 = new HashSet<UserDto>();
    public ICollection<UserDto> Users1 { get { return _users1; } }

    // This is a field returning the same instance
    // which was created when the UserGroupDto was instantiated.
    public virtual ICollection<UserDto> Users2 = new HashSet<UserDto>();
```

```
    // This is a property which returns a new HashSet<UserDto> as
    // an ICollection<UserDto> on each call to it.
    public ICollection<UserDto> Users3 { get { return new HashSet<UserDto>(); } }
}
```

.

.

```
var dict = new Dictionary<string, int>()
{
    ["foo"] = 34,
    ["bar"] = 42
};
```

getter setter .

```
class Program
{
    public class MyClassWithIndexer
    {
        public int this[string index]
        {
            set
            {
                Console.WriteLine($"Index: {index}, value: {value}");
            }
        }
    }

    public static void Main()
    {
        var x = new MyClassWithIndexer()
        {
            ["foo"] = 34,
            ["bar"] = 42
        };

        Console.ReadKey();
    }
}
```

:

: foo, : 34
: , : 42

.

```
class Program
{
    public class MyClassWithIndexer
    {
        public int this[string index]
        {
```

```
            set
            {
                Console.WriteLine($"Index: {index}, value: {value}");
            }
        }
        public string this[int index]
        {
            set
            {
                Console.WriteLine($"Index: {index}, value: {value}");
            }
        }
    }

    public static void Main()
    {
        var x = new MyClassWithIndexer()
        {
            ["foo"] = 34,
            ["bar"] = 42,
            [10] = "Ten",
            [42] = "Meaning of life"
        };
    }
}
```

:

- : foo, : 34
- : , : 42
- : 10, : 10
- : 42, :

`set` ( ) `Add` .

:

```
var d = new Dictionary<string, int>
{
    ["foo"] = 34,
    ["foo"] = 42,
}; // does not throw, second value overwrites the first one
```

:

```
var d = new Dictionary<string, int>
{
    { "foo", 34 },
    { "foo", 42 },
}; // run-time ArgumentException: An item with the same key has already been added.
```

`variables` .

---

int foo bar bar .

```
int foo = 34;
int bar = 42;

string resultString = $"The foo is {foo}, and the bar is {bar}.";

Console.WriteLine(resultString);
```

:

foo 34 bar 42.

.

```
var foo = 34;
var bar = 42;

// String interpolation notation (new style)
Console.WriteLine($"The foo is {{foo}}, and the bar is {{bar}}.");
```

.

foo {foo} bar {bar}.

@ . . .

```
Console.WriteLine($@"In case it wasn't clear:
\u00B9
The foo
is {foo},
and the bar
is {bar}.");
```

:

:
\ u00B9

34,

42.

{} . . foo bar Math.Max .

```
Console.WriteLine($"And the greater one is: { Math.Max(foo, bar) }");
```

:

: 42

: (, CRLF /) .

, .

```
Console.WriteLine($"Foo formatted as a currency to 4 decimal places: {foo:c4}");
```

:

Foo 4 : $ 34.0000

.

```
Console.WriteLine($"Today is: {DateTime.Today:dddd, MMMM dd - yyyy}");
```

:

: 7 20 , 2015

(3) . .

```
Console.WriteLine($"{(foo > bar ? "Foo is larger than bar!" : "Bar is larger than foo!")}");
```

:

foo !

.

```
Console.WriteLine($"Environment: {(Environment.Is64BitProcess ? 64 : 32):00'-bit'} process");
```

:

: 32

(\ ) (") . , .

```
Console.WriteLine($"Foo is: {foo}. In a non-verbatim string, we need to escape \" and \\ with
backslashes.");
Console.WriteLine($@"Foo is: {foo}. In a verbatim string, we need to escape "" with an extra
quote, but we don't need to escape \");
```

:

Foo 34. " \ .

Foo 34. " \

{ } {{ }} .

```
$"{{foo}} is: {foo}"
```

:

{foo} : 34

FormattableString

$"..." . .

```
string s = $"hello, {name}";
System.FormattableString s = $"Hello, {name}";
System.IFormattable s = $"Hello, {name}";
```

.

System.FormattableString . .

```
public void AddLogItem(FormattableString formattableString)
{
    foreach (var arg in formattableString.GetArguments())
    {
        // do something to interpolation argument 'arg'
    }

    // use the standard interpolation and the current culture info
    // to get an ordinary String:
    var formatted = formattableString.ToString();

    // ...
}
```

.

```
AddLogItem($"The foo is {foo}, and the bar is {bar}.");
```

, .

.

```
var s = $"Foo: {foo}";
System.IFormattable s = $"Foo: {foo}";
```

IFormattable .

```
var s = $"Bar: {bar}";
System.FormattableString s = $"Bar: {bar}";
```

CA1305 (Specify `IFormatProvider`) . .

```
public static class Culture
{
    public static string Current(FormattableString formattableString)
    {
        return formattableString?.ToString(CultureInfo.CurrentCulture);
    }
    public static string Invariant(FormattableString formattableString)
    {
        return formattableString?.ToString(CultureInfo.InvariantCulture);
    }
}
```

.

```
Culture.Current($"interpolated {typeof(string).Name} string.")
Culture.Invariant($"interpolated {typeof(string).Name} string.")
```

: `Current` `Invariant` `String` :

```
$"interpolated {typeof(string).Name} string.".Current();
```

`FormattableString` `Invariant()` using static using static .

```
using static System.FormattableString;


string invariant = Invariant($"Now = {DateTime.Now}");
string current = $"Now = {DateTime.Now}";
```

`String.Format()` . (Roslyn) `String.Format` .

```
var text = $"Hello {name + lastName}";
```

:

```
string text = string.Format("Hello {0}", new object[] {
    name + lastName
});
```

Linq

Linq .

```
var fooBar = (from DataRow x in fooBarTable.Rows
          select string.Format("{0}{1}", x["foo"], x["bar"])).ToList();
```

:

```
var fooBar = (from DataRow x in fooBarTable.Rows
          select $"{x["foo"]}{x["bar"]}").ToList();
```

---

string.Format .

```
public const string ErrorFormat = "Exception caught:\r\n{0}";

// ...

Logger.Log(string.Format(ErrorFormat, ex));
```

. .

```
public const string ErrorFormat = $"Exception caught:\r\n{error}";
// CS0103: The name 'error' does not exist in the current context
```

String Func<> .

```
public static Func<Exception, string> FormatError =
    error => $"Exception caught:\r\n{error}";

// ...

Logger.Log(FormatError(ex));
```

. String .

```
"My name is {name} {middlename} {surname}"
```

:

```
"My name is {0} {1} {2}"
```

String string.Format . ., .

```
var FirstName = "John";
```

```
// method using different resource file "strings"
// for French ("strings.fr.resx"), German ("strings.de.resx"),
// and English ("strings.en.resx")
void ShowMyNameLocalized(string name, string middlename = "", string surname = "")
{
    // get localized string
    var localizedMyNameIs = Properties.strings.Hello;
    // insert spaces where necessary
    name = (string.IsNullOrWhiteSpace(name) ? "" : name + " ");
    middlename = (string.IsNullOrWhiteSpace(middlename) ? "" : middlename + " ");
    surname = (string.IsNullOrWhiteSpace(surname) ? "" : surname + " ");
    // display it
    Console.WriteLine($"{localizedMyNameIs} {name}{middlename}{surname}".Trim());
}

// switch to French and greet John
Thread.CurrentThread.CurrentUICulture = CultureInfo.GetCultureInfo("fr-FR");
ShowMyNameLocalized(FirstName);

// switch to German and greet John
Thread.CurrentThread.CurrentUICulture = CultureInfo.GetCultureInfo("de-DE");
ShowMyNameLocalized(FirstName);

// switch to US English and greet John
Thread.CurrentThread.CurrentUICulture = CultureInfo.GetCultureInfo("en-US");
ShowMyNameLocalized(FirstName);
```

.

Bonjour, mon nom est John

. .

,

. culture switch case . C # .

. .

`string` .

```
Console.WriteLine($"String has {$"My class is called {nameof(MyClass)}.".Length} chars:");
Console.WriteLine($"My class is called {nameof(MyClass)}.");
```

:

27 .

MyClass.

.

`await` (TResult) `catch` `finally` C # 6 .

`catch`

`finally` `await` . C # 6 `await` .

```
try
{
    //since C#5
    await service.InitializeAsync();
}
catch (Exception e)
{
    //since C#6
    await logger.LogAsync(e);
}
finally
{
    //since C#6
    await service.CloseAsync();
}
```

C # 5 `bool` try catch `Exception` . .

```
bool error = false;
Exception ex = null;

try
{
    // Since C#5
    await service.InitializeAsync();
}
catch (Exception e)
{
    // Declare bool or place exception inside variable
    error = true;
    ex = e;
}

// If you don't use the exception
if (error)
{
    // Handle async task
}

// If want to use information from the exception
if (ex != null)
{
    await logger.LogAsync(e);
}

// Close the service, since this isn't possible in the finally
await service.CloseAsync();
```

`?.` `?[...]` null . .

. ( ) `null` `NullReferenceException` throw. `?.` (null-conditional) throw null .

`?.` `?.` . .

```
var teacherName = classroom.GetTeacher().Name;
// throws NullReferenceException if GetTeacher() returns null
```

`classroom` `GetTeacher()` `null` `.Name` `null` `Name` `NullReferenceException` throw.

`?.` `null` .

```
var teacherName = classroom.GetTeacher()?.Name;
// teacherName is null if GetTeacher() returns null
```

`classroom` `null` .

```
var teacherName = classroom?.GetTeacher()?.Name;
// teacherName is null if GetTeacher() returns null OR classroom is null
```

. (NULL) null .

null `Nullable<T>` `?.` `?.` .

```
bool hasCertification = classroom.GetTeacher().HasCertification;
// compiles without error but may throw a NullReferenceException at runtime

bool hasCertification = classroom?.GetTeacher()?.HasCertification;
// compile time error: implicit conversion from bool? to bool not allowed

bool? hasCertification = classroom?.GetTeacher()?.HasCertification;
// works just fine, hasCertification will be null if any part of the chain is null

bool hasCertification = classroom?.GetTeacher()?.HasCertification.GetValueOrDefault();
// must extract value from nullable to assign to a value type variable
```

Null (??)

null Null (`??`) `null` . .

```
var teacherName = classroom?.GetTeacher()?.Name ?? "No Name";
// teacherName will be "No Name" when GetTeacher()
// returns null OR classroom is null OR Name is null
```

---

null .

```
var firstStudentName = classroom?.Students?[0]?.Name;
```

:

- `?.` `classroom` `null` .
- `?` `Students` `null` .
-

?. [0] null . IndexOutOfRangeException .

---

void

null void . null . NullReferenceException .

```
List<string> list = null;
list?.Add("hi");          // Does not evaluate to null
```

.

```
private event EventArgs OnCompleted;
```

null .

```
var handler = OnCompleted;
if (handler != null)
{
    handler(EventArgs.Empty);
}
```

null .

```
OnCompleted?.Invoke(EventArgs.Empty);
```

Null lvalue rvalue . , , . .

```
// Error: The left-hand side of an assignment must be a variable, property or indexer
Process.GetProcessById(1337)?.EnableRaisingEvents = true;
// Error: The event can only appear on the left hand side of += or -=
Process.GetProcessById(1337)?.Exited += OnProcessExited;
```

:

```
int? nameLength = person?.Name.Length;    // safe if 'person' is null
```

:

```
int? nameLength = (person?.Name).Length;  // avoid this
```

.

```
int? nameLength = person != null ? (int?)person.Name.Length : null;
```

.

```
int? nameLength = (person != null ? person.Name : null).Length;
```

?: . . .

```
void Main()
{
    var foo = new Foo();
    Console.WriteLine("Null propagation");
    Console.WriteLine(foo.Bar?.Length);

    Console.WriteLine("Ternary");
    Console.WriteLine(foo.Bar != null ? foo.Bar.Length : (int?)null);
}

class Foo
{
    public string Bar
    {
        get
        {
            Console.WriteLine("I was read");
            return string.Empty;
        }
    }
}
```

:

.
0

.
.
0

.

```
var interimResult = foo.Bar;
Console.WriteLine(interimResult != null ? interimResult.Length : (int?)null);
```

.

using static [Namespace.Type] . .

6.0

```
using static System.Console;
using static System.ConsoleColor;
using static System.Math;

class Program
{
    static void Main()
    {
        BackgroundColor = DarkBlue;
        WriteLine(Sqrt(2));
    }
}
```

6.0

```
using System;

class Program
{
    static void Main()
    {
        Console.BackgroundColor = ConsoleColor.DarkBlue;
        Console.WriteLine(Math.Sqrt(2));
    }
}
```

( ) . **C # 6** .

```
using System;
public class Program
{
    public static void Main()
    {
        Overloaded(DoSomething);
    }

    static void Overloaded(Action action)
    {
       Console.WriteLine("overload with action called");
    }

    static void Overloaded(Func<int> function)
    {
       Console.WriteLine("overload with Func<int> called");
    }

    static int DoSomething()
    {
        Console.WriteLine(0);
        return 0;
    }
}
```

:

6.0

Func <int> overload .

5.0

CS0121 : 'Program.Overloaded (System.Action)' 'Program.Overloaded (System.Func)' .

C # 6 C # 5 .

```
using System;

class Program
{
    static void Foo(Func<Func<long>> func) {}
    static void Foo(Func<Func<int>> func) {}

    static void Main()
    {
        Foo(() => () => 7);
    }
}
```

. C # 5 C # 6 .

5.0

```
Console.WriteLine((value: 23));
```

is as . C # 5 C # 6 .

5.0

```
var result = "".Any is byte;
```

( ) 1.Any is string IDisposable.Dispose is object IDisposable.Dispose is object .

.

IEnumerable Add .

Add . C # 6 .

```
public class CollectionWithAdd : IEnumerable
{
    public void Add<T>(T item)
    {
        Console.WriteLine("Item added with instance add method: " + item);
    }

    public IEnumerator GetEnumerator()
    {
        // Some implementation here
    }
```

```
}

public class CollectionWithoutAdd : IEnumerable
{
    public IEnumerator GetEnumerator()
    {
        // Some implementation here
    }
}

public static class Extensions
{
    public static void Add<T>(this CollectionWithoutAdd collection, T item)
    {
        Console.WriteLine("Item added with extension add method: " + item);
    }
}

public class Program
{
    public static void Main()
    {
        var collection1 = new CollectionWithAdd{1,2,3}; // Valid in all C# versions
        var collection2 = new CollectionWithoutAdd{4,5,6}; // Valid only since C# 6
    }
}
```

.

: 1
: 2
: 3
: 4
: 5
: 6

C # 5.0 . Roslyn Analyzers C # . C # 6.0 pragma .

:

```
#pragma warning disable 0501
```

C # 6.0 :

```
#pragma warning disable CS0501
```

C # 6.0 : https://riptutorial.com/ko/csharp/topic/24/c-sharp-6-0-

14: C # 7.0

C # 7.0 C # 7 . , , `out var` , , , , `ref return` , `ref return ref local` .

: C # 7

Examples

out var

C # `bool TryParse(object input, out object value)` `bool TryParse(object input, out object value)` .

`out var` . out .

.

C # 7.0 `TryParse` .

7.0

```
int value;
if (int.TryParse(input, out value))
{
    Foo(value); // ok
}
else
{
    Foo(value); // value is zero
}

Foo(value); // ok
```

C # 7.0 `out` .

7.0

```
if (int.TryParse(input, out var value))
{
    Foo(value); // ok
}
else
{
    Foo(value); // value is zero
}

Foo(value); // still ok, the value in scope within the remainder of the body
```

`out` _ .

```
p.GetCoordinates(out var x, out _); // I only care about x
```

out var out . out var . .

out var .

7.0

```
var a = new[] { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
var groupedByMod2 = a.Select(x => new
                                  {
                                      Source = x,
                                      Mod2 = x % 2
                                  })
                     .GroupBy(x => x.Mod2)
                     .ToDictionary(g => g.Key, g => g.ToArray());
if (groupedByMod2.TryGetValue(1, out var oddElements))
{
    Console.WriteLine(oddElements.Length);
}
```

int key Dictionary . C # TryGetValue TryGetValue (out). out var out .

---

var LINQ lambdas . .

```
var nums =
    from item in seq
    let success = int.TryParse(item, out var tmp)
    select success ? tmp : 0; // Error: The name 'tmp' does not exist in the current context
```

- GitHub var

0b .

0 1 2 . .

34 (= $2^5 + 2^1$) int .

```
// Using a binary literal:
//   bits: 76543210
int a1 = 0b00100010;          // binary: explicitly specify bits

// Existing methods:
int a2 = 0x22;                // hexadecimal: every digit corresponds to 4 bits
int a3 = 34;                  // decimal: hard to visualise which bits are set
int a4 = (1 << 5) | (1 << 1); // bitwise arithmetic: combining non-zero bits
```

---

enum .

```
[Flags]
public enum DaysOfWeek
{
    // Previously available methods:
    //          decimal        hex       bit shifting
    Monday    =  1,    //    = 0x01    = 1 << 0
    Tuesday   =  2,    //    = 0x02    = 1 << 1
    Wednesday =  4,    //    = 0x04    = 1 << 2
    Thursday  =  8,    //    = 0x08    = 1 << 3
    Friday    = 16,    //    = 0x10    = 1 << 4
    Saturday  = 32,    //    = 0x20    = 1 << 5
    Sunday    = 64,    //    = 0x40    = 1 << 6

    Weekdays = Monday | Tuesday | Wednesday | Thursday | Friday,
    Weekends = Saturday | Sunday
}
```

16 .

```
[Flags]
public enum DaysOfWeek
{
    Monday    = 0b00000001,
    Tuesday   = 0b00000010,
    Wednesday = 0b00000100,
    Thursday  = 0b00001000,
    Friday    = 0b00010000,
    Saturday  = 0b00100000,
    Sunday    = 0b01000000,

    Weekdays = Monday | Tuesday | Wednesday | Thursday | Friday,
    Weekends = Saturday | Sunday
}
```

_ . .

. .

._ . .

```
int bin = 0b1001_1010_0001_0100;
int hex = 0x1b_a0_44_fe;
int dec = 33_554_432;
int weird = 1_2__3___4____5_____6______7_______8________9;
double real = 1_000.111_1e-1_000;
```

_ :

- (`_121`)
- (`121_ 121.05_`)
- 10 (`10_.0`)
- (`1.1e_1`)
- (`10_f`)
- `0x 0b` 2 16 (: 0b_1001_1000)

. (,) .

C # 7.0 . ValueTuple .

```
public (int sum, int count) GetTallies()
{
    return (1, 2);
}
```

: Visual Studio 2017 `System.ValueTuple` `System.ValueTuple` .

.

```
var result = GetTallies();
// > result.sum
// 1
// > result.count
// 2
```

(tuple deconstruction) .

, `GetTallies` .

```
(int tallyOne, int tallyTwo) = GetTallies();
```

`var` .

```
(var s, var c) = GetTallies();
```

`var` of `()` .

```
var (s, c) = GetTallies();
```

.

```
int s, c;
(s, c) = GetTallies();
```

( ).

```
(b, a) = (a, b);
```

`Deconstruct` .

```
class Person
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
```

```
    public void Deconstruct(out string firstName, out string lastName)
    {
        firstName = FirstName;
        lastName = LastName;
    }
}

var person = new Person { FirstName = "John", LastName = "Smith" };
var (localFirstName, localLastName) = person;
```

, (localFirstName, localLastName) = person person Deconstruct .

. .

```
public static class PersonExtensions
{
    public static void Deconstruct(this Person person, out string firstName, out string
lastName)
    {
        firstName = person.FirstName;
        lastName = person.LastName;
    }
}

var (localFirstName, localLastName) = person;
```

Person Name Tuple . :

```
class Person
{
    public (string First, string Last) Name { get; }

    public Person((string FirstName, string LastName) name)
    {
        Name = name;
    }
}
```

( ).

```
var person = new Person(("Jane", "Smith"));

var firstName = person.Name.First; // "Jane"
var lastName = person.Name.Last;   // "Smith"
```

.

```
var name = ("John", "Smith");
Console.WriteLine(name.Item1);
// Outputs John

Console.WriteLine(name.Item2);
// Outputs Smith
```

.

```
var name = (first: "John", middle: "Q", last: "Smith");
Console.WriteLine(name.first);
// Outputs John
```

( ) . :

```
public (int sum, double average) Measure(List<int> items)
{
    var stats = (sum: 0, average: 0d);
    stats.sum = items.Sum();
    stats.average = items.Average();
    return stats;
}
```

stats stats .

. . object .

ValueTuple TupleElementNamesAttribute . , , . . , (, int) TupleElementNameAttribute . "" "".

async

(ValueTuple) . async / await await .

```
public async Task<(string value, int count)> GetValueAsync()
{
    string fooBar = await _stackoverflow.GetStringAsync();
    int num = await _stackoverflow.GetIntAsync();

    return (fooBar, num);
}
```

( ) .

:

```
private readonly List<Tuple<string, string, string>> labels = new List<Tuple<string, string,
string>>()
{
    new Tuple<string, string, string>("test1", "test2", "Value"),
    new Tuple<string, string, string>("test1", "test1", "Value2"),
    new Tuple<string, string, string>("test2", "test2", "Value3"),
```

```
};

public string FindMatchingValue(string firstElement, string secondElement)
{
    var result = labels
        .Where(w => w.Item1 == firstElement && w.Item2 == secondElement)
        .FirstOrDefault();

    if (result == null)
        throw new ArgumentException("combo not found");

    return result.Item3;
}
```

.

```
private readonly List<(string firstThingy, string secondThingyLabel, string foundValue)>
labels = new List<(string firstThingy, string secondThingyLabel, string foundValue)>()
{
    ("test1", "test2", "Value"),
    ("test1", "test1", "Value2"),
    ("test2", "test2", "Value3"),
}

public string FindMatchingValue(string firstElement, string secondElement)
{
    var result = labels
        .Where(w => w.firstThingy == firstElement && w.secondThingyLabel == secondElement)
        .FirstOrDefault();

    if (result == null)
        throw new ArgumentException("combo not found");

    return result.foundValue;
}
```

"item1", "item2" "item3" .

ValueTuple Tuple

`ValueTuple` .

	ValueTuple	Tuple
	struct	class
( )		
		( )

- GitHub
- C # 7.0 VS 15

- NuGet

. iterators, `async` / `await` lambda . , . .

```
double GetCylinderVolume(double radius, double height)
{
    return getVolume();

    double getVolume()
    {
        // You can declare inner-local functions in a local function
        double GetCircleArea(double r) => Math.PI * r * r;

        // ALL parents' variables are accessible even though parent doesn't have any input.
        return GetCircleArea(radius) * height;
    }
}
```

LINQ . .

```
public static IEnumerable<TSource> Where<TSource>(
    this IEnumerable<TSource> source,
    Func<TSource, bool> predicate)
{
    if (source == null) throw new ArgumentNullException(nameof(source));
    if (predicate == null) throw new ArgumentNullException(nameof(predicate));

    return iterator();

    IEnumerable<TSource> iterator()
    {
        foreach (TSource element in source)
            if (predicate(element))
                yield return element;
    }
}
```

`async await` .

```
async Task WriteEmailsAsync()
{
    var emailRegex = new Regex(@"(?i)[a-z0-9_.+-]+@[a-z0-9-]+\.[a-z0-9-.]+");
    IEnumerable<string> emails1 = await getEmailsFromFileAsync("input1.txt");
    IEnumerable<string> emails2 = await getEmailsFromFileAsync("input2.txt");
    await writeLinesToFileAsync(emails1.Concat(emails2), "output.txt");

    async Task<IEnumerable<string>> getEmailsFromFileAsync(string fileName)
    {
        string text;

        using (StreamReader reader = File.OpenText(fileName))
        {
```

```
            text = await reader.ReadToEndAsync();
        }

        return from Match emailMatch in emailRegex.Matches(text) select emailMatch.Value;
    }

    async Task writeLinesToFileAsync(IEnumerable<string> lines, string fileName)
    {
        using (StreamWriter writer = File.CreateText(fileName))
        {
            foreach (string line in lines)
            {
                await writer.WriteLineAsync(line);
            }
        }
    }
}
```

`return` . "lowerCamelCase" .

C

switch

`switch` .

```
class Geometry {}

class Triangle : Geometry
{
    public int Width { get; set; }
    public int Height { get; set; }
    public int Base { get; set; }
}

class Rectangle : Geometry
{
    public int Width { get; set; }
    public int Height { get; set; }
}

class Square : Geometry
{
    public int Width { get; set; }
}

public static void PatternMatching()
{
    Geometry g = new Square { Width = 5 };

    switch (g)
    {
        case Triangle t:
            Console.WriteLine($"{t.Width} {t.Height} {t.Base}");
            break;
        case Rectangle sq when sq.Width == sq.Height:
            Console.WriteLine($"Square rectangle: {sq.Width} {sq.Height}");
            break;
```

```
        case Rectangle r:
            Console.WriteLine($"{r.Width} {r.Height}");
            break;
        case Square s:
            Console.WriteLine($"{s.Width}");
            break;
        default:
            Console.WriteLine("<other>");
            break;
    }
}
```

is

is .

7.0

```
string s = o as string;
if(s != null)
{
    // do something with s
}
```

:

7.0

```
if(o is string s)
{
    //Do something with s
};
```

s if (:

```
if(someCondition)
{
   if(o is string s)
   {
      //Do something with s
   }
   else
   {
     // s is unassigned here, but accessible
   }

   // s is unassigned here, but accessible
}
// s is not accessible here
```

ref return ref local

Ref .

```
public static ref TValue Choose<TValue>(
    Func<bool> condition, ref TValue left, ref TValue right)
{
    return condition() ? ref left : ref right;
}
```

.

```
Matrix3D left = …, right = …;
Choose(chooser, ref left, ref right).M20 = 1.0;
```

```
public static ref int Max(ref int first, ref int second, ref int third)
{
    ref int max = first > second ? ref first : ref second;
    return max > third ? ref max : ref third;
}
…
int a = 1, b = 2, c = 3;
Max(ref a, ref b, ref c) = 4;
Debug.Assert(a == 1); // true
Debug.Assert(b == 2); // true
Debug.Assert(c == 4); // true
```

System.Runtime.CompilerServices.Unsafe ref .

, (ref) .

```
byte[] b = new byte[4] { 0x42, 0x42, 0x42, 0x42 };

ref int r = ref Unsafe.As<byte, int>(ref b[0]);
Assert.Equal(0x42424242, r);

0x0EF00EF0;
Assert.Equal(0xFE, b[0] | b[1] | b[2] | b[3]);
```

BitConverter.IsLittleEndian . BitConverter.IsLittleEndian .

.

```
int[] a = new int[] { 0x123, 0x234, 0x345, 0x456 };

ref int r1 = ref Unsafe.Add(ref a[0], 1);
Assert.Equal(0x234, r1);

ref int r2 = ref Unsafe.Add(ref r1, 2);
Assert.Equal(0x456, r2);

ref int r3 = ref Unsafe.Add(ref r2, -3);
```

```
Assert.Equal(0x123, r3);
```

Subtract :

```
string[] a = new string[] { "abc", "def", "ghi", "jkl" };

ref string r1 = ref Unsafe.Subtract(ref a[0], -2);
Assert.Equal("ghi", r1);

ref string r2 = ref Unsafe.Subtract(ref r1, -1);
Assert.Equal("jkl", r2);

ref string r3 = ref Unsafe.Subtract(ref r2, 3);
Assert.Equal("abc", r3);
```

ref .

```
long[] a = new long[2];

Assert.True(Unsafe.AreSame(ref a[0], ref a[0]));
Assert.False(Unsafe.AreSame(ref a[0], ref a[1]));
```

Roslyn Github Issue

github System.Runtime.CompilerServices.Unsafe

C # 7.0 throw .

```
class Person
{
    public string Name { get; }

    public Person(string name) => Name = name ?? throw new
ArgumentNullException(nameof(name));

    public string GetFirstName()
    {
        var parts = Name.Split(' ');
        return (parts.Length > 0) ? parts[0] : throw new InvalidOperationException("No
name!");
    }

    public string GetLastName() => throw new NotImplementedException();
}
```

C # 7.0 throw .

```
var spoons = "dinner,desert,soup".Split(',');

var spoonsArray = spoons.Length > 0 ? spoons : null;

if (spoonsArray == null)
{
```

```
    throw new Exception("There are no spoons");
}
```

```
var spoonsArray = spoons.Length > 0
    ? spoons
    : new Func<string[]>(() =>
      {
          throw new Exception("There are no spoons");
      })();
```

C # 7.0 .

```
var spoonsArray = spoons.Length > 0 ? spoons : throw new Exception("There are no spoons");
```

C # 7.0 , .

```
class Person
{
    private static ConcurrentDictionary<int, string> names = new ConcurrentDictionary<int,
string>();

    private int id = GetId();

    public Person(string name) => names.TryAdd(id, name); // constructors

    ~Person() => names.TryRemove(id, out _);              // finalizers

    public string Name
    {
        get => names[id];                                 // getters
        set => names[id] = value;                         // setters
    }
}
```

out var .

ValueTask

Task<T> .

ValueTask<T> Task .

ValueTask<T> .

1.

Task<T> .

-
- JIT 120ns

```
async Task<int> TestTask(int d)
{
    await Task.Delay(d);
    return 10;
}
```

`ValueTask<T>` .

- (`Task.Delay` , `async` / `await`)
- JIT 65ns

```
async ValueTask<int> TestValueTask(int d)
{
    await Task.Delay(d);
    return 10;
}
```

2.

`Task.Run` `Task.FromResult` ( ). .

`ValueTask<T>` .

.

```
interface IFoo<T>
{
    ValueTask<T> BarAsync();
}
```

... .

```
IFoo<T> thing = getThing();
var x = await thing.BarAsync();
```

`ValueTask` .

:

```
class SynchronousFoo<T> : IFoo<T>
{
    public ValueTask<T> BarAsync()
    {
        var value = default(T);
        return new ValueTask<T>(value);
    }
}
```

```
class AsynchronousFoo<T> : IFoo<T>
```

```
{
    public async ValueTask<T> BarAsync()
    {
        var value = default(T);
        await Task.Delay(1);
        return value;
    }
}
```

`ValueTask` C # 7.0 , . ValueTask <T> `System.Threading.Tasks.Extensions` Nuget Gallery .

C # 7.0 : https://riptutorial.com/ko/csharp/topic/1936/c-sharp-7-0-

15: C # Structs Union (C)

, C "" . , . .

Struct . . . .

Examples

C # C

C "" . , . . IP . IP Byte1.Byte2.Byte3.Byte4 Byte . Int32 long .

Explicit Layout Structs C # .

```
using System;
using System.Runtime.InteropServices;

// The struct needs to be annotated as "Explicit Layout"
[StructLayout(LayoutKind.Explicit)]
struct IpAddress
{
    // The "FieldOffset" means that this Integer starts, an offset in bytes.
    // sizeof(int) 4, sizeof(byte) = 1
    [FieldOffset(0)] public int Address;
    [FieldOffset(0)] public byte Byte1;
    [FieldOffset(1)] public byte Byte2;
    [FieldOffset(2)] public byte Byte3;
    [FieldOffset(3)] public byte Byte4;

    public IpAddress(int address) : this()
    {
        // When we init the Int, the Bytes will change too.
        Address = address;
    }

    // Now we can use the explicit layout to access the
    // bytes separately, without doing any conversion.
    public override string ToString() => $"{Byte1}.{Byte2}.{Byte3}.{Byte4}";
}
```

Struct C Union . IP Random Integer '100' 'ABCD' '100.BCD' :

```
var ip = new IpAddress(new Random().Next());
Console.WriteLine($"{ip} = {ip.Address}");
ip.Byte1 = 100;
Console.WriteLine($"{ip} = {ip.Address}");
```

:

```
75.49.5.32 = 537211211
100.49.5.32 = 537211236
```

C # .

C # Explicit Layout () Structs . Union .

```
using System;
using System.Runtime.InteropServices;

// The struct needs to be annotated as "Explicit Layout"
[StructLayout(LayoutKind.Explicit)]
struct IpAddress
{
    // Same definition of IpAddress, from the example above
}

// Now let's see if we can fit a whole URL into a long

// Let's define a short enum to hold protocols
enum Protocol : short { Http, Https, Ftp, Sftp, Tcp }

// The Service struct will hold the Address, the Port and the Protocol
[StructLayout(LayoutKind.Explicit)]
struct Service
{
    [FieldOffset(0)] public IpAddress Address;
    [FieldOffset(4)] public ushort Port;
    [FieldOffset(6)] public Protocol AppProtocol;
    [FieldOffset(0)] public long Payload;

    public Service(IpAddress address, ushort port, Protocol protocol)
    {
        Payload = 0;
        Address = address;
        Port  = port;
        AppProtocol = protocol;
    }

    public Service(long payload)
    {
        Address = new IpAddress(0);
        Port = 80;
        AppProtocol = Protocol.Http;
        Payload = payload;
    }

    public Service Copy() => new Service(Payload);

    public override string ToString() => $"{AppProtocol}//{Address}:{Port}/";
}
```

(Service Union) (8) .

```
var ip = new IpAddress(new Random().Next());
Console.WriteLine($"Size: {Marshal.SizeOf(ip)} bytes. Value: {ip.Address} = {ip}.");

var s1 = new Service(ip, 8080, Protocol.Https);
var s2 = new Service(s1.Payload);
s2.Address.Byte1 = 100;
s2.AppProtocol = Protocol.Ftp;
```

```
Console.WriteLine($"Size: {Marshal.SizeOf(s1)} bytes. Value: {s1.Address} = {s1}.");
Console.WriteLine($"Size: {Marshal.SizeOf(s2)} bytes. Value: {s2.Address} = {s2}.");
```

C # Structs Union (C) : https://riptutorial.com/ko/csharp/topic/5626/c-sharp-structs--union-----c--

16: C

Examples

HashSet

O (1) .

```
HashSet<int> validStoryPointValues = new HashSet<int>() { 1, 2, 3, 5, 8, 13, 21 };
bool containsEight = validStoryPointValues.Contains(8); // O(1)
```

List `Contains` .

```
List<int> validStoryPointValues = new List<int>() { 1, 2, 3, 5, 8, 13, 21 };
bool containsEight = validStoryPointValues.Contains(8); // O(n)
```

`HashSet.Contains` .

SortedSet

```
// create an empty set
var mySet = new SortedSet<int>();

// add something
// note that we add 2 before we add 1
mySet.Add(2);
mySet.Add(1);

// enumerate through the set
foreach(var item in mySet)
{
    Console.WriteLine(item);
}

// output:
// 1
// 2
```

T [] (T)

```
// create an array with 2 elements
var myArray = new [] { "one", "two" };

// enumerate through the array
foreach(var item in myArray)
{
    Console.WriteLine(item);
}

// output:
// one
```

```
// two

// exchange the element on the first position
// note that all collections start with the index 0
myArray[0] = "something else";


// enumerate through the array again
foreach(var item in myArray)
{
    Console.WriteLine(item);
}

// output:
// something else
// two
```

List<T> . ,, .

```
using System.Collections.Generic;

var list = new List<int>() { 1, 2, 3, 4, 5 };
list.Add(6);
Console.WriteLine(list.Count); // 6
list.RemoveAt(3);
Console.WriteLine(list.Count); // 5
Console.WriteLine(list[3]);    // 5
```

List<T> . . Dictionary<T> .

<TKey, TValue>. .

```
using System.Collections.Generic;

var people = new Dictionary<string, int>
{
    { "John", 30 }, {"Mary", 35}, {"Jack", 40}
};

// Reading data
Console.WriteLine(people["John"]); // 30
Console.WriteLine(people["George"]); // throws KeyNotFoundException

int age;
if (people.TryGetValue("Mary", out age))
{
    Console.WriteLine(age); // 35
}

// Adding and changing data
people["John"] = 40;    // Overwriting values this way is ok
people.Add("John", 40); // Throws ArgumentException since "John" already exists

// Iterating through contents
foreach(KeyValuePair<string, int> person in people)
{
    Console.WriteLine("Name={0}, Age={1}", person.Key, person.Value);
}
```

```
foreach(string name in people.Keys)
{
    Console.WriteLine("Name={0}", name);
}

foreach(int age in people.Values)
{
    Console.WriteLine("Age={0}", age);
}
```

```
var people = new Dictionary<string, int>
{
    { "John", 30 }, {"Mary", 35}, {"Jack", 40}, {"Jack", 40}
}; // throws ArgumentException since "Jack" already exists
```

```
// Initialize a stack object of integers
var stack = new Stack<int>();

// add some data
stack.Push(3);
stack.Push(5);
stack.Push(8);

// elements are stored with "first in, last out" order.
// stack from top to bottom is: 8, 5, 3

// We can use peek to see the top element of the stack.
Console.WriteLine(stack.Peek()); // prints 8

// Pop removes the top element of the stack and returns it.
Console.WriteLine(stack.Pop()); // prints 8
Console.WriteLine(stack.Pop()); // prints 5
Console.WriteLine(stack.Pop()); // prints 3
```

LinkedList

```
// initialize a LinkedList of integers
LinkedList list = new LinkedList<int>();

// add some numbers to our list.
list.AddLast(3);
list.AddLast(5);
list.AddLast(8);

// the list currently is 3, 5, 8

list.AddFirst(2);
// the list now is 2, 3, 5, 8

list.RemoveFirst();
// the list is now 3, 5, 8

list.RemoveLast();
// the list is now 3, 5
```

LinkedList<T> . T . .

```
// Initalize a new queue of integers
var queue = new Queue<int>();

// Add some data
queue.Enqueue(6);
queue.Enqueue(4);
queue.Enqueue(9);

// Elements in a queue are stored in "first in, first out" order.
// The queue from first to last is: 6, 4, 9

// View the next element in the queue, without removing it.
Console.WriteLine(queue.Peek()); // prints 6

// Removes the first element in the queue, and returns it.
Console.WriteLine(queue.Dequeue()); // prints 6
Console.WriteLine(queue.Dequeue()); // prints 4
Console.WriteLine(queue.Dequeue()); // prints 9
```

! ConcurrentQueue .

C # : https://riptutorial.com/ko/csharp/topic/2344/c-sharp--

17: C

Examples

C # .

```
int value = await CSharpScript.EvaluateAsync<int>("15 * 89 + 95");
var span = await CSharpScript.EvaluateAsync<TimeSpan>("new DateTime(2016,1,1) -
DateTime.Now");
```

type `object` .

```
object value = await CSharpScript.EvaluateAsync("15 * 89 + 95");
```

C # : https://riptutorial.com/ko/csharp/topic/3780/c-sharp-

18: C

Examples

```
 public class AuthenticationHandler : DelegatingHandler
    {
        /// <summary>
        /// Holds request's header name which will contains token.
        /// </summary>
        private const string securityToken = "__RequestAuthToken";

        /// <summary>
        /// Default overridden method which performs authentication.
        /// </summary>
        /// <param name="request">Http request message.</param>
        /// <param name="cancellationToken">Cancellation token.</param>
        /// <returns>Returns http response message of type <see cref="HttpResponseMessage"/>
class asynchronously.</returns>
        protected override Task<HttpResponseMessage> SendAsync(HttpRequestMessage request,
CancellationToken cancellationToken)
        {
            if (request.Headers.Contains(securityToken))
            {
                bool authorized = Authorize(request);
                if (!authorized)
                {
                    return ApiHttpUtility.FromResult(request, false,
HttpStatusCode.Unauthorized, MessageTypes.Error, Resource.UnAuthenticatedUser);
                }
            }
            else
            {
                return ApiHttpUtility.FromResult(request, false, HttpStatusCode.BadRequest,
MessageTypes.Error, Resource.UnAuthenticatedUser);
            }

            return base.SendAsync(request, cancellationToken);
        }

        /// <summary>
        /// Authorize user by validating token.
        /// </summary>
        /// <param name="requestMessage">Authorization context.</param>
        /// <returns>Returns a value indicating whether current request is authenticated or
not.</returns>
        private bool Authorize(HttpRequestMessage requestMessage)
        {
            try
            {
                HttpRequest request = HttpContext.Current.Request;
                string token = request.Headers[securityToken];
                return SecurityUtility.IsTokenValid(token, request.UserAgent,
HttpContext.Current.Server.MapPath("~/Content/"), requestMessage);
            }
            catch (Exception)
            {
                return false;
```

```
                }
            }
        }
```

C # : https://riptutorial.com/ko/csharp/topic/5430/c-sharp--

19: C # SQLite

Examples

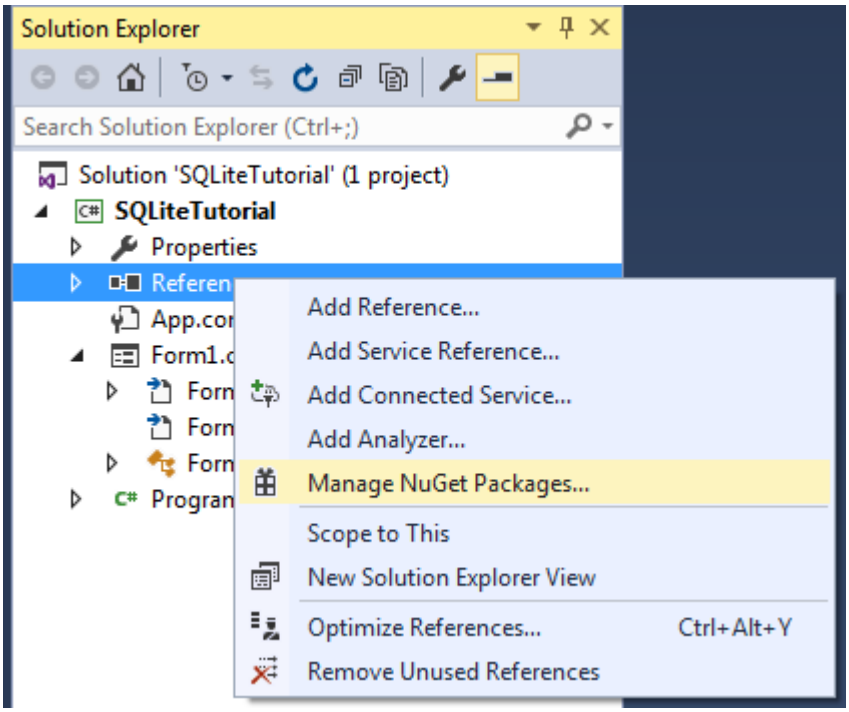
C # SQLite CRUD

SQLite . 2

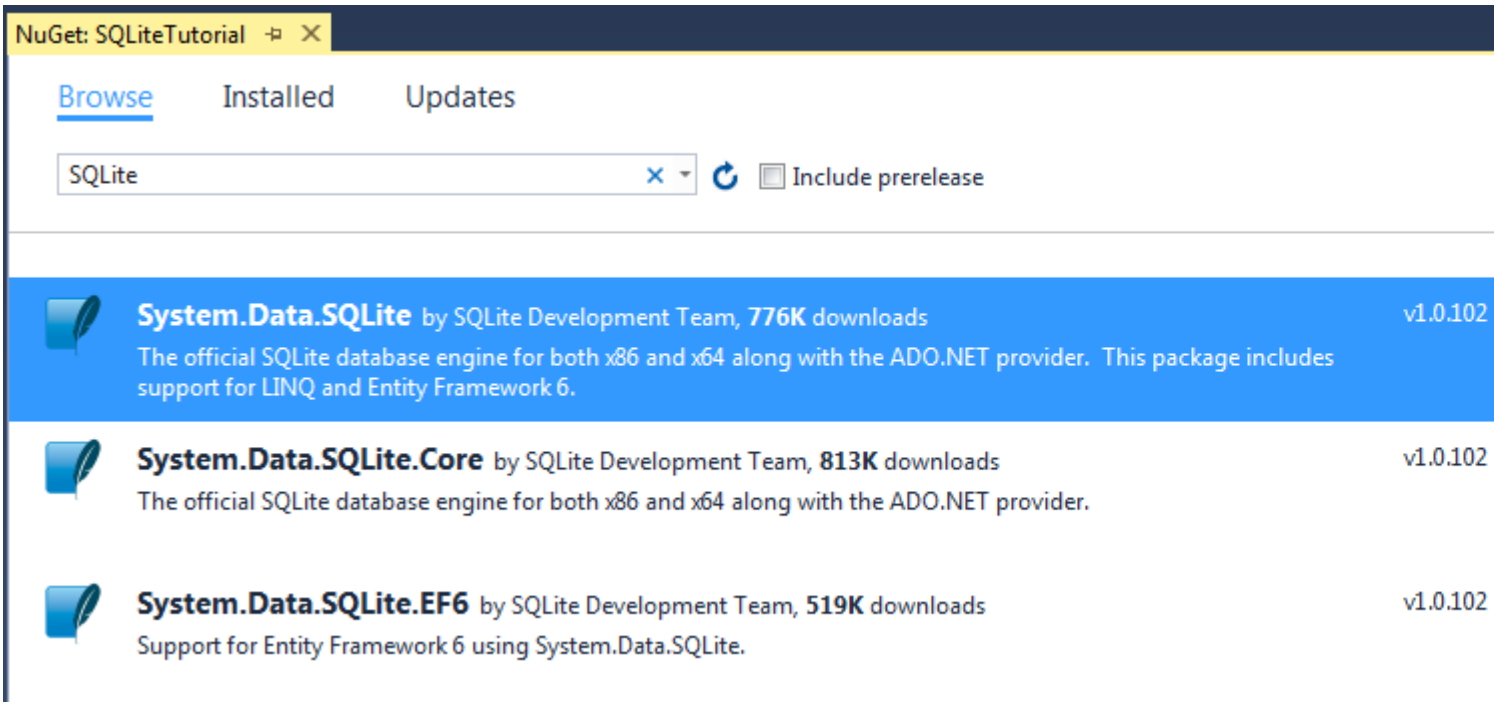
- SQLite DLL .
- NuGet SQLite

.

NuGet .



System.Data.SQLite Install .



.

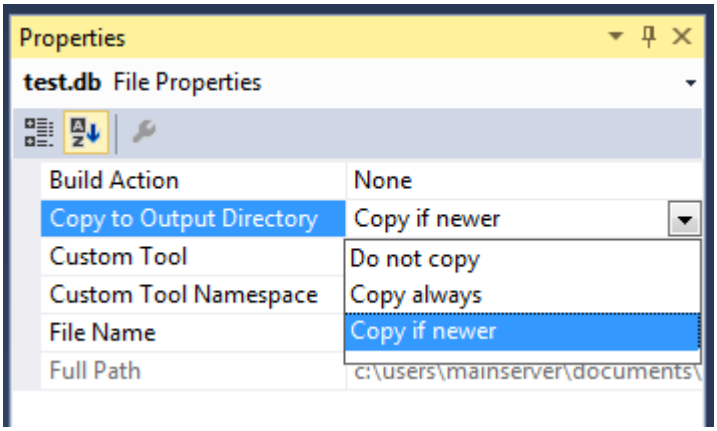
```
PM> Install-Package System.Data.SQLite
```

```
PM> Install-Package System.Data.SQLite.Core
```

.

SQLite .

```
CREATE TABLE User(
  Id INTEGER PRIMARY KEY AUTOINCREMENT,
  FirstName TEXT NOT NULL,
  LastName TEXT NOT NULL
);
```



User . .

```
private class User
{
```

```
    public string FirstName { get; set; }
    public string Lastname { get; set; }
}
```

, , .

```
private int ExecuteWrite(string query, Dictionary<string, object> args)
{
    int numberOfRowsAffected;

    //setup the connection to the database
    using (var con = new SQLiteConnection("Data Source=test.db"))
    {
        con.Open();

        //open a new command
        using (var cmd = new SQLiteCommand(query, con))
        {
            //set the arguments given in the query
            foreach (var pair in args)
            {
                cmd.Parameters.AddWithValue(pair.Key, pair.Value);
            }

            //execute the query and get the number of row affected
            numberOfRowsAffected = cmd.ExecuteNonQuery();
        }

        return numberOfRowsAffected;
    }
}
```

```
private DataTable Execute(string query)
{
    if (string.IsNullOrEmpty(query.Trim()))
        return null;

    using (var con = new SQLiteConnection("Data Source=test.db"))
    {
        con.Open();
        using (var cmd = new SQLiteCommand(query, con))
        {
            foreach (KeyValuePair<string, object> entry in args)
            {
                cmd.Parameters.AddWithValue(entry.Key, entry.Value);
            }

            var da = new SQLiteDataAdapter(cmd);

            var dt = new DataTable();
            da.Fill(dt);

            da.Dispose();
            return dt;
        }
    }
}
```

CRUD .

```
private int AddUser(User user)
{
    const string query = "INSERT INTO User(FirstName, LastName) VALUES(@firstName,
@lastName)";

    //here we are setting the parameter values that will be actually
    //replaced in the query in Execute method
    var args = new Dictionary<string, object>
    {
        {"@firstName", user.FirstName},
        {"@lastName", user.Lastname}
    };

    return ExecuteWrite(query, args);
}
```

```
private int EditUser(User user)
{
    const string query = "UPDATE User SET FirstName = @firstName, LastName = @lastName WHERE
Id = @id";

    //here we are setting the parameter values that will be actually
    //replaced in the query in Execute method
    var args = new Dictionary<string, object>
    {
        {"@id", user.Id},
        {"@firstName", user.FirstName},
        {"@lastName", user.Lastname}
    };

    return ExecuteWrite(query, args);
}
```

```
private int DeleteUser(User user)
{
    const string query = "Delete from User WHERE Id = @id";

    //here we are setting the parameter values that will be actually
    //replaced in the query in Execute method
    var args = new Dictionary<string, object>
    {
        {"@id", user.Id}
    };

    return ExecuteWrite(query, args);
}
```

Id

```
private User GetUserById(int id)
{
    var query = "SELECT * FROM User WHERE Id = @id";

    var args = new Dictionary<string, object>
    {
```

```
        {"@id", id}
    };

    DataTable dt = ExecuteRead(query, args);

    if (dt == null || dt.Rows.Count == 0)
    {
        return null;
    }

    var user = new User
    {
        Id = Convert.ToInt32(dt.Rows[0]["Id"]),
        FirstName = Convert.ToString(dt.Rows[0]["FirstName"]),
        Lastname = Convert.ToString(dt.Rows[0]["LastName"])
    };

    return user;
}
```

```
using (SQLiteConnection conn = new SQLiteConnection(@"Data
Source=data.db;Pooling=true;FailIfMissing=false"))
{
    conn.Open();
    using (SQLiteCommand cmd = new SQLiteCommand(conn))
    {
       cmd.CommandText = "query";
       using (SqlDataReader dr = cmd.ExecuteReader())
       {
           while(dr.Read())
           {
               //do stuff
           }
       }
    }
}
```

: `FailIfMissing` true `data.db` `data.db` . . .

C # SQLite : https://riptutorial.com/ko/csharp/topic/4960/c-sharp-sqlite-

20: C

OOP . . , , , , .

Examples

:

.

<> :

[]:

```
[private/public/protected/internal] class <Desired Class Name> [:[Inherited
class][,][[Interface Name 1],[Interface Name 2],...]
{
    //Your code
}
```

. . .

```
class MyClass
{
    int i = 100;
    public void getMyValue()
    {
        Console.WriteLine(this.i);//Will print number 100 in output
    }
}
```

`int i` public `getMyValue()` .

C # : https://riptutorial.com/ko/csharp/topic/9856/c-sharp---

21: C

- ()
- (int)
- int ()
- int (int maxValue)
- int (int minValue, int maxValue)

	. .
	. 0.
maxValue	. `Int32.MaxValue` .
	.

.

.

Examples

int

0 2147483647 .

```
Random rnd = new Random();
int randomNumber = rnd.Next();
```

0 1.0 . (1.0)

```
Random rnd = new Random();
var randomDouble = rnd.NextDouble();
```

int .

`minValue maxValue - 1` .

```
Random rnd = new Random();
var randomBetween10And20 = rnd.Next(10, 20);
```

`Random`

.

```
int seed = 5;
for (int i = 0; i < 2; i++)
{
   Console.WriteLine("Random instance " + i);
   Random rnd = new Random(seed);
   for (int j = 0; j < 5; j++)
   {
      Console.Write(rnd.Next());
      Console.Write(" ");
   }

   Console.WriteLine();
}
```

:

```
Random instance 0
726643700 610783965 564707973 1342984399 995276750
Random instance 1
726643700 610783965 564707973 1342984399 995276750
```

.

System.Guid.NewGuid().GetHashCode() .

```
Random rnd1 = new Random();
Random rnd2 = new Random();
Console.WriteLine("First 5 random number in rnd1");
for (int i = 0; i < 5; i++)
    Console.WriteLine(rnd1.Next());

Console.WriteLine("First 5 random number in rnd2");
for (int i = 0; i < 5; i++)
    Console.WriteLine(rnd2.Next());

rnd1 = new Random(Guid.NewGuid().GetHashCode());
rnd2 = new Random(Guid.NewGuid().GetHashCode());
Console.WriteLine("First 5 random number in rnd1 using Guid");
for (int i = 0; i < 5; i++)
    Console.WriteLine(rnd1.Next());
Console.WriteLine("First 5 random number in rnd2 using Guid");
for (int i = 0; i < 5; i++)
    Console.WriteLine(rnd2.Next());
```

Random .

```
Random rndSeeds = new Random();
Random rnd1 = new Random(rndSeeds.Next());
Random rnd2 = new Random(rndSeeds.Next());
```

rndSeeds Random . .

Next() a z Next() int char .

```
Random rnd = new Random();
char randomChar = (char)rnd.Next('a','z');
//'a' and 'z' are interpreted as ints for parameters for Next()
```

.

X% . NextDouble() .

```
var rnd = new Random();
var maxValue = 5000;
var percentage = rnd.NextDouble();
var result = maxValue * percentage;
//suppose NextDouble() returns .65, result will hold 65% of 5000: 3250.
```

C # : https://riptutorial.com/ko/csharp/topic/1975/c-sharp--

22: CLSCompliantAttribute

1. [: CLSCompliant (true)]
2. [CLSCompliant (true)]

CLSCompliantAttribute ()	CLS CLSCompliantAttribute .

CLS (Common Language Specification) CLI (CLS (Common Language Infrastructure)) CLS .

CLI

CLSCompliant . CLS . , CLR ( ) .

`CLSCompliantAttribute` CLS .

Examples

CLS

```
using System;

[assembly:CLSCompliant(true)]
namespace CLSDoc
{

    public class Cat
    {
        internal UInt16 _age = 0;
        private UInt16 _daysTillVacination = 0;

        //Warning CS3003  Type of 'Cat.DaysTillVacination' is not CLS-compliant
        protected UInt16 DaysTillVacination
        {
            get { return _daysTillVacination; }
        }

        //Warning    CS3003    Type of 'Cat.Age' is not CLS-compliant
        public UInt16 Age
        { get { return _age; } }

        //valid behaviour by CLS-compliant rules
        public int IncreaseAge()
        {
            int increasedAge = (int)_age + 1;

            return increasedAge;
        }

    }
}
```

CLS / .

CLS : / sbyte

```
using System;

[assembly:CLSCompliant(true)]
namespace CLSDoc
{

    public class Car
    {
        internal UInt16 _yearOfCreation = 0;

        //Warning CS3008  Identifier '_numberOfDoors' is not CLS-compliant
        //Warning CS3003  Type of 'Car._numberOfDoors' is not CLS-compliant
        public UInt32 _numberOfDoors = 0;

        //Warning    CS3003    Type of 'Car.YearOfCreation' is not CLS-compliant
        public UInt16 YearOfCreation
        {
            get { return _yearOfCreation; }
        }


        //Warning CS3002  Return type of 'Car.CalculateDistance()' is not CLS-compliant
        public UInt64 CalculateDistance()
        {
            return 0;
        }


        //Warning CS3002  Return type of 'Car.TestDummyUnsignedPointerMethod()' is not CLS-
compliant
        public UIntPtr TestDummyUnsignedPointerMethod()
        {
            int[] arr = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
            UIntPtr ptr = (UIntPtr)arr[0];


            return ptr;
        }

        //Warning CS3003  Type of 'Car.age' is not CLS-compliant
        public sbyte age = 120;


    }
}
```

CLS :

```
using System;

[assembly:CLSCompliant(true)]
namespace CLSDoc
{
```

```
    public class Car
    {
        //Warning    CS3005    Identifier 'Car.CALCULATEAge()' differing only in case is not
CLS-compliant
        public int CalculateAge()
        {
            return 0;
        }

        public int CALCULATEAge()
        {
            return 0;
        }

    }
}
```

Visual Basic / .

CLS : _

```
using System;

[assembly:CLSCompliant(true)]
namespace CLSDoc
{

    public class Car
    {
        //Warning CS3008  Identifier '_age' is not CLS-complian
        public int _age = 0;
    }

}
```

_ .

CLS : CLSComplaint

```
using System;

[assembly:CLSCompliant(true)]
namespace CLSDoc
{

    [CLSCompliant(false)]
    public class Animal
    {
        public int age = 0;
    }

    //Warning    CS3009    'Dog': base type 'Animal' is not CLS-compliant
    public class Dog : Animal
    {
    }

}
```

CLSCompliantAttribute : https://riptutorial.com/ko/csharp/topic/7214/clscompliantattribute

23: DateTime

Examples

DateTime.Add (TimeSpan)

```
// Calculate what day of the week is 36 days from this instant.
System.DateTime today = System.DateTime.Now;
System.TimeSpan duration = new System.TimeSpan(36, 0, 0, 0);
System.DateTime answer = today.Add(duration);
System.Console.WriteLine("{0:dddd}", answer);
```

DateTime.AddDays (Double)

dateTime .

```
DateTime today = DateTime.Now;
DateTime answer = today.AddDays(36);
Console.WriteLine("Today: {0:dddd}", today);
Console.WriteLine("36 days from today: {0:dddd}", answer);
```

.

```
DateTime today = DateTime.Now;
DateTime answer = today.AddDays(-3);
Console.WriteLine("Today: {0:dddd}", today);
Console.WriteLine("-3 days from today: {0:dddd}", answer);
```

DateTime.AddHours (Double)

```
double[] hours = {.08333, .16667, .25, .33333, .5, .66667, 1, 2,
                  29, 30, 31, 90, 365};
DateTime dateValue = new DateTime(2009, 3, 1, 12, 0, 0);

foreach (double hour in hours)
  Console.WriteLine("{0} + {1} hour(s) = {2}", dateValue, hour,
                    dateValue.AddHours(hour));
```

DateTime.AddMilliseconds (Double)

```
string dateFormat = "MM/dd/yyyy hh:mm:ss.fffffff";
DateTime date1 = new DateTime(2010, 9, 8, 16, 0, 0);
Console.WriteLine("Original date: {0} ({1:N0} ticks)\n",
                  date1.ToString(dateFormat), date1.Ticks);

DateTime date2 = date1.AddMilliseconds(1);
Console.WriteLine("Second date:   {0} ({1:N0} ticks)",
                  date2.ToString(dateFormat), date2.Ticks);
Console.WriteLine("Difference between dates: {0} ({1:N0} ticks)\n",
```

```
                  date2 - date1, date2.Ticks - date1.Ticks);

DateTime date3 = date1.AddMilliseconds(1.5);
Console.WriteLine("Third date:    {0} ({1:N0} ticks)",
                  date3.ToString(dateFormat), date3.Ticks);
Console.WriteLine("Difference between dates: {0} ({1:N0} ticks)",
                  date3 - date1, date3.Ticks - date1.Ticks);
```

DateTime.Compare (t1, t2)

```
DateTime date1 = new DateTime(2009, 8, 1, 0, 0, 0);
DateTime date2 = new DateTime(2009, 8, 1, 12, 0, 0);
int result = DateTime.Compare(date1, date2);
string relationship;

if (result < 0)
    relationship = "is earlier than";
else if (result == 0)
    relationship = "is the same time as";
else relationship = "is later than";

Console.WriteLine("{0} {1} {2}", date1, relationship, date2);
```

DateTime.DaysInMonth (Int32, Int32)

```
const int July = 7;
const int Feb = 2;

int daysInJuly = System.DateTime.DaysInMonth(2001, July);
Console.WriteLine(daysInJuly);

// daysInFeb gets 28 because the year 1998 was not a leap year.
int daysInFeb = System.DateTime.DaysInMonth(1998, Feb);
Console.WriteLine(daysInFeb);

// daysInFebLeap gets 29 because the year 1996 was a leap year.
int daysInFebLeap = System.DateTime.DaysInMonth(1996, Feb);
Console.WriteLine(daysInFebLeap);
```

DateTime.AddYears (Int32)

dateTime .

```
DateTime baseDate = new DateTime(2000, 2, 29);
Console.WriteLine("Base Date: {0:d}\n", baseDate);

// Show dates of previous fifteen years.
for (int ctr = -1; ctr >= -15; ctr--)
   Console.WriteLine("{0,2} year(s) ago:{1:d}",
                     Math.Abs(ctr), baseDate.AddYears(ctr));

Console.WriteLine();

// Show dates of next fifteen years.
for (int ctr = 1; ctr <= 15; ctr++)
```

```
   Console.WriteLine("{0,2} year(s) from now: {1:d}",
                     ctr, baseDate.AddYears(ctr));
```

DateTime

Wikipedia .

1. . I / O .
2. (: I / O).

. DateTime . . :

```
        DateTime sample = new DateTime(2016, 12, 25);
        sample.AddDays(1);
        Console.WriteLine(sample.ToShortDateString());
```

'26 / 12 / 2016 ' . AddDays . AddDays .

```
        sample = sample.AddDays(1);
```

DateTime.Parse (String)

```
// Converts the string representation of a date and time to its DateTime equivalent

var dateTime = DateTime.Parse("14:23 22 Jul 2016");

Console.WriteLine(dateTime.ToString());
```

DateTime.TryParse (String, DateTime)

```
// Converts the specified string representation of a date and time to its DateTime equivalent
and returns a value that indicates whether the conversion succeeded

string[] dateTimeStrings = new []{
    "14:23 22 Jul 2016",
    "99:23 2x Jul 2016",
    "22/7/2016 14:23:00"
};

foreach(var dateTimeString in dateTimeStrings){

    DateTime dateTime;

    bool wasParsed = DateTime.TryParse(dateTimeString, out dateTime);

    string result = dateTimeString +
        (wasParsed
            ? $"was parsed to {dateTime}"
            : "can't be parsed to DateTime");

    Console.WriteLine(result);
}
```

TryParse

() DateTimes .

```
DateTime dateResult;
var dutchDateString = "31 oktober 1999 04:20";
var dutchCulture = CultureInfo.CreateSpecificCulture("nl-NL");
DateTime.TryParse(dutchDateString, dutchCulture, styles, out dateResult);
// output {31/10/1999 04:20:00}
```

:

```
DateTime.Parse(dutchDateString, dutchCulture)
// output {31/10/1999 04:20:00}
```

for-loop DateTime

```
// This iterates through a range between two DateTimes
// with the given iterator (any of the Add methods)

DateTime start = new DateTime(2016, 01, 01);
DateTime until = new DateTime(2016, 02, 01);

// NOTICE: As the add methods return a new DateTime you have
// to overwrite dt in the iterator like dt = dt.Add()
for (DateTime dt = start; dt < until; dt = dt.AddDays(1))
{
    Console.WriteLine("Added {0} days. Resulting DateTime: {1}",
                      (dt - start).Days, dt.ToString());
}
```

TimeSpan .

DateTime ToString, ToShortDateString, ToLongDateString ToString

```
using System;

public class Program
{
    public static void Main()
    {
        var date = new DateTime(2016,12,31);

        Console.WriteLine(date.ToString());        //Outputs: 12/31/2016 12:00:00 AM
        Console.WriteLine(date.ToShortDateString()); //Outputs: 12/31/2016
        Console.WriteLine(date.ToLongDateString()); //Outputs: Saturday, December 31, 2016
        Console.WriteLine(date.ToString("dd/MM/yyyy"));    //Outputs: 31/12/2016
    }
}
```

DateTime.Today . DateTime . .ToString() .

:

```
Console.WriteLine(DateTime.Today);
```

.

DateTime

DateTime

DateTimeFormatInfo . DateTimeFormatInfo .

```
//Create datetime
DateTime dt = new DateTime(2016,08,01,18,50,23,230);

var t = String.Format("{0:t}", dt); // "6:50 PM"                                ShortTime
var d = String.Format("{0:d}", dt); // "8/1/2016"                               ShortDate
var T = String.Format("{0:T}", dt); // "6:50:23 PM"                             LongTime
var D = String.Format("{0:D}", dt); // "Monday, August 1, 2016"                 LongDate
var f = String.Format("{0:f}", dt); // "Monday, August 1, 2016 6:50 PM"
LongDate+ShortTime
var F = String.Format("{0:F}", dt); // "Monday, August 1, 2016 6:50:23 PM"     FullDateTime
var g = String.Format("{0:g}", dt); // "8/1/2016 6:50 PM"
ShortDate+ShortTime
var G = String.Format("{0:G}", dt); // "8/1/2016 6:50:23 PM"
ShortDate+LongTime
var m = String.Format("{0:m}", dt); // "August 1"                              MonthDay
var y = String.Format("{0:y}", dt); // "August 2016"                           YearMonth
var r = String.Format("{0:r}", dt); // "SMon, 01 Aug 2016 18:50:23 GMT"        RFC1123
var s = String.Format("{0:s}", dt); // "2016-08-01T18:50:23"                   SortableDateTime
var u = String.Format("{0:u}", dt); // "2016-08-01 18:50:23Z"
UniversalSortableDateTime
```

.

- `y` ()
- `M` ()
- `d` ()
- `h` (12)
- `H` (24)
- `m` ()
- `s` ()
- `f` ( )
- `F` (, 0)
- `t` (PM AM)
- `z` ().

```
var year =       String.Format("{0:y yy yyy yyyy}", dt); // "16 16 2016 2016"  year
var month =      String.Format("{0:M MM MMM MMMM}", dt); // "8 08 Aug August"  month
var day =        String.Format("{0:d dd ddd dddd}", dt); // "1 01 Mon Monday"  day
var hour =       String.Format("{0:h hh H HH}",     dt); // "6 06 18 18"       hour 12/24
var minute =     String.Format("{0:m mm}",          dt); // "50 50"            minute
var secound =    String.Format("{0:s ss}",          dt); // "23 23"            second
var fraction =   String.Format("{0:f ff fff ffff}", dt); // "2 23 230 2300"    sec.fraction
var fraction2 =  String.Format("{0:F FF FFF FFFF}", dt); // "2 23 23 23"       without zeroes
```

```
var period =     String.Format("{0:t tt}",          dt); // "P PM"             A.M. or P.M.
var zone =       String.Format("{0:z zz zzz}",      dt); // "+0 +00 +00:00"    time zone
```

/ () sepatator :) .

MSDN .

DateTime.ParseExact (String, String, IFormatProvider)

culture DateTime . .

DateTime

culture DateTime `08-07-2016 11:30:12 PM` `MM-dd-yyyy hh:mm:ss tt` DateTime

```
string str = "08-07-2016 11:30:12 PM";
DateTime date = DateTime.ParseExact(str, "MM-dd-yyyy hh:mm:ss tt",
CultureInfo.CurrentCulture);
```

DateTime

`dd-MM-yy hh:mm:ss tt` DateTime `DateTime` .

```
string str = "17-06-16 11:30:12 PM";
DateTime date = DateTime.ParseExact(str, "dd-MM-yy hh:mm:ss tt",
CultureInfo.InvariantCulture);
```

DateTime

, '23 -12-2016 ' '12 / 23 / 2016' `DateTime` .

```
   string date =  '23-12-2016' or date = 12/23/2016';
   string[] formats = new string[] {"dd-MM-yyyy","MM/dd/yyyy"}; // even can add more possible
formats.
   DateTime date = DateTime.ParseExact(date,formats,
CultureInfo.InvariantCulture,DateTimeStyles.None);
```

: CultureInfo **`System.Globalization` .**

DateTime.TryParseExact (, , IFormatProvider, DateTimeStyles, DateTime)

, DateTime . . .

```
CultureInfo enUS = new CultureInfo("en-US");
string dateString;
System.DateTime dateValue;
```

.

```
dateString = " 5/01/2009 8:30 AM";
if (DateTime.TryParseExact(dateString, "g", enUS, DateTimeStyles.None, out dateValue))
```

```
{
   Console.WriteLine("Converted '{0}' to {1} ({2}).", dateString, dateValue, dateValue.Kind);
}
else
{
   Console.WriteLine("'{0}' is not in an acceptable format.", dateString);
}


// Allow a leading space in the date string.
if(DateTime.TryParseExact(dateString, "g", enUS, DateTimeStyles.AllowLeadingWhite, out
dateValue))
{
   Console.WriteLine("Converted '{0}' to {1} ({2}).", dateString, dateValue, dateValue.Kind);
else
{
   Console.WriteLine("'{0}' is not in an acceptable format.", dateString);
}
```

M MM .

```
dateString = "5/01/2009 09:00";
if(DateTime.TryParseExact(dateString, "M/dd/yyyy hh:mm", enUS, DateTimeStyles.None, out
dateValue))
{
    Console.WriteLine("Converted '{0}' to {1} ({2}).", dateString, dateValue, dateValue.Kind);
}
else
{
   Console.WriteLine("'{0}' is not in an acceptable format.", dateString);
}

// Allow a leading space in the date string.
if(DateTime.TryParseExact(dateString, "MM/dd/yyyy hh:mm", enUS, DateTimeStyles.None, out
dateValue))
{
   Console.WriteLine("Converted '{0}' to {1} ({2}).", dateString, dateValue, dateValue.Kind);
}
else
{
   Console.WriteLine("'{0}' is not in an acceptable format.", dateString);
}
```

.

```
dateString = "05/01/2009 01:30:42 PM -05:00";
if (DateTime.TryParseExact(dateString, "MM/dd/yyyy hh:mm:ss tt zzz", enUS,
DateTimeStyles.None, out dateValue))
{
    Console.WriteLine("Converted '{0}' to {1} ({2}).", dateString, dateValue, dateValue.Kind);
}
else
{
   Console.WriteLine("'{0}' is not in an acceptable format.", dateString);
}


// Allow a leading space in the date string.
if (DateTime.TryParseExact(dateString, "MM/dd/yyyy hh:mm:ss tt zzz", enUS,
```

```
DateTimeStyles.AdjustToUniversal, out dateValue))
{
   Console.WriteLine("Converted '{0}' to {1} ({2}).", dateString, dateValue, dateValue.Kind);
}
else
{
   Console.WriteLine("'{0}' is not in an acceptable format.", dateString);
}
```

UTC represengting .

```
dateString = "2008-06-11T16:11:20.0904778Z";
if(DateTime.TryParseExact(dateString, "o", CultureInfo.InvariantCulture, DateTimeStyles.None,
out dateValue))
{
   Console.WriteLine("Converted '{0}' to {1} ({2}).", dateString, dateValue, dateValue.Kind);
}
else
{
  Console.WriteLine("'{0}' is not in an acceptable format.", dateString);
}

if (DateTime.TryParseExact(dateString, "o", CultureInfo.InvariantCulture,
DateTimeStyles.RoundtripKind, out dateValue))
{
   Console.WriteLine("Converted '{0}' to {1} ({2}).", dateString, dateValue, dateValue.Kind);
}
else
{
   Console.WriteLine("'{0}' is not in an acceptable format.", dateString);
}
```

```
' 5/01/2009 8:30 AM' is not in an acceptable format.
Converted ' 5/01/2009 8:30 AM' to 5/1/2009 8:30:00 AM (Unspecified).
Converted '5/01/2009 09:00' to 5/1/2009 9:00:00 AM (Unspecified).
'5/01/2009 09:00' is not in an acceptable format.
Converted '05/01/2009 01:30:42 PM -05:00' to 5/1/2009 11:30:42 AM (Local).
Converted '05/01/2009 01:30:42 PM -05:00' to 5/1/2009 6:30:42 PM (Utc).
Converted '2008-06-11T16:11:20.0904778Z' to 6/11/2008 9:11:20 AM (Local).
Converted '2008-06-11T16:11:20.0904778Z' to 6/11/2008 4:11:20 PM (Utc).
```

DateTime : https://riptutorial.com/ko/csharp/topic/1587/datetime-

24: FileSystemWatcher

- Public FileSystemWatcher ()
- FileSystemWatcher ( )
- FileSystemWatcher (,)

(UNC) . . "* .txt" .	

Examples

FileWatcher

`FileSystemWatcher` . **LastWrite LastAccess** , , . , . .

System.Diagnostics System.IO .

```
FileSystemWatcher watcher;

private void watch()
{
  // Create a new FileSystemWatcher and set its properties.
  watcher = new FileSystemWatcher();
  watcher.Path = path;

 /* Watch for changes in LastAccess and LastWrite times, and
       the renaming of files or directories. */
  watcher.NotifyFilter = NotifyFilters.LastAccess | NotifyFilters.LastWrite
                         | NotifyFilters.FileName | NotifyFilters.DirectoryName;

  // Only watch text files.
  watcher.Filter = "*.txt*";

  // Add event handler.
  watcher.Changed += new FileSystemEventHandler(OnChanged);
  // Begin watching.
  watcher.EnableRaisingEvents = true;
}

// Define the event handler.
private void OnChanged(object source, FileSystemEventArgs e)
{
  //Copies file to another directory or another action.
  Console.WriteLine("File: " +  e.FullPath + " " + e.ChangeType);
}
```

IsFileReady

FileSystemWatcher FileWatcher . .

:

1GB . apr (Explorer.exe) . .

.

```
 public static bool IsFileReady(String sFilename)
{
    // If the file can be opened for exclusive access it means that the file
    // is no longer locked by another process.
    try
    {
        using (FileStream inputStream = File.Open(sFilename, FileMode.Open, FileAccess.Read,
FileShare.None))
        {
            if (inputStream.Length > 0)
            {
                return true;
            }
            else
            {
                return false;
            }

        }
    }
    catch (Exception)
    {
        return false;
    }
}
```

FileSystemWatcher : https://riptutorial.com/ko/csharp/topic/5061/filesystemwatcher

25: Func

- `public delegate TResult Func<in T, out TResult>(T arg)`
- `public delegate TResult Func<in T1, in T2, out TResult>(T1 arg1, T2 arg2)`
- `public delegate TResult Func<in T1, in T2, in T3, out TResult>(T1 arg1, T2 arg2, T3 arg3)`
- `public delegate TResult Func<in T1, in T2, in T3, in T4, out TResult>(T1 arg1, T2 arg2, T3 arg3, T4 arg4)`

arg arg1	()
arg2	
arg3	
arg4	
T T1	()
T2	
T3	
T4	
TResult	

Examples

.

```
static DateTime UTCNow()
{
    return DateTime.UtcNow;
}

static DateTime LocalNow()
{
    return DateTime.Now;
}

static void Main(string[] args)
{
    Func<DateTime> method = UTCNow;
    // method points to the UTCNow method
    // that retuns current UTC time
    DateTime utcNow = method();

    method = LocalNow;
    // now method points to the LocalNow method
    // that returns local time

    DateTime localNow = method();
}
```

```
static int Sum(int a, int b)
{
    return a + b;
}

static int Multiplication(int a, int b)
{
    return a * b;
}

static void Main(string[] args)
{
    Func<int, int, int> method = Sum;
    // method points to the Sum method
    // that retuns 1 int variable and takes 2 int variables
    int sum = method(1, 1);

    method = Multiplication;
    // now method points to the Multiplication method

    int multiplication = method(1, 1);
}
```

.

```
Func<int, int> square = delegate (int x) { return x * x; }
```

.

```
Func<int, int> square = x => x * x;
```

square .

```
var sq = square.Invoke(2);
```

:

```
var sq = square(2);
```

.

```
Func<int, int> sum = delegate (int x, int y) { return x + y; } // error
Func<int, int> sum = (x, y) => x + y; // error
```

Func .

```
// Simple hierarchy of classes.
public class Person { }
public class Employee : Person { }

class Program
{
    static Employee FindByTitle(String title)
```

```
    {
        // This is a stub for a method that returns
        // an employee that has the specified title.
        return new Employee();
    }

    static void Test()
    {
        // Create an instance of the delegate without using variance.
        Func<String, Employee> findEmployee = FindByTitle;

        // The delegate expects a method to return Person,
        // but you can assign it a method that returns Employee.
        Func<String, Person> findPerson = FindByTitle;

        // You can also assign a delegate
        // that returns a more derived type
        // to a delegate that returns a less derived type.
        findPerson = findEmployee;

    }
}
```

Func : https://riptutorial.com/ko/csharp/topic/2769/func-

26: Google

JSON Google .

Examples

ASP.NET MVC Google (Gmail) "Google API " . .

```
using Google.Contacts;
using Google.GData.Client;
using Google.GData.Contacts;
using Google.GData.Extensions;
```

.

```
using Google.Contacts;
using Google.GData.Client;
using Google.GData.Contacts;
using Google.GData.Extensions;
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using System.Text;
using System.Web;
using System.Web.Mvc;

namespace GoogleContactImport.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        public ActionResult Import()
        {
            string clientId = "";  // here you need to add your google client id
            string redirectUrl = "http://localhost:1713/Home/AddGoogleContacts"; // here your
redirect action method NOTE: you need to configure same url in google console
            Response.Redirect("https://accounts.google.com/o/oauth2/auth?redirect_uri=" +
redirectUrl + "&amp&response_type=code&amp&client_id=" + clientId +
"&amp&scope=https://www.google.com/m8/feeds/&amp;approval_prompt=force&amp;access_type=offline");

            return View();
        }

        public ActionResult AddGoogleContacts()
        {
            string code = Request.QueryString["code"];
            if (!string.IsNullOrEmpty(code))
            {
```

```
                var contacts = GetAccessToken().ToArray();
                if (contacts.Length > 0)
                {
                  // You will get all contacts here
                    return View("Index",contacts);
                }
                else
                {
                    return RedirectToAction("Index","Home");
                }
            }
            else
            {
                return RedirectToAction("Index", "Home");
            }
        }
        public List<GmailContacts> GetAccessToken()
        {
            string code = Request.QueryString["code"];
            string google_client_id = ""; //your google client Id
            string google_client_sceret = ""; // your google secret key
            string google_redirect_url = "http://localhost:1713/MyContact/AddGoogleContacts";


            HttpWebRequest webRequest =
(HttpWebRequest)WebRequest.Create("https://accounts.google.com/o/oauth2/token");
            webRequest.Method = "POST";
            string parameters = "code=" + code + "&client_id=" + google_client_id +
"&client_secret=" + google_client_sceret + "&redirect_uri=" + google_redirect_url +
"&grant_type=authorization_code";
            byte[] byteArray = Encoding.UTF8.GetBytes(parameters);
            webRequest.ContentType = "application/x-www-form-urlencoded";
            webRequest.ContentLength = byteArray.Length;
            Stream postStream = webRequest.GetRequestStream();
            // Add the post data to the web request
            postStream.Write(byteArray, 0, byteArray.Length);
            postStream.Close();
            WebResponse response = webRequest.GetResponse();
            postStream = response.GetResponseStream();
            StreamReader reader = new StreamReader(postStream);
            string responseFromServer = reader.ReadToEnd();
            GooglePlusAccessToken serStatus =
JsonConvert.DeserializeObject<GooglePlusAccessToken>(responseFromServer);
            /*End*/
            return GetContacts(serStatus);
        }

        public List<GmailContacts> GetContacts(GooglePlusAccessToken serStatus)
        {
            string google_client_id = "";  //client id
            string google_client_sceret = ""; //secret key
            /*Get Google Contacts From Access Token and Refresh Token*/
            // string refreshToken = serStatus.refresh_token;
            string accessToken = serStatus.access_token;
            string scopes = "https://www.google.com/m8/feeds/contacts/default/full/";
            OAuth2Parameters oAuthparameters = new OAuth2Parameters()
            {
                ClientId = google_client_id,
                ClientSecret = google_client_sceret,
                RedirectUri = "http://localhost:1713/Home/AddGoogleContacts",
                Scope = scopes,
```

```
                AccessToken = accessToken,
                //  RefreshToken = refreshToken
            };

            RequestSettings settings = new RequestSettings("App Name", oAuthparameters);
            ContactsRequest cr = new ContactsRequest(settings);
            ContactsQuery query = new
ContactsQuery(ContactsQuery.CreateContactsUri("default"));
            query.NumberToRetrieve = 5000;
            Feed<Contact> ContactList = cr.GetContacts();

            List<GmailContacts> olist = new List<GmailContacts>();
            foreach (Contact contact in ContactList.Entries)
            {
                foreach (EMail email in contact.Emails)
                {
                    GmailContacts gc = new GmailContacts();
                    gc.EmailID = email.Address;
                    var a = contact.Name.FullName;
                    olist.Add(gc);
                }
            }
            return olist;
        }

        public class GmailContacts
        {
            public string EmailID
            {
                get { return _EmailID; }
                set { _EmailID = value; }
            }
            private string _EmailID;
        }

        public class GooglePlusAccessToken
        {

            public GooglePlusAccessToken()
            { }


            public string access_token
            {
                get { return _access_token; }
                set { _access_token = value; }
            }
            private string _access_token;

            public string token_type
            {
                get { return _token_type; }
                set { _token_type = value; }
            }
            private string _token_type;

            public string expires_in
            {
                get { return _expires_in; }
                set { _expires_in = value; }
            }
```

```
            private string _expires_in;

        }
    }
}
```

.

.

```
<a href='@Url.Action("Import", "Home")'>Import Google Contacts</a>
```

Google : https://riptutorial.com/ko/csharp/topic/6744/google---

27: HTTP

Examples

HTTP POST

```
using System.Net;
using System.IO;

...

string requestUrl = "https://www.example.com/submit.html";
HttpWebRequest request = HttpWebRequest.CreateHttp(requestUrl);
request.Method = "POST";

// Optionally, set properties of the HttpWebRequest, such as:
request.AutomaticDecompression = DecompressionMethods.Deflate | DecompressionMethods.GZip;
request.ContentType = "application/x-www-form-urlencoded";
// Could also set other HTTP headers such as Request.UserAgent, Request.Referer,
// Request.Accept, or other headers via the Request.Headers collection.

// Set the POST request body data. In this example, the POST data is in
// application/x-www-form-urlencoded format.
string postData = "myparam1=myvalue1&myparam2=myvalue2";
using (var writer = new StreamWriter(request.GetRequestStream()))
{
    writer.Write(postData);
}

// Submit the request, and get the response body from the remote server.
string responseFromRemoteServer;
using (HttpWebResponse response = (HttpWebResponse)request.GetResponse())
{
    using (StreamReader reader = new StreamReader(response.GetResponseStream()))
    {
        responseFromRemoteServer = reader.ReadToEnd();
    }
}
```

HTTP GET

```
using System.Net;
using System.IO;

...

string requestUrl = "https://www.example.com/page.html";
HttpWebRequest request = HttpWebRequest.CreateHttp(requestUrl);

// Optionally, set properties of the HttpWebRequest, such as:
request.AutomaticDecompression = DecompressionMethods.GZip | DecompressionMethods.Deflate;
request.Timeout = 2 * 60 * 1000; // 2 minutes, in milliseconds

// Submit the request, and get the response body.
string responseBodyFromRemoteServer;
```

```
using (HttpWebResponse response = (HttpWebResponse)request.GetResponse())
{
    using (StreamReader reader = new StreamReader(response.GetResponseStream()))
    {
        responseBodyFromRemoteServer = reader.ReadToEnd();
    }
}
```

HTTP (: 404)

```
using System.Net;

...

string serverResponse;
try
{
    // Call a method that performs an HTTP request (per the above examples).
    serverResponse = PerformHttpRequest();
}
catch (WebException ex)
{
    if (ex.Status == WebExceptionStatus.ProtocolError)
    {
        HttpWebResponse response = ex.Response as HttpWebResponse;
        if (response != null)
        {
            if ((int)response.StatusCode == 404) // Not Found
            {
                // Handle the 404 Not Found error
                // ...
            }
            else
            {
                // Could handle other response.StatusCode values here.
                // ...
            }
        }
    }
    else
    {
        // Could handle other error conditions here, such as
WebExceptionStatus.ConnectFailure.
        // ...
    }
}
```

JSON HTTP POST

```
public static async Task PostAsync(this Uri uri, object value)
{
    var content = new ObjectContext(value.GetType(), value, new JsonMediaTypeFormatter());

    using (var client = new HttpClient())
    {
        return await client.PostAsync(uri, content);
    }
```

```
}

. . .

var uri = new Uri("http://stackoverflow.com/documentation/c%23/1971/performing-http-
requests");
await uri.PostAsync(new { foo = 123.45, bar = "Richard Feynman" });
```

HTTP GET JSON

```
public static async Task<TResult> GetAnsync<TResult>(this Uri uri)
{
    using (var client = new HttpClient())
    {
        var message = await client.GetAsync(uri);

        if (!message.IsSuccessStatusCode)
            throw new Exception();

        return message.ReadAsAsync<TResult>();
    }
}

. . .

public class Result
{
    public double foo { get; set; }

    public string bar { get; set; }
}

var uri = new Uri("http://stackoverflow.com/documentation/c%23/1971/performing-http-
requests");
var result = await uri.GetAsync<Result>();
```

HTML ()

```
string contents = "";
string url = "http://msdn.microsoft.com";

using (System.Net.WebClient client = new System.Net.WebClient())
{
    contents = client.DownloadString(url);
}

Console.WriteLine(contents);
```

HTTP : https://riptutorial.com/ko/csharp/topic/1971/http--

28: ICloneable

- ICloneable.Clone () { Clone (); } // Clone () .
- Foo Clone () { Foo (this); } // .

CLR object Clone() . .

. . .

: C# .

Examples

ICloneable

ICloneable . Clone() , Clone() object Clone() object Clone() .

```
public class Person : ICloneable
{
    // Contents of class
    public string Name { get; set; }
    public int Age { get; set; }
    // Constructor
    public Person(string name, int age)
    {
        this.Name=name;
        this.Age=age;
    }
    // Copy Constructor
    public Person(Person other)
    {
        this.Name=other.Name;
        this.Age=other.Age;
    }

    #region ICloneable Members
    // Type safe Clone
    public Person Clone() { return new Person(this); }
    // ICloneable implementation
    object ICloneable.Clone()
    {
        return Clone();
    }
    #endregion
}
```

.

```
{
    Person bob=new Person("Bob", 25);
    Person bob_clone=bob.Clone();
    Debug.Assert(bob_clone.Name==bob.Name);
```

```
    bob.Age=56;
    Debug.Assert(bob.Age!=bob.Age);
}
```

`bob` `bob_clone` . () .

ICloneable

= ICloneable . `ICloneable` .

( ) .

```
// Structs are recommended to be immutable objects
[ImmutableObject(true)]
public struct Person : ICloneable
{
    // Contents of class
    public string Name { get; private set; }
    public int Age { get; private set; }
    // Constructor
    public Person(string name, int age)
    {
        this.Name=name;
        this.Age=age;
    }
    // Copy Constructor
    public Person(Person other)
    {
        // The assignment operator copies all members
        this=other;
    }

    #region ICloneable Members
    // Type safe Clone
    public Person Clone() { return new Person(this); }
    // ICloneable implementation
    object ICloneable.Clone()
    {
        return Clone();
    }
    #endregion
}
```

.

```
static void Main(string[] args)
{
    Person bob=new Person("Bob", 25);
    Person bob_clone=bob.Clone();
    Debug.Assert(bob_clone.Name==bob.Name);
}
```

ICloneable : https://riptutorial.com/ko/csharp/topic/7917/icloneable

29: IComparable

Examples

:

```
public class Version : IComparable<Version>
{
    public int[] Parts { get; }

    public Version(string value)
    {
        if (value == null)
            throw new ArgumentNullException();
        if (!Regex.IsMatch(value, @"^[0-9]+(\.[0-9]+)*$"))
            throw new ArgumentException("Invalid format");
        var parts = value.Split('.');
        Parts = new int[parts.Length];
        for (var i = 0; i < parts.Length; i++)
            Parts[i] = int.Parse(parts[i]);
    }

    public override string ToString()
    {
        return string.Join(".", Parts);
    }

    public int CompareTo(Version that)
    {
        if (that == null) return 1;
        var thisLength = this.Parts.Length;
        var thatLength = that.Parts.Length;
        var maxLength = Math.Max(thisLength, thatLength);
        for (var i = 0; i < maxLength; i++)
        {
            var thisPart = i < thisLength ? this.Parts[i] : 0;
            var thatPart = i < thatLength ? that.Parts[i] : 0;
            if (thisPart < thatPart) return -1;
            if (thisPart > thatPart) return 1;
        }
        return 0;
    }
}
```

:

```
Version a, b;

a = new Version("4.2.1");
b = new Version("4.2.6");
a.CompareTo(b); // a < b : -1

a = new Version("2.8.4");
b = new Version("2.8.0");
a.CompareTo(b); // a > b : 1
```

```
a = new Version("5.2");
b = null;
a.CompareTo(b); // a > b : 1

a = new Version("3");
b = new Version("3.6");
a.CompareTo(b); // a < b : -1

var versions = new List<Version>
{
    new Version("2.0"),
    new Version("1.1.5"),
    new Version("3.0.10"),
    new Version("1"),
    null,
    new Version("1.0.1")
};

versions.Sort();

foreach (var version in versions)
    Console.WriteLine(version?.ToString() ?? "NULL");
```

:

1
1.0.1
1.1.5
2.0
3.0.10

:

Ideone□ □□□ □□

IComparable : https://riptutorial.com/ko/csharp/topic/4222/icomparable

30: IDisposable

- `IDisposable` `Dispose` . `Dispose` CLR .
- finalizer . `Dispose` `Dispose` .
- `Dispose` . `private bool` `Dispose` `true` .

Examples

. BCL `SqlConnection` . `IDisposable` . .

finalizer .

```
public class ObjectWithManagedResourcesOnly : IDisposable
{
    private SqlConnection sqlConnection = new SqlConnection();

    public void Dispose()
    {
        sqlConnection.Dispose();
    }
}
```

. . . `Dispose(bool)` Finalizer `Dispose` .

```
public class ManagedAndUnmanagedObject : IDisposable
{
    private SqlConnection sqlConnection = new SqlConnection();
    private UnmanagedHandle unmanagedHandle = Win32.SomeUnmanagedResource();
    private bool disposed;

    public void Dispose()
    {
        Dispose(true); // client called dispose
        GC.SuppressFinalize(this); // tell the GC to not execute the Finalizer
    }

    protected virtual void Dispose(bool disposeManaged)
    {
        if (!disposed)
        {
            if (disposeManaged)
            {
                if (sqlConnection != null)
                {
                    sqlConnection.Dispose();
                }
            }

            unmanagedHandle.Release();

            disposed = true;
        }
```

```
    }

    ~ManagedAndUnmanagedObject()
    {
        Dispose(false);
    }
}
```

IDisposable, Dispose

.NET Framework .

```
public interface IDisposable
{
  void Dispose();
}
```

`Dispose()` . . GC `Dispose()` .

```
public void Dispose()
{
   if (null != this.CurrentDatabaseConnection)
   {
       this.CurrentDatabaseConnection.Dispose();
       this.CurrentDatabaseConnection = null;
   }
}
```

win32 `SafeHandle` / .

`IDisposable` . `Dispose virtual` .

```
public class Parent : IDisposable
{
    private ManagedResource parentManagedResource = new ManagedResource();

    public virtual void Dispose()
    {
        if (parentManagedResource != null)
        {
            parentManagedResource.Dispose();
        }
    }
}

public class Child : Parent
{
    private ManagedResource childManagedResource = new ManagedResource();

    public override void Dispose()
    {
        if (childManagedResource != null)
        {
            childManagedResource.Dispose();
        }
        //clean up the parent's resources
```

```
        base.Dispose();
    }
}
```

IDisposable using .

```
using (var foo = new Foo())
{
    // do foo stuff
} // when it reaches here foo.Dispose() will get called

public class Foo : IDisposable
{
    public void Dispose()
    {
        Console.WriteLine("dispose called");
    }
}
```

using syntatic A try/finally ; .

```
{
    var foo = new Foo();
    try
    {
        // do foo stuff
    }
    finally
    {
        if (foo != null)
            ((IDisposable)foo).Dispose();
    }
}
```

IDisposable : https://riptutorial.com/ko/csharp/topic/1795/idisposable-

31: IEnumerable

`IEnumerable` ArrayList . `IEnumerator<T>` List <> .

`IEnumerable GetEnumerator` . `GetEnumerator` foreach `IEnumerator` .

IEnumerable .

Examples

IEnumerable

IEnumerable . c # `foreach` .

`sequenceOfNumbers` IEnumerable . . `foreach` .

```
int AddNumbers(IEnumerable<int> sequenceOfNumbers) {
    int returnValue = 0;
    foreach(int i in sequenceOfNumbers) {
        returnValue += i;
    }
    return returnValue;
}
```

IEnumerable

IEnumerable BCL . .

.

-
- DFS BFS

```
public static void Main(string[] args) {

    foreach (var coffee in new CoffeeCollection()) {
        Console.WriteLine(coffee);
    }
}

public class CoffeeCollection : IEnumerable {
    private CoffeeEnumerator enumerator;

    public CoffeeCollection() {
        enumerator = new CoffeeEnumerator();
    }

    public IEnumerator GetEnumerator() {
        return enumerator;
    }

    public class CoffeeEnumerator : IEnumerator {
```

```
        string[] beverages = new string[3] { "espresso", "macchiato", "latte" };
        int currentIndex = -1;

        public object Current {
            get {
                return beverages[currentIndex];
            }
        }

        public bool MoveNext() {
            currentIndex++;

            if (currentIndex < beverages.Length) {
                return true;
            }

            return false;
        }

        public void Reset() {
            currentIndex = 0;
        }
    }
}
```

IEnumerable : https://riptutorial.com/ko/csharp/topic/2220/ienumerable

32: ILGenerator

Examples

UnixTimestamp DynamicAssembly .

ILGenerator . C # DynamicAssembly .

```
public static class UnixTimeHelper
{
    private readonly static DateTime EpochTime = new DateTime(1970, 1, 1);

    public static int UnixTimestamp(DateTime input)
    {
        int totalSeconds;
        try
        {
            totalSeconds =
checked((int)input.Subtract(UnixTimeHelper.EpochTime).TotalSeconds);
        }
        catch (OverflowException overflowException)
        {
            throw new InvalidOperationException("It's to late for an Int32 timestamp.",
overflowException);
        }
        return totalSeconds;
    }
}
```

```
//Get the required methods
var dateTimeCtor = typeof (DateTime)
    .GetConstructor(new[] {typeof (int), typeof (int), typeof (int)});
var dateTimeSubstract = typeof (DateTime)
    .GetMethod(nameof(DateTime.Subtract), new[] {typeof (DateTime)});
var timeSpanSecondsGetter = typeof (TimeSpan)
    .GetProperty(nameof(TimeSpan.TotalSeconds)).GetGetMethod();
var invalidOperationCtor = typeof (InvalidOperationException)
    .GetConstructor(new[] {typeof (string), typeof (Exception)});

if (dateTimeCtor == null || dateTimeSubstract == null ||
    timeSpanSecondsGetter == null || invalidOperationCtor == null)
{
    throw new Exception("Could not find a required Method, can not create Assembly.");
}

//Setup the required members
var an = new AssemblyName("UnixTimeAsm");
var dynAsm = AppDomain.CurrentDomain.DefineDynamicAssembly(an,
AssemblyBuilderAccess.RunAndSave);
var dynMod = dynAsm.DefineDynamicModule(an.Name, an.Name + ".dll");

var dynType = dynMod.DefineType("UnixTimeHelper",
    TypeAttributes.Abstract | TypeAttributes.Sealed | TypeAttributes.Public);

var epochTimeField = dynType.DefineField("EpochStartTime", typeof (DateTime),
```

```
    FieldAttributes.Private | FieldAttributes.Static | FieldAttributes.InitOnly);

var cctor =
    dynType.DefineConstructor(
        MethodAttributes.Private | MethodAttributes.HideBySig | MethodAttributes.SpecialName |
        MethodAttributes.RTSpecialName | MethodAttributes.Static, CallingConventions.Standard,
        Type.EmptyTypes);

var cctorGen = cctor.GetILGenerator();
cctorGen.Emit(OpCodes.Ldc_I4, 1970); //Load the DateTime constructor arguments onto the stack
cctorGen.Emit(OpCodes.Ldc_I4_1);
cctorGen.Emit(OpCodes.Ldc_I4_1);
cctorGen.Emit(OpCodes.Newobj, dateTimeCtor); //Call the constructor
cctorGen.Emit(OpCodes.Stsfld, epochTimeField); //Store the object in the static field
cctorGen.Emit(OpCodes.Ret);

var unixTimestampMethod = dynType.DefineMethod("UnixTimestamp",
    MethodAttributes.Public | MethodAttributes.HideBySig | MethodAttributes.Static,
    CallingConventions.Standard, typeof (int), new[] {typeof (DateTime)});

unixTimestampMethod.DefineParameter(1, ParameterAttributes.None, "input");

var methodGen = unixTimestampMethod.GetILGenerator();
methodGen.DeclareLocal(typeof (TimeSpan));
methodGen.DeclareLocal(typeof (int));
methodGen.DeclareLocal(typeof (OverflowException));

methodGen.BeginExceptionBlock(); //Begin the try block
methodGen.Emit(OpCodes.Ldarga_S, (byte) 0); //To call a method on a struct we need to load the
address of it
methodGen.Emit(OpCodes.Ldsfld, epochTimeField);
    //Load the object of the static field we created as argument for the following call
methodGen.Emit(OpCodes.Call, dateTimeSubstract); //Call the substract method on the input
DateTime
methodGen.Emit(OpCodes.Stloc_0); //Store the resulting TimeSpan in a local
methodGen.Emit(OpCodes.Ldloca_S, (byte) 0); //Load the locals address to call a method on it
methodGen.Emit(OpCodes.Call, timeSpanSecondsGetter); //Call the TotalSeconds Get method on the
TimeSpan
methodGen.Emit(OpCodes.Conv_Ovf_I4); //Convert the result to Int32; throws an exception on
overflow
methodGen.Emit(OpCodes.Stloc_1); //store the result for returning later
//The leave instruction to jump behind the catch block will be automatically emitted
methodGen.BeginCatchBlock(typeof (OverflowException)); //Begin the catch block
//When we are here, an OverflowException was thrown, that is now on the stack
methodGen.Emit(OpCodes.Stloc_2); //Store the exception in a local.
methodGen.Emit(OpCodes.Ldstr, "It's to late for an Int32 timestamp.");
    //Load our error message onto the stack
methodGen.Emit(OpCodes.Ldloc_2); //Load the exception again
methodGen.Emit(OpCodes.Newobj, invalidOperationCtor);
    //Create an InvalidOperationException with our message and inner Exception
methodGen.Emit(OpCodes.Throw); //Throw the created exception
methodGen.EndExceptionBlock(); //End the catch block
//When we are here, everything is fine
methodGen.Emit(OpCodes.Ldloc_1); //Load the result value
methodGen.Emit(OpCodes.Ret); //Return it

dynType.CreateType();

dynAsm.Save(an.Name + ".dll");
```

ToString .

```
// create an Assembly and new type
var name = new AssemblyName("MethodOverriding");
var dynAsm = AppDomain.CurrentDomain.DefineDynamicAssembly(name,
AssemblyBuilderAccess.RunAndSave);
var dynModule = dynAsm.DefineDynamicModule(name.Name, $"{name.Name}.dll");
var typeBuilder = dynModule.DefineType("MyClass", TypeAttributes.Public |
TypeAttributes.Class);

// define a new method
var toStr = typeBuilder.DefineMethod(
    "ToString", // name
    MethodAttributes.Public | MethodAttributes.Virtual, // modifiers
    typeof(string), // return type
    Type.EmptyTypes); // argument types
var ilGen = toStr.GetILGenerator();
ilGen.Emit(OpCodes.Ldstr, "Hello, world!");
ilGen.Emit(OpCodes.Ret);

// set this method as override of object.ToString
typeBuilder.DefineMethodOverride(toStr, typeof(object).GetMethod("ToString"));
var type = typeBuilder.CreateType();

// now test it:
var instance = Activator.CreateInstance(type);
Console.WriteLine(instance.ToString());
```

ILGenerator : https://riptutorial.com/ko/csharp/topic/667/ilgenerator

33: INotifyPropertyChanged

`INotifyPropertyChanged` . `PropertyChanged` .

XAML `PropertyChanged` INotifyPropertyChanged XAML .

Examples

C # 6 INotifyPropertyChanged

`INotifyPropertyChange` . (), `CallerMemberName` .

```
class C : INotifyPropertyChanged
{
    // backing field
    int offset;
    // property
    public int Offset
    {
        get
        {
            return offset;
        }
        set
        {
            if (offset == value)
                return;
            offset = value;
            RaisePropertyChanged();
        }
    }

    // helper method for raising PropertyChanged event
    void RaisePropertyChanged([CallerMemberName] string propertyName = null) =>
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

    // interface implemetation
    public event PropertyChangedEventHandler PropertyChanged;
}
```

`INotifyPropertyChanged` .

```
class NotifyPropertyChangedImpl : INotifyPropertyChanged
{
    protected void RaisePropertyChanged([CallerMemberName] string propertyName = null) =>
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

    // interface implemetation
    public event PropertyChangedEventHandler PropertyChanged;
}

class C : NotifyPropertyChangedImpl
{
    int offset;
```

```
    public int Offset
    {
        get { return offset; }
        set { if (offset != value) { offset = value; RaisePropertyChanged(); } }
    }
}
```

Set INotifyPropertyChanged

`NotifyPropertyChangedBase` Set .

```
public class NotifyPropertyChangedBase : INotifyPropertyChanged
{
    protected void RaisePropertyChanged([CallerMemberName] string propertyName = null) =>
    PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

    public event PropertyChangedEventHandler PropertyChanged;

    public virtual bool Set<T>(ref T field, T value, [CallerMemberName] string propertyName =
null)
    {
        if (Equals(field, value))
            return false;
        storage = value;
        RaisePropertyChanged(propertyName);
        return true;
    }
}
```

Set NotifyPropertyChangedBase .

```
public class SomeViewModel : NotifyPropertyChangedBase
{
    private string _foo;
    private int _bar;

    public string Foo
    {
        get { return _foo; }
        set { Set(ref _foo, value); }
    }

    public int Bar
    {
        get { return _bar; }
        set { Set(ref _bar, value); }
    }
}
```

`Set(ref _fieldName, value);` `Set(ref _fieldName, value);` PropertyChanged .

PropertyChanged .

```
public class SomeListener
{
    public SomeListener()
```

```
    {
        _vm = new SomeViewModel();
        _vm.PropertyChanged += OnViewModelPropertyChanged;
    }

    private void OnViewModelPropertyChanged(object sender, PropertyChangedEventArgs e)
    {
        Console.WriteLine($"Property {e.PropertyName} was changed.");
    }

    private readonly SomeViewModel _vm;

}
```

INotifyPropertyChanged : https://riptutorial.com/ko/csharp/topic/2990/inotifypropertychanged-

34: IQueryable

Examples

LINQ SQL

`IQueryable` `IQueryable<T>` LINQ (' ') (:) . C # LINQ .

```
var query = from book in books
            where book.Author == "Stephen King"
            select book;
```

`books` `IQueryable<Book>` (`IQueryable.Provider`) . .

.

- `Book` `Author` .
- 'equals'(`==`).
- `"Stephen King"` .

C # SQL .

```
select *
from Books
where Author = 'Stephen King'
```

`query` (`IQueryable` `IEnumerable`).

C # `IQueryable` .

IQueryable : https://riptutorial.com/ko/csharp/topic/3094/iqueryable-

35: json.net

JSON.net JsonConverter .

Examples

JsonConverter

JsonCoverter Movies Timespan API deserialize

JSON (http://www.omdbapi.com/?i=tt1663662)


```
{
    Title: "Pacific Rim",
    Year: "2013",
    Rated: "PG-13",
    Released: "12 Jul 2013",
    Runtime: "131 min",
    Genre: "Action, Adventure, Sci-Fi",
    Director: "Guillermo del Toro",
    Writer: "Travis Beacham (screenplay), Guillermo del Toro (screenplay), Travis Beacham
(story)",
    Actors: "Charlie Hunnam, Diego Klattenhoff, Idris Elba, Rinko Kikuchi",
    Plot: "As a war between humankind and monstrous sea creatures wages on, a former pilot and
a trainee are paired up to drive a seemingly obsolete special weapon in a desperate effort to
save the world from the apocalypse.",
    Language: "English, Japanese, Cantonese, Mandarin",
    Country: "USA",
    Awards: "Nominated for 1 BAFTA Film Award. Another 6 wins & 46 nominations.",
    Poster: "https://images-na.ssl-images-
amazon.com/images/M/MV5BMTY3MTI5NjQ4Nl5BMl5BanBnXkFtZTcwOTU1OTU0OQ@@._V1_SX300.jpg",
    Ratings: [{
            Source: "Internet Movie Database",
            Value: "7.0/10"
        },
        {
            Source: "Rotten Tomatoes",
            Value: "71%"
        },
        {
            Source: "Metacritic",
            Value: "64/100"
        }
    ],
    Metascore: "64",
    imdbRating: "7.0",
    imdbVotes: "398,198",
    imdbID: "tt1663662",
    Type: "movie",
    DVD: "15 Oct 2013",
    BoxOffice: "$101,785,482.00",
```

```
    Production: "Warner Bros. Pictures",
    Website: "http://pacificrimmovie.com",
    Response: "True"
}
```

```
using Project.Serializers;
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.Threading.Tasks;

namespace Project.Models
{
    [DataContract]
    public class Movie
    {
        public Movie() { }

        [DataMember]
        public int Id { get; set; }

        [DataMember]
        public string ImdbId { get; set; }

        [DataMember]
        public string Title { get; set; }

        [DataMember]
        public DateTime Released { get; set; }

        [DataMember]
        [JsonConverter(typeof(RuntimeSerializer))]
        public TimeSpan Runtime { get; set; }

    }
}
```

RuntimeSerializer

```
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text.RegularExpressions;
using System.Threading.Tasks;

namespace Project.Serializers
{
    public class RuntimeSerializer : JsonConverter
    {
        public override bool CanConvert(Type objectType)
        {
```

```
            return objectType == typeof(TimeSpan);
        }

        public override object ReadJson(JsonReader reader, Type objectType, object
existingValue, JsonSerializer serializer)
        {
            if (reader.TokenType == JsonToken.Null)
                return null;

            JToken jt = JToken.Load(reader);
            String value = jt.Value<String>();

            Regex rx = new Regex("(\\s*)min$");
            value = rx.Replace(value, (m) => "");

            int timespanMin;
            if(!Int32.TryParse(value, out timespanMin))
            {
                throw new NotSupportedException();
            }

            return new TimeSpan(0, timespanMin, 0);
        }

        public override void WriteJson(JsonWriter writer, object value, JsonSerializer
serializer)
        {
            serializer.Serialize(writer, value);
        }
    }
}
```

```
Movie m = JsonConvert.DeserializeObject<Movie>(apiResponse));
```

JSON

```
using Newtonsoft.Json.Linq;
using System.Collections.Generic;

public class JsonFieldsCollector
{
    private readonly Dictionary<string, JValue> fields;

    public JsonFieldsCollector(JToken token)
    {
        fields = new Dictionary<string, JValue>();
        CollectFields(token);
    }

    private void CollectFields(JToken jToken)
    {
        switch (jToken.Type)
        {
            case JTokenType.Object:
                foreach (var child in jToken.Children<JProperty>())
                    CollectFields(child);
                break;
```

```
            case JTokenType.Array:
                foreach (var child in jToken.Children())
                    CollectFields(child);
                break;
            case JTokenType.Property:
                CollectFields(((JProperty) jToken).Value);
                break;
            default:
                fields.Add(jToken.Path, (JValue)jToken);
                break;
        }
    }

    public IEnumerable<KeyValuePair<string, JValue>> GetAllFields() => fields;
}
```

:

```
var json = JToken.Parse(/* JSON string */);
var fieldsCollector = new JsonFieldsCollector(json);
var fields = fieldsCollector.GetAllFields();

foreach (var field in fields)
    Console.WriteLine($"{field.Key}: '{field.Value}'");
```

JSON

```
{
  "User": "John",
  "Workdays": {
    "Monday": true,
    "Tuesday": true,
    "Friday": false
  },
  "Age": 42
}
```

.

```
User: 'John'
Workdays.Monday: 'True'
Workdays.Tuesday: 'True'
Workdays.Friday: 'False'
Age: '42'
```

json.net : https://riptutorial.com/ko/csharp/topic/9879/json-net-

36: LINQ to XML

Examples

LINQ to XML XML

```
<?xml version="1.0" encoding="utf-8" ?>
<Employees>
 <Employee>
  <EmpId>1</EmpId>
  <Name>Sam</Name>
  <Sex>Male</Sex>
  <Phone Type="Home">423-555-0124</Phone>
  <Phone Type="Work">424-555-0545</Phone>
  <Address>
   <Street>7A Cox Street</Street>
   <City>Acampo</City>
   <State>CA</State>
   <Zip>95220</Zip>
   <Country>USA</Country>
  </Address>
</Employee>
<Employee>
 <EmpId>2</EmpId>
 <Name>Lucy</Name>
 <Sex>Female</Sex>
 <Phone Type="Home">143-555-0763</Phone>
 <Phone Type="Work">434-555-0567</Phone>
 <Address>
  <Street>Jess Bay</Street>
  <City>Alta</City>
  <State>CA</State>
  <Zip>95701</Zip>
  <Country>USA</Country>
  </Address>
 </Employee>
</Employees>
```

LINQ XML

```
XDocument xdocument = XDocument.Load("Employees.xml");
IEnumerable<XElement> employees = xdocument.Root.Elements();
foreach (var employee in employees)
{
    Console.WriteLine(employee);
}
```

```
XElement xelement = XElement.Load("Employees.xml");
IEnumerable<XElement> employees = xelement.Root.Elements();
Console.WriteLine("List of all Employee Names :");
foreach (var employee in employees)
{
    Console.WriteLine(employee.Element("Name").Value);
}
```

```
XElement xelement = XElement.Load("Employees.xml");
IEnumerable<XElement> employees = xelement.Root.Elements();
Console.WriteLine("List of all Employee Names along with their ID:");
foreach (var employee in employees)
{
    Console.WriteLine("{0} has Employee ID {1}",
    employee.Element("Name").Value,
    employee.Element("EmpId").Value);
}
```

```
XElement xelement = XElement.Load("Employees.xml");
var name = from nm in xelement.Root.Elements("Employee")
       where (string)nm.Element("Sex") == "Female"
       select nm;
Console.WriteLine("Details of Female Employees:");
foreach (XElement xEle in name)
Console.WriteLine(xEle);
```

```
XElement xelement = XElement.Load("..\\..\\Employees.xml");
var homePhone = from phoneno in xelement.Root.Elements("Employee")
            where (string)phoneno.Element("Phone").Attribute("Type") == "Home"
            select phoneno;
Console.WriteLine("List HomePhone Nos.");
foreach (XElement xEle in homePhone)
{
    Console.WriteLine(xEle.Element("Phone").Value);
}
```

LINQ to XML : https://riptutorial.com/ko/csharp/topic/2773/linq-to-xml

37: Linq

LINQ to Objects IEnumerable LINQ .

Examples

LINQ to Object

LINQ . . (: for , ToList, Count, Max, Average, First).

. .

:

```
var query = from n in numbers
            where n % 2 != 0
            select n;
```

query . .

foreach .

```
foreach(var n in query) {
    Console.WriteLine($"Number selected {n}");
}
```

LINQ , Count , First , Max , Average . . ToList ToArray List Array .

LINQ . . .

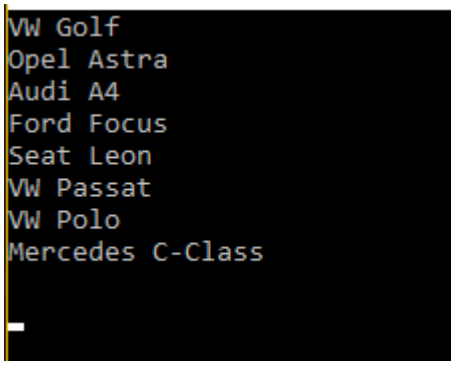
C # LINQ

Linq SELECT

```
static void Main(string[] args)
{
    string[] cars = { "VW Golf",
                      "Opel Astra",
                      "Audi A4",
                      "Ford Focus",
                      "Seat Leon",
                      "VW Passat",
                      "VW Polo",
                      "Mercedes C-Class" };

    var list = from car in cars
               select car;

    StringBuilder sb = new StringBuilder();

    foreach (string entry in list)
```

```
    {
        sb.Append(entry + "\n");
    }

    Console.WriteLine(sb.ToString());
    Console.ReadLine();
}
```

() LINQ . LINQ from () () . . .



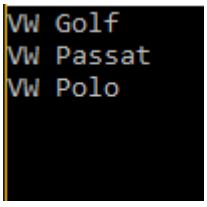
.

WHERE SELECT

```
var list = from car in cars
           where car.Contains("VW")
           select car;
```

WHERE () WHERE .

.




```
var list = from car in cars
           orderby car ascending
           select car;
```

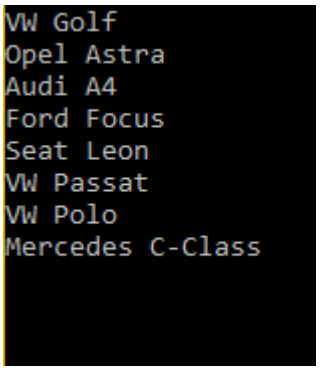
. orderby .

.

.

```
public class Car
{
    public String Name { get; private set; }
    public int UnitsSold { get; private set; }

    public Car(string name, int unitsSold)
    {
        Name = name;
        UnitsSold = unitsSold;
    }
}

class Program
{
    static void Main(string[] args)
    {

        var car1 = new Car("VW Golf", 270952);
        var car2 = new Car("Opel Astra", 56079);
        var car3 = new Car("Audi A4", 52493);
        var car4 = new Car("Ford Focus", 51677);
        var car5 = new Car("Seat Leon", 42125);
        var car6 = new Car("VW Passat", 97586);
        var car7 = new Car("VW Polo", 69867);
        var car8 = new Car("Mercedes C-Class", 67549);

        var cars = new List<Car> {
            car1, car2, car3, car4, car5, car6, car7, car8 };
        var list = from car in cars
                   select car.Name;

        foreach (var entry in list)
        {
            Console.WriteLine(entry);
        }
        Console.ReadLine();
    }
}
```

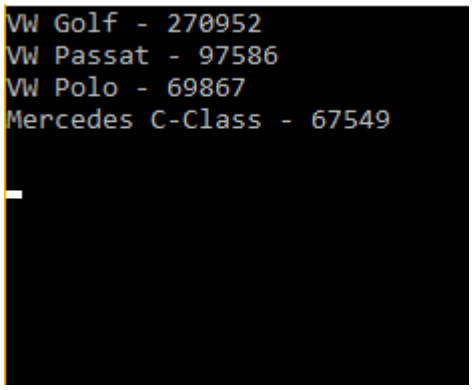
.



. LINQ to Object .

60000 .

```
var list = from car in cars
           where car.UnitsSold > 60000
           orderby car.UnitsSold descending
           select car;

StringBuilder sb = new StringBuilder();

foreach (var entry in list)
{
    sb.AppendLine($"{entry.Name} - {entry.UnitsSold}");
}
Console.WriteLine(sb.ToString());
```

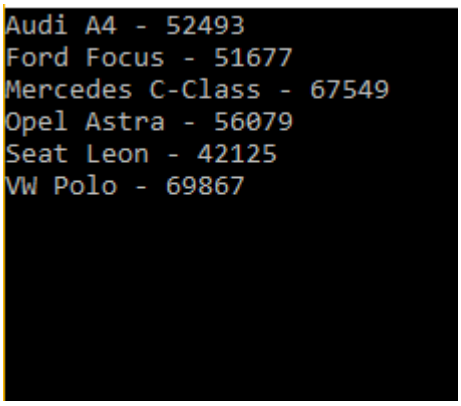


.

.

```
var list = from car in cars
           where car.UnitsSold % 2 != 0
           orderby car.Name ascending
           select car;
```

.



Linq : https://riptutorial.com/ko/csharp/topic/9405/linq-

38: LINQ

LINQ tegrated **Q** uery **IN L** anguage . . XML , SQL , ADO.NET , .NET LINQ .

- :
  - <collection> <range variable>
  - [<collection> <range variable>, ...]
  - <, , , , ...> < >
  - <select groupBy > < >
- :
  - Enumerable.Aggregate (func)
  - Enumerable.Aggregate (seed, func)
  - Enumerable.Aggregate (seed, func, resultSelector)
  - Enumerable.All ()
  - Enumerable.Any ()
  - Enumerable.Any ()
  - Enumerable.AsEnumerable ()
  - . ()
  - . ()
  - . <> ()
  - Enumerable.Concat ()
  - Enumerable.Contains (value)
  - Enumerable.Contains (value, comparer)
  - Enumerable.Count ()
  - Enumerable.Count ()
  - Enumerable.DefaultIfEmpty ()
  - Enumerable.DefaultIfEmpty (defaultValue)
  - Enumerable.Distinct ()
  - Enumerable.Distinct ( )
  - Enumerable.ElementAt (index)
  - Enumerable.ElementAtOrDefault (index)
  - Enumerable.Empty ()
  - Enumerable.Except ()
  - Enumerable.Except (,)
  - Enumerable.First ()
  - Enumerable.First ()
  - Enumerable.FirstOrDefault ()
  - Enumerable.FirstOrDefault ( )
  - Enumerable.GroupBy (keySelector)
  - Enumerable.GroupBy (keySelector, resultSelector)
  - Enumerable.GroupBy (keySelector, elementSelector)
  - Enumerable.GroupBy (keySelector, comparer)
  - Enumerable.GroupBy (keySelector, resultSelector, comparer)

- Enumerable.GroupBy (keySelector, elementSelector, resultSelector)
- Enumerable.GroupBy (keySelector, elementSelector, comparer)
- Enumerable.GroupBy (keySelector, elementSelector, resultSelector, comparer)
- Enumerable.Intersect ()
- Enumerable.Intersect (second, comparer)
- Enumerable.Join (inner, outerKeySelector, innerKeySelector, resultSelector)
- Enumerable.Join (inner, outerKeySelector, innerKeySelector, resultSelector, comparer)
- . ()
- Enumerable.Last ()
- Enumerable.LastOrDefault ()
- Enumerable.LastOrDefault ( )
- Enumerable.LongCount ()
- Enumerable.LongCount ()
- Enumerable.Max ()
- Enumerable.Max ()
- Enumerable.Min ()
- Enumerable.Min ()
- Enumerable.OfType <TResult> ()
- Enumerable.OrderBy (keySelector)
- Enumerable.OrderBy (keySelector, comparer)
- Enumerable.OrderByDescending (keySelector)
- Enumerable.OrderByDescending (keySelector, comparer)
- Enumerable.Range (,)
- . (,)
- Enumerable.Reverse ()
- Enumerable.Select (selector)
- Enumerable.SelectMany (selector)
- Enumerable.SelectMany (collectionSelector, resultSelector)
- Enumerable.SequenceEqual ()
- Enumerable.SequenceEqual (second, comparer)
- Enumerable.Single ()
- Enumerable.Single ()
- Enumerable.SingleOrDefault ()
- Enumerable.SingleOrDefault ( )
- Enumerable.Skip (count)
- Enumerable.SkipWhile ( )
- Enumerable.Sum ()
- Enumerable.Sum ()
- Enumerable.Take ()
- Enumerable.TakeWhile ()
- orderedEnumerable.ThenBy (keySelector)
- orderedEnumerable.ThenBy (keySelector, comparer)
- orderedEnumerable.ThenByDescending (keySelector)
- orderedEnumerable.ThenByDescending (keySelector, comparer)
- .Array ()
- Enumerable.ToDictionary (keySelector)

- Enumerable.ToDictionary (keySelector, elementSelector)
- Enumerable.ToDictionary (keySelector, comparer)
- Enumerable.ToDictionary (keySelector, elementSelector, comparer)
- Enumerable.ToList ()
- Enumerable.ToLookup (keySelector)
- Enumerable.ToLookup (keySelector, elementSelector)
- Enumerable.ToLookup (keySelector, comparer)
- Enumerable.ToLookup (keySelector, elementSelector, comparer)
- Enumerable.Union ()
- Enumerable.Union (second, comparer)
- . ()
- Enumerable.Zip (second, resultSelector)

LINQ `System.Linq` .

Method Syntax Query Syntax . Query .

.

Examples

.

```
List<string> trees = new List<string>{ "Oak", "Birch", "Beech", "Elm", "Hazel", "Maple" };
```

```
// Select all trees with name of length 3
var shortTrees = trees.Where(tree => tree.Length == 3); // Oak, Elm
```

```
var shortTrees = from tree in trees
                 where tree.Length == 3
                 select tree; // Oak, Elm
```

-

Select IEnumerable .

:

```
List<String> trees = new List<String>{ "Oak", "Birch", "Beech", "Elm", "Hazel", "Maple" };
```

()

```
//The below select stament transforms each element in tree into its first character.
IEnumerable<String> initials = trees.Select(tree => tree.Substring(0, 1));
foreach (String initial in initials) {
    System.Console.WriteLine(initial);
}
```

:

H

.NET Fiddle

LINQ

```
initials = from tree in trees
           select tree.Substring(0, 1);
```

LINQ `IEnumerable<TSource>` `IEnumerable<TResult>`. `TSource` `TResult` .

.

```
public static IEnumerable<TResult> Select<TSource, TResult>(
    this IEnumerable<TSource> source,
    Func<TSource, TResult> selector
)

public static IEnumerable<TSource> Where<TSource>(
    this IEnumerable<TSource> source,
    Func<TSource, int, bool> predicate
)

public static IOrderedEnumerable<TSource> OrderBy<TSource, TKey>(
    this IEnumerable<TSource> source,
    Func<TSource, TKey> keySelector
)
```

LINQ **MSDN** . LINQ () ().

wiki . . .

`Select` , `Where` `OrderBy` , .

```
var someNumbers = { 4, 3, 2, 1 };

var processed = someNumbers
        .Select(n => n * 2)    // Multiply each number by 2
        .Where(n => n != 6)    // Keep all the results, except for 6
        .OrderBy(n => n);      // Sort in ascending order
```

:

2
4
8

.NET Fiddle

IEnumerable<T> . , .

Enumerable Range Repeat .

Enumerable.Range() .

```
// Generate a collection containing the numbers 1-100 ([1, 2, 3, ..., 98, 99, 100])
var range = Enumerable.Range(1,100);
```

.NET Fiddle

Enumerable.Repeat() .

```
// Generate a collection containing "a", three times (["a","a","a"])
var repeatedValues = Enumerable.Repeat("a", 3);
```

.NET Fiddle

Skip . . .

Take . . .

```
var values = new [] { 5, 4, 3, 2, 1 };

var skipTwo        = values.Skip(2);         // { 3, 2, 1 }
var takeThree      = values.Take(3);         // { 5, 4, 3 }
var skipOneTakeTwo = values.Skip(1).Take(2); // { 4, 3 }
var takeZero       = values.Take(0);         // An IEnumerable<int> with 0 items
```

.NET Fiddle

Skip and Take .

```
IEnumerable<T> GetPage<T>(IEnumerable<T> collection, int pageNumber, int resultsPerPage) {
    int startIndex = (pageNumber - 1) * resultsPerPage;
    return collection.Skip(startIndex).Take(resultsPerPage);
}
```

: LINQ to Entities . "Skip ' LINQ to Entities 'Skip ' 'OrderBy ' ."
NotSupportedException .

, FirstOrDefault, Last, LastOrDefault, Single SingleOrDefault

.

predicate predicate .

()

- predicate .
- " " InvalidOperationException InvalidOperationException .
- predicate " " InvalidOperationException InvalidOperationException .

```
// Returns "a":
new[] { "a" }.First();

// Returns "a":
new[] { "a", "b" }.First();

// Returns "b":
new[] { "a", "b" }.First(x => x.Equals("b"));

// Returns "ba":
new[] { "ba", "be" }.First(x => x.Contains("b"));

// Throws InvalidOperationException:
new[] { "ca", "ce" }.First(x => x.Contains("b"));

// Throws InvalidOperationException:
new string[0].First();
```

.NET Fiddle

FirstOrDefault ()

- predicate .
- V predicate default(T) default(T) .

```
// Returns "a":
new[] { "a" }.FirstOrDefault();

// Returns "a":
new[] { "a", "b" }.FirstOrDefault();

// Returns "b":
new[] { "a", "b" }.FirstOrDefault(x => x.Equals("b"));

// Returns "ba":
new[] { "ba", "be" }.FirstOrDefault(x => x.Contains("b"));

// Returns null:
new[] { "ca", "ce" }.FirstOrDefault(x => x.Contains("b"));

// Returns null:
new string[0].FirstOrDefault();
```

.NET Fiddle

()

- predicate .
- " ." InvalidOperationException .
- predicate " " InvalidOperationException InvalidOperationException .

```
// Returns "a":
new[] { "a" }.Last();

// Returns "b":
new[] { "a", "b" }.Last();

// Returns "a":
new[] { "a", "b" }.Last(x => x.Equals("a"));

// Returns "be":
new[] { "ba", "be" }.Last(x => x.Contains("b"));

// Throws InvalidOperationException:
new[] { "ca", "ce" }.Last(x => x.Contains("b"));

// Throws InvalidOperationException:
new string[0].Last();
```

LastOrDefault ()

- predicate .
- V predicate default(T) default(T) .

```
// Returns "a":
new[] { "a" }.LastOrDefault();

// Returns "b":
new[] { "a", "b" }.LastOrDefault();

// Returns "a":
new[] { "a", "b" }.LastOrDefault(x => x.Equals("a"));

 // Returns "be":
new[] { "ba", "be" }.LastOrDefault(x => x.Contains("b"));

// Returns null:
new[] { "ca", "ce" }.LastOrDefault(x => x.Contains("b"));

// Returns null:
new string[0].LastOrDefault();
```

()

- V predicate , .
- predicate

" " InvalidOperationException InvalidOperationException .

- predicate " " InvalidOperationException InvalidOperationException .
- : 1 , 2 .

```
// Returns "a":
new[] { "a" }.Single();

// Throws InvalidOperationException because sequence contains more than one element:
new[] { "a", "b" }.Single();

// Returns "b":
new[] { "a", "b" }.Single(x => x.Equals("b"));

// Throws InvalidOperationException:
new[] { "a", "b" }.Single(x => x.Equals("c"));

// Throws InvalidOperationException:
new string[0].Single();

// Throws InvalidOperationException because sequence contains more than one element:
new[] { "a", "a" }.Single();
```

SingleOrDefault ()

- V predicate , .
- V predicate ,default(T) .
- predicate " " InvalidOperationException InvalidOperationException .
- V predicate default(T) default(T) .
- : 1 , 2 .

```
// Returns "a":
new[] { "a" }.SingleOrDefault();

// returns "a"
new[] { "a", "b" }.SingleOrDefault(x => x == "a");

// Returns null:
new[] { "a", "b" }.SingleOrDefault(x => x == "c");

// Throws InvalidOperationException:
new[] { "a", "a" }.SingleOrDefault(x => x == "a");

// Throws InvalidOperationException:
new[] { "a", "b" }.SingleOrDefault();

// Returns null:
new string[0].SingleOrDefault();
```

- FirstOrDefault ,LastOrDefault SingleOrDefault Any Count . default(T) / / default(T) .
- . , Single . First . Single throw. "* OrDefault"- .

- : (Single) 0 (SingleOrDefault) , . First .

Except . IEqualityComparer . IEqualityComparer .

:

```
int[] first = { 1, 2, 3, 4 };
int[] second = { 0, 2, 3, 5 };

IEnumerable<int> inFirstButNotInSecond = first.Except(second);
// inFirstButNotInSecond = { 1, 4 }
```

:

1
4

.NET Fiddle

.Except(second) second , 2 3 (0, 5 first).

Except Distinct (,). :

```
int[] third = { 1, 1, 1, 2, 3, 4 };

IEnumerable<int> inThirdButNotInSecond = third.Except(second);
// inThirdButNotInSecond = { 1, 4 }
```

:

1
4

.NET Fiddle

1 4 .

IEquatable IEqualityComparer . GetHashCode IEquatable object .

IEquatable :

```
class Holiday : IEquatable<Holiday>
{
    public string Name { get; set; }

    public bool Equals(Holiday other)
    {
        return Name == other.Name;
    }

    // GetHashCode must return true whenever Equals returns true.
    public override int GetHashCode()
```

```
    {
        //Get hash code for the Name field if it is not null.
        return Name?.GetHashCode() ?? 0;
    }
}

public class Program
{
    public static void Main()
    {
        List<Holiday> holidayDifference = new List<Holiday>();

        List<Holiday> remoteHolidays = new List<Holiday>
        {
            new Holiday { Name = "Xmas" },
            new Holiday { Name = "Hanukkah" },
            new Holiday { Name = "Ramadan" }
        };

        List<Holiday> localHolidays = new List<Holiday>
        {
            new Holiday { Name = "Xmas" },
            new Holiday { Name = "Ramadan" }
        };

        holidayDifference = remoteHolidays
            .Except(localHolidays)
            .ToList();

        holidayDifference.ForEach(x => Console.WriteLine(x.Name));
    }
}
```

:

.NET Fiddle

SelectMany :

```
var sequenceOfSequences = new [] { new [] { 1, 2, 3 }, new [] { 4, 5 }, new [] { 6 } };
var sequence = sequenceOfSequences.SelectMany(x => x);
// returns { 1, 2, 3, 4, 5, 6 }
```

`SelectMany()` .

LINQ :

```
var sequence = from subSequence in sequenceOfSequences
               from item in subSequence
               select item;
```

`SelectMany` .

.

```
public class BlogPost
{
    public int Id { get; set; }
    public string Content { get; set; }
    public List<Comment> Comments { get; set; }
}

public class Comment
{
    public int Id { get; set; }
    public string Content { get; set; }
}
```

.

```
List<BlogPost> posts = new List<BlogPost>()
{
    new BlogPost()
    {
        Id = 1,
        Comments = new List<Comment>()
        {
            new Comment()
            {
                Id = 1,
                Content = "It's really great!",
            },
            new Comment()
            {
                Id = 2,
                Content = "Cool post!"
            }
        }
    },
    new BlogPost()
    {
        Id = 2,
        Comments = new List<Comment>()
        {
            new Comment()
            {
                Id = 3,
                Content = "I don't think you're right",
            },
            new Comment()
            {
                Id = 4,
                Content = "This post is a complete nonsense"
            }
        }
    }
};
```

BlogPost Id Content . SelectMany .

```
var commentsWithIds = posts.SelectMany(p => p.Comments, (post, comment) => new { PostId =
post.Id, CommentContent = comment.Content });
```

commentsWithIds .

```
{
    PostId = 1,
    CommentContent = "It's really great!"
},
{
    PostId = 1,
    CommentContent = "Cool post!"
},
{
    PostId = 2,
    CommentContent = "I don't think you're right"
},
{
    PostId = 2,
    CommentContent = "This post is a complete nonsense"
}
```

SelectMany

SelectMany linq IEnumerable<IEnumerable<T>> IEnumerable<T> ". IEnumerable IEnumerable T IEnumerable .

```
var words = new [] { "a,b,c", "d,e", "f" };
var splitAndCombine = words.SelectMany(x => x.Split(','));
// returns { "a", "b", "c", "d", "e", "f" }
```

.

Select() .

```
class School
{
    public Student[] Students { get; set; }
}

class Student
{
    public string Name { get; set; }
}

var schools = new [] {
    new School(){ Students = new [] { new Student { Name="Bob"}, new Student { Name="Jack"}
}},
    new School(){ Students = new [] { new Student { Name="Jim"}, new Student { Name="John"} }}
};

var allStudents = schools.SelectMany(s=> s.Students);

foreach(var student in allStudents)
{
    Console.WriteLine(student.Name);
}
```

:

.NET Fiddle

All .
: *.Any*

1.

All : .

2.

All : true false .

```
var numbers = new List<int>(){ 1, 2, 3, 4, 5};
bool result = numbers.All(i => i < 10); // true
bool result = numbers.All(i => i >= 3); // false
```

3.

: true true .

```
var numbers = new List<int>();
bool result = numbers.All(i => i >= 0); // true
```

: All ., . .

/ Cast

```
interface IFoo { }
class Foo : IFoo { }
class Bar : IFoo { }
```

```
var item0 = new Foo();
var item1 = new Foo();
var item2 = new Bar();
var item3 = new Bar();
var collection = new IFoo[] { item0, item1, item2, item3 };
```

OfType

```
var foos = collection.OfType<Foo>(); // result: IEnumerable<Foo> with item0 and item1
var bars = collection.OfType<Bar>(); // result: IEnumerable<Bar> item item2 and item3
var foosAndBars = collection.OfType<IFoo>(); // result: IEnumerable<IFoo> with all four items
```

Where Where

```
var foos = collection.Where(item => item is Foo); // result: IEnumerable<IFoo> with item0 and
item1
var bars = collection.Where(item => item is Bar); // result: IEnumerable<IFoo> with item2 and
item3
```

Cast

```
var bars = collection.Cast<Bar>();               // throws InvalidCastException on the 1st
item
var foos = collection.Cast<Foo>();               // throws InvalidCastException on the 3rd
item
var foosAndBars = collection.Cast<IFoo>();       // OK
```

.

```
int[] numbers1 = { 1, 2, 3 };
int[] numbers2 = { 2, 3, 4, 5 };

var allElement = numbers1.Union(numbers2);   // AllElement now contains 1,2,3,4,5
```

.NET Fiddle

.

SQL LINQ .

,,, .

.

```
var first = new List<string>(){ "a","b","c"}; // Left data
var second = new List<string>(){ "a", "c", "d"}; // Right data
```

()

```
var result = from f in first
             join s in second on f equals s
             select new { f, s };

var result = first.Join(second,
                        f => f,
                        s => s,
                        (f, s) => new { f, s });

// Result: {"a","a"}
//         {"c","c"}
```

```
var leftOuterJoin = from f in first
                    join s in second on f equals s into temp
                    from t in temp.DefaultIfEmpty()
```

```
                    select new { First = f, Second = t};

// Or can also do:
var leftOuterJoin = from f in first
                    from s in second.Where(x => x == f).DefaultIfEmpty()
                    select new { First = f, Second = s};

// Result: {"a","a"}
//         {"b", null}
//         {"c","c"}


// Left outer join method syntax
var leftOuterJoinFluentSyntax = first.GroupJoin(second,
                                      f => f,
                                      s => s,
                                      (f, s) => new { First = f, Second = s })
                                    .SelectMany(temp => temp.Second.DefaultIfEmpty(),
                                      (f, s) => new { First = f.First, Second = s });
```

```
var rightOuterJoin = from s in second
                     join f in first on s equals f into temp
                     from t in temp.DefaultIfEmpty()
                     select new {First=t,Second=s};

// Result: {"a","a"}
//         {"c","c"}
//         {null,"d"}
```

```
var CrossJoin = from f in first
                from s in second
                select new { f, s };

// Result: {"a","a"}
//         {"a","c"}
//         {"a","d"}
//         {"b","a"}
//         {"b","c"}
//         {"b","d"}
//         {"c","a"}
//         {"c","c"}
//         {"c","d"}
```

```
var fullOuterjoin = leftOuterJoin.Union(rightOuterJoin);

// Result: {"a","a"}
//         {"b", null}
//         {"c","c"}
//         {null,"d"}
```

LINQ .

`Region` . (`first` `second` `ID`).

: .

```
public class Region
{
    public Int32 ID;
    public string RegionDescription;

    public Region(Int32 pRegionID, string pRegionDescription=null)
    {
        ID = pRegionID; RegionDescription = pRegionDescription;
    }
}
```

(,).

```
// Left data
var first = new List<Region>()
                { new Region(1), new Region(3), new Region(4) };
// Right data
var second = new List<Region>()
                {
                   new Region(1, "Eastern"),  new Region(2, "Western"),
                   new Region(3, "Northern"), new Region(4, "Southern")
                };
```

first second . .

```
// do the inner join
var result = from f in first
             join s in second on f.ID equals s.ID
             select new { f.ID, s.RegionDescription };


 // Result: {1,"Eastern"}
 //         {3, Northern}
 //         {4,"Southern"}
```

. select new { f.ID, s.RegionDescription }; select new { f.ID, s.RegionDescription }; select new Region(f.ID, s.RegionDescription); select new Region(f.ID, s.RegionDescription); Region .

.NET

IEnumerable . .

```
int[] array = { 1, 2, 3, 4, 2, 5, 3, 1, 2 };

var distinct = array.Distinct();
// distinct = { 1, 2, 3, 4, 5 }
```

IEquatable<T> GetHashCode Equals . .

```
class SSNEqualityComparer : IEqualityComparer<Person> {
    public bool Equals(Person a, Person b) => return a.SSN == b.SSN;
    public int GetHashCode(Person p) => p.SSN;
}

List<Person> people;
```

```
distinct = people.Distinct(SSNEqualityComparer);
```

GroupBy

:

```
public class Film {
    public string Title { get; set; }
    public string Category { get; set; }
    public int Year { get; set; }
}
```

:

```
foreach (var grp in films.GroupBy(f => f.Category)) {
    var groupCategory = grp.Key;
    var numberOfFilmsInCategory = grp.Count();
}
```

:

```
foreach (var grp in films.GroupBy(f => new { Category = f.Category, Year = f.Year })) {
    var groupCategory = grp.Key.Category;
    var groupYear = grp.Key.Year;
    var numberOfFilmsInCategory = grp.Count();
}
```

Range Linq

Linq Enumerable Linq .

:

```
var asciiCharacters = new List<char>();
for (var x = 0; x < 256; x++)
{
    asciiCharacters.Add((char)x);
}
```

:

```
var asciiCharacters = Enumerable.Range(0, 256).Select(a => (char) a);
```

100 .

```
var evenNumbers = Enumerable.Range(1, 100).Where(a => a % 2 == 0);
```

- OrderBy () ThenBy () OrderByDescending () ThenByDescending ()

```
string[] names= { "mark", "steve", "adam" };
```

:

```
var sortedNames =
    from name in names
    orderby name
    select name;
```

```
var sortedNames = names.OrderBy(name => name);
```

sortedNames "adam", "mark", "steve" .

:

```
var sortedNames =
    from name in names
    orderby name descending
    select name;
```

```
var sortedNames = names.OrderByDescending(name => name);
```

sortedNames "steve", "mark", "adam" .

```
Person[] people =
{
    new Person { FirstName = "Steve", LastName = "Collins", Age = 30},
    new Person { FirstName = "Phil" , LastName = "Collins", Age = 28},
    new Person { FirstName = "Adam" , LastName = "Ackerman", Age = 29},
    new Person { FirstName = "Adam" , LastName = "Ackerman", Age = 15}
};
```

```
var sortedPeople = from person in people
                   orderby person.LastName, person.FirstName, person.Age descending
                   select person;
```

```
 sortedPeople = people.OrderBy(person => person.LastName)
                      .ThenBy(person => person.FirstName)
                      .ThenByDescending(person => person.Age);
```

```
1. Adam Ackerman 29
2. Adam Ackerman 15
3. Phil Collins  28
4. Steve Collins 30
```

LINQ ( ) .

.

```
var classroom = new Classroom
```

```
{
    new Student { Name = "Alice", Grade = 97, HasSnack = true  },
    new Student { Name = "Bob",   Grade = 82, HasSnack = false },
    new Student { Name = "Jimmy", Grade = 71, HasSnack = true  },
    new Student { Name = "Greg",  Grade = 90, HasSnack = false },
    new Student { Name = "Joe",   Grade = 59, HasSnack = false }
}
```

LINQ "" . .

```
var studentsWithSnacks = from s in classroom.Students
                         where s.HasSnack
                         select s;
```

90 `Student` .

```
var topStudentNames = from s in classroom.Students
                      where s.Grade >= 90
                      select s.Name;
```

LINQ . . . .

```
var topStudentNames = classroom.Students
                              .Where(s => s.Grade >= 90)
                              .Select(s => s.Name);
```

GroupBy

GroupBy `IEnumerable<T>` .

.

```
List<int> iList = new List<int>() { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
var grouped = iList.GroupBy(x => x % 2 == 0);

//Groups iList into odd [13579] and even[2468] items

foreach(var group in grouped)
{
    foreach (int item in group)
    {
        Console.Write(item); // 135792468  (first odd then even)
    }
}
```

. Person .

```
public class Person
{
    public int Age {get; set;}
    public string Name {get; set;}
}
```

.

```
List<Person> people = new List<Person>();
people.Add(new Person{Age = 20, Name = "Mouse"});
people.Add(new Person{Age = 30, Name = "Neo"});
people.Add(new Person{Age = 40, Name = "Morpheus"});
people.Add(new Person{Age = 30, Name = "Trinity"});
people.Add(new Person{Age = 40, Name = "Dozer"});
people.Add(new Person{Age = 40, Name = "Smith"});
```

LINQ .

```
var query = people.GroupBy(x => x.Age);
```

.

```
foreach(var result in query)
{
    Console.WriteLine(result.Key);

    foreach(var person in result)
        Console.WriteLine(person.Name);
}
```

.

```
20
Mouse
30
Neo
Trinity
40
Morpheus
Dozer
Smith
```

.NET Fiddle .

Any .
: .All , Any FirstOrDefault :

1.

Any : `true` `false` .

```
var numbers = new List<int>();
bool result = numbers.Any(); // false

var numbers = new List<int>(){ 1, 2, 3, 4, 5};
bool result = numbers.Any(); //true
```

2.

Any : `true` `true` .

```
var arrayOfStrings = new string[] { "a", "b", "c" };
arrayOfStrings.Any(item => item == "a");    // true
arrayOfStrings.Any(item => item == "d");    // false
```

3.

Any : `false` .

```
var numbers = new List<int>();
bool result = numbers.Any(i => i >= 0); // false
```

: `Any` . , . .

.NET Fiddle

ToDictionary

`ToDictionary()` LINQ `IEnumerable<T>` `Dictionary<TKey, TElement>` .

```
IEnumerable<User> users = GetUsers();
Dictionary<int, User> usersById = users.ToDictionary(x => x.Id);
```

`ToDictionary` `Func<TSource, TKey>` .

.

```
Dictionary<int, User> usersById = new Dictionary<int User>();
foreach (User u in users)
{
  usersById.Add(u.Id, u);
}
```

`Func<TSource, TElement>` `ToDictionary` `Value` .

```
IEnumerable<User> users = GetUsers();
Dictionary<int, string> userNamesById = users.ToDictionary(x => x.Id, x => x.Name);
```

`IComparer` . / .

```
IEnumerable<User> users = GetUsers();
Dictionary<string, User> usersByCaseInsenstiveName = users.ToDictionary(x => x.Name,
StringComparer.InvariantCultureIgnoreCase);

var user1 = usersByCaseInsenstiveName["john"];
var user2 = usersByCaseInsenstiveName["JOHN"];
```

```
user1 == user2; // Returns true
```

: ToDictionary . .ArgumentException: An item with the same key has already been added.
ToLookup .

Aggregate .

```
int[] intList = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
int sum = intList.Aggregate((prevSum, current) => prevSum + current);
// sum = 55
```

- prevSum = 1
- prevSum = prevSum(at the first step) + 2
- i prevSum = prevSum(at the (i-1) step) + i-th element of the array

```
string[] stringList = { "Hello", "World", "!" };
string joinedString = stringList.Aggregate((prev, current) => prev + " " + current);
// joinedString = "Hello World !"
```

Aggregate seed . .

```
List<int> items = new List<int> { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
```

items

1. .Count
2.
3. .

Aggregate .

```
var result = items.Aggregate(new { Total = 0, Even = 0, FourthItems = new List<int>() },
                (accumelative,item) =>
                new {
                    Total = accumelative.Total + 1,
                    Even = accumelative.Even + (item % 2 == 0 ? 1 : 0),
                    FourthItems = (accumelative.Total + 1)%4 == 0 ?
                        new List<int>(accumelative.FourthItems) { item } :
                        accumelative.FourthItems
                });
// Result:
// Total = 12
// Even = 6
// FourthItems = [4, 8, 12]
```

. *new* *(seed)*.

Linq (let)

linq **let** . . .

```
 int[] numbers = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

 var aboveAverages = from number in numbers
                     let average = numbers.Average()
                     let nSquared = Math.Pow(number,2)
                     where nSquared > average
                     select number;

 Console.WriteLine("The average of the numbers is {0}.", numbers.Average());

 foreach (int n in aboveAverages)
 {
   Console.WriteLine("Query result includes number {0} with square of {1}.", n,
Math.Pow(n,2));
 }
```

:

4.5.
9 3 .
16 4 .
25 5 .
6 6 .
49 7 .
64 8 .
81 9 .

SkipWhile

`SkipWhile()` `SkipWhile()` ( ).

```
int[] list = { 42, 42, 6, 6, 6, 42 };
var result = list.SkipWhile(i => i == 42);
// Result: 6, 6, 6, 42
```

DefaultIfEmpty

DefaultIfEmpty . . :

```
var chars = new List<string>() { "a", "b", "c", "d" };

chars.DefaultIfEmpty("N/A").FirstOrDefault(); // returns "a";

chars.Where(str => str.Length > 1)
     .DefaultIfEmpty("N/A").FirstOrDefault(); // return "N/A"

chars.Where(str => str.Length > 1)
        .DefaultIfEmpty().First(); // returns null;
```

:

`DefaultIfEmpty` Linq Join . SQL . :

```
var leftSequence = new List<int>() { 99, 100, 5, 20, 102, 105 };
var rightSequence = new List<char>() { 'a', 'b', 'c', 'i', 'd' };

var numbersAsChars = from l in leftSequence
                     join r in rightSequence
                     on l equals (int)r into leftJoin
                     from result in leftJoin.DefaultIfEmpty('?')
                     select new
                     {
                         Number = l,
                         Character = result
                     };

foreach(var item in numbersAsChars)
{
    Console.WriteLine("Num = {0} ** Char = {1}", item.Number, item.Character);
}

ouput:

Num = 99         Char = c
Num = 100        Char = d
Num = 5          Char = ?
Num = 20         Char = ?
Num = 102        Char = ?
Num = 105        Char = i
```

`DefaultIfEmpty` ( ) `null` . `NullReferenceException` .:

```
var leftSequence = new List<int> { 1, 2, 5 };
var rightSequence = new List<dynamic>()
    {
        new { Value = 1 },
        new { Value = 2 },
        new { Value = 3 },
        new { Value = 4 },
    };

var numbersAsChars = (from l in leftSequence
                      join r in rightSequence
                      on l equals r.Value into leftJoin
                      from result in leftJoin.DefaultIfEmpty()
                      select new
                      {
                          Left = l,
                          // 5 will not have a matching object in the right so result
                          // will be equal to null.
                          // To avoid an error use:
                          //    -  C# 6.0 or above - ?.
                          //    -  Under           - result == null ? 0 : result.Value
                          Right = result?.Value
                      }).ToList();
```

SequenceEqual

`SequenceEqual` `IEnumerable<T>` .

```
int[] a = new int[] {1, 2, 3};
```

```
int[] b = new int[] {1, 2, 3};
int[] c = new int[] {1, 3, 2};

bool returnsTrue = a.SequenceEqual(b);
bool returnsFalse = a.SequenceEqual(c);
```

Count IEnumerable<T> Count . Count .

```
int[] array = { 1, 2, 3, 4, 2, 5, 3, 1, 2 };

int n = array.Count(); // returns the number of elements in the array
int x = array.Count(i => i > 2); // returns the number of elements in the array greater than 2
```

LongCount Count long int.MaxValue IEnumerable<T> .

```
int[] array = GetLargeArray();

long n = array.LongCount(); // returns the number of elements in the array
long x = array.LongCount(i => i > 100); // returns the number of elements in the array greater
than 100
```

LINQ . .

```
IEnumerable<VehicleModel> BuildQuery(int vehicleType, SearchModel search, int start = 1, int
count = -1) {
    IEnumerable<VehicleModel> query = _entities.Vehicles
        .Where(x => x.Active && x.Type == vehicleType)
        .Select(x => new VehicleModel {
            Id = v.Id,
            Year = v.Year,
            Class = v.Class,
            Make = v.Make,
            Model = v.Model,
            Cylinders = v.Cylinders ?? 0
        });
```

.

```
    if (!search.Years.Contains("all", StringComparer.OrdinalIgnoreCase))
        query = query.Where(v => search.Years.Contains(v.Year));

    if (!search.Makes.Contains("all", StringComparer.OrdinalIgnoreCase)) {
        query = query.Where(v => search.Makes.Contains(v.Make));
    }

    if (!search.Models.Contains("all", StringComparer.OrdinalIgnoreCase)) {
        query = query.Where(v => search.Models.Contains(v.Model));
    }

    if (!search.Cylinders.Equals("all", StringComparer.OrdinalIgnoreCase)) {
        decimal minCylinders = 0;
        decimal maxCylinders = 0;
        switch (search.Cylinders) {
            case "2-4":
                maxCylinders = 4;
                break;
```

```
            case "5-6":
                minCylinders = 5;
                maxCylinders = 6;
                break;
            case "8":
                minCylinders = 8;
                maxCylinders = 8;
                break;
            case "10+":
                minCylinders = 10;
                break;
        }
        if (minCylinders > 0) {
            query = query.Where(v => v.Cylinders >= minCylinders);
        }
        if (maxCylinders > 0) {
            query = query.Where(v => v.Cylinders <= maxCylinders);
        }
    }
```

.

```
    switch (search.SortingColumn.ToLower()) {
        case "make_model":
            query = query.OrderBy(v => v.Make).ThenBy(v => v.Model);
            break;
        case "year":
            query = query.OrderBy(v => v.Year);
            break;
        case "engine_size":
            query = query.OrderBy(v => v.EngineSize).ThenBy(v => v.Cylinders);
            break;
        default:
            query = query.OrderBy(v => v.Year); //The default sorting.
    }
```

:

```
    query = query.Skip(start - 1);
```

.

```
    if (count > -1) {
        query = query.Take(count);
    }
    return query;
}
```

`foreach` `ToList` `ToArray` LINQ .

```
SearchModel sm;

// populate the search model here
// ...
```

```
List<VehicleModel> list = BuildQuery(5, sm).ToList();
```

Zip . .Func Zip C # . .

"C # in a Nutshell" ,

```
int[] numbers = { 3, 5, 7 };
string[] words = { "three", "five", "seven", "ignored" };
IEnumerable<string> zip = numbers.Zip(words, (n, w) => n + "=" + w);
```

:

3 = 3
5 = 5
7 = 7

GroupJoin

```
Customer[] customers = Customers.ToArray();
Purchase[] purchases = Purchases.ToArray();

var groupJoinQuery =
    from c in customers
    join p in purchases on c.ID equals p.CustomerID
    into custPurchases
    select new
    {
        CustName = c.Name,
        custPurchases
    };
```

ElementAt ElementAtOrDefault

ElementAt n .n ArgumentOutOfRangeException .

```
int[] numbers  = { 1, 2, 3, 4, 5 };
numbers.ElementAt(2);  // 3
numbers.ElementAt(10); // throws ArgumentOutOfRangeException
```

ElementAtOrDefault n .n default(T) .

```
int[] numbers  = { 1, 2, 3, 4, 5 };
numbers.ElementAtOrDefault(2);  // 3
numbers.ElementAtOrDefault(10); // 0 = default(int)
```

ElementAt ElementAtOrDefault IList<T> .

ElementAt IList<T> ArgumentOutOfRangeException throw ( ).

Linq

. LINQ to Objects . LINQ 3 .

`All` - . :

```
int[] array = { 10, 20, 30 };

// Are all elements >= 10? YES
array.All(element => element >= 10);

// Are all elements >= 20? NO
array.All(element => element >= 20);

// Are all elements < 40? YES
array.All(element => element < 40);
```

`Any` - . :

```
int[] query=new int[] { 2, 3, 4 }
query.Any (n => n == 3);
```

`Contains` - . :

```
//for int array
int[] query =new int[] { 1,2,3 };
query.Contains(1);

//for string array
string[] query={"Tom","grey"};
query.Contains("Tom");

//for a string
var stringValue="hello";
stringValue.Contains("h");
```

`Customer` , `Purchase` `PurchaseItem` .

```
public class Customer
{
   public string Id { get; set } // A unique Id that identifies customer
   public string Name  {get; set; }
}

public class Purchase
{
   public string Id { get; set }
   public string CustomerId {get; set; }
   public string Description { get; set; }
}

public class PurchaseItem
{
   public string Id { get; set }
   public string PurchaseId {get; set; }
   public string Detail { get; set; }
}
```

.

```
var customers = new List<Customer>()
 {
    new Customer() {
        Id = Guid.NewGuid().ToString(),
        Name = "Customer1"
    },

    new Customer() {
        Id = Guid.NewGuid().ToString(),
        Name = "Customer2"
    }
 };

 var purchases = new List<Purchase>()
 {
     new Purchase() {
         Id = Guid.NewGuid().ToString(),
         CustomerId = customers[0].Id,
         Description = "Customer1-Purchase1"
     },

     new Purchase() {
         Id = Guid.NewGuid().ToString(),
         CustomerId = customers[0].Id,
         Description = "Customer1-Purchase2"
     },

     new Purchase() {
         Id = Guid.NewGuid().ToString(),
         CustomerId = customers[1].Id,
         Description = "Customer2-Purchase1"
     },

     new Purchase() {
         Id = Guid.NewGuid().ToString(),
         CustomerId = customers[1].Id,
         Description = "Customer2-Purchase2"
     }
  };

 var purchaseItems = new List<PurchaseItem>()
 {
     new PurchaseItem() {
         Id = Guid.NewGuid().ToString(),
         PurchaseId= purchases[0].Id,
         Detail = "Purchase1-PurchaseItem1"
     },

     new PurchaseItem() {
         Id = Guid.NewGuid().ToString(),
         PurchaseId= purchases[1].Id,
         Detail = "Purchase2-PurchaseItem1"
     },

     new PurchaseItem() {
         Id = Guid.NewGuid().ToString(),
         PurchaseId= purchases[1].Id,
         Detail = "Purchase2-PurchaseItem2"
     },
```

```
     new PurchaseItem() {
         Id = Guid.NewGuid().ToString(),
         PurchaseId= purchases[3].Id,
         Detail = "Purchase3-PurchaseItem1"
     }
 };
```

, linq :

```
var result = from c in customers
             join p in purchases on c.Id equals p.CustomerId           // first join
             join pi in purchaseItems on p.Id equals pi.PurchaseId     // second join
             select new
             {
                c.Name, p.Description, pi.Detail
             };
```

:

```
foreach(var resultItem in result)
{
    Console.WriteLine($"{resultItem.Name}, {resultItem.Description}, {resultItem.Detail}");
}
```

.

1, 1 - 1, 1 - 1

1, 1 - 2, 2 - 1

1, 1 - 2, 2 - 2

2, 2 - 2, 3 - 1

.NET Fiddle

```
  PropertyInfo[] stringProps = typeof (string).GetProperties();//string properties
  PropertyInfo[] builderProps = typeof(StringBuilder).GetProperties();//stringbuilder
properties

    var query =
        from s in stringProps
        join b in builderProps
            on new { s.Name, s.PropertyType } equals new { b.Name, b.PropertyType }
        select new
        {
            s.Name,
            s.PropertyType,
            StringToken = s.MetadataToken,
            StringBuilderToken = b.MetadataToken
        };
```

join . .

Func . selector - .

Select index select . .

" "

```
var rowNumbers = collection.OrderBy(item => item.Property1)
                           .ThenBy(item => item.Property2)
                           .ThenByDescending(item => item.Property3)
                           .Select((item, index) => new { Item = item, RowNumber = index })
                           .ToList();
```

.

```
var rankInGroup = collection.GroupBy(item => item.Property1)
                            .OrderBy(group => group.Key)
                            .SelectMany(group => group.OrderBy(item => item.Property2)
                                                      .ThenByDescending(item => item.Property3)
                                                      .Select((item, index) => new
                                                      {
                                                          Item = item,
                                                          RankInGroup = index
                                                      })).ToList();
```

(Oracle dense_rank) .

```
var rankOfBelongingGroup = collection.GroupBy(item => item.Property1)
                            .OrderBy(group => group.Key)
                            .Select((group, index) => new
                            {
                                Items = group,
                                Rank = index
                            })
                            .SelectMany(v => v.Items, (s, i) => new
                            {
                                Item = i,
                                DenseRank = s.Rank
                            }).ToList();
```

.

```
public class SomeObject
{
    public int Property1 { get; set; }
    public int Property2 { get; set; }
    public int Property3 { get; set; }

    public override string ToString()
    {
        return string.Join(", ", Property1, Property2, Property3);
    }
}
```

:

```
List<SomeObject> collection = new List<SomeObject>
{
    new SomeObject { Property1 = 1, Property2 = 1, Property3 = 1},
    new SomeObject { Property1 = 1, Property2 = 2, Property3 = 1},
    new SomeObject { Property1 = 1, Property2 = 2, Property3 = 2},
    new SomeObject { Property1 = 2, Property2 = 1, Property3 = 1},
    new SomeObject { Property1 = 2, Property2 = 2, Property3 = 1},
    new SomeObject { Property1 = 2, Property2 = 2, Property3 = 1},
    new SomeObject { Property1 = 2, Property2 = 3, Property3 = 1}
};
```

TakeWhile

TakeWhile .

```
int[] list = { 1, 10, 40, 50, 44, 70, 4 };
var result = list.TakeWhile(item => item < 50).ToList();
// result = { 1, 10, 40 }
```

Enumerable.Sum .

.

```
int[] numbers = new int[] { 1, 4, 6 };
Console.WriteLine( numbers.Sum() ); //outputs 11
```

.

```
var totalMonthlySalary = employees.Sum( employee => employee.MonthlySalary );
```

Sum .

- Int32
- Int64
-
-
-

nullable null-coalescing null .

```
int?[] numbers = new int?[] { 1, null, 6 };
Console.WriteLine( numbers.Sum( number => number ?? 0 ) ); //outputs 7
```

ToLookup . . foreach ILookup . . - dotnetperls

```
string[] array = { "one", "two", "three" };
//create lookup using string length as key
var lookup = array.ToLookup(item => item.Length);

//join the values whose lengths are 3
Console.WriteLine(string.Join(",",lookup[3]));
//output: one,two
```

:

```
int[] array = { 1,2,3,4,5,6,7,8 };
//generate lookup for odd even numbers (keys will be 0 and 1)
var lookup = array.ToLookup(item => item % 2);

//print even numbers after joining
Console.WriteLine(string.Join(",",lookup[0]));
//output: 2,4,6,8

//print odd numbers after joining
Console.WriteLine(string.Join(",",lookup[1]));
//output: 1,3,5,7
```

IEnumerable Linq .

Linq . `IEnumerable<T>` .

```
public namespace MyNamespace
{
    public static class LinqExtensions
    {
        public static IEnumerable<List<T>> Batch<T>(this IEnumerable<T> source, int batchSize)
        {
            var batch = new List<T>();
            foreach (T item in source)
            {
                batch.Add(item);
                if (batch.Count == batchSize)
                {
                    yield return batch;
                    batch = new List<T>();
                }
            }
            if (batch.Count > 0)
                yield return batch;
        }
    }
}
```

`IEnumerable<T>` .`this` (`source`) . `yield` `IEnumerable<T>` (yield).

.

```
//using MyNamespace;
var items = new List<int> { 2, 3, 4, 5, 6 };
foreach (List<int> sublist in items.Batch(3))
{
    // do something
}
```

`{2, 3, 4}` `{5, 6}` .

LinQ LinQ . :

```
//using MyNamespace;
var result = Enumerable.Range(0, 13)          // generate a list
                       .Where(x => x%2 == 0) // filter the list or do something other
                       .Batch(3)             // call our extension method
                       .ToList()             // call other standard methods
```

`{0, 2, 4}, {6, 8, 10}, {12}` .

`using MyNamespace;` `using MyNamespace;`

SelectMany

2

```
var list1 = new List<string> { "a", "b", "c" };
var list2 = new List<string> { "1", "2", "3", "4" };
```

.

```
var result = new List<string>();
foreach (var s1 in list1)
    foreach (var s2 in list2)
        result.Add($"{s1}{s2}");
```

SelectMany .

```
var result = list1.SelectMany(x => list2.Select(y => $"{x}{y}", x, y)).ToList();
```

Any and First (OrDefault) -

`Any` `FirstOrDefault` `FirstOrDefault` . Any First, FirstOrDefault, Last, LastOrDefault, Single SingleOrDefault .

.

```
if (myEnumerable.Any(t=>t.Foo == "Bob"))
{
    var myFoo = myEnumerable.First(t=>t.Foo == "Bob");
    //Do stuff
}
```

.

```
var myFoo = myEnumerable.FirstOrDefault(t=>t.Foo == "Bob");
if (myFoo != null)
{
    //Do stuff
}
```

. `Single` .

GroupBy

.

```
public class Transaction
{
    public string Category { get; set; }
    public DateTime Date { get; set; }
    public decimal Amount { get; set; }
}
```

.

```
var transactions = new List<Transaction>
{
   new Transaction { Category = "Saving Account", Amount = 56, Date =
DateTime.Today.AddDays(1) },
   new Transaction { Category = "Saving Account", Amount = 10, Date = DateTime.Today.AddDays(-
10) },
   new Transaction { Category = "Credit Card", Amount = 15, Date = DateTime.Today.AddDays(1)
},
   new Transaction { Category = "Credit Card", Amount = 56, Date = DateTime.Today },
   new Transaction { Category = "Current Account", Amount = 100, Date =
DateTime.Today.AddDays(5) },
};
```

GroupBy .

```
var summaryApproach1 = transactions.GroupBy(t => t.Category)
                           .Select(t => new
                           {
                               Category = t.Key,
                               Count = t.Count(),
                               Amount = t.Sum(ta => ta.Amount),
                           }).ToList();

Console.WriteLine("-- Summary: Approach 1 --");
summaryApproach1.ForEach(
            row => Console.WriteLine($"Category: {row.Category}, Amount: {row.Amount}, Count:
{row.Count}"));
```

.

```
var summaryApproach2 = transactions.GroupBy(t => t.Category, (key, t) =>
{
        var transactionArray = t as Transaction[] ?? t.ToArray();
        return new
        {
            Category = key,
            Count = transactionArray.Length,
            Amount = transactionArray.Sum(ta => ta.Amount),
        };
}).ToList();

Console.WriteLine("-- Summary: Approach 2 --");
summaryApproach2.ForEach(
```

```
row => Console.WriteLine($"Category: {row.Category}, Amount: {row.Amount}, Count:
{row.Count}"));
```

.

: , : 66, : 2

: , : 71, : 2

: , : 100, : 1

.NET Fiddle

- .
- `ArgumentNullException: source is null.`

:

```
// Create an array.
int[] array = { 1, 2, 3, 4 };                          //Output:
// Call reverse extension method on the array.         //4
var reverse = array.Reverse();                         //3
// Write contents of array to screen.                  //2
foreach (int value in reverse)                         //1
    Console.WriteLine(value);
```

Remeber `Reverse()` LINQ .

```
        //Create List of chars
        List<int> integerlist = new List<int>() { 1, 2, 3, 4, 5, 6 };

        //Reversing the list then taking the two first elements
        IEnumerable<int> reverseFirst = integerlist.Reverse<int>().Take(2);

        //Taking 2 elements and then reversing only thos two
        IEnumerable<int> reverseLast = integerlist.Take(2).Reverse();

        //reverseFirst output: 6, 5
        //reverseLast output:  2, 1
```

Reverse () . OrderBy .

LINQ-to-Objects (Reverse, OrderBy, GroupBy) (Where, Take, Skip) .

:

```
public static IEnumerable<T> Reverse<T>(this IList<T> list) {
    for (int i = list.Count - 1; i >= 0; i--)
        yield return list[i];
}
```

.

IEnumerable <T> LINQ . .

Collection <T> , Array , List <T> , Dictionary <TKey, TValue> HashSet <T> .

IEnumerable <T> IEnumerator <T> . " " .

" " . . .OrderBy() IEnumerable <T> . (: .ToList()). ( ) .

. ( ) ( ) .

IEnumerable <T> . . , IEnumerable <T> .

. Where IEnumerable , .

```
void Main()
{
    Fibonacci Fibo = new Fibonacci();
    IEnumerable<long> quadrillionplus = Fibo.Where(i => i > 1000000000000);
    Console.WriteLine("Enumerable built");
    Console.WriteLine(quadrillionplus.Take(2).Sum());
    Console.WriteLine(quadrillionplus.Skip(2).First());

    IEnumerable<long> fibMod612 = Fibo.OrderBy(i => i % 612);
    Console.WriteLine("Enumerable built");
    Console.WriteLine(fibMod612.First());//smallest divisible by 612
}

public class Fibonacci : IEnumerable<long>
{
    private int max = 90;

    //Enumerator called typically from foreach
    public IEnumerator GetEnumerator() {
        long n0 = 1;
        long n1 = 1;
        Console.WriteLine("Enumerating the Enumerable");
        for(int i = 0; i < max; i++){
            yield return n0+n1;
            n1 += n0;
            n0 = n1-n0;
        }
    }

    //Enumerable called typically from linq
    IEnumerator<long> IEnumerable<long>.GetEnumerator() {
        long n0 = 1;
        long n1 = 1;
        Console.WriteLine("Enumerating the Enumerable");
        for(int i = 0; i < max; i++){
            yield return n0+n1;
            n1 += n0;
            n0 = n1-n0;
        }
    }
}
```

```
Enumerable built
Enumerating the Enumerable
```

```
4052739537881
Enumerating the Enumerable
4052739537881
Enumerable built
Enumerating the Enumerable
14930352
```

(fibMod612) , .First `.First()` O (n) 1 . 1 . `.Take(5)` `.First()` 5 , 5 . 5 , .

.

, **double** **float** . , 0 (1).

char chars *ascii* (2).

OrderBy CultureInfo (a, b, c ...) .

. (OrderByDescending).

1 :

```
int[] numbers = {2, 1, 0, -1, -2};
IEnumerable<int> ascending = numbers.OrderBy(x => x);
// returns {-2, -1, 0, 1, 2}
```

2 :

```
 char[] letters = {' ', '!', '?', '[', '{', '+', '1', '9', 'a', 'A', 'b', 'B', 'y', 'Y', 'z',
'Z'};
 IEnumerable<char> ascending = letters.OrderBy(x => x);
 // returns { ' ', '!', '+', '1', '9', '?', 'A', 'B', 'Y', 'Z', '[', 'a', 'b', 'y', 'z', '{' }
```

:

```
class Person
{
   public string Name { get; set; }
   public int Age { get; set; }
}

var people = new[]
{
    new Person {Name = "Alice", Age = 25},
    new Person {Name = "Bob", Age = 21},
    new Person {Name = "Carol", Age = 43}
};
var youngestPerson = people.OrderBy(x => x.Age).First();
var name = youngestPerson.Name; // Bob
```

OrderByDescending

.

, **double float** . , 0 (1).

char chars *ascii* (2).

OrderBy CultureInfo (z, y, x, ...) .

. (OrderBy).

1 :

```
int[] numbers = {-2, -1, 0, 1, 2};
IEnumerable<int> descending = numbers.OrderByDescending(x => x);
// returns {2, 1, 0, -1, -2}
```

2 :

```
char[] letters = {' ', '!', '?', '[', '{', '+', '1', '9', 'a', 'A', 'b', 'B', 'y', 'Y', 'z',
'Z'};
IEnumerable<char> descending = letters.OrderByDescending(x => x);
// returns { '{', 'z', 'y', 'b', 'a', '[', 'Z', 'Y', 'B', 'A', '?', '9', '1', '+', '!', ' ' }
```

3 :

```
class Person
{
   public  string Name { get; set; }
   public  int Age { get; set; }
}

var people = new[]
{
    new Person {Name = "Alice", Age = 25},
    new Person {Name = "Bob", Age = 21},
    new Person {Name = "Carol", Age = 43}
};
var oldestPerson = people.OrderByDescending(x => x.Age).First();
var name = oldestPerson.Name; // Carol
```

( ).

```
List<int> foo = new List<int> { 1, 2, 3 };
List<int> bar = new List<int> { 3, 4, 5 };

// Through Enumerable static class
var result = Enumerable.Concat(foo, bar).ToList(); // 1,2,3,3,4,5

// Through extension method
var result = foo.Concat(bar).ToList(); // 1,2,3,3,4,5
```

MSDN :

`IEqualityComparer<T>` .

```
List<int> numbers = new List<int> { 1, 2, 3, 4, 5 };
var result1 = numbers.Contains(4); // true
var result2 = numbers.Contains(8); // false

List<int> secondNumberCollection = new List<int> { 4, 5, 6, 7 };
// Note that can use the Intersect method in this case
var result3 = secondNumberCollection.Where(item => numbers.Contains(item)); // will be true
only for 4,5
```

:

```
public class Person
{
   public string Name { get; set; }
}

List<Person> objects = new List<Person>
{
    new Person { Name = "Nikki"},
    new Person { Name = "Gilad"},
    new Person { Name = "Phil"},
    new Person { Name = "John"}
};

//Using the Person's Equals method - override Equals() and GetHashCode() - otherwise it
//will compare by reference and result will be false
var result4 = objects.Contains(new Person { Name = "Phil" }); // true
```

`Enumerable.Contains(value, comparer)` :

```
public class Compare : IEqualityComparer<Person>
{
    public bool Equals(Person x, Person y)
    {
        return x.Name == y.Name;
    }
    public int GetHashCode(Person codeh)
    {
        return codeh.Name.GetHashCode();
    }
}

var result5 = objects.Contains(new Person { Name = "Phil" }, new Compare()); // true
```

Contains if Contains .

:

```
if(status == 1 || status == 3 || status == 4)
{
    //Do some business operation
}
else
{
    //Do something else
}
```

:

```
if(new int[] {1, 3, 4 }.Contains(status)
{
    //Do some business operaion
}
else
{
    //Do something else
}
```

LINQ : https://riptutorial.com/ko/csharp/topic/68/linq-

39: Microsoft.Exchange.WebServices

Examples

`ExchangeManager .  OofSettings  GetOofSettings .`

```
using System;
using System.Web.Configuration;
using Microsoft.Exchange.WebServices.Data;

namespace SetOutOfOffice
{
    class ExchangeManager
    {
        private ExchangeService Service;

        public ExchangeManager()
        {
            var password =
WebConfigurationManager.ConnectionStrings["Password"].ConnectionString;
            Connect("exchangeadmin", password);
        }
        private void Connect(string username, string password)
        {
            var service = new ExchangeService(ExchangeVersion.Exchange2010_SP2);
            service.Credentials = new WebCredentials(username, password);
            service.AutodiscoverUrl("autodiscoveremail@domain.com" ,
RedirectionUrlValidationCallback);

            Service = service;
        }
        private static bool RedirectionUrlValidationCallback(string redirectionUrl)
        {
            return
redirectionUrl.Equals("https://mail.domain.com/autodiscover/autodiscover.xml");
        }
        public OofSettings GetOofSettings(string email)
        {
            return Service.GetUserOofSettings(email);
        }
    }
}
```

.

```
var em = new ExchangeManager();
var oofSettings = em.GetOofSettings("testemail@domain.com");
```

Exchange `UpdateUserOof` .

```
using System;
using System.Web.Configuration;
using Microsoft.Exchange.WebServices.Data;
```

```
class ExchangeManager
{
    private ExchangeService Service;

    public ExchangeManager()
    {
        var password = WebConfigurationManager.ConnectionStrings["Password"].ConnectionString;
        Connect("exchangeadmin", password);
    }
    private void Connect(string username, string password)
    {
        var service = new ExchangeService(ExchangeVersion.Exchange2010_SP2);
        service.Credentials = new WebCredentials(username, password);
        service.AutodiscoverUrl("autodiscoveremail@domain.com" ,
RedirectionUrlValidationCallback);

        Service = service;
    }
    private static bool RedirectionUrlValidationCallback(string redirectionUrl)
    {
        return redirectionUrl.Equals("https://mail.domain.com/autodiscover/autodiscover.xml");
    }
    /// <summary>
    /// Updates the given user's Oof settings with the given details
    /// </summary>
    public void UpdateUserOof(int oofstate, DateTime starttime, DateTime endtime, int
externalaudience, string internalmsg, string externalmsg, string emailaddress)
    {
        var newSettings = new OofSettings
        {
            State = (OofState)oofstate,
            Duration = new TimeWindow(starttime, endtime),
            ExternalAudience = (OofExternalAudience)externalaudience,
            InternalReply = internalmsg,
            ExternalReply = externalmsg
        };

        Service.SetUserOofSettings(emailaddress, newSettings);
    }
}
```

.

```
var oofState = 1;
var startDate = new DateTime(01,08,2016);
var endDate = new DateTime(15,08,2016);
var externalAudience = 1;
var internalMessage = "I am not in the office!";
var externalMessage = "I am not in the office <strong>and neither are you!</strong>"
var theUser = "theuser@domain.com";

var em = new ExchangeManager();
em.UpdateUserOof(oofstate, startDate, endDate, externalAudience, internalMessage,
externalMessage, theUser);
```

html html .

Microsoft.Exchange.WebServices : https://riptutorial.com/ko/csharp/topic/4863/microsoft-exchange-webservices

40: Null

- `Nullable<int> i = 10;`
- int? j = 11;
- int? k = null;
- ? DateOfBirth = DateTime.Now;
- ? = 1.0m;
- ? IsAvailable = true;
- ? = 'a';
- ()? variableName

Nullable `null` .

`T?` `Nullable<T>` .

Nullable `System.ValueType` boxed unbox . , `null` null `null` , .

Nullable boxing null `null` `null` Nullable .

```
DateTime? dt = null;
var o = (object)dt;
var result = (o == null); // is true

DateTime? dt = new DateTime(2015, 12, 11);
var o = (object)dt;
var dt2 = (DateTime)dt; // correct cause o contains DateTime value
```

.

```
DateTime? dt = new DateTime(2015, 12, 11);
var o = (object)dt;
var type = o.GetType(); // is DateTime, not Nullable<DateTime>
```

:

```
DateTime? dt = new DateTime(2015, 12, 11);
var type = dt.GetType(); // is DateTime, not Nullable<DateTime>
```

Examples

nullable

`null` :

```
Nullable<int> i = null;
```

:

```
int? i = null;
```

:

```
var i = (int?)null;
```

null :

```
Nullable<int> i = 0;
```

:

```
int? i = 0;
```

Nullable .

```
int? i = null;

if (i != null)
{
    Console.WriteLine("i is not null");
}
else
{
    Console.WriteLine("i is null");
}
```

:

```
if (i.HasValue)
{
    Console.WriteLine("i is not null");
}
else
{
    Console.WriteLine("i is null");
}
```

nullable

nullable `int`

```
int? i = 10;
```

, `GetValueOrDefault` nullable int `HasValue` .

```
int j = i ?? 0;
int j = i.GetValueOrDefault(0);
int j = i.HasValue ? i.Value : 0;
```

. i System.InvalidOperationException . Use of unassigned local variable 'i' .

```
int j = i.Value;
```

nullable

.GetValueOrDefault() .HasValue false (throw Value).

```
class Program
{
    static void Main()
    {
        int? nullableExample = null;
        int result = nullableExample.GetValueOrDefault();
        Console.WriteLine(result); // will output the default value for int - 0
        int secondResult = nullableExample.GetValueOrDefault(1);
        Console.WriteLine(secondResult) // will output our specified default - 1
        int thirdResult = nullableExample ?? 1;
        Console.WriteLine(secondResult) // same as the GetValueOrDefault but a bit shorter
    }
}
```

:

```
0
1
```

nullable .

```
public bool IsTypeNullable<T>()
{
    return Nullable.GetUnderlyingType( typeof(T) )!=null;
}
```

nullable null.

```
public class NullableTypesExample
{
    static int? _testValue;

    public static void Main()
    {
        if(_testValue == null)
            Console.WriteLine("null");
        else
            Console.WriteLine(_testValue.ToString());
    }
}
```

:

Nullable

nullable . nullable .

/ Nullable.GetUnderlyingType .

```
public static class TypesHelper {
    public static bool IsNullable(this Type type) {
        Type underlyingType;
        return IsNullable(type, out underlyingType);
    }
    public static bool IsNullable(this Type type, out Type underlyingType) {
        underlyingType = Nullable.GetUnderlyingType(type);
        return underlyingType != null;
    }
    public static Type GetNullable(Type type) {
        Type underlyingType;
        return IsNullable(type, out underlyingType) ? type : NullableTypesCache.Get(type);
    }
    public static bool IsExactOrNullable(this Type type, Func<Type, bool> predicate) {
        Type underlyingType;
        if(IsNullable(type, out underlyingType))
            return IsExactOrNullable(underlyingType, predicate);
        return predicate(type);
    }
    public static bool IsExactOrNullable<T>(this Type type)
        where T : struct {
        return IsExactOrNullable(type, t => Equals(t, typeof(T)));
    }
}
```

:

```
Type type = typeof(int).GetNullable();
Console.WriteLine(type.ToString());

if(type.IsNullable())
    Console.WriteLine("Type is nullable.");
Type underlyingType;
if(type.IsNullable(out underlyingType))
    Console.WriteLine("The underlying type is " + underlyingType.Name + ".");
if(type.IsExactOrNullable<int>())
    Console.WriteLine("Type is either exact or nullable Int32.");
if(!type.IsExactOrNullable(t => t.IsEnum))
    Console.WriteLine("Type is neither exact nor nullable enum.");
```

:

```
System.Nullable`1[System.Int32]
Type is nullable.
The underlying type is Int32.
Type is either exact or nullable Int32.
Type is neither exact nor nullable enum.
```

. NullableTypesCache .

```
static class NullableTypesCache {
    readonly static ConcurrentDictionary<Type, Type> cache = new ConcurrentDictionary<Type,
Type>();
    static NullableTypesCache() {
        cache.TryAdd(typeof(byte), typeof(Nullable<byte>));
        cache.TryAdd(typeof(short), typeof(Nullable<short>));
        cache.TryAdd(typeof(int), typeof(Nullable<int>));
        cache.TryAdd(typeof(long), typeof(Nullable<long>));
        cache.TryAdd(typeof(float), typeof(Nullable<float>));
        cache.TryAdd(typeof(double), typeof(Nullable<double>));
        cache.TryAdd(typeof(decimal), typeof(Nullable<decimal>));
        cache.TryAdd(typeof(sbyte), typeof(Nullable<sbyte>));
        cache.TryAdd(typeof(ushort), typeof(Nullable<ushort>));
        cache.TryAdd(typeof(uint), typeof(Nullable<uint>));
        cache.TryAdd(typeof(ulong), typeof(Nullable<ulong>));
        //...
    }
    readonly static Type NullableBase = typeof(Nullable<>);
    internal static Type Get(Type type) {
        // Try to avoid the expensive MakeGenericType method call
        return cache.GetOrAdd(type, t => NullableBase.MakeGenericType(t));
    }
}
```

Null : https://riptutorial.com/ko/csharp/topic/1240/null--

41: Null-Coalescing

- var result = possibleNullObject ?? ;

possibleNullObject	null . null , . Nullable .
defaultValue	possibleNullObject null possibleNullObject possibleNullObject .

null . ??

.

```
possibleNullObject != null ? possibleNullObject : defaultValue
```

( ) nullable .

The ?? .

Examples

null-coalescing operator (??) null nullable .

```
string testString = null;
Console.WriteLine("The specified string is - " + (testString ?? "not provided"));
```

.NET Fiddle

.

```
string testString = null;
if (testString == null)
{
    Console.WriteLine("The specified string is - not provided");
}
else
{
    Console.WriteLine("The specified string is - " + testString);
}
```

(? :) :

```
string testString = null;
Console.WriteLine("The specified string is - " + (testString == null ? "not provided" :
testString));
```

Null

Nullable . .

null

```
int? a = null;
int b = 3;
var output = a ?? b;
var type = output.GetType();

Console.WriteLine($"Output Type :{type}");
Console.WriteLine($"Output value :{output}");
```

:

: System.Int32
: 3

Nullable

```
int? a = null;
int? b = null;
var output = a ?? b;
```

`output int? int? b , null`

.

```
int? a = null;
int? b = null;
int c = 3;
var output = a ?? b ?? c;

var type = output.GetType();
Console.WriteLine($"Type :{type}");
Console.WriteLine($"value :{output}");
```

:

: System.Int32
: 3

Null

null null .

```
object o = null;
var output = o?.ToString() ?? "Default Value";
```

:

: System.String
:

null

null `null` .

.

```
string name = GetName();

if (name == null)
    name = "Unknown!";
```

:

```
string name = GetName() ?? "Unknown!";
```

.

```
IEnumerable<MyClass> myList = GetMyList();
var item = myList.SingleOrDefault(x => x.Id == 2) ?? new MyClass { Id = 2 };
```

null

```
private List<FooBar> _fooBars;

public List<FooBar> FooBars
{
    get { return _fooBars ?? (_fooBars = new List<FooBar>()); }
}
```

`.FooBars _fooBars null _fooBars .`

(lazy) . thread-safe laziness , .NET Framework `Lazy<T>` .

C # 6

C # 6 .

```
private List<FooBar> _fooBars;

public List<FooBar> FooBars => _fooBars ?? ( _fooBars = new List<FooBar>() );
```

`_fooBars .`

MVVM

MVVM . .

```
private ICommand _actionCommand = null;
public ICommand ActionCommand =>
   _actionCommand ?? ( _actionCommand = new DelegateCommand( DoAction ) );
```

Null-Coalescing : https://riptutorial.com/ko/csharp/topic/37/null-coalescing-

42: NullReferenceException

Examples

NullReferenceException

`NullReferenceException` (, ,) null.

```
Car myFirstCar = new Car();
Car mySecondCar = null;
Color myFirstColor = myFirstCar.Color; // No problem as myFirstCar exists / is not null
Color mySecondColor = mySecondCar.Color; // Throws a NullReferenceException
// as mySecondCar is null and yet we try to access its color.
```

. . " [', ' ('.

```
myGarage.CarCollection[currentIndex.Value].Color = theCarInTheStreet.Color;
```

? :

- `myGarage` null
- `myGarage.CarCollection` null
- `currentIndex` null
- `myGarage.CarCollection[currentIndex.Value]` null
- `theCarInTheStreet` null

null . . .

/ ?

```
myGarage.CarCollection = new Car[10];
```

null ?

```
if (myGarage == null)
{
    Console.WriteLine("Maybe you should buy a garage first!");
}
```

, .

```
if (theCarInTheStreet == null)
{
    throw new ArgumentNullException("theCarInTheStreet");
}
```

NullReferenceException . , .

NullReferenceException : https://riptutorial.com/ko/csharp/topic/2702/nullreferenceexception

43: O (n)

. ( ) . .

Examples

.

```
    public static void Main()
    {
        int[] array = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
        int shiftCount = 1;
        Rotate(ref array, shiftCount);
        Console.WriteLine(string.Join(", ", array));
        // Output: [10, 1, 2, 3, 4, 5, 6, 7, 8, 9]

        array = new []{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
        shiftCount = 15;
        Rotate(ref array, shiftCount);
        Console.WriteLine(string.Join(", ", array));
        // Output: [6, 7, 8, 9, 10, 1, 2, 3, 4, 5]

        array = new[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
        shiftCount = -1;
        Rotate(ref array, shiftCount);
        Console.WriteLine(string.Join(", ", array));
        // Output: [2, 3, 4, 5, 6, 7, 8, 9, 10, 1]

        array = new[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
        shiftCount = -35;
        Rotate(ref array, shiftCount);
        Console.WriteLine(string.Join(", ", array));
        // Output: [6, 7, 8, 9, 10, 1, 2, 3, 4, 5]
    }

    private static void Rotate<T>(ref T[] array, int shiftCount)
    {
        T[] backupArray= new T[array.Length];

        for (int index = 0; index < array.Length; index++)
        {
            backupArray[(index + array.Length + shiftCount % array.Length) % array.Length] =
array[index];
        }

        array = backupArray;
    }
```

.

(index + array.Length + shiftCount % array.Length) % array.Length

:

(shiftCount % array.Length) -> (10 1 11 -1 -11) .

array.Length + (shiftCount % array.Length) -> . . 0 -1 10 -1 -1 9 . (10 + (-1 % 10) = 9)

index + array.Length + (shiftCount % array.Length) -> . (0 + 10 + (-1 % 10) = 9)

index + array.Length + (shiftCount % array.Length) % array.Length -> . . 9 10 1 10 0 . ((9 + 10 + (1 % 10)) % 10 = 0)

O (n) : https://riptutorial.com/ko/csharp/topic/9770/o--n-----

44: ObservableCollection

Examples

ObservableCollection

ObservableCollection T List<T> T .

.

ObservableCollection , .

ObservableCollection INotifyCollectionChanged INotifyPropertyChanged .

UI UI .

```
using System.Collections.ObjectModel
```

string .

```
ObservableCollection<string> collection = new ObservableCollection<string>();
```

```
 ObservableCollection<string> collection = new ObservableCollection<string>()
 {
  "First_String", "Second_String"
 };
```

IList 0 (IList.Item).

ObservableCollection : https://riptutorial.com/ko/csharp/topic/7351/observablecollection--t-

45: String.Format

`Format` `System.String` . .NET Framework `String.Format` , `WriteLine` .

- string.Format (, params [] args)
- string.Format (IFormatProvider , , params [] args)
- $ "string {text} blablabla"// C # 6

	args .
args	. `params` .
	. CultureInfo.InvariantCulture CultureInfo.CurrentCulture .

:

- `String.Format()` throw `null` .
- `args` , .

Examples

String.Format ''

`String.Format` . `string format, params object[] args` (:

```
Console.WriteLine(String.Format("{0} - {1}", name, value));
```

:

```
Console.WriteLine("{0} - {1}", name, value);
```

`String.Format` :

```
Debug.WriteLine(); // and Print()
StringBuilder.AppendFormat();
```

`NumberFormatInfo` .

```
// invariantResult is "1,234,567.89"
var invarianResult = string.Format(CultureInfo.InvariantCulture, "{0:#,###,##}", 1234567.89);

// NumberFormatInfo is one of classes that implement IFormatProvider
var customProvider = new NumberFormatInfo
{
    NumberDecimalSeparator = "_NS_", // will be used instead of ','
    NumberGroupSeparator = "_GS_", // will be used instead of '.'
```

```
};

// customResult is "1_GS_234_GS_567_NS_89"
var customResult = string.Format(customProvider, "{0:#,###.##}", 1234567.89);
```

```
public class CustomFormat : IFormatProvider, ICustomFormatter
{
    public string Format(string format, object arg, IFormatProvider formatProvider)
    {
        if (!this.Equals(formatProvider))
        {
            return null;
        }

        if (format == "Reverse")
        {
            return String.Join("", arg.ToString().Reverse());
        }

        return arg.ToString();
    }

    public object GetFormat(Type formatType)
    {
        return formatType==typeof(ICustomFormatter) ? this:null;
    }
}
```

:

```
String.Format(new CustomFormat(), "-> {0:Reverse} <-", "Hello World");
```

:

```
-> dlroW olleH <-
```

,

. .

```
string.Format("LEFT:  string: ->{0,-5}<- int: ->{1,-5}<-", "abc", 123);
string.Format("RIGHT: string: ->{0,5}<- int: ->{1,5}<-", "abc", 123);
```

:

```
LEFT:  string: ->abc  <- int: ->123  <-
RIGHT: string: ->  abc<- int: ->  123<-
```

```
// Integral types as hex
string.Format("Hexadecimal: byte2: {0:x2}; byte4: {0:X4}; char: {1:x2}", 123, (int)'A');

// Integers with thousand separators
string.Format("Integer, thousand sep.: {0:#,#}; fixed length: >{0,10:#,#}<", 1234567);
```

```
// Integer with leading zeroes
string.Format("Integer, leading zeroes: {0:00}; ", 1);

// Decimals
string.Format("Decimal, fixed precision: {0:0.000}; as percents: {0:0.00%}", 0.12);
```

:

```
Hexadecimal: byte2: 7b; byte4: 007B; char: 41
Integer, thousand sep.: 1,234,567; fixed length: > 1,234,567<
Integer, leading zeroes: 01;
Decimal, fixed precision: 0.120; as percents: 12.00%
```

"c"( ) .

```
string.Format("{0:c}", 112.236677) // $112.23 - defaults to system
```

2. c1, c2, c3 .

```
string.Format("{0:C1}", 112.236677) //$112.2
string.Format("{0:C3}", 112.236677) //$112.237
string.Format("{0:C4}", 112.236677) //$112.2367
string.Format("{0:C9}", 112.236677) //$112.236677000
```

1. `CultureInfo` culture .

```
string.Format(new CultureInfo("en-US"), "{0:c}", 112.236677); //$112.24
string.Format(new CultureInfo("de-DE"), "{0:c}", 112.236677); //112,24 €
string.Format(new CultureInfo("hi-IN"), "{0:c}", 112.236677); //  112.24
```

2. . `NumberFormatInfo` .

```
NumberFormatInfo nfi = new CultureInfo( "en-US", false ).NumberFormat;
nfi = (NumberFormatInfo) nfi.Clone();
nfi.CurrencySymbol = "?";
string.Format(nfi, "{0:C}", 112.236677); //?112.24
nfi.CurrencySymbol = "?%^&";
string.Format(nfi, "{0:C}", 112.236677); //?%^&112.24
```

CurrencyPositivePattern , CurrencyNegativePattern .

```
NumberFormatInfo nfi = new CultureInfo( "en-US", false ).NumberFormat;
nfi.CurrencyPositivePattern = 0;
string.Format(nfi, "{0:C}", 112.236677); //$112.24 - default
nfi.CurrencyPositivePattern = 1;
string.Format(nfi, "{0:C}", 112.236677); //112.24$
nfi.CurrencyPositivePattern = 2;
string.Format(nfi, "{0:C}", 112.236677); //$ 112.24
nfi.CurrencyPositivePattern = 3;
string.Format(nfi, "{0:C}", 112.236677); //112.24 $
```

. .

```
NumberFormatInfo nfi = new CultureInfo( "en-US", false ).NumberFormat;
nfi.CurrencyPositivePattern = 0;
nfi.CurrencyDecimalSeparator = "..";
string.Format(nfi, "{0:C}", 112.236677); //$112..24
```

C # 6.0

6.0

C # 6.0 `String.Format` .

```
string name = "John";
string lastname = "Doe";
Console.WriteLine($"Hello {name} {lastname}!");
```

John Doe!

C # 6.0 . .

String.Format () .

```
string outsidetext = "I am outside of bracket";
string.Format("{{I am in brackets!}} {0}", outsidetext);

//Outputs "{I am in brackets!} I am outside of bracket"
```

```
DateTime date = new DateTime(2016, 07, 06, 18, 30, 14);
// Format: year, month, day hours, minutes, seconds

Console.Write(String.Format("{0:dd}",date));

//Format by Culture info
String.Format(new System.Globalization.CultureInfo("mn-MN"),"{0:dddd}",date);
```

6.0

```
Console.Write($"{date:ddd}");
```

:

```
06
Лхагва
06
```

		`{0:d}`	2014 7 6
DD		`{0:dd}`	06

ddd		{0:ddd}	
dddd		{0:dddd}	
		{0:D}	2016 7 6
	,	{0:f}	2016 7 6 6:30
ff	, 2	{0:ff}	20
fff	, 3	{0:fff}	201
ffff	, 4	{0:ffff}	2016
		{0:F}	2016 7 6 6 30 14
		{0:g}	2014 7 6 6 30
gg		{0:gg}	
hh	(2 , 12H)	{0:hh}	06
HH	(2 , 24H)	{0:HH}	18
		{0:M}	7 6
mm		{0:mm}	30
MM		{0:MM}	07
MMM	3	{0:MMM}	7
MMMM		{0:MMMM}	
SS		{0:ss}	14
	RFC1123	{0:r}	2016 7 6 , , 18:30:14 GMT
		{0:s}	2016-07-06T18 : 30 : 14
		{0:t}	6:30
		{0:T}	6:30:14
tt		{0:tt}	
		{0:u}	2016-07-06 18 : 30 : 14Z
	GMT	{0:U}	2016 7 6 9:30:14
		{0:Y}	2016 7

	2	`{0:yy}`	16
yyyy	4	`{0:yyyy}`	2016
zz	2	`{0:zz}`	+09
		`{0:zzz}`	+09 : 00

ToString ()

ToString () . ToString () Object . ToString () . "" .

```
public class User
{
    public string Name { get; set; }
    public int Id { get; set; }
}

...

var user = new User {Name = "User1", Id = 5};
Console.WriteLine(user.ToString());
```

ToString () . "Id : 5, Name : User1" .

```
public class User
{
    public string Name { get; set; }
    public int Id { get; set; }
    public override ToString()
    {
        return string.Format("Id: {0}, Name: {1}", Id, Name);
    }
}

...

var user = new User {Name = "User1", Id = 5};
Console.WriteLine(user.ToString());
```

ToString ()

`String.Format()` , .

```
String.Format("{0:C}", money);  // yields "$42.00"
```

C # `ToString()` . .

```
money.ToString("C");  // yields "$42.00"
```

`ToString()` `String.Format()` .

```
String.Format("{0,10:C}", money);  // yields "    $42.00"
```

ToString() PadLeft() PadRight() .

```
money.ToString("C").PadLeft(10);  // yields "    $42.00"
```

String.Format : https://riptutorial.com/ko/csharp/topic/79/string-format

46: StringBuilder

Examples

StringBuilder ?

StringBuilder . ., .

```
string myString = "Apples";
mystring += " are my favorite fruit";
```

myString "Apples" . `` " .

- myString .
- myString .
- myString .

. ?

```
String myString = "";
for (int i = 0; i < 10000; i++)
    myString += " "; // puts 10,000 spaces into our string
```

. StringBuilder .

```
StringBuilder myStringBuilder = new StringBuilder();
for (int i = 0; i < 10000; i++)
    myStringBuilder.Append(' ');
```

. StringBuilder StringBuilder ToString() .

StringBuilder . .

```
StringBuilder sb = new StringBuilder(10000); // initializes the capacity to 10000
```

StringBuilder .

```
sb.Append('k', 2000);
```

StringBuilder StringBuilder .

StringBuilder ToString() string . StringBuilder string string .

, StringBuilder string .

```
string RepeatCharacterTimes(char character, int times)
```

```
{
    StringBuilder builder = new StringBuilder("");
    for (int counter = 0; counter < times; counter++)
    {
        //Append one instance of the character to the StringBuilder.
        builder.Append(character);
    }
    //Convert the result to string and return it.
    return builder.ToString();
}
```

StringBuilder .

StringBuilder

```
public string GetCustomerNamesCsv()
{
    List<CustomerData> customerDataRecords = GetCustomerData(); // Returns a large number of
records, say, 10000+

    StringBuilder customerNamesCsv = new StringBuilder();
    foreach (CustomerData record in customerDataRecords)
    {
       customerNamesCsv
            .Append(record.LastName)
            .Append(',')
            .Append(record.FirstName)
            .Append(Environment.NewLine);
    }

    return customerNamesCsv.ToString();
}
```

StringBuilder : https://riptutorial.com/ko/csharp/topic/4675/stringbuilder

47: System.DirectoryServices.Protocols.LdapConnec

Examples

SSL LDAP , SSL DNS .

. LDAPv3 .

```
// Authentication, and the name of the server.
private const string LDAPUser =
"cn=example:app:mygroup:accts,ou=Applications,dc=example,dc=com";
private readonly char[] password = { 'p', 'a', 's', 's', 'w', 'o', 'r', 'd' };
private const string TargetServer = "ldap.example.com";

// Specific to your company. Might start "cn=manager" instead of "ou=people", for example.
private const string CompanyDN = "ou=people,dc=example,dc=com";
```

LdapDirectoryIdentifier () NetworkCredentials .

```
// Configure server and port. LDAP w/ SSL, aka LDAPS, uses port 636.
// If you don't have SSL, don't give it the SSL port.
LdapDirectoryIdentifier identifier = new LdapDirectoryIdentifier(TargetServer, 636);

// Configure network credentials (userid and password)
var secureString = new SecureString();
foreach (var character in password)
        secureString.AppendChar(character);
NetworkCredential creds = new NetworkCredential(LDAPUser, secureString);

// Actually create the connection
LdapConnection connection = new LdapConnection(identifier, creds)
{
    AuthType = AuthType.Basic,
    SessionOptions =
    {
        ProtocolVersion = 3,
        SecureSocketLayer = true
    }
};

// Override SChannel reverse DNS lookup.
// This gets us past the "The LDAP server is unavailable." exception
// Could be
//    connection.SessionOptions.VerifyServerCertificate += { return true; };
// but some certificate validation is probably good.
connection.SessionOptions.VerifyServerCertificate +=
    (sender, certificate) => certificate.Subject.Contains(string.Format("CN={0},",
TargetServer));
```

LDAP (: objectClass userid). objectClass . "and".

```
 SearchRequest searchRequest = new SearchRequest(
        CompanyDN,
        string.Format((&(objectClass=*)(uid={0})), uid),
        SearchScope.Subtree,
        null
);

// Look at your results
foreach (SearchResultEntry entry in searchResponse.Entries) {
    // do something
}
```

LDAP

LDAPv3 . LDAPv3 LdapConnection .

```
private const string TargetServer = "ldap.example.com";
```

LdapDirectoryIdentifier () NetworkCredentials .

```
// Configure server and credentials
LdapDirectoryIdentifier identifier = new LdapDirectoryIdentifier(TargetServer);
NetworkCredential creds = new NetworkCredential();
LdapConnection connection = new LdapConnection(identifier, creds)
{
    AuthType=AuthType.Anonymous,
    SessionOptions =
    {
        ProtocolVersion = 3
    }
};
```

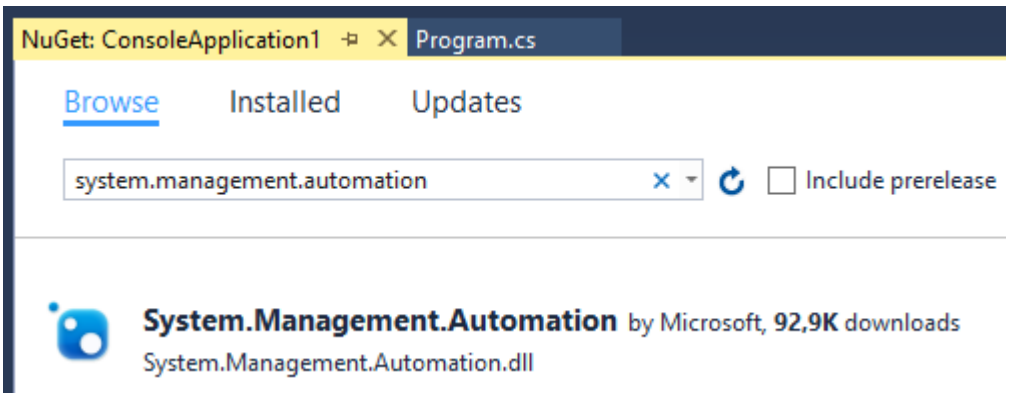
```
SearchRequest searchRequest = new SearchRequest("dn=example,dn=com", "(sn=Smith)",
SearchScope.Subtree,null);
```

System.DirectoryServices.Protocols.LdapConnection :
https://riptutorial.com/ko/csharp/topic/5177/system-directoryservices-protocols-ldapconnection

48: System.Management.Automation

System.Management.Automation Windows PowerShell .

System.Management.Automation Microsoft NuGet Visual Studio .

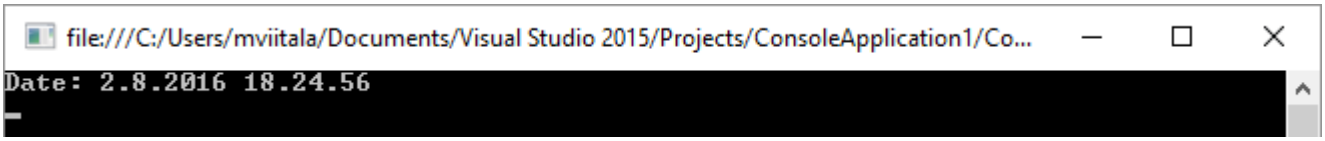



```
PM> Install-Package System.Management.Automation
```

Examples

.

```
public class Program
{
    static void Main()
    {
        // create empty pipeline
        PowerShell ps = PowerShell.Create();

        // add command
        ps.AddCommand("Get-Date");

        // run command(s)
        Console.WriteLine("Date: {0}", ps.Invoke().First());

        Console.ReadLine();
    }
}
```



System.Management.Automation : https://riptutorial.com/ko/csharp/topic/4988/system-management-automation

49: T4

- **T4**
- `<#@...#>` // ,
- `Plain Text` //
- `<#=...#>` //
- `<#+...#>` //
- `<#...#>` //

Examples

```
<#@ template language="C#" #> //Language of your project
<#@ assembly name="System.Core" #>
<#@ import namespace="System.Linq" #>
<#@ import namespace="System.Text" #>
<#@ import namespace="System.Collections.Generic" #>
```

T4 : https://riptutorial.com/ko/csharp/topic/4824/t4--

50: Windows Communication Foundation

Windows Communication Foundation (WCF) . WCF . IIS . XML .

Examples

.

```
// Define a service contract.
[ServiceContract(Namespace="http://StackOverflow.ServiceModel.Samples")]
public interface ICalculator
{
    [OperationContract]
    double Add(double n1, double n2);
}
```

.

```
// Service class that implements the service contract.
public class CalculatorService : ICalculator
{
    public double Add(double n1, double n2)
    {
        return n1 + n2;
    }
}
```

(Web.config) .

```
<services>
    <service
        name="StackOverflow.ServiceModel.Samples.CalculatorService"
        behaviorConfiguration="CalculatorServiceBehavior">
        <!-- ICalculator is exposed at the base address provided by
         host: http://localhost/servicemodelsamples/service.svc.  -->
       <endpoint address=""
             binding="wsHttpBinding"
             contract="StackOverflow.ServiceModel.Samples.ICalculator" />
       ...
    </service>
</services>
```

. ServiceMetadataBehavior http : //localhost/servicemodelsamples/service.svc/mex (MEX) .

.

```
<system.serviceModel>
  <services>
    <service
        name="StackOverflow.ServiceModel.Samples.CalculatorService"
        behaviorConfiguration="CalculatorServiceBehavior">
      ...
      <!-- the mex endpoint is explosed at
```

```
       http://localhost/servicemodelsamples/service.svc/mex -->
      <endpoint address="mex"
                binding="mexHttpBinding"
                contract="IMetadataExchange" />
    </service>
  </services>

  <!--For debugging purposes set the includeExceptionDetailInFaults
   attribute to true-->
  <behaviors>
    <serviceBehaviors>
      <behavior name="CalculatorServiceBehavior">
        <serviceMetadata httpGetEnabled="True"/>
        <serviceDebug includeExceptionDetailInFaults="False" />
      </behavior>
    </serviceBehaviors>
  </behaviors>
</system.serviceModel>
```

ServiceModel Metadata Utility Tool (Svcutil.exe) .

SDK .

```
svcutil.exe /n:"http://StackOverflow.ServiceModel.Samples,StackOverflow.ServiceModel.Samples"
http://localhost/servicemodelsamples/service.svc/mex /out:generatedClient.cs
```

(App.config) . , .

```
<client>
     <endpoint
         address="http://localhost/servicemodelsamples/service.svc"
         binding="wsHttpBinding"
         contract="StackOverflow.ServiceModel.Samples.ICalculator" />
</client>
```

.

```
// Create a client.
CalculatorClient client = new CalculatorClient();

// Call the Add service operation.
double value1 = 100.00D;
double value2 = 15.99D;
double result = client.Add(value1, value2);
Console.WriteLine("Add({0},{1}) = {2}", value1, value2, result);

//Closing the client releases all communication resources.
client.Close();
```

Windows Communication Foundation : https://riptutorial.com/ko/csharp/topic/10760/windows-communication-foundation

51: Windows Form MessageBox

MessageBox .

MessageBox , . / .

MessageBox dll MessageBox Control .

- ' DialogResult result = DialogResult.No; // DialogResult . '

Examples

MessageBox .

MessageBox .

1. Visual Studio (VS 2008/2010/2012/2015/2017)
2. -> -> Windows Forms -> .
3. ( ) ( ).

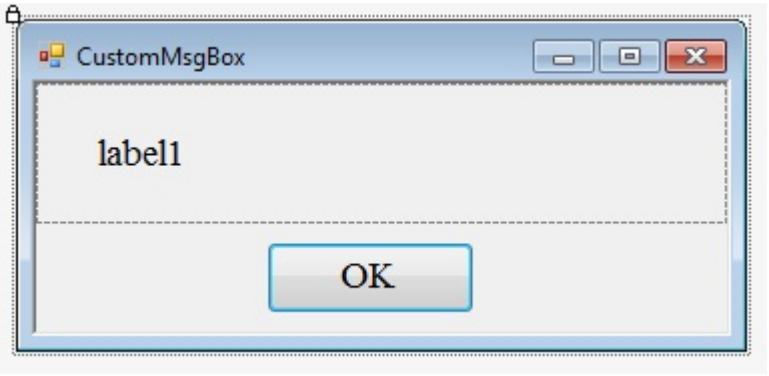
4. .
5. ( ).

```
namespace MsgBoxExample {
    public partial class MsgBoxExampleForm : Form {
        //Constructor, called when the class is initialised.
        public MsgBoxExampleForm() {
            InitializeComponent();
        }
```

```
        //Called whenever the button is clicked.
        private void btnShowMessageBox_Click(object sender, EventArgs e) {
           CustomMsgBox.Show($"I'm a {nameof(CustomMsgBox)}!", "MSG", "OK");
        }
    }
}
```

6. -> -> -> Windows Form "CustomMsgBox.cs" .

7. & ( ).



8. .

```
private DialogResult result = DialogResult.No;
public static DialogResult Show(string text, string caption, string btnOkText) {
    var msgBox = new CustomMsgBox();
    msgBox.lblText.Text = text; //The text for the label...
    msgBox.Text = caption; //Title of form
    msgBox.btnOk.Text = btnOkText; //Text on the button
    //This method is blocking, and will only return once the user
    //clicks ok or closes the form.
    msgBox.ShowDialog();
    return result;
}

private void btnOk_Click(object sender, EventArgs e) {
    result = DialogResult.Yes;
    MsgBox.Close();
}
```

9. F5 . . .

Windows Form MessageBox

.cs Visual Studio .

1. Visual Studio -> (Windows Form) -> -> -> -> -> -> .cs .

2. . using .

```
using System;
using System.Collections.Generic;
```

```
using System.ComponentModel;
using System.Data;
using System.Drawing;
.
.
.
using CustomMsgBox; //Here's the using statement for our dependency.
```

3. messagebox .

   CustomMsgBox.Show (" ...", "MSG", "OK");

Windows Form MessageBox : https://riptutorial.com/ko/csharp/topic/9788/windows-form----messagebox-

52: XDocument System.Xml.Linq

Examples

XML

XML .

```
<FruitBasket xmlns="http://www.fruitauthority.fake">
  <Fruit ID="F0001">
    <FruitName>Banana</FruitName>
    <FruitColor>Yellow</FruitColor>
  </Fruit>
  <Fruit ID="F0002">
    <FruitName>Apple</FruitName>
    <FruitColor>Red</FruitColor>
  </Fruit>
</FruitBasket>
```

:

```
XNamespace xns = "http://www.fruitauthority.fake";
XDeclaration xDeclaration = new XDeclaration("1.0", "utf-8", "yes");
XDocument xDoc = new XDocument(xDeclaration);
XElement xRoot = new XElement(xns + "FruitBasket");
xDoc.Add(xRoot);

XElement xelFruit1 = new XElement(xns + "Fruit");
XAttribute idAttribute1 = new XAttribute("ID", "F0001");
xelFruit1.Add(idAttribute1);
XElement xelFruitName1 = new XElement(xns + "FruitName", "Banana");
XElement xelFruitColor1 = new XElement(xns + "FruitColor", "Yellow");
xelFruit1.Add(xelFruitName1);
xelFruit1.Add(xelFruitColor1);
xRoot.Add(xelFruit1);

XElement xelFruit2 = new XElement(xns + "Fruit");
XAttribute idAttribute2 = new XAttribute("ID", "F0002");
xelFruit2.Add(idAttribute2);
XElement xelFruitName2 = new XElement(xns + "FruitName", "Apple");
XElement xelFruitColor2 = new XElement(xns + "FruitColor", "Red");
xelFruit2.Add(xelFruitName2);
xelFruit2.Add(xelFruitColor2);
xRoot.Add(xelFruit2);
```

XML

XML `XDocument` , `XDocument` , , . XML .

XML :

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<FruitBasket xmlns="http://www.fruitauthority.fake">
  <Fruit>
    <FruitName>Banana</FruitName>
    <FruitColor>Yellow</FruitColor>
  </Fruit>
  <Fruit>
    <FruitName>Apple</FruitName>
    <FruitColor>Red</FruitColor>
  </Fruit>
</FruitBasket>
```

:

1. .
2. `XDocument.Load` URI ( ).
3. xml AND .
4. null C # 6 Linq . . null . C # 6 null . `<Fruit>` . `IEnumerable<XElement>` `FirstOfDefault()` .
5. FruitColor . . .
6. null , FruitColor "Brown" .
7. `XDocument` .

```
// 1.
string xmlFilePath = "c:\\users\\public\\fruit.xml";

// 2.
XDocument xdoc = XDocument.Load(xmlFilePath);

// 3.
XNamespace ns = "http://www.fruitauthority.fake";

//4.
var elBanana = xdoc.Descendants()?.
    Elements(ns + "FruitName")?.
    Where(x => x.Value == "Banana")?.
    Ancestors(ns + "Fruit");

// 5.
var elColor = elBanana.Elements(ns + "FruitColor").FirstOrDefault();

// 6.
if (elColor != null)
{
    elColor.Value = "Brown";
}

// 7.
xdoc.Save(xmlFilePath);
```

.

```
<?xml version="1.0" encoding="utf-8"?>
<FruitBasket xmlns="http://www.fruitauthority.fake">
  <Fruit>
    <FruitName>Banana</FruitName>
    <FruitColor>Brown</FruitColor>
```

```
  </Fruit>
  <Fruit>
    <FruitName>Apple</FruitName>
    <FruitColor>Red</FruitColor>
  </Fruit>
</FruitBasket>
```

XML

:

```
<FruitBasket xmlns="http://www.fruitauthority.fake">
  <Fruit>
    <FruitName>Banana</FruitName>
    <FruitColor>Yellow</FruitColor>
  </Fruit>
  <Fruit>
    <FruitName>Apple</FruitName>
    <FruitColor>Red</FruitColor>
  </Fruit>
</FruitBasket>
```

:

```
XNamespace xns = "http://www.fruitauthority.fake";
XDocument xDoc =
    new XDocument(new XDeclaration("1.0", "utf-8", "yes"),
        new XElement(xns + "FruitBasket",
            new XElement(xns + "Fruit",
                new XElement(xns + "FruitName", "Banana"),
                new XElement(xns + "FruitColor", "Yellow")),
            new XElement(xns + "Fruit",
                new XElement(xns + "FruitName", "Apple"),
                new XElement(xns + "FruitColor", "Red"))
                ));
```

XDocument System.Xml.Linq : https://riptutorial.com/ko/csharp/topic/1866/xdocument--system-xml-linq--

53: XML

XML . .

.

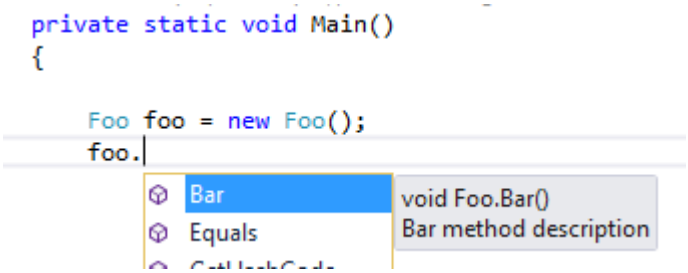
-
-
- NDoc
- DocFX

Examples

. (/// XML .

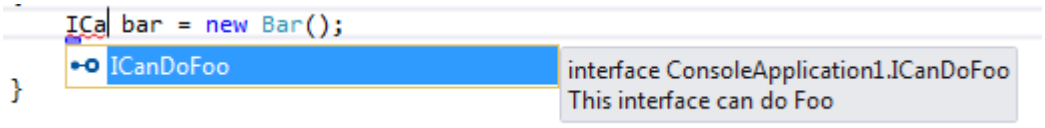
```
/// <summary>
/// Bar method description
/// </summary>
public void Bar()
{

}
```

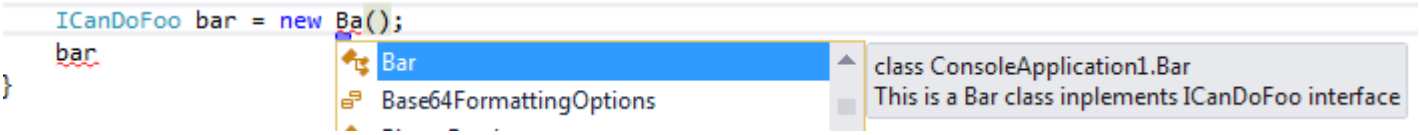
Visual Studio IntelliSense .



Microsoft .

```
/// <summary>
/// This interface can do Foo
/// </summary>
public interface ICanDoFoo
{
    // ...
}

/// <summary>
/// This Bar class implements ICanDoFoo interface
/// </summary>
public class Bar : ICanDoFoo
{
    // ...
}
```

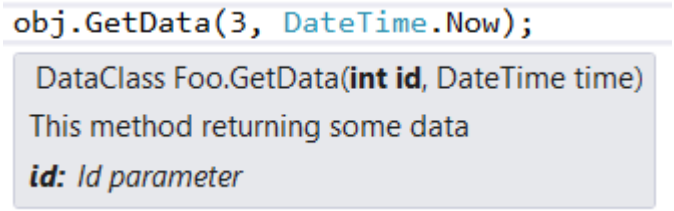




param

```
/// <summary>
/// Returns the data for the specified ID and timestamp.
/// </summary>
/// <param name="id">The ID for which to get data. </param>
/// <param name="time">The DateTime for which to get data. </param>
/// <returns>A DataClass instance with the result. </returns>
public DataClass GetData(int id, DateTime time)
{
   // ...
}
```

IntelliSense .

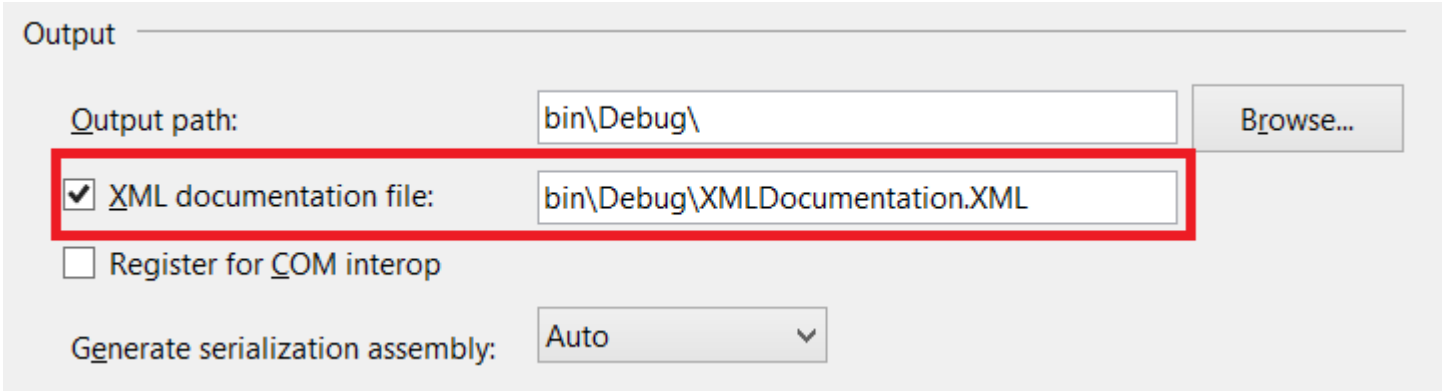


: Intellisense Visual Studio .

XML

XML `csc.exe` C # `/doc` .

Visual Studio 2013/2015 -> -> -> `XML documentation file` .



(: `XMLDocumentation.dll` -> `XMLDocumentation.xml`) XML .

XML DLL .

:

```
/// <summary>
/// Data class description
/// </summary>
public class DataClass
{
    /// <summary>
    /// Name property description
    /// </summary>
    public string Name { get; set; }
}


/// <summary>
/// Foo function
/// </summary>
public class Foo
{
    /// <summary>
    /// This method returning some data
    /// </summary>
    /// <param name="id">Id parameter</param>
    /// <param name="time">Time parameter</param>
    /// <returns>Data will be returned</returns>
    public DataClass GetData(int id, DateTime time)
    {
        return new DataClass();
    }
}
```

XML .

```
<?xml version="1.0"?>
<doc>
    <assembly>
        <name>XMLDocumentation</name>
    </assembly>
    <members>
        <member name="T:XMLDocumentation.DataClass">
            <summary>
            Data class description
            </summary>
        </member>
        <member name="P:XMLDocumentation.DataClass.Name">
            <summary>
            Name property description
            </summary>
        </member>
        <member name="T:XMLDocumentation.Foo">
            <summary>
            Foo function
            </summary>
        </member>
        <member name="M:XMLDocumentation.Foo.GetData(System.Int32,System.DateTime)">
            <summary>
            This method returning some data
```

```
            </summary>
            <param name="id">Id parameter</param>
            <param name="time">Time parameter</param>
            <returns>Data will be returned</returns>
        </member>
    </members>
</doc>
```

`<see>` . `cref` . Visual Studio Intellsense .

```
/// <summary>
/// You might also want to check out <see cref="SomeOtherClass"/>.
/// </summary>
public class SomeClass
{
}
```

Visual Studio Intellisense .

.

```
/// <summary>
/// An enhanced version of <see cref="List{T}"/>.
/// </summary>
public class SomeGenericClass<T>
{
}
```

XML : https://riptutorial.com/ko/csharp/topic/740/xml--

54: XmlDocument System.Xml

Examples

XML

```
public static void Main()
{
    var xml  = new XmlDocument();
    var root = xml.CreateElement("element");
        // Creates an attribute, so the element will now be "<element attribute='value' />"
        root.SetAttribute("attribute", "value");

    // All XML documents must have one, and only one, root element
    xml.AppendChild(root);

    // Adding data to an XML document
    foreach (var dayOfWeek in Enum.GetNames((typeof(DayOfWeek))))
    {
        var day = xml.CreateElement("dayOfWeek");
            day.SetAttribute("name", dayOfWeek);

        // Don't forget to add the new value to the current document!
        root.AppendChild(day);
    }

    // Looking for data using XPath; BEWARE, this is case-sensitive
    var monday = xml.SelectSingleNode("//dayOfWeek[@name='Monday']");
    if (monday != null)
    {
        // Once you got a reference to a particular node, you can delete it
        // by navigating through its parent node and asking for removal
        monday.ParentNode.RemoveChild(monday);
    }

    // Displays the XML document in the screen; optionally can be saved to a file
    xml.Save(Console.Out);
}
```

XML

XML

```
    <Sample>
    <Account>
        <One number="12"/>
        <Two number="14"/>
    </Account>
    <Account>
        <One number="14"/>
        <Two number="16"/>
    </Account>
    </Sample>
```

XML :

```
    using System.Xml;
    using System.Collections.Generic;

    public static void Main(string fullpath)
    {
        var xmldoc = new XmlDocument();
        xmldoc.Load(fullpath);

        var oneValues = new List<string>();

        // Getting all XML nodes with the tag name
        var accountNodes = xmldoc.GetElementsByTagName("Account");
        for (var i = 0; i < accountNodes.Count; i++)
        {
            // Use Xpath to find a node
            var account = accountNodes[i].SelectSingleNode("./One");
            if (account != null && account.Attributes != null)
            {
                // Read node attribute
                oneValues.Add(account.Attributes["number"].Value);
            }
        }
}
```

XmlDocument XDocument ( )

Xml .

1. XML
2. XDocument
3. XmlReader / XmlWriter

XML LINQ , XML XMLDocument . LINQ to XML XDocument . XMLDocument .

XDocument XmlDocument mutch. XmlDoucument XML .

XmlDocument XDocument .

XML

```
string filename = @"C:\temp\test.xml";
```

XmlDocument

```
XmlDocument _doc = new XmlDocument();
_doc.Load(filename);
```

XDocument

```
XDocument _doc = XDocument.Load(fileName);
```

XmlDocument

XmlDocument

```
XmlDocument doc = new XmlDocument();
XmlElement root = doc.CreateElement("root");
root.SetAttribute("name", "value");
XmlElement child = doc.CreateElement("child");
child.InnerText = "text node";
root.AppendChild(child);
doc.AppendChild(root);
```

XDocument

```
 XDocument doc = new XDocument(
    new XElement("Root", new XAttribute("name", "value"),
    new XElement("Child", "text node"))
);

/*result*/
<root name="value">
    <child>"TextNode"</child>
</root>
```

XML InnerText

XmlDocument

```
XmlNode node = _doc.SelectSingleNode("xmlRootNode");
node.InnerText = value;
```

XDocument

```
 XElement rootNote = _doc.XPathSelectElement("xmlRootNode");
rootNode.Value = "New Value";
```

xml .

```
// Safe XmlDocument and XDocument
_doc.Save(filename);
```

XML

XmlDocument

```
 XmlNode node = _doc.SelectSingleNode("xmlRootNode/levelOneChildNode");
string text = node.InnerText;
```

XDocument

```
 XElement node = _doc.XPathSelectElement("xmlRootNode/levelOneChildNode");
 string text = node.Value;
```

attribute = something .

XmlDocument

```
List<string> valueList = new List<string>();
    foreach (XmlNode n in nodelist)
    {
        if(n.Attributes["type"].InnerText == "City")
        {
            valueList.Add(n.Attributes["type"].InnerText);
        }
    }
```

XDocument

```
var accounts = _doc.XPathSelectElements("/data/summary/account").Where(c =>
c.Attribute("type").Value == "setting").Select(c => c.Value);
```

XmlDocument

```
XmlNode nodeToAppend = doc.CreateElement("SecondLevelNode");
nodeToAppend.InnerText = "This title is created by code";

/* Append node to parent */
XmlNode firstNode= _doc.SelectSingleNode("xmlRootNode/levelOneChildNode");
firstNode.AppendChild(nodeToAppend);

/*After a change make sure to safe the document*/
_doc.Save(fileName);
```

XDocument

```
_doc.XPathSelectElement("ServerManagerSettings/TcpSocket").Add(new
XElement("SecondLevelNode"));

 /*After a change make sure to safe the document*/
_doc.Save(fileName);
```

XmlDocument System.Xml : https://riptutorial.com/ko/csharp/topic/1528/xmldocument--system-xml--

55: .

Examples

Parallel.For

```
using System;
using System.Threading;
using System.Threading.Tasks;

class Program
{
    static void Main( string[] args )
    {
        object sync = new object();
        int sum = 0;
        Parallel.For( 1, 1000, ( i ) => {
            lock( sync ) sum = sum + i; // lock is necessary

            // As a practical matter, ensure this `parallel for` executes
            // on multiple threads by simulating a lengthy operation.
            Thread.Sleep( 1 );
        } );
        Console.WriteLine( "Correct answer should be 499500.  sum is: {0}", sum );
    }
}
```

- - sum = sum + i . sum sum , sum + i sum .

. : https://riptutorial.com/ko/csharp/topic/4140/----

56:

- : public void Double (ref int numberToDouble) {}

. . 3D , .

() struct . .

2 .

-
-

. . , ( ) finalizers .

C # .

-
-
-

() class . , .

.

,

C # int . int mine = 0, mine 0 . ( ). C ++ .

. .

, .

.

. .

.

?

"ref" . , . . .

null

, null . , . . Nullable nullable ( ). . .

Examples

```
public static void Main(string[] args)
{
    var studentList = new List<Student>();
    studentList.Add(new Student("Scott", "Nuke"));
    studentList.Add(new Student("Vincent", "King"));
    studentList.Add(new Student("Craig", "Bertt"));

    // make a separate list to print out later
    var printingList = studentList; // this is a new list object, but holding the same student
objects inside it

    // oops, we've noticed typos in the names, so we fix those
    studentList[0].LastName = "Duke";
    studentList[1].LastName = "Kong";
    studentList[2].LastName = "Brett";

    // okay, we now print the list
    PrintPrintingList(printingList);
}

private static void PrintPrintingList(List<Student> students)
{
    foreach (Student student in students)
    {
        Console.WriteLine(string.Format("{0} {1}", student.FirstName, student.LastName));
    }
}
```

printingList PrintPrintingList .

```
Scott Duke
Vincent Kong
Craig Brett
```

. .

.

```
public class Student
{
    public string FirstName { get; set; }
    public string LastName { get; set; }

    public Student(string firstName, string lastName)
    {
        this.FirstName = firstName;
        this.LastName = lastName;
    }
}
```

ref .

```
public static void Main(string[] args)
{
    ...
    DoubleNumber(ref number); // calling code
    Console.WriteLine(number); // outputs 8
```

```
    ...
}
```

```
public void DoubleNumber(ref int number)
{
    number += number;
}
```

number 8.

ref .

:

C # . , , , , . ref out .

ref out out ends.in ref .

```
using System;

class Program
{
    static void Main(string[] args)
    {
        int a = 20;
        Console.WriteLine("Inside Main - Before Callee: a = {0}", a);
        Callee(a);
        Console.WriteLine("Inside Main - After Callee: a = {0}", a);

        Console.WriteLine("Inside Main - Before CalleeRef: a = {0}", a);
        CalleeRef(ref a);
        Console.WriteLine("Inside Main - After CalleeRef: a = {0}", a);

        Console.WriteLine("Inside Main - Before CalleeOut: a = {0}", a);
        CalleeOut(out a);
        Console.WriteLine("Inside Main - After CalleeOut: a = {0}", a);

        Console.ReadLine();
    }

    static void Callee(int a)
    {
        a = 5;
        Console.WriteLine("Inside Callee a : {0}", a);
    }

    static void CalleeRef(ref int a)
    {
        a = 6;
        Console.WriteLine("Inside CalleeRef a : {0}", a);
    }

    static void CalleeOut(out int a)
    {
        a = 7;
        Console.WriteLine("Inside CalleeOut a : {0}", a);
    }
```

```
}
```

:

```
Inside Main - Before Callee: a = 20
Inside Callee a : 5
Inside Main - After Callee: a = 20
Inside Main - Before CalleeRef: a = 20
Inside CalleeRef a : 6
Inside Main - After CalleeRef: a = 6
Inside Main - Before CalleeOut: a = 6
Inside CalleeOut a : 7
Inside Main - After CalleeOut: a = 7
```

```
var a = new List<int>();
var b = a;
a.Add(5);
Console.WriteLine(a.Count); // prints 1
Console.WriteLine(b.Count); // prints 1 as well
```

(A) `List<int>` `List<int>` . `List<int>` . .

ref out

ref out . ref out . out .

```
public void ByRef(ref int value)
{
    Console.WriteLine(nameof(ByRef) + value);
    value += 4;
    Console.WriteLine(nameof(ByRef) + value);
}

public void ByOut(out int value)
{
    value += 4 // CS0269: Use of unassigned out parameter `value'
    Console.WriteLine(nameof(ByOut) + value); // CS0269: Use of unassigned out parameter
`value'

    value = 4;
    Console.WriteLine(nameof(ByOut) + value);
}

public void TestOut()
{
    int outValue1;
    ByOut(out outValue1); // prints 4

    int outValue2 = 10;   // does not make any sense for out
    ByOut(out outValue2); // prints 4
}

public void TestRef()
{
    int refValue1;
    ByRef(ref refValue1); // S0165  Use of unassigned local variable 'refValue'
```

```
    int refValue2 = 0;
    ByRef(ref refValue2); // prints 0 and 4

    int refValue3 = 10;
    ByRef(ref refValue3); // prints 10 and 14
}
```

out must ref out :

```
public void EmtyRef(bool condition, ref int value)
{
    if (condition)
    {
        value += 10;
    }
}

public void EmtyOut(bool condition, out int value)
{
    if (condition)
    {
        value = 10;
    }
} //CS0177: The out parameter 'value' must be assigned before control leaves the current
method
```

condition condition value .

ref out

```
class Program
{
    static void Main(string[] args)
    {
        int a = 20;
        Console.WriteLine("Inside Main - Before Callee: a = {0}", a);
        Callee(a);
        Console.WriteLine("Inside Main - After Callee: a = {0}", a);
        Console.WriteLine();

        Console.WriteLine("Inside Main - Before CalleeRef: a = {0}", a);
        CalleeRef(ref a);
        Console.WriteLine("Inside Main - After CalleeRef: a = {0}", a);
        Console.WriteLine();

        Console.WriteLine("Inside Main - Before CalleeOut: a = {0}", a);
        CalleeOut(out a);
        Console.WriteLine("Inside Main - After CalleeOut: a = {0}", a);
        Console.ReadLine();
    }

    static void Callee(int a)
    {
        a += 5;
        Console.WriteLine("Inside Callee a : {0}", a);
    }
```

```
    static void CalleeRef(ref int a)
    {
        a += 10;
        Console.WriteLine("Inside CalleeRef a : {0}", a);
    }

    static void CalleeOut(out int a)
    {
        // can't use a+=15 since for this method 'a' is not intialized only declared in the
method declaration
        a = 25; //has to be initialized
        Console.WriteLine("Inside CalleeOut a : {0}", a);
    }
}
```

```
Inside Main - Before Callee: a = 20
Inside Callee a : 25
Inside Main - After Callee: a = 20

Inside Main - Before CalleeRef: a = 20
Inside CalleeRef a : 30
Inside Main - After CalleeRef: a = 30

Inside Main - Before CalleeOut: a = 30
Inside CalleeOut a : 25
Inside Main - After CalleeOut: a = 25
```

: https://riptutorial.com/ko/csharp/topic/3014/---

57: GetHashCode

Equals .

- **(Reflexive)** : .
  x.Equals(x) true true .
- : x y x y . .
  x.Equals(y) y.Equals(x) .
- : .
  if (x.Equals(y) && y.Equals(z)) true x.Equals(z) true true .
- : .
  x.Equals(y) x y .
- **null** : null .
  x.Equals(null) false .

GetHashCode :

- **Equals** : (Equals true), GetHashCode .
- : (Equals) . .
- : .

: Equals () ==

Examples

.

Equals Object .

```
public virtual bool Equals(Object obj);
```

Equals .

- Equals .
- Equals true .
- string . .

```
namespace ConsoleApplication
{
    public class Program
```

```
    {
        public static void Main(string[] args)
        {
            //areFooClassEqual: False
            Foo fooClass1 = new Foo("42");
            Foo fooClass2 = new Foo("42");
            bool areFooClassEqual = fooClass1.Equals(fooClass2);
            Console.WriteLine("fooClass1 and fooClass2 are equal: {0}", areFooClassEqual);
            //False

            //areFooIntEqual: True
            int fooInt1 = 42;
            int fooInt2 = 42;
            bool areFooIntEqual = fooInt1.Equals(fooInt2);
            Console.WriteLine("fooInt1 and fooInt2 are equal: {0}", areFooIntEqual);

            //areFooStringEqual: True
            string fooString1 = "42";
            string fooString2 = "42";
            bool areFooStringEqual = fooString1.Equals(fooString2);
            Console.WriteLine("fooString1 and fooString2 are equal: {0}", areFooStringEqual);
        }
    }

    public class Foo
    {
        public string Bar { get; }

        public Foo(string bar)
        {
            Bar = bar;
        }
    }
}
```

GetHashCode

`GetHashCode` Dictionary <> HashTable .

`GetHashCode`

- .
  - .
  - (: '999')
- .
  - .
  - .
- `Equals` true HashCode .
  - (: `GetHashCode`) `List` , `Dictionary` .

`GetHashCode` .

```
public override int GetHashCode()
{
    unchecked // Overflow is fine, just wrap
    {
```

```
        int hash = 3049; // Start value (prime number).

        // Suitable nullity checks etc, of course :)
        hash = hash * 5039 + field1.GetHashCode();
        hash = hash * 883 + field2.GetHashCode();
        hash = hash * 9719 + field3.GetHashCode();
        return hash;
    }
}
```

Equals .

Dictionary / HashTables IEqualityComparer .

Equals GetHashCode

Person :

```
public class Person
{
    public string Name { get; set; }
    public int Age { get; set; }
    public string Clothes { get; set; }
}

var person1 = new Person { Name = "Jon", Age = 20, Clothes = "some clothes" };
var person2 = new Person { Name = "Jon", Age = 20, Clothes = "some other clothes" };

bool result = person1.Equals(person2); //false because it's reference Equals
```

Equals GetHashCode .

```
public class Person
{
    public string Name { get; set; }
    public int Age { get; set; }
    public string Clothes { get; set; }

    public override bool Equals(object obj)
    {
        var person = obj as Person;
        if(person == null) return false;
        return Name == person.Name && Age == person.Age; //the clothes are not important when
comparing two persons
    }

    public override int GetHashCode()
    {
        return Name.GetHashCode()*Age;
    }
}

var person1 = new Person { Name = "Jon", Age = 20, Clothes = "some clothes" };
var person2 = new Person { Name = "Jon", Age = 20, Clothes = "some other clothes" };

bool result = person1.Equals(person2); // result is true
```

LINQ Equals GetHashCode .

```
var persons = new List<Person>
{
     new Person{ Name = "Jon", Age = 20, Clothes = "some clothes"},
     new Person{ Name = "Dave", Age = 20, Clothes = "some other clothes"},
     new Person{ Name = "Jon", Age = 20, Clothes = ""}
};

var distinctPersons = persons.Distinct().ToList();//distinctPersons has Count = 2
```

IEqualityComparator Equals GetHashCode

Person :

```
public class Person
{
    public string Name { get; set; }
    public int Age { get; set; }
    public string Clothes { get; set; }
}

List<Person> persons = new List<Person>
{
    new Person{ Name = "Jon", Age = 20, Clothes = "some clothes"},
    new Person{ Name = "Dave", Age = 20, Clothes = "some other clothes"},
    new Person{ Name = "Jon", Age = 20, Clothes = ""}
};

var distinctPersons = persons.Distinct().ToList();// distinctPersons has Count = 3
```

Equals GetHashCode IEqualityComparator :

```
public class PersonComparator : IEqualityComparer<Person>
{
    public bool Equals(Person x, Person y)
    {
        return x.Name == y.Name && x.Age == y.Age; //the clothes are not important when
comparing two persons;
    }

    public int GetHashCode(Person obj) { return obj.Name.GetHashCode() * obj.Age; }
}

var distinctPersons = persons.Distinct(new PersonComparator()).ToList();// distinctPersons has
Count = 2
```

Equals true GetHashCode .

GetHashCode : https://riptutorial.com/ko/csharp/topic/3429/--gethashcode

58:

- SomeClass sc = SomeClass {Property1 = value1, Property2 = value2, ...};
- SomeClass sc = SomeClass (param1, param2, ...) {Property1 = value1, Property2 = value2, ...}

( ) .

Examples

. .

```
public class Book
{
    public string Title { get; set; }
    public string Author { get; set; }

    // the rest of class definition
}
```

:

```
Book theBook = new Book { Title = "Don Quixote", Author = "Miguel de Cervantes" };
```

```
Book theBook = new Book();
theBook.Title = "Don Quixote";
theBook.Author = "Miguel de Cervantes";
```

.

```
var album = new { Band = "Beatles", Title = "Abbey Road" };
```

LINQ select . LINQ select .

```
var albumTitles = from a in albums
                  select new
                  {
                     Title = a.Title,
                     Artist = a.Band
                  };
```

. .

```
public class Book {
    public string Title { get; set; }
    public string Author { get; set; }

    public Book(int id) {
        //do things
```

```
    }

    // the rest of class definition
}

var someBook = new Book(16) { Title = "Don Quixote", Author = "Miguel de Cervantes" }
```

`Book(int)` `Book` . :

```
var someBook = new Book(16);
someBook.Title = "Don Quixote";
someBook.Author = "Miguel de Cervantes";
```

: https://riptutorial.com/ko/csharp/topic/738/--

59:

- DerivedClass : BaseClass
- DerivedClass : BaseClass, IExampleInterface
- DerivedClass : BaseClass, IExampleInterface, IAnotherInterface

( ) .

. System.ValueType System.Object .

.

Examples

.

```
public class Animal
{
    public string Name { get; set; }
    // Methods and attributes common to all animals
    public void Eat(Object dinner)
    {
        // ...
    }
    public void Stare()
    {
        // ...
    }
    public void Roll()
    {
        // ...
    }
}
```

Animal .

```
public class Cat : Animal
{
    public Cat()
    {
        Name = "Cat";
    }
    // Methods for scratching furniture and ignoring owner
    public void Scratch(Object furniture)
    {
        // ...
    }
}
```

Cat Animal . ( ) , , . . . (Gopher Sloth .)

```
public class Animal
```

```
{
    public string Name { get; set; }
}

public interface INoiseMaker
{
    string MakeNoise();
}

//Note that in C#, the base class name must come before the interface names
public class Cat : Animal, INoiseMaker
{
    public Cat()
    {
        Name = "Cat";
    }

    public string MakeNoise()
    {
        return "Nyan";
    }
}
```

```
public class LivingBeing
{
    string Name { get; set; }
}

public interface IAnimal
{
    bool HasHair { get; set; }
}

public interface INoiseMaker
{
    string MakeNoise();
}

//Note that in C#, the base class name must come before the interface names
public class Cat : LivingBeing, IAnimal, INoiseMaker
{
    public Cat()
    {
        Name = "Cat";
        HasHair = true;
    }

    public bool HasHair { get; set; }

    public string Name { get; set; }

    public string MakeNoise()
    {
        return "Nyan";
    }
}
```

```
interface BaseInterface {}
class BaseClass : BaseInterface {}
```

```
interface DerivedInterface {}
class DerivedClass : BaseClass, DerivedInterface {}

var baseInterfaceType = typeof(BaseInterface);
var derivedInterfaceType = typeof(DerivedInterface);
var baseType = typeof(BaseClass);
var derivedType = typeof(DerivedClass);

var baseInstance = new BaseClass();
var derivedInstance = new DerivedClass();

Console.WriteLine(derivedInstance is DerivedClass); //True
Console.WriteLine(derivedInstance is DerivedInterface); //True
Console.WriteLine(derivedInstance is BaseClass); //True
Console.WriteLine(derivedInstance is BaseInterface); //True
Console.WriteLine(derivedInstance is object); //True

Console.WriteLine(derivedType.BaseType.Name);  //BaseClass
Console.WriteLine(baseType.BaseType.Name);  //Object
Console.WriteLine(typeof(object).BaseType);  //null

Console.WriteLine(baseType.IsInstanceOfType(derivedInstance));  //True
Console.WriteLine(derivedType.IsInstanceOfType(baseInstance));  //False

Console.WriteLine(
    string.Join(",",
    derivedType.GetInterfaces().Select(t => t.Name).ToArray()));
//BaseInterface,DerivedInterface

Console.WriteLine(baseInterfaceType.IsAssignableFrom(derivedType)); //True
Console.WriteLine(derivedInterfaceType.IsAssignableFrom(derivedType)); //True
Console.WriteLine(derivedInterfaceType.IsAssignableFrom(baseType)); //False
```

.

( ) .

.

```
public abstract class Car
{
    public void HonkHorn() {
        // Implementation of horn being honked
    }
}

public class Mustang : Car
{
    // Simply by extending the abstract class Car, the Mustang can HonkHorn()
    // If Car were an interface, the HonkHorn method would need to be included
    // in every class that implemented it.
}
```

Car HonkHorn . , .

: base .

```
class Instrument
{
    string type;
    bool clean;

    public Instrument (string type, bool clean)
    {
        this.type = type;
        this.clean = clean;
    }
}

class Trumpet : Instrument
{
    bool oiled;

    public Trumpet(string type, bool clean, bool oiled) : base(type, clean)
    {
        this.oiled = oiled;
    }
}
```

.

Animal . Dog

```
class Animal
{
    public Animal()
    {
        Console.WriteLine("In Animal's constructor");
    }
}

class Dog : Animal
{
    public Dog()
    {
        Console.WriteLine("In Dog's constructor");
    }
}
```

Object .

.

```
class Animal : Object
{
    public Animal()
    {
        Console.WriteLine("In Animal's constructor");
    }
}
```

Dog . , Object's , Animal's , Dog's .

```
public class Program
{
    public static void Main()
    {
        Dog dog = new Dog();
    }
}
```

.

Dog Animal .   .   .

.

```
class Animal
{
    protected string name;

    public Animal()
    {
        Console.WriteLine("Animal's default constructor");
    }

    public Animal(string name)
    {
        this.name = name;
        Console.WriteLine("Animal's constructor with 1 parameter");
        Console.WriteLine(this.name);
    }
}

class Dog : Animal
{
    public Dog() : base()
    {
        Console.WriteLine("Dog's default constructor");
    }

    public Dog(string name) : base(name)
    {
        Console.WriteLine("Dog's constructor with 1 parameter");
        Console.WriteLine(this.name);
    }
}
```

?

2 .

base ?

base . Dog

```
Dog dog = new Dog();
```

Dog() . . .base() . Dog . Dog() .

.

Dog's ?

```
Dog dog = new Dog("Rex");
```

., Dog name .
. (), name , .

```
Animal's constructor with 1 parameter
Rex
Dog's constructor with 1 parameter
Rex
```

:

. . Object Object . .

1. base .
2. . .

.

```
public abstract class Car
{
    public void HonkHorn() {
        // Implementation of horn being honked
    }

    // virtual methods CAN be overridden in derived classes
    public virtual void ChangeGear() {
        // Implementation of gears being changed
    }

    // abstract methods MUST be overridden in derived classes
    public abstract void Accelerate();
}

public class Mustang : Car
{
    // Before any code is added to the Mustang class, it already contains
    // implementations of HonkHorn and ChangeGear.

    // In order to compile, it must be given an implementation of Accelerate,
    // this is done using the override keyword
    public override void Accelerate() {
        // Implementation of Mustang accelerating
    }

    // If the Mustang changes gears differently to the implementation in Car
    // this can be overridden using the same override keyword as above
```

```
    public override void ChangeGear() {
        // Implementation of Mustang changing gears
    }
}
```

Foo Bar Foo . Foo Do1 Do2 . Bar Do1 Foo Do2 Do2 .

: Foo Do2() Bar Do2() Bar throw Exception throw Exception

```
public class Bar : Foo
{
    public override void Do2()
    {
        //Does something completely different that you would expect Foo to do
        //or simply throws new Exception
    }
}
```

Foo Do1() Baz Baz Foo Bar Do2() .

```
public class Baz
{
    public void Do1()
    {
        // magic
    }
}

public class Foo : Baz
{
    public void Do2()
    {
        // foo way
    }
}

public class Bar : Baz
{
    public void Do2()
    {
        // bar way or not have Do2 at all
    }
}
```

. nr2 Foo Bar Foo Bar . Foo Bar commonalty Baz .

. .

```
/// <summary>
/// Generic base class for a tree structure
/// </summary>
/// <typeparam name="T">The node type of the tree</typeparam>
public abstract class Tree<T> where T : Tree<T>
{
    /// <summary>
```

```
    /// Constructor sets the parent node and adds this node to the parent's child nodes
    /// </summary>
    /// <param name="parent">The parent node or null if a root</param>
    protected Tree(T parent)
    {
        this.Parent=parent;
        this.Children=new List<T>();
        if(parent!=null)
        {
            parent.Children.Add(this as T);
        }
    }
    public T Parent { get; private set; }
    public List<T> Children { get; private set; }
    public bool IsRoot { get { return Parent==null; } }
    public bool IsLeaf { get { return Children.Count==0; } }
    /// <summary>
    /// Returns the number of hops to the root object
    /// </summary>
    public int Level { get { return IsRoot ? 0 : Parent.Level+1; } }
}
```

. .

```
public class MyNode : Tree<MyNode>
{
    // stuff
}
```

, . Control XmlElement Tree<T> .

, .

```
public class Part : Tree<Part>
{
    public static readonly Part Empty = new Part(null) { Weight=0 };
    public Part(Part parent) : base(parent) { }
    public Part Add(float weight)
    {
        return new Part(this) { Weight=weight };
    }
    public float Weight { get; set; }

    public float TotalWeight { get { return Weight+Children.Sum((part) => part.TotalWeight); }
}
}
```

.

```
// [Q:2.5] -- [P:4.2] -- [R:0.4]
//    \
//      - [Z:0.8]
var Q = Part.Empty.Add(2.5f);
var P = Q.Add(4.2f);
var R = P.Add(0.4f);
var Z = Q.Add(0.9f);
```

```
// 2.5+(4.2+0.4)+0.9 = 8.0
float weight = Q.TotalWeight;
```

. .

```
public class RelativeCoordinate : Tree<RelativeCoordinate>
{
    public static readonly RelativeCoordinate Start = new RelativeCoordinate(null,
PointF.Empty) { };
    public RelativeCoordinate(RelativeCoordinate parent, PointF local_position)
        : base(parent)
    {
        this.LocalPosition=local_position;
    }
    public PointF LocalPosition { get; set; }
    public PointF GlobalPosition
    {
        get
        {
            if(IsRoot) return LocalPosition;
            var parent_pos = Parent.GlobalPosition;
            return new PointF(parent_pos.X+LocalPosition.X, parent_pos.Y+LocalPosition.Y);
        }
    }
    public float TotalDistance
    {
        get
        {
            float dist =
(float)Math.Sqrt(LocalPosition.X*LocalPosition.X+LocalPosition.Y*LocalPosition.Y);
            return IsRoot ? dist : Parent.TotalDistance+dist;
        }
    }
    public RelativeCoordinate Add(PointF local_position)
    {
        return new RelativeCoordinate(this, local_position);
    }
    public RelativeCoordinate Add(float x, float y)
    {
        return Add(new PointF(x, y));
    }
}
```

.

```
// Define the following coordinate system hierarchy
//
// o--> [A1] --+--> [B1] -----> [C1]
//             |
//             +--> [B2] --+--> [C2]
//                         |
//                         +--> [C3]

var A1 = RelativeCoordinate.Start;
var B1 = A1.Add(100, 20);
var B2 = A1.Add(160, 10);
```

```
var C1 = B1.Add(120, -40);
var C2 = B2.Add(80, -20);
var C3 = B2.Add(60, -30);

double dist1 = C1.TotalDistance;
```

: https://riptutorial.com/ko/csharp/topic/29/

60:

Examples

. , .int 2147483647

```
int x = int.MaxValue;                //MaxValue is 2147483647
x = unchecked(x + 1);                //make operation explicitly unchecked so that the example
also works when the check for arithmetic overflow/underflow is enabled in the project settings

Console.WriteLine(x);                //Will print -2147483648
Console.WriteLine(int.MinValue);     //Same as Min value
```

BigInteger System.Numerics . . https://msdn.microsoft.com/en-us/library/system.numerics.biginteger(v=vs.110).aspx

. x int 1 int . int . int . .

```
int x = int.MaxValue;                //MaxValue is 2147483647
long y = x + 1;                      //It will be overflown
Console.WriteLine(y);                //Will print -2147483648
Console.WriteLine(int.MinValue);     //Same as Min value
```

1L . 1 long long .

```
int x = int.MaxValue;                //MaxValue is 2147483647
long y = x + 1L;                     //It will be OK
Console.WriteLine(y);                //Will print 2147483648
```

.

```
int x = int.MaxValue;
Console.WriteLine(x + x + 1L);  //prints -1
```

.

```
int x = int.MaxValue;
Console.WriteLine(x + 1L + x);  //prints 4294967295
```

. x + x long . x + 1L long x .

: https://riptutorial.com/ko/csharp/topic/3303/

61:

C # 7.5.3 . 7.5.2 ( ) 7.6.5 ( ) .

C # 7 . Microsoft ( ) .

Examples

Hello .

```
class Example
{
    public static void Hello(int arg)
    {
        Console.WriteLine("int");
    }

    public static void Hello(double arg)
    {
        Console.WriteLine("double");
    }

    public static void Main(string[] args)
    {
        Hello(0);
        Hello(0.0);
    }
}
```

Main .

```
int
double
```

```
Hello(0)   Hello   .   .    .         .

Hello(0)  Hello(int)    Hello(double)    .    .

Hello(0.0)  0.0  int     Hello(int)    .    .
```

"params" .

:

```
class Program
{
    static void Method(params Object[] objects)
    {
        System.Console.WriteLine(objects.Length);
    }
    static void Method(Object a, Object b)
    {
```

```
        System.Console.WriteLine("two");
    }
    static void Main(string[] args)
    {
        object[] objectArray = new object[5];

        Method(objectArray);
        Method(objectArray, objectArray);
        Method(objectArray, objectArray, objectArray);
    }
}
```

:

```
5
two
3
```

Method(objectArray) . Object .(1 . Method params , , 5 .

Method(objectArray, objectArray) . two .

Method(objectArray, objectArray, objectArray) 3 .

null

```
void F1(MyType1 x) {
    // do something
}

void F1(MyType2 x) {
    // do something else
}
```

F1 x = null

```
F1(null);
```

. .

```
F1(null as MyType1);
```

: https://riptutorial.com/ko/csharp/topic/77/-

62:

. 7 . : , , , , ,

Examples

"" .

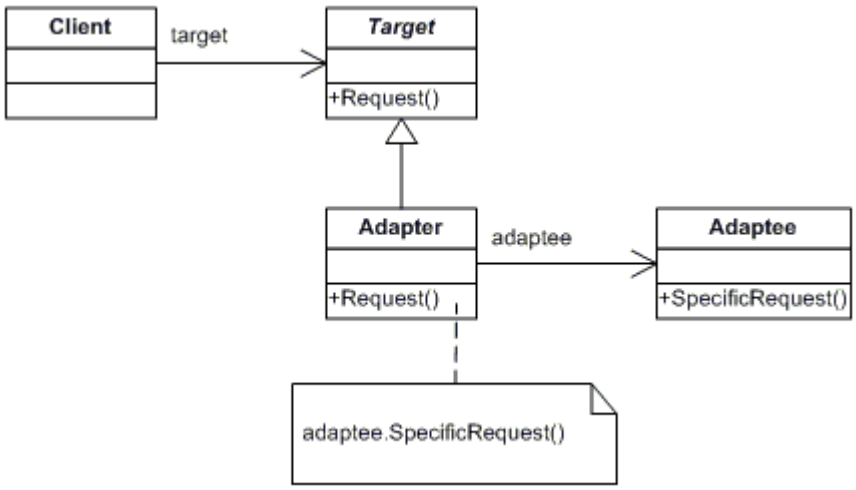
: 8 ( ) . jac USB . . " ".

: . . .

4 .

1. **ITarget :** .
2. **Adaptee :** .
3. **:** .
4. **:** ITarget Adaptee .

UML



( ) .

```
public interface ITarget
{
    void MethodA();
}

public class Adaptee
{
    public void MethodB()
    {
        Console.WriteLine("MethodB() is called");
    }
}

public class Client
```

```
{
    private ITarget target;

    public Client(ITarget target)
    {
        this.target = target;
    }

    public void MakeRequest()
    {
        target.MethodA();
    }
}

public class Adapter : Adaptee, ITarget
{
    public void MethodA()
    {
        MethodB();
    }
}
```

( )

```
/// <summary>
///  Interface: This is the interface which is used by the client to achieve functionality.
/// </summary>
public interface ITarget
{
    List<string> GetEmployeeList();
}

/// <summary>
/// Adaptee: This is the functionality which the client desires but its interface is not
compatible with the client.
/// </summary>
public class CompanyEmplyees
{
    public string[][] GetEmployees()
    {
        string[][] employees = new string[4][];

        employees[0] = new string[] { "100", "Deepak", "Team Leader" };
        employees[1] = new string[] { "101", "Rohit", "Developer" };
        employees[2] = new string[] { "102", "Gautam", "Developer" };
        employees[3] = new string[] { "103", "Dev", "Tester" };

        return employees;
    }
}

/// <summary>
/// Client: This is the class which wants to achieve some functionality by using the adaptee’s
code (list of employees).
/// </summary>
public class ThirdPartyBillingSystem
{
    /*
     * This class is from a thirt party and you do'n have any control over it.
     * But it requires a Emplyee list to do its work
```

```
     */

    private ITarget employeeSource;

    public ThirdPartyBillingSystem(ITarget employeeSource)
    {
        this.employeeSource = employeeSource;
    }

    public void ShowEmployeeList()
    {
        // call the clietn list in the interface
        List<string> employee = employeeSource.GetEmployeeList();

        Console.WriteLine("######### Employee List ##########");
        foreach (var item in employee)
        {
            Console.Write(item);
        }

    }
}

/// <summary>
/// Adapter: This is the class which would implement ITarget and would call the Adaptee code
which the client wants to call.
/// </summary>
public class EmployeeAdapter : CompanyEmplyees, ITarget
{
    public List<string> GetEmployeeList()
    {
        List<string> employeeList = new List<string>();
        string[][] employees = GetEmployees();
        foreach (string[] employee in employees)
        {
            employeeList.Add(employee[0]);
            employeeList.Add(",");
            employeeList.Add(employee[1]);
            employeeList.Add(",");
            employeeList.Add(employee[2]);
            employeeList.Add("\n");
        }

        return employeeList;
    }
}

///
/// Demo
///
class Programs
{
    static void Main(string[] args)
    {
        ITarget Itarget = new EmployeeAdapter();
        ThirdPartyBillingSystem client = new ThirdPartyBillingSystem(Itarget);
        client.ShowEmployeeList();
        Console.ReadKey();
    }
}
```

?

- .
- .
- Ado.Net SqlAdapter, OracleAdapter, MySqlAdapter .

: https://riptutorial.com/ko/csharp/topic/9764/--

63:

struct . , struct . struct GC struct .

struct s protected . struct .

Examples

```
public struct Vector
{
    public int X;
    public int Y;
    public int Z;
}

public struct Point
{
    public decimal x, y;

    public Point(decimal pointX, decimal pointY)
    {
        x = pointX;
        y = pointY;
    }
}
```

- struct struct .
- private .
- struct System.ValueType .
- Structs .
- . , .
- nullable null .

```
Vector v1 = null; //illegal
Vector? v2 = null; //OK
Nullable<Vector> v3 = null // OK
```

- new .

```
//Both of these are acceptable
Vector v1 = new Vector();
v1.X = 1;
v1.Y = 2;
v1.Z = 3;

Vector v2;
v2.X = 1;
```

```
v2.Y = 2;
v2.Z = 3;
```

`new` .

```
Vector v1 = new MyStruct { X=1, Y=2, Z=3 }; // OK
Vector v2 { X=1, Y=2, Z=3 }; // illegal
```

.

- . `struct` `struct` . private .
- protected .
- const static .

:

```
Vector v1 = new Vector();
v1.X = 1;
v1.Y = 2;
v1.Z = 3;

Console.WriteLine("X = {0}, Y = {1}, Z = {2}",v1.X,v1.Y,v1.Z);
// Output X=1,Y=2,Z=3

Vector v1 = new Vector();
//v1.X is not assigned
v1.Y = 2;
v1.Z = 3;

Console.WriteLine("X = {0}, Y = {1}, Z = {2}",v1.X,v1.Y,v1.Z);
// Output X=0,Y=2,Z=3

Point point1 = new Point();
point1.x = 0.5;
point1.y = 0.6;

Point point2 = new Point(0.5, 0.6);
```

:

```
Vector v1;
v1.Y = 2;
v1.Z = 3;

Console.WriteLine("X = {0}, Y = {1}, Z = {2}",v1.X,v1.Y,v1.Z);
//Output ERROR "Use of possibly unassigned field 'X'

Vector v1;
v1.X = 1;
v1.Y = 2;
v1.Z = 3;

Console.WriteLine("X = {0}, Y = {1}, Z = {2}",v1.X,v1.Y,v1.Z);
// Output X=1,Y=2,Z=3

Point point3;
```

```
point3.x = 0.5;
point3.y = 0.6;
```

(null).

. , new . struct null .

```
public interface IShape
{
    decimal Area();
}

public struct Rectangle : IShape
{
    public decimal Length { get; set; }
    public decimal Width { get; set; }

    public decimal Area()
    {
        return Length * Width;
    }
}
```

.

. p1 p2 p1 p2 .

```
var p1 = new Point {
    x = 1,
    y = 2
};

Console.WriteLine($"{p1.x} {p1.y}"); // 1 2

var p2 = p1;
Console.WriteLine($"{p2.x} {p2.y}"); // Same output: 1 2

p1.x = 3;
Console.WriteLine($"{p1.x} {p1.y}"); // 3 2
Console.WriteLine($"{p2.x} {p2.y}"); // p2 remain the same: 1 2
```

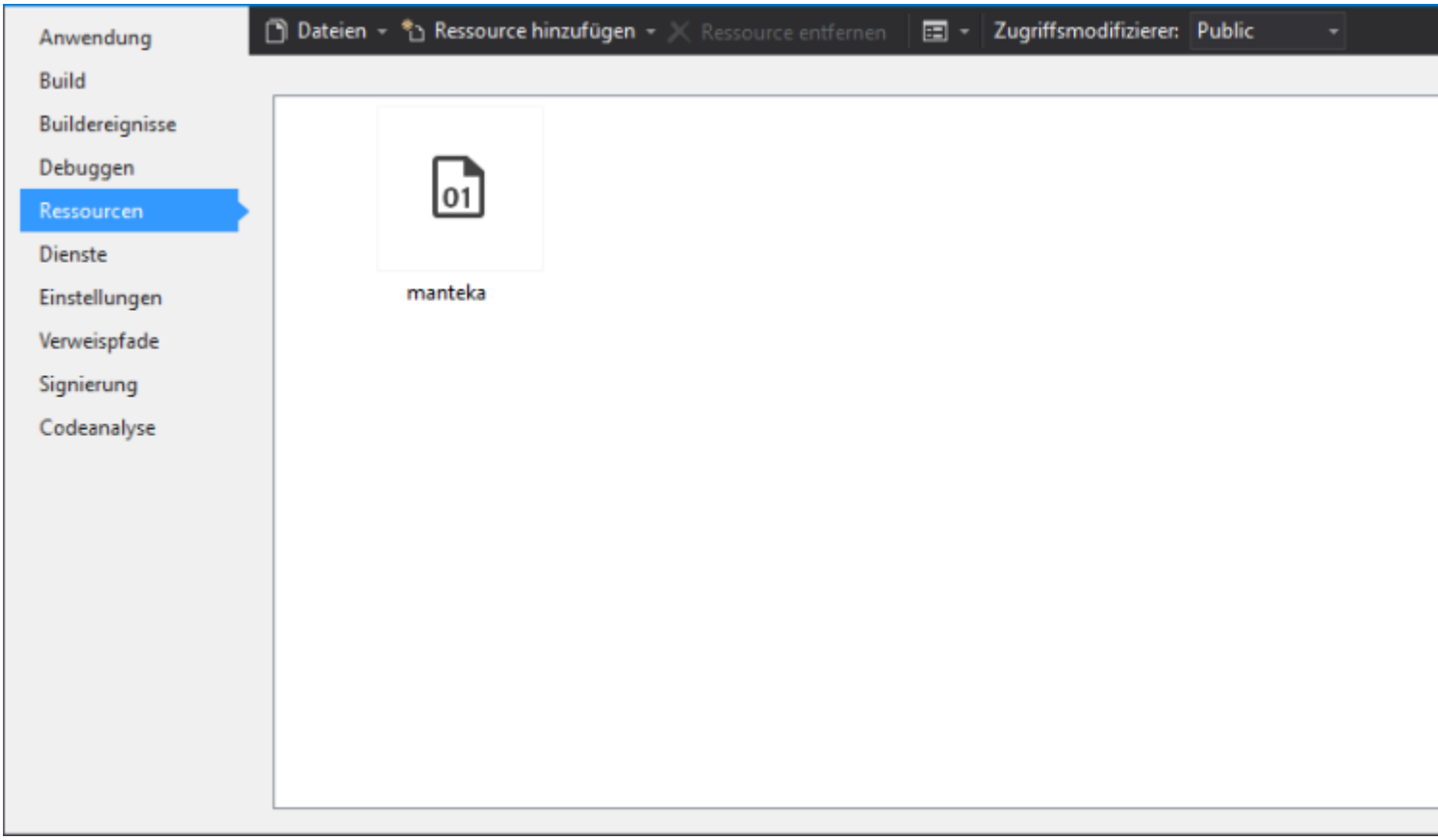
: https://riptutorial.com/ko/csharp/topic/778/

64:

Examples

'Fontfamily'

```
public FontFamily Maneteke = GetResourceFontFamily(Properties.Resources.manteka);
```




```
public static FontFamily GetResourceFontFamily(byte[] fontbytes)
{
    PrivateFontCollection pfc = new PrivateFontCollection();
    IntPtr fontMemPointer = Marshal.AllocCoTaskMem(fontbytes.Length);
    Marshal.Copy(fontbytes, 0, fontMemPointer, fontbytes.Length);
    pfc.AddMemoryFont(fontMemPointer, fontbytes.Length);
    Marshal.FreeCoTaskMem(fontMemPointer);
    return pfc.Families[0];
}
```

"

```
    public static class Res
    {
```

```
        /// <summary>
        /// URL: https://www.behance.net/gallery/2846011/Manteka
        /// </summary>
        public static FontFamily Maneteke =
GetResourceFontFamily(Properties.Resources.manteka);

        public static FontFamily GetResourceFontFamily(byte[] fontbytes)
        {
            PrivateFontCollection pfc = new PrivateFontCollection();
            IntPtr fontMemPointer = Marshal.AllocCoTaskMem(fontbytes.Length);
            Marshal.Copy(fontbytes, 0, fontMemPointer, fontbytes.Length);
            pfc.AddMemoryFont(fontMemPointer, fontbytes.Length);
            Marshal.FreeCoTaskMem(fontMemPointer);
            return pfc.Families[0];
        }
    }

    public class FlatButton : Button
    {
        public FlatButton() : base()
        {
            Font = new Font(Res.Maneteke, Font.Size);
        }

        protected override void OnFontChanged(EventArgs e)
        {
            base.OnFontChanged(e);
            this.Font = new Font(Res.Maneteke, this.Font.Size);
        }
    }
```

: https://riptutorial.com/ko/csharp/topic/9789/--

65:

Examples

Func . .

```
// square a number.
Func<double, double> square = (x) => { return x * x; };

// get the square root.
// note how the signature matches the built in method.
Func<double, double> squareroot = Math.Sqrt;

// provide your workings.
Func<double, double, string> workings = (x, y) =>
    string.Format("The square of {0} is {1}.", x, square(y))
```

void . .

```
// right-angled triangle.
class Triangle
{
    public double a;
    public double b;
    public double h;
}

// Pythagorean theorem.
Action<Triangle> pythagoras = (x) =>
    x.h = squareroot(square(x.a) + square(x.b));

Triangle t = new Triangle { a = 3, b = 4 };
pythagoras(t);
Console.WriteLine(t.h); // 5.
```

.

.

```
public class Address ()
{
    public string Line1 { get; set; }
    public string Line2 { get; set; }
    public string City  { get; set; }
}
```

.

.

```
public class Address ()
{
```

```
    public readonly string Line1;
    public readonly string Line2;
    public readonly string City;

    public Address(string line1, string line2, string city)
    {
        Line1 = line1;
        Line2 = line2;
        City  = city;
    }
}
```

. ,

```
public class Classroom
{
    public readonly List<Student> Students;

    public Classroom(List<Student> students)
    {
        Students = students;
    }
}
```

( ). IEnumerable . IEnumerable ReadOnlyCollection .

```
public class Classroom
{
    public readonly ReadOnlyCollection<Student> Students;

    public Classroom(ReadOnlyCollection<Student> students)
    {
        Students = students;
    }
}

List<Students> list = new List<Student>();
// add students
Classroom c = new Classroom(list.AsReadOnly());
```

.

- ( ).
- .
- .
- .

Null

C # null . F # Option . Option <> (<>) Some None . (NULL) .

, .

```
var user = _repository.GetUser(id);
```

null .

```
var username = user != null ? user.Name : string.Empty;
```

Option <> ?

```
Option<User> maybeUser = _repository.GetUser(id);
```

None Some None .

```
var username = maybeUser.HasValue ? maybeUser.Value.Name : string.Empty;
```

Option <> .

```
public Option<User> GetUser(int id)
{
    var users = new List<User>
    {
        new User { Id = 1, Name = "Joe Bloggs" },
        new User { Id = 2, Name = "John Smith" }
    };

    var user = users.FirstOrDefault(user => user.Id == id);

    return user != null ? new Option<User>(user) : new Option<User>();
}
```

Option <> .

```
public struct Option<T>
{
    private readonly T _value;

    public T Value
    {
        get
        {
            if (!HasValue)
                throw new InvalidOperationException();

            return _value;
        }
    }

    public bool HasValue
    {
        get { return _value != null; }
    }

    public Option(T value)
    {
        _value = value;
    }

    public static implicit operator Option<T>(T value)
```

```
    {
        return new Option<T>(value);
    }
}
```

avoidNull.csx C # REPL .

, . ""NuGet .

.

LINQ Where lambda .

```
var results = data.Where(p => p.Items == 0);
```

Where () .

. , Sorting Sort .

System.Collections.Immutable NuGet .

```
var stack = ImmutableStack.Create<int>();
var stack2 = stack.Push(1); // stack is still empty, stack2 contains 1
var stack3 = stack.Push(2); // stack2 still contains only one, stack3 has 2, 1
```

Builder .

```
var builder = ImmutableList.CreateBuilder<int>(); // returns ImmutableList.Builder
builder.Add(1);
builder.Add(2);
var list = builder.ToImmutable();
```

IEnumerable

```
var numbers = Enumerable.Range(1, 5);
var list = ImmutableList.CreateRange<int>(numbers);
```

:

- System.Collections.Immutable.ImmutableArray<T>
- System.Collections.Immutable.ImmutableDictionary<TKey,TValue>
- System.Collections.Immutable.ImmutableHashSet<T>
- System.Collections.Immutable.ImmutableList<T>
- System.Collections.Immutable.ImmutableQueue<T>
- System.Collections.Immutable.ImmutableSortedDictionary<TKey,TValue>
- System.Collections.Immutable.ImmutableSortedSet<T>
- System.Collections.Immutable.ImmutableStack<T>

: https://riptutorial.com/ko/csharp/topic/2564/-

66:

Examples

C# .

C #	.NET Framework
	System.Boolean
	System.Byte
	System.SByte
	System.Char
	System.Decimal
	System.Double
	System.Single
int	System.Int32
uint	System.UInt32
	System.Int64
ulong	System.UInt64
	System.Object
	System.Int16
	System.UInt16
	System.String

C# . , .

```
int number = 123;
System.Int32 number = 123;
```

: https://riptutorial.com/ko/csharp/topic/1862/--

67:

Examples

-

```
// assign string from a string literal
string s = "hello";

// assign string from an array of characters
char[] chars = new char[] { 'h', 'e', 'l', 'l', 'o' };
string s = new string(chars, 0, chars.Length);

// assign string from a char pointer, derived from a string
string s;
unsafe
{
    fixed (char* charPointer = "hello")
    {
        s = new string(charPointer);
    }
}
```

- char

```
// single character s
char c = 's';

// character s: casted from integer value
char c = (char)115;

// unicode character: single character s
char c = '\u0073';

// unicode character: smiley face
char c = '\u263a';
```

- short, int, long (16 , 32 , 64)

```
// assigning a signed short to its minimum value
short s = -32768;

// assigning a signed short to its maximum value
short s = 32767;

// assigning a signed int to its minimum value
int i = -2147483648;

// assigning a signed int to its maximum value
int i = 2147483647;

// assigning a signed long to its minimum value (note the long postfix)
long l = -9223372036854775808L;
```

```
// assigning a signed long to its maximum value (note the long postfix)
long l = 92233720368547758O7L;
```

nullable . , null . nullable 0 null. Null (?) .

```
int a; //This is now 0.
int? b; //This is now null.
```

- ushort, uint, ulong (16 , 32 , 64)

```
// assigning an unsigned short to its minimum value
ushort s = 0;

// assigning an unsigned short to its maximum value
ushort s = 65535;

// assigning an unsigned int to its minimum value
uint i = 0;

// assigning an unsigned int to its maximum value
uint i = 4294967295;

// assigning an unsigned long to its minimum value (note the unsigned long postfix)
ulong l = 0UL;

// assigning an unsigned long to its maximum value (note the unsigned long postfix)
ulong l = 18446744073709551615UL;
```

nullable . , null . nullable 0 null. Null (?) .

```
uint a; //This is now 0.
uint? b; //This is now null.
```

- bool

```
// default value of boolean is false
bool b;
//default value of nullable boolean is null
bool? z;
b = true;
if(b) {
    Console.WriteLine("Boolean has true value");
}
```

bool System.Boolean . (`true` `false` .

`object` . `System.Object` . `==` . :

```
object left = (int)1;  // int in an object box
object right = (int)1; // int in an object box

var comparison1 = left == right;      // false
```

Equals .

```
var comparison2 = left.Equals(right); // true
```

left right int .

```
var comparison3 = (int)left == (int)right; // true
```

Type Type unbox . :

```
object boxedInt = (int)1; // int boxed in an object

long unboxedInt1 = (long)boxedInt; // invalid cast
```

Type unboxing . :

```
long unboxedInt2 = (long)(int)boxedInt; // valid
```

: https://riptutorial.com/ko/csharp/topic/42/

68:

- X? Y; // X null null. else XY
- X? Y? Z; // X null Y null null else XYZ
- X? [index]; // X null null. else X [index]
- X? .ValueMethod (); // X null null X.ValueMethod () ;
- X? .VoidMethod (); // X null . else call X.VoidMethod ();

`T  Nullable<T>` .

Examples

Null

`?.` null . .

.

```
public class Person
{
    public int Age { get; set; }
    public string Name { get; set; }
    public Person Spouse { get; set; }
}
```

null ( ), `NullReferenceException` , null . null .

```
Person person = GetPerson();

int? age = null;
if (person != null)
    age = person.Age;
```

null .

```
Person person = GetPerson();

var age = person?.Age;    // 'age' will be of type 'int?', even if 'person' is not null
```

---

null .

```
// Will be null if either `person` or `person.Spouse` are null
int? spouseAge = person?.Spouse?.Age;
```

Null

.

```
// spouseDisplayName will be "N/A" if person, Spouse, or Name is null
var spouseDisplayName = person?.Spouse?.Name ?? "N/A";
```

Null

the `?.` `?.` , .

```
string item = collection?[index];
```

.

```
string item = null;
if(collection != null)
{
    item = collection[index];
}
```

NullReferenceExceptions

```
var person = new Person
{
    Address = null;
};

var city = person.Address.City; //throws a NullReferenceException
var nullableCity = person.Address?.City; //returns the value of null
```

.

```
var person = new Person
{
    Address = new Address
    {
        State = new State
        {
            Country = null
        }
    }
};

// this will always return a value of at least "null" to be stored instead
// of throwing a NullReferenceException
var countryName = person?.Address?.State?.Country?.Name;
```

Null .

null `?.` `?.` .

```
public class Person
{
```

```
    public string Name {get; set;}
}

public static class PersonExtensions
{
    public static int GetNameLength(this Person person)
    {
        return person == null ? -1 : person.Name.Length;
    }
}
```

`null` -1 .

```
Person person = null;
int nameLength = person.GetNameLength(); // returns -1
```

`?.` `?.` `null` , `int?` :

```
Person person = null;
int? nameLength = person?.GetNameLength(); // nameLength is null.
```

`?.` operator works : `NullReferenceExceptions` . .

- null .

: https://riptutorial.com/ko/csharp/topic/41/--

69:

- TcpClient (, int);

`client.GetStream()` `TcpClient` `NetworkStream` `client.GetStream()` `StreamReader/StreamWriter` .

Examples

TCP

TCP "Hello World" .

```
// Declare Variables
string host = "stackoverflow.com";
int port = 9999;
int timeout = 5000;

// Create TCP client and connect
using (var _client = new TcpClient(host, port))
using (var _netStream = _client.GetStream())
{
    _netStream.ReadTimeout = timeout;

    // Write a message over the socket
    string message = "Hello World!";
    byte[] dataToSend = System.Text.Encoding.ASCII.GetBytes(message);
    _netStream.Write(dataToSend, 0, dataToSend.Length);

    // Read server response
    byte[] recvData = new byte[256];
    int bytes = _netStream.Read(recvData, 0, recvData.Length);
    message = System.Text.Encoding.ASCII.GetString(recvData, 0, bytes);
    Console.WriteLine(string.Format("Server: {0}", message));
};// The client and stream will close as control exits the using block (Equivilent but safer
than calling Close();
```

.

" System.Net.WebClient " .

"using" .

```
using (var webClient = new WebClient())
{
    webClient.DownloadFile("http://www.server.com/file.txt", "C:\\file.txt");
}
```

"" URL .

" System.Uri " " System.String ".

. .

.

```
var webClient = new WebClient())
webClient.DownloadFileCompleted += new AsyncCompletedEventHandler(Completed);
webClient.DownloadProgressChanged += new DownloadProgressChangedEventHandler(ProgressChanged);
webClient.DownloadFileAsync("http://www.server.com/file.txt", "C:\\file.txt");
```

"'' .

. . using , .

" .

TCP

C # `async/await` . TcpClient `async/await` .

```
// Declare Variables
string host = "stackoverflow.com";
int port = 9999;
int timeout = 5000;

// Create TCP client and connect
// Then get the netstream and pass it
// To our StreamWriter and StreamReader
using (var client = new TcpClient())
using (var netstream = client.GetStream())
using (var writer = new StreamWriter(netstream))
using (var reader = new StreamReader(netstream))
{
    // Asynchronsly attempt to connect to server
    await client.ConnectAsync(host, port);

    // AutoFlush the StreamWriter
    // so we don't go over the buffer
    writer.AutoFlush = true;

    // Optionally set a timeout
    netstream.ReadTimeout = timeout;

    // Write a message over the TCP Connection
    string message = "Hello World!";
    await writer.WriteLineAsync(message);

    // Read server response
    string response = await reader.ReadLineAsync();
    Console.WriteLine(string.Format($"Server: {response}"));
}
// The client and stream will close as control exits
// the using block (Equivilent but safer than calling Close();
```

UDP

UDP "Hello World" . UDP . .

```
byte[] data = Encoding.ASCII.GetBytes("Hello World");
string ipAddress = "192.168.1.141";
string sendPort = 55600;
try
{
     using (var client = new UdpClient())
     {
         IPEndPoint ep = new IPEndPoint(IPAddress.Parse(ipAddress), sendPort);
         client.Connect(ep);
         client.Send(data, data.Length);
     }
}
catch (Exception ex)
{
     Console.WriteLine(ex.ToString());
}
```

UDP . . 'done' .

```
bool done = false;
int listenPort = 55600;
using(UdpClinet listener = new UdpClient(listenPort))
{
    IPEndPoint listenEndPoint = new IPEndPoint(IPAddress.Any, listenPort);
    while(!done)
    {
        byte[] receivedData = listener.Receive(ref listenPort);

        Console.WriteLine("Received broadcast message from client {0}",
listenEndPoint.ToString());

        Console.WriteLine("Decoded data is:");
        Console.WriteLine(Encoding.ASCII.GetString(receivedData)); //should be "Hello World"
sent from above client
    }
}
```

: https://riptutorial.com/ko/csharp/topic/1352/

70:

Examples

OOP . ''.

. Vehicle .

Derived Ducati Lamborghini Vehicle Display() NumberOfWheels .

```
public class Vehicle
{
    protected int NumberOfWheels { get; set; } = 0;
    public Vehicle()
    {
    }

    public virtual void Display()
    {
        Console.WriteLine($"The number of wheels for the {nameof(Vehicle)} is
{NumberOfWheels}");
    }
}

public class Ducati : Vehicle
{
    public Ducati()
    {
        NoOfWheels = 2;
    }

    public override void Display()
    {
        Console.WriteLine($"The number of wheels for the {nameof(Ducati)} is
{NumberOfWheels}");
    }
}

public class Lamborghini : Vehicle
{
    public Lamborghini()
    {
        NoOfWheels = 4;
    }

    public override void Display()
    {
        Console.WriteLine($"The number of wheels for the {nameof(Lamborghini)} is
{NumberOfWheels}");
    }
}
```

. Vehicle vehicle 1 Display() 2 .

```
 static void Main(string[] args)
```

```
 {
    Vehicle vehicle = new Vehicle();    //Line 1
    vehicle.Display();                  //Line 2
    vehicle = new Ducati();             //Line 3
    vehicle.Display();                  //Line 4
    vehicle = new Lamborghini();        //Line 5
    vehicle.Display();                  //Line 6
 }
```

Line 3 `vehicle Ducati Display()` . . (polymorphic) `vehicle Vehicle , Display() Ducati Display() , vehicle Ducati .`

`Lamborghini Display()` .

.

```
The number of wheels for the Vehicle is 0        // Line 2
The number of wheels for the Ducati is 2         // Line 4
The number of wheels for the Lamborghini is 4    // Line 6
```

.

.

- :

  `function overloading function overloading .` .
- :

  .
- :

  .

`Ad hoc polymorphism` . `sumInt(par1, par2)` .

```
public static int sumInt( int a, int b)
{
    return a + b;
}

public static int sumInt( string a, string b)
{
    int _a, _b;

    if(!Int32.TryParse(a, out _a))
        _a = 0;

    if(!Int32.TryParse(b, out _b))
        _b = 0;

    return _a + _b;
}

public static int sumInt(string a, int b)
```

```
{
    int _a;

    if(!Int32.TryParse(a, out _a))
        _a = 0;

    return _a + b;
}

public static int sumInt(int a, string b)
{
    return sumInt(b,a);
}
```

.

```
public static void Main()
{
    Console.WriteLine(sumInt( 1 , 2 ));  //  3
    Console.WriteLine(sumInt("3","4"));  //  7
    Console.WriteLine(sumInt("5", 6 ));  // 11
    Console.WriteLine(sumInt( 7 ,"8"));  // 15
}
```

.

```
public interface Car{
    void refuel();
}

public class NormalCar : Car
{
    public void refuel()
    {
        Console.WriteLine("Refueling with petrol");
    }
}

public class ElectricCar : Car
{
    public void refuel()
    {
        Console.WriteLine("Charging battery");
    }
}
```

NormalCar ElectricCar . .

```
public static void Main()
{
    List<Car> cars = new List<Car>(){
        new NormalCar(),
        new ElectricCar()
    };
```

```
    cars.ForEach(x => x.refuel());
}
```

.

: https://riptutorial.com/ko/csharp/topic/1589/

71:

. . .

: Action<...> , Predicate<T> Func<...,TResult>

System Action<...> , Predicate<T> Func<...,TResult> () "..."0 16 (0 , Action -).

Func TResult Action (void) . , .

Predicate boolean , T .

delegate .

. .

.

-
-
- delegate

+ . - .

Examples

.

- .
- .

```
public class Greeter
{
    public void WriteInstance()
    {
        Console.WriteLine("Instance");
    }

    public static void WriteStatic()
    {
        Console.WriteLine("Static");
```

```
    }
}

// ...

Greeter greeter1 = new Greeter();
Greeter greeter2 = new Greeter();

Action instance1 = greeter1.WriteInstance;
Action instance2 = greeter2.WriteInstance;
Action instance1Again = greeter1.WriteInstance;

Console.WriteLine(instance1.Equals(instance2)); // False
Console.WriteLine(instance1.Equals(instance1Again)); // True

Action @static = Greeter.WriteStatic;
Action staticAgain = Greeter.WriteStatic;

Console.WriteLine(@static.Equals(staticAgain)); // True
```

delegate NumberInOutDelegate , int int .

```
public delegate int NumberInOutDelegate(int input);
```

.

```
public static class Program
{
    static void Main()
    {
        NumberInOutDelegate square = MathDelegates.Square;
        int answer1 = square(4);
        Console.WriteLine(answer1); // Will output 16

        NumberInOutDelegate cube = MathDelegates.Cube;
        int answer2 = cube(4);
        Console.WriteLine(answer2); // Will output 64
    }
}

public static class MathDelegates
{
    static int Square (int x)
    {
        return x*x;
    }

    static int Cube (int x)
    {
        return x*x*x;
    }
}
```

example Square . . . .

. (out) (in) . :

```
public delegate TTo Converter<in TFrom, out TTo>(TFrom input);
```

`where TFrom : struct, IConvertible where TTo : new()` .

contravariance . (+) . .

```
public delegate void EventHandler<in TEventArgs>(object sender, TEventArgs e);
```

.

```
public delegate void EventHandler<TEventArgs>(object sender, TEventArgs e);
```

`ref out` .

```
public delegate bool TryParser<T>(string input, out T result);
```

(`TryParser<decimal> example = decimal.TryParse;`), `params` . ( ). `int* char*` (`unsafe`). .

,

`Func<..., TResult>` 0 15 `TResult` .

```
private void UseFunc(Func<string> func)
{
    string output = func(); // Func with a single generic type parameter returns that type
    Console.WriteLine(output);
}

private void UseFunc(Func<int, int, string> func)
{
    string output = func(4, 2); // Func with multiple generic type parameters takes all but
the first as parameters of that type
    Console.WriteLine(output);
}
```

System (0 16) `Action<...>` . `Func<T1, .., Tn>` `void` .

```
private void UseAction(Action action)
{
    action(); // The non-generic Action has no parameters
}

private void UseAction(Action<int, string> action)
{
    action(4, "two"); // The generic action is invoked with parameters matching its type
arguments
}
```

`Predicate<T>` `Func` `bool` . . `true` `false` . `Predicate<T>` `Func<T, bool>` .

```
Predicate<string> predicate = s => s.StartsWith("a");
Func<string, bool> func = s => s.StartsWith("a");

// Both of these return true
var predicateReturnsTrue = predicate("abc");
var funcReturnsTrue = func("abc");

// Both of these return false
var predicateReturnsFalse = predicate("xyz");
var funcReturnsFalse = func("xyz");
```

`Predicate<T>` `Func<T, bool>` . `Predicate<T>` , `Func<T, bool>` C # .

API . `List<T>` `Array<T>` `Predicate<T>` LINQ `Func<T, bool>` .

.

```
public static class Example
{
    public static int AddOne(int input)
    {
        return input + 1;
    }
}


Func<int,int> addOne = Example.AddOne
```

`Example.AddOne` `int` `int` `Func<int,int>` . `Example.AddOne` `addOne` .

`.Equals()` `.Equals()` .

```
Action action1 = () => Console.WriteLine("Hello delegates");
Action action2 = () => Console.WriteLine("Hello delegates");
Action action1Again = action1;

Console.WriteLine(action1.Equals(action1)) // True
Console.WriteLine(action1.Equals(action2)) // False
Console.WriteLine(action1Again.Equals(action1)) // True
```

`+=` `-=` (:).

Lambda .

```
Func<int,int> addOne = x => x+1;
```

.

```
var addOne = x => x+1; // Does not work
```

.

```
class FuncAsParameters
```

```
{
  public void Run()
  {
    DoSomething(ErrorHandler1);
    DoSomething(ErrorHandler2);
  }

  public bool ErrorHandler1(string message)
  {
    Console.WriteLine(message);
    var shouldWeContinue = ...
    return shouldWeContinue;
  }

  public bool ErrorHandler2(string message)
  {
    // ...Write message to file...
    var shouldWeContinue = ...
    return shouldWeContinue;
  }

  public void DoSomething(Func<string, bool> errorHandler)
  {
    // In here, we don't care what handler we got passed!
    ...
    if (...error...)
    {
      if (!errorHandler("Some error occurred!"))
      {
        // The handler decided we can't continue
        return;
      }
    }
  }
}
```

( )

+ - . .

```
using System;
using System.Reflection;
using System.Reflection.Emit;

namespace DelegatesExample {
    class MainClass {
        private delegate void MyDelegate(int a);

        private static void PrintInt(int a) {
            Console.WriteLine(a);
        }

        private static void PrintType<T>(T a) {
            Console.WriteLine(a.GetType());
        }

        public static void Main (string[] args)
        {
            MyDelegate d1 = PrintInt;
```

```
            MyDelegate d2 = PrintType;

            // Output:
            // 1
            d1(1);

            // Output:
            // System.Int32
            d2(1);

            MyDelegate d3 = d1 + d2;
            // Output:
            // 1
            // System.Int32
            d3(1);

            MyDelegate d4 = d3 - d2;
            // Output:
            // 1
            d4(1);

            // Output:
            // True
            Console.WriteLine(d1 == d4);
        }
    }
}
```

d3 d1 d2 1 System.Int32 .

:

nonvoid . .

```
    class Program
    {
        public delegate int Transformer(int x);

        static void Main(string[] args)
        {
            Transformer t = Square;
            t += Cube;
            Console.WriteLine(t(2));  // O/P 8
        }

        static int Square(int x) { return x * x; }

        static int Cube(int x) { return x*x*x; }
    }
```

t(2) Square Cube . Square Cube .

. , , AggregateException :

```
public static class DelegateExtensions
{
```

```
    public static void SafeInvoke(this Delegate del,params object[] args)
    {
        var exceptions = new List<Exception>();

        foreach (var handler in del.GetInvocationList())
        {
            try
            {
                handler.Method.Invoke(handler.Target, args);
            }
            catch (Exception ex)
            {
                exceptions.Add(ex);
            }
        }

        if(exceptions.Any())
        {
            throw new AggregateException(exceptions);
        }
    }
}

public class Test
{
    public delegate void SampleDelegate();

    public void Run()
    {
        SampleDelegate delegateInstance = this.Target2;
        delegateInstance += this.Target1;

        try
        {
            delegateInstance.SafeInvoke();
        }
        catch(AggregateException ex)
        {
            // Do any exception handling here
        }
    }

    private void Target1()
    {
        Console.WriteLine("Target 1 executed");
    }

    private void Target2()
    {
        Console.WriteLine("Target 2 executed");
        throw new Exception();
    }
}
```

.

```
Target 2 executed
Target 1 executed
```

SaveInvoke

2 .

Parent .

, . , . . **- Jon Skeet**

```
delegate int testDel();
static void Main(string[] args)
{
    int foo = 4;
    testDel myClosure = delegate()
    {
        return foo;
    };
    int bar = myClosure();

}
```

.NET (Closures) .

```
public class MyObject{
    public DateTime? TestDate { get; set; }

    public Func<MyObject, bool> DateIsValid = myObject => myObject.TestDate.HasValue &&
myObject.TestDate > DateTime.Now;

    public void DoSomething(){
        //We can do this:
        if(this.TestDate.HasValue && this.TestDate > DateTime.Now){
            CallAnotherMethod();
        }

        //or this:
        if(DateIsValid(this)){
            CallAnotherMethod();
        }
    }
}
```

Func . DateTime ? Func . Func .

```
public void CheckForIntegrity(){
    if(ShipDateIsValid(this) && TestResultsHaveBeenIssued(this) && !TestResultsFail(this)){
        SendPassingTestNotification();
    }
}
```

: https://riptutorial.com/ko/csharp/topic/1194/

72:

Examples

DisplayNameAttribute ( )

`DisplayName` , (0) public void .

```
public class Employee
{
    [DisplayName(@"Employee first name")]
    public string FirstName { get; set; }
}
```

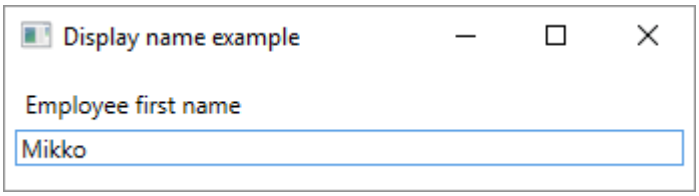
XAML

```
<Window x:Class="WpfApplication.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:wpfApplication="clr-namespace:WpfApplication"
        Height="100" Width="360" Title="Display name example">

    <Window.Resources>
        <wpfApplication:DisplayNameConverter x:Key="DisplayNameConverter"/>
    </Window.Resources>

    <StackPanel Margin="5">
        <!-- Label (DisplayName attribute) -->
        <Label Content="{Binding Employee, Converter={StaticResource DisplayNameConverter},
ConverterParameter=FirstName}" />
        <!-- TextBox (FirstName property value) -->
        <TextBox Text="{Binding Employee.FirstName, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}" />
    </StackPanel>

</Window>
```

```
namespace WpfApplication
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        private Employee _employee = new Employee();

        public MainWindow()
        {
            InitializeComponent();
            DataContext = this;
        }

        public Employee Employee
```

```
        {
            get { return _employee; }
            set { _employee = value; }
        }
    }
}
```

```
namespace WpfApplication
{
    public class DisplayNameConverter : IValueConverter
    {
        public object Convert(object value, Type targetType, object parameter, CultureInfo
culture)
        {
            // Get display name for given instance type and property name
            var attribute = value.GetType()
                .GetProperty(parameter.ToString())
                .GetCustomAttributes(false)
                .OfType<DisplayNameAttribute>()
                .FirstOrDefault();

            return attribute != null ? attribute.DisplayName : string.Empty;
        }

        public object ConvertBack(object value, Type targetType, object parameter, CultureInfo
culture)
        {
            throw new NotImplementedException();
        }
    }
}
```



EditableAttribute ( )

EditableAttribute .

```
public class Employee
{
    [Editable(false)]
    public string FirstName { get; set; }
}
```

XAML

```
<Window x:Class="WpfApplication.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:wpfApplication="clr-namespace:WpfApplication"
```

```
        Height="70" Width="360" Title="Display name example">

    <Window.Resources>
        <wpfApplication:EditableConverter x:Key="EditableConverter"/>
    </Window.Resources>

    <StackPanel Margin="5">
        <!-- TextBox Text (FirstName property value) -->
        <!-- TextBox IsEnabled (Editable attribute) -->
        <TextBox Text="{Binding Employee.FirstName, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
                 IsEnabled="{Binding Employee, Converter={StaticResource EditableConverter},
ConverterParameter=FirstName}"/>
    </StackPanel>

</Window>
```

```
namespace WpfApplication
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        private Employee _employee = new Employee() { FirstName = "This is not editable"};

        public MainWindow()
        {
            InitializeComponent();
            DataContext = this;
        }

        public Employee Employee
        {
            get { return _employee; }
            set { _employee = value; }
        }
    }
}
```

```
namespace WpfApplication
{
    public class EditableConverter : IValueConverter
    {
        public object Convert(object value, Type targetType, object parameter, CultureInfo
culture)
        {
            // return editable attribute's value for given instance property,
            // defaults to true if not found
            var attribute = value.GetType()
                .GetProperty(parameter.ToString())
                .GetCustomAttributes(false)
                .OfType<EditableAttribute>()
                .FirstOrDefault();

            return attribute != null ? attribute.AllowEdit : true;
        }
```

```
        public object ConvertBack(object value, Type targetType, object parameter, CultureInfo
culture)
        {
            throw new NotImplementedException();
        }
    }
}
```

. ValidationAttribute .

: RequiredAttribute

ValidationAttribute.Validate ValidationAttribute.Validate Name null .

```
public class ContactModel
{
    [Required(ErrorMessage = "Please provide a name.")]
    public string Name { get; set; }
}
```

: StringLengthAttribute

StringLengthAttribute . . .

```
public class ContactModel
{
    [StringLength(20, MinimumLength = 5, ErrorMessage = "A name must be between five and
twenty characters.")]
    public string Name { get; set; }
}
```

: RangeAttribute

RangeAttribute .

```
public class Model
{
    [Range(0.01, 100.00,ErrorMessage = "Price must be between 0.01 and 100.00")]
    public decimal Price { get; set; }
}
```

: CustomValidationAttribute

CustomValidationAttribute static . static ValidationResult [MethodName] (object input) .

```
public class Model
{
    [CustomValidation(typeof(MyCustomValidation), "IsNotAnApple")]
    public string FavoriteFruit { get; set; }
}
```

:

```
public static class MyCustomValidation
{
    public static ValidationResult IsNotAnApple(object input)
    {
        var result = ValidationResult.Success;

        if (input?.ToString()?.ToUpperInvariant() == "APPLE")
        {
            result = new ValidationResult("Apples are not allowed.");
        }

        return result;
    }
}
```

ValidationAttribute virtual .

```
[AttributeUsage(AttributeTargets.Property, AllowMultiple = false, Inherited = false)]
public class NotABananaAttribute : ValidationAttribute
{
    public override bool IsValid(object value)
    {
        var inputValue = value as string;
        var isValid = true;

        if (!string.IsNullOrEmpty(inputValue))
        {
            isValid = inputValue.ToUpperInvariant() != "BANANA";
        }

        return isValid;
    }
}
```

.

```
public class Model
{
    [NotABanana(ErrorMessage = "Bananas are not allowed.")]
    public string FavoriteFruit { get; set; }
}
```

. .

- : .
- :
- : .

ValidationAttribute DisplayAttribute .

```
class Kid
{
    [Range(0, 18)] // The age cannot be over 18 and cannot be negative
    public int Age { get; set; }
    [StringLength(MaximumLength = 50, MinimumLength = 3)] // The name cannot be under 3 chars
or more than 50 chars
    public string Name { get; set; }
    [DataType(DataType.Date)] // The birthday will be displayed as a date only (without the
time)
    public DateTime Birthday { get; set; }
}
```

ASP.NET . ASP.NET MVC ModelState.IsValid() ValidationAttribute ModelState.IsValid() . DisplayAttribute ASP.NET MVC .

ASP.NET . . ? Validator ( ).

. , , .

```
ValidationContext vc = new ValidationContext(objectToValidate); // The simplest form of
validation context. It contains only a reference to the object being validated.
```

.

```
ICollection<ValidationResult> results = new List<ValidationResult>(); // Will contain the
results of the validation
bool isValid = Validator.TryValidateObject(objectToValidate, vc, results, true); // Validates
the object and its properties using the previously created context.
// The variable isValid will be true if everything is valid
// The results variable contains the results of the validation
```

```
ICollection<ValidationResult> results = new List<ValidationResult>(); // Will contain the
results of the validation
bool isValid = Validator.TryValidatePropery(objectToValidate.PropertyToValidate, vc, results,
true); // Validates the property using the previously created context.
// The variable isValid will be true if everything is valid
// The results variable contains the results of the validation
```

.

- ValidationContext
- Validator

: https://riptutorial.com/ko/csharp/topic/4942/-

73:

Examples

ADO.NET

ADO.NET C # . .

.

- `System.Data.SqlClient` `SqlConnection` , `SqlCommand` , `SqlDataReader`
- `OleDbConnection` , `OleDbCommand` , `System.Data.OleDb` `OleDbDataReader`
- `MySqlConnection` , `MySqlCommand` , `MySqlDbDataReader` `MySql.Data`

C # . `FooConnection` , `FooCommand` , `FooDataReader` .

ADO.NET

ADO.NET .

```
// This scopes the connection (your specific class may vary)
using(var connection = new SqlConnection("{your-connection-string}")
{
    // Build your query
    var query = "SELECT * FROM YourTable WHERE Property = @property");
    // Scope your command to execute
    using(var command = new SqlCommand(query, connection))
    {
         // Open your connection
         connection.Open();

         // Add your parameters here if necessary

         // Execute your query as a reader (again scoped with a using statement)
         using(var reader = command.ExecuteReader())
         {
               // Iterate through your results here
         }
    }
}
```

.

```
using(var connection = new SqlConnection("{your-connection-string}"))
{
     var query = "UPDATE YourTable SET Property = Value WHERE Foo = @foo";
     using(var command = new SqlCommand(query,connection))
     {
          connection.Open();

          // Add parameters here
```

```
          // Perform your update
          command.ExecuteNonQuery();
     }
}
```

. ADO.NET .

- IDbConnection -
- IDbCommand - SQL
- IDbTransaction -
- IDataReader -
- IDataAdapter -

```
var connectionString = "{your-connection-string}";
var providerName = "{System.Data.SqlClient}"; //for Oracle use
"Oracle.ManagedDataAccess.Client"
//most likely you will get the above two from ConnectionStringSettings object

var factory = DbProviderFactories.GetFactory(providerName);

using(var connection = new factory.CreateConnection()) {
    connection.ConnectionString = connectionString;
    connection.Open();

    using(var command = new connection.CreateCommand()) {
        command.CommandText = "{sql-query}";    //this needs to be tailored for each database
system

        using(var reader = command.ExecuteReader()) {
            while(reader.Read()) {
                ...
            }
        }
    }
}
```

Entity Framework `DbContext` . `DbSet<T>` .

```
public class ExampleContext: DbContext
{
    public virtual DbSet<Widgets> Widgets { get; set; }
}
```

`DbContext` .

```
public class ExampleContext: DbContext
{
    // The parameter being passed in to the base constructor indicates the name of the
    // connection string
    public ExampleContext() : base("ExampleContextEntities")
    {
    }

    public virtual DbSet<Widgets> Widgets { get; set; }
```

```
}
```

Entity Framework

Entity Framework

```
using(var context = new ExampleContext())
{
      // Retrieve all of the Widgets in your database
      var data = context.Widgets.ToList();
}
```

Entity Framework `SaveChanges()` .

```
using(var context = new ExampleContext())
{
      // Grab the widget you wish to update
      var widget = context.Widgets.Find(w => w.Id == id);
      // If it exists, update it
      if(widget != null)
      {
           // Update your widget and save your changes
           widget.Updated = DateTime.UtcNow;
           context.SaveChanges();
      }
}
```

, .

```
Server=myServerAddress;Database=myDataBase;User Id=myUsername;Password=myPassword;
```

(: ASP.NET `app.config` `web.config`) . .

```
<connectionStrings>
   <add name="WidgetsContext" providerName="System.Data.SqlClient"
connectionString="Server=.\SQLEXPRESS;Database=Widgets;Integrated Security=True;"/>
</connectionStrings>

<connectionStrings>
   <add name="WidgetsContext" providerName="System.Data.SqlClient"
connectionString="Server=.\SQLEXPRESS;Database=Widgets;Integrated Security=SSPI;"/>
</connectionStrings>
```

`WidgetsContext` . `Integrated Security=SSPI` `Integrated Security=True` , `Integrated Security=SSPI` `Integrated Security=true` SQLClient OleDB OleDb throw.

(SQL Server, MySQL, Azure) . ConnectionStrings.com .

: https://riptutorial.com/ko/csharp/topic/4811/-

74:

:

- .
-
- .

Examples

. , . Decorator DP .

. Starbobs . - , ! , , , . .

, .

```
public abstract class AbstractCoffee
{
    protected AbstractCoffee k = null;

    public AbstractCoffee(AbstractCoffee k)
    {
        this.k = k;
    }

    public abstract string ShowCoffee();
}
```

, , . AbstractCoffee - :

```
public class Milk : AbstractCoffee
{
    public Milk(AbstractCoffee c) : base(c) { }
    public override string ShowCoffee()
    {
        if (k != null)
            return k.ShowCoffee() + " with Milk";
        else return "Milk";
    }
}
public class Sugar : AbstractCoffee
{
    public Sugar(AbstractCoffee c) : base(c) { }

    public override string ShowCoffee()
    {
        if (k != null) return k.ShowCoffee() + " with Sugar";
        else return "Sugar";
    }
}
public class Topping : AbstractCoffee
{
    public Topping(AbstractCoffee c) : base(c) { }
```

```
    public override string ShowCoffee()
    {
        if (k != null) return k.ShowCoffee() + " with Topping";
        else return "Topping";
    }
}
```

.

```
public class Program
{
    public static void Main(string[] args)
    {
        AbstractCoffee coffee = null; //we cant create instance of abstract class
        coffee = new Topping(coffee); //passing null
        coffee = new Sugar(coffee); //passing topping instance
        coffee = new Milk(coffee);  //passing sugar
        Console.WriteLine("Coffee with " + coffee.ShowCoffee());

    }
}
```

.

: https://riptutorial.com/ko/csharp/topic/4798/---

75:

`dynamic` . `dynamic` .

"Metaprogramming in .NET" C # `dynamic` .

`dynamic` `CallSite` . `CallSite` . , , `System.Object` .

Examples

```
dynamic foo = 123;
Console.WriteLine(foo + 234);
// 357    Console.WriteLine(foo.ToUpper())
// RuntimeBinderException, since int doesn't have a ToUpper method

foo = "123";
Console.WriteLine(foo + 234);
// 123234
Console.WriteLine(foo.ToUpper()):
// NOW A STRING
```

```
using System;

public static void Main()
{
    var value = GetValue();
    Console.WriteLine(value);
    // dynamics are useful!
}

private static dynamic GetValue()
{
    return "dynamics are useful!";
}
```

```
using System;
using System.Dynamic;

dynamic info = new ExpandoObject();
info.Id = 123;
info.Another = 456;

Console.WriteLine(info.Another);
// 456

Console.WriteLine(info.DoesntExist);
// Throws RuntimeBinderException
```

.

```
class IfElseExample
{
```

```
    public string DebugToString(object a)
    {
        if (a is StringBuilder)
        {
            return DebugToStringInternal(a as StringBuilder);
        }
        else if (a is List<string>)
        {
            return DebugToStringInternal(a as List<string>);
        }
        else
        {
            return a.ToString();
        }
    }

    private string DebugToStringInternal(object a)
    {
        // Fall Back
        return a.ToString();
    }

    private string DebugToStringInternal(StringBuilder sb)
    {
        return $"StringBuilder - Capacity: {sb.Capacity}, MaxCapacity: {sb.MaxCapacity},
Value: {sb.ToString()}";
    }

    private string DebugToStringInternal(List<string> list)
    {
        return $"List<string> - Count: {list.Count}, Value: {Environment.NewLine + "\t" +
string.Join(Environment.NewLine + "\t", list.ToArray())}";
    }
}

class DynamicExample
{
    public string DebugToString(object a)
    {
        return DebugToStringInternal((dynamic)a);
    }

    private string DebugToStringInternal(object a)
    {
        // Fall Back
        return a.ToString();
    }

    private string DebugToStringInternal(StringBuilder sb)
    {
        return $"StringBuilder - Capacity: {sb.Capacity}, MaxCapacity: {sb.MaxCapacity},
Value: {sb.ToString()}";
    }

    private string DebugToStringInternal(List<string> list)
    {
        return $"List<string> - Count: {list.Count}, Value: {Environment.NewLine + "\t" +
string.Join(Environment.NewLine + "\t", list.ToArray())}";
    }
}
```

DebugToStringInternal . .

: https://riptutorial.com/ko/csharp/topic/762/-

76:

. C# :

, . .

=> . ., . . , {code block} . void return .

Examples

```
List<int> l2 = l1.FindAll(x => x > 6);
```

x => x > 6 6 .

```
public delegate int ModifyInt(int input);
ModifyInt multiplyByTwo = x => x * 2;
```

.

```
public delegate int ModifyInt(int input);

ModifyInt multiplyByTwo = delegate(int x){
    return x * 2;
};
```

Lambda 'Func' 'Action' .

lambda (linq) .

```
var incremented = myEnumerable.Select(x => x + 1);
```

return .

lambda .

```
myObservable.Do(x => Console.WriteLine(x));
```

=> .

```
delegate int ModifyInt(int input1, int input2);
ModifyInt multiplyTwoInts = (x,y) => x * y;
```

.

```
delegate string ReturnString();
ReturnString getGreeting = () => "Hello world.";
```

(lambda) .

```
delegate void ModifyInt(int input);

ModifyInt addOneAndTellMe = x =>
{
    int result = x + 1;
    Console.WriteLine(result);
};
```

{} .

.

`Func``Expression` .

Person .

```
public class Person
{
    public string Name { get; set; }
    public int Age { get; set; }
}
```

:

```
p => p.Age > 18
```

.

```
public void AsFunc(Func<Person, bool> func)
public void AsExpression(Expression<Func<Person, bool>> expr)
```

Expression

LINQ Expression (IQueryable<T>) .

.

- .
- .

:

```
smtpClient.SendCompleted += (sender, args) => Console.WriteLine("Email sent");
```

/ .

```
EventHandler handler = (sender, args) => Console.WriteLine("Email sent");

smtpClient.SendCompleted += handler;
```

```
smtpClient.SendCompleted -= handler;
```

(-=) C # .

```
EventHandler handlerA = (sender, args) => Console.WriteLine("Email sent");
EventHandler handlerB = (sender, args) => Console.WriteLine("Email sent");
Console.WriteLine(handlerA.Equals(handlerB)); // May return "False"
```

. :

```
smtpClient.SendCompleted += (sender, args) => Console.WriteLine("Email sent");
emailSendButton.Enabled = true;
```

(sender, args) => Console.WriteLine("Email sent"); emailSendButton.Enabled = true; . . ,

, .

```
smtpClient.SendCompleted += ((sender, args) => Console.WriteLine("Email sent"));
//Adding an extra statement will result in a compile-time error
```

: https://riptutorial.com/ko/csharp/topic/46/-

77:

. . '' .

:

```
Func<object, bool> safeApplyFiltererPredicate = o => (o != null) && filterer.Predicate(i);
```

`safeApplyFilterPredicate` `filterer` private `Invoke` .

```
o => (o != null) && filterer.Predicate(i);
```

`safeApplyFilterPredicate` , `filterer` . garbage collection `filterer` .

.

:

```
var logger = new Logger();
Func<int, int> Add1AndLog = i => {
    logger.Log("adding 1 to " + i);
    return (i + 1);
};
```

.

```
Func<int, int> MyAddingMachine() {
    var i = 0;
    return x => i += x;
};
```

Examples

```
Func<int, int> add1 = i => i + 1;

Func<int, int, int> add = (i, j) => i + j;

// Behaviourally equivalent to:

int Add1(int i)
{
    return i + 1;
}

int Add(int i, int j)
{
    return i + j;
}

...
```

```
Console.WriteLine(add1(42)); //43
Console.WriteLine(Add1(42)); //43
Console.WriteLine(add(100, 250)); //350
Console.WriteLine(Add(100, 250)); //350
```

LINQ

```
// assume source is {0, 1, 2, ..., 10}

var evens = source.Where(n => n%2 == 0);
// evens = {0, 2, 4, ... 10}

var strings = source.Select(n => n.ToString());
// strings = {"0", "1", ..., "10"}
```

. .

```
public interface IMachine<TState, TInput>
{
    TState State { get; }
    public void Input(TInput input);
}
```

.

```
IMachine<int, int> machine = ...;
Func<int, int> machineClosure = i => {
    machine.Input(i);
    return machine.State;
};
```

machineClosure int int IMachine machine . machine machineClosure IMachine " .

: (:). . .

```
Func<int, string> doubleThenAddElevenThenQuote = i => {
    var doubled = 2 * i;
    var addedEleven = 11 + doubled;
    return $"'{addedEleven}'";
};
```

System.Linq.Expressions

```
Expression<Func<int, bool>> checkEvenExpression = i => i%2 == 0;
// lambda expression is automatically converted to an Expression<Func<int, bool>>
```

: https://riptutorial.com/ko/csharp/topic/7057/-

78:

Examples

RoslynScript

`Microsoft.CodeAnalysis.CSharp.Scripting.CSharpScript` C # .

```
var code = "(1 + 2).ToString()";
var run = await CSharpScript.RunAsync(code, ScriptOptions.Default);
var result = (string)run.ReturnValue;
Console.WriteLine(result); //output 3
```

,,, .

CSharpCodeProvider

`Microsoft.CSharp.CSharpCodeProvider` C # .

```
var code = @"
    public class Abc {
       public string Get() { return ""abc""; }
    }
";

var options = new CompilerParameters();
options.GenerateExecutable = false;
options.GenerateInMemory = false;

var provider = new CSharpCodeProvider();
var compile = provider.CompileAssemblyFromSource(options, code);

var type = compile.CompiledAssembly.GetType("Abc");
var abc = Activator.CreateInstance(type);

var method = type.GetMethod("Get");
var result = method.Invoke(abc, null);

Console.WriteLine(result); //output: abc
```

: https://riptutorial.com/ko/csharp/topic/3139/-

79:

Examples

. .

```
/// loop while the condition satisfies
while(condition)
{
    /// do something
}
```

while . .

```
do
{
    /// do something
} while(condition) /// loop while the condition satisfies
```

. (i) . .

```
for ( int i = 0; i < array.Count; i++ )
{
    var currentItem = array[i];
    /// do something with "currentItem"
}
```

IEnumarable . .

```
foreach ( var item in someList )
{
    /// do something with "item"
}
```

Foreach

.

```
list.ForEach(item => item.DoSomething());

// or
list.ForEach(item => DoSomething(item));

// or using a method group
list.ForEach(Console.WriteLine);

// using an array
Array.ForEach(myArray, Console.WriteLine);
```

List<T> Array . Linq .

Linq Parallel Foreach

Linq Foreach , . .

```
collection.AsParallel().ForAll(item => item.DoSomething());

/// or
collection.AsParallel().ForAll(item => DoSomething(item));
```

. .

```
for (;;)
{
    // precondition code that can change the value of should_end_loop expression

    if (should_end_loop)
        break;

    // do something
}
```

:

```
bool endLoop = false;
for (; !endLoop;)
{
    // precondition code that can set endLoop flag

    if (!endLoop)
    {
        // do something
    }
}
```

: / `switch` `break` .

Foreach

foreach `IEnumerable` (`IEnumerable<T>`). `List<T>` , `T[]` ( ), `Dictionary<TKey, TSource>` `IQueryable` `ICollection` .

```
foreach(ItemType itemVariable in enumerableObject)
    statement;
```

1. `ItemType` .
2. `ItemType var` `IEnumerable` generic enumerableObject
3. , (;) .
4. `enumerableObject` `IEnumerable` .
5. `ItemType` (`var` `var`) `InvalidCastException` .

.

```
var list = new List<string>();
list.Add("Ion");
list.Add("Andrei");
foreach(var name in list)
{
    Console.WriteLine("Hello " + name);
}
```

.

```
var list = new List<string>();
list.Add("Ion");
list.Add("Andrei");
IEnumerator enumerator;
try
{
    enumerator = list.GetEnumerator();
    while(enumerator.MoveNext())
    {
        string name = (string)enumerator.Current;
        Console.WriteLine("Hello " + name);
    }
}
finally
{
    if (enumerator != null)
        enumerator.Dispose();
}
```

While

```
int n = 0;
while (n < 5)
{
    Console.WriteLine(n);
    n++;
}
```

:

0
1
2

4

IEnumerator while .

```
// Call a custom method that takes a count, and returns an IEnumerator for a list
// of strings with the names of theh largest city metro areas.
IEnumerator<string> largestMetroAreas = GetLargestMetroAreas(4);

while (largestMetroAreas.MoveNext())
{
    Console.WriteLine(largestMetroAreas.Current);
```

```
}
```

:

/

/

For Loop

For . While .

For .

```
for (Initialization; Condition; Increment)
{
    // Code
}
```

- .
- .

Increment - .

:

```
for (int i = 0; i < 5; i++)
{
    Console.WriteLine(i);
}
```

:

0
1
2

4

For Loop .

```
int input = Console.ReadLine();

for ( ; input < 10; input + 2)
{
    Console.WriteLine(input);
}
```

3 :

5
7
9
11

Do - While

`while` . Do - While true .

```
int[] numbers = new int[] { 6, 7, 8, 10 };

// Sum values from the array until we get a total that's greater than 10,
// or until we run out of values.
int sum = 0;
int i = 0;
do
{
    sum += numbers[i];
    i++;
} while (sum <= 10 && i < numbers.Length);

System.Console.WriteLine(sum); // 13
```

```
// Print the multiplication table up to 5s
for (int i = 1; i <= 5; i++)
{
    for (int j = 1; j <= 5; j++)
    {
        int product = i * j;
        Console.WriteLine("{0} times {1} is {2}", i, j, product);
    }
}
```

`break` `continue` . . .

.

```
for (int i = 1; i <= 10; i++)
{
    if (i < 9)
        continue;

    Console.WriteLine(i);
}
```

.

```
9
10
```

: `Continue` while do-while . for .

: https://riptutorial.com/ko/csharp/topic/2064/

80: (Rx)

Examples

TextBox TextChanged

TextBox TextChanged . 0.5 . .

```
Observable
     .FromEventPattern(textBoxInput, "TextChanged")
     .Select(s => ((TextBox) s.Sender).Text)
     .Throttle(TimeSpan.FromSeconds(0.5))
     .DistinctUntilChanged()
     .Subscribe(text => Console.WriteLine(text));
```

`IEnumerable<T>` , fe .

```
private IEnumerable<T> GetData()
{
    try
    {
        // return results from database
    }
    catch(Exception exception)
    {
        throw;
    }
}
```

Observable , . `SelectMany` 200 `Buffer` .

```
int bufferSize = 200;

Observable
    .Start(() => GetData())
    .SelectMany(s => s)
    .Buffer(bufferSize)
    .ObserveOn(SynchronizationContext.Current)
    .Subscribe(items =>
    {
        Console.WriteLine("Loaded {0} elements", items.Count);

        // do something on the UI like incrementing a ProgressBar
    },
    () => Console.WriteLine("Completed loading"));
```

(Rx) : https://riptutorial.com/ko/csharp/topic/5770/----rx-

81:

- **bool :** true false
- **byte :** None, int
- **sbyte :** None, int
- **char :** .
- **:** M m
- **double :** D, d
- **float :** F f
- **int :** None, int .
- **uint :** U, u uint
- **long :** L, l long
- **ulong :** UL, ul, Ul, uL, LU, lu, Lu, lU ulong
- **short :** None, int
- **ushort :** None, int
- **:** , `@`
- **null** : `null`

Examples

int

`int` `int` :

```
int i = 5;
```

uint

`uint` `U` `u` `uint` .

```
uint ui = 5U;
```

`string` ."

```
string s = "hello, this is a string literal";
```

 . .

, C # ( ). `"` , `@` . :

```
string s = @"The path is:
C:\Windows\System32";
//The backslashes and newline are included in the string
```

char

char .'

```
char c = 'h';
```

. .

( ). . (default(char) new char()) '\0', NULL (null null).

byte . int .

```
byte b = 127;
```

sbyte

sbyte . int .

```
sbyte sb = 127;
```

decimal M m .

```
decimal m = 30.5M;
```

double D d .

```
double d = 30.5D;
```

float F f .

```
float f = 30.5F;
```

long L l long .

```
long l = 5L;
```

ulong

ulong UL, ul, Ul, uL, LU, lu, Lu lU ulong .

```
ulong ul = 5UL;
```

short . int .

```
short s = 127;
```

ushort

ushort . int .

```
ushort us = 127;
```

bool true false .

```
bool b = true;
```

: https://riptutorial.com/ko/csharp/topic/2655/

82:

IDisposable .

- () {}
- (IDisposable disposable = new MyDisposable ()) {}

using IDisposable .

```
using(var obj = new MyObject())
{
}

class MyObject : IDisposable
{
    public void Dispose()
    {
        // Cleanup
    }
}
```

IDisposable MSDN .

Examples

using try-finally . , .

IDisposable (FileStream Stream .NET) Dispose .

```
int Foo()
{
    var fileName = "file.txt";

    {
        FileStream disposable = null;

        try
        {
            disposable = File.Open(fileName, FileMode.Open);

            return disposable.ReadByte();
        }
        finally
        {
            // finally blocks are always run
            if (disposable != null) disposable.Dispose();
        }
    }
}
```

using try-finally using .

```
int Foo()
{
    var fileName = "file.txt";

    using (var disposable = File.Open(fileName, FileMode.Open))
    {
        return disposable.ReadByte();
    }
    // disposable.Dispose is called even if we return earlier
}
```

`finally` `using` `Dispose()` :

```
int Foo()
{
    var fileName = "file.txt";

    using (var disposable = File.Open(fileName, FileMode.Open))
    {
        throw new InvalidOperationException();
    }
    // disposable.Dispose is called even if we throw an exception earlier
}
```

: `Dispose` , `IDisposable` `Dispose` throw . .

```
using ( var disposable = new DisposableItem() )
{
    return disposable.SomeProperty;
}
```

`try..finally` `using` `return` . `finally` . .

1. `try`
2.
3. finally
4. .

`disposable` ( ).

`using` . :

```
using (var input = File.OpenRead("input.txt"))
{
    using (var output = File.OpenWrite("output.txt"))
    {
        input.CopyTo(output);
    } // output is disposed here
} // input is disposed here
```

.

```
using (var input = File.OpenRead("input.txt"))
using (var output = File.OpenWrite("output.txt"))
```

```
{
    input.CopyTo(output);
} // output and then input are disposed here
```

.

: `using` Microsoft Code Analysis CS2002 ( ). `using` .

`using` ( ).

```
using (FileStream file = File.Open("MyFile.txt"), file2 = File.Open("MyFile2.txt"))
{
}
```

.

```
using (Stream file = File.Open("MyFile.txt"), data = new MemoryStream())
{
}
```

`var` . . .

Gotcha :

`db` .

```
public IDBContext GetDBContext()
{
    using (var db = new DBContext())
    {
        return db;
    }
}
```

:

```
public IEnumerable<Person> GetPeople(int age)
{
    using (var db = new DBContext())
    {
        return db.Persons.Where(p => p.Age == age);
    }
}
```

catch LINQ `DBContext` `DBContext` `DBContext` .

`using` . `using` . `ToList()` , `ToArray()` . Entity Framework `ToListAsync()` `ToArrayAsync()` `async` .

.

```
public IEnumerable<Person> GetPeople(int age)
```

```
{
    using (var db = new DBContext())
    {
        return db.Persons.Where(p => p.Age == age).ToList();
    }
}
```

ToList() ToArray() ., .

null .

IDisposable null .using Dispose() .

```
DisposableObject TryOpenFile()
{
    return null;
}

// disposable is null here, but this does not throw an exception
using (var disposable = TryOpenFile())
{
    // this will throw a NullReferenceException because disposable is null
    disposable.DoSomething();

    if(disposable != null)
    {
        // here we are safe because disposable has been checked for null
        disposable.DoSomething();
    }
}
```

Gotcha : Dispose

.

```
try
{
    using (var disposable = new MyDisposable())
    {
        throw new Exception("Couldn't perform operation.");
    }
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}

class MyDisposable : IDisposable
{
    public void Dispose()
    {
        throw new Exception("Couldn't dispose successfully.");
    }
}
```

" " " " . Dispose throw

. .

using . IDisposable .

IDisposable . using using .

IDisposable

IDisposable using using .

- SqlConnection , SqlCommand , SqlDataReader
- OleDbConnection , OleDbCommand , OleDbDataReader
- MySqlConnection , MySqlCommand , MySqlDbDataReader
- DbContext

C # . FooConnection , FooCommand , FooDataReader .

ADO.NET

ADO.NET .

```
// This scopes the connection (your specific class may vary)
using(var connection = new SqlConnection("{your-connection-string}")
{
    // Build your query
    var query = "SELECT * FROM YourTable WHERE Property = @property");
    // Scope your command to execute
    using(var command = new SqlCommand(query, connection))
    {
         // Open your connection
         connection.Open();

         // Add your parameters here if necessary

         // Execute your query as a reader (again scoped with a using statement)
         using(var reader = command.ExecuteReader())
         {
               // Iterate through your results here
         }
    }
}
```

.

```
using(var connection = new SqlConnection("{your-connection-string}"))
{
     var query = "UPDATE YourTable SET Property = Value WHERE Foo = @foo";
     using(var command = new SqlCommand(query,connection))
     {
          connection.Open();

          // Add parameters here
```

```
            // Perform your update
            command.ExecuteNonQuery();
        }
}
```

DataContext

Entity Framework ORM `DbContext` `DbContext` . `IDisposable` `using` .

```
using(var context = new YourDbContext())
{
      // Access your context and perform your query
      var data = context.Widgets.ToList();
}
```

Dispose

`using` . , .

```
public class CultureContext : IDisposable
{
    private readonly CultureInfo originalCulture;

    public CultureContext(string culture)
    {
        originalCulture = CultureInfo.CurrentCulture;
        Thread.CurrentThread.CurrentCulture = new CultureInfo(culture);
    }

    public void Dispose()
    {
        Thread.CurrentThread.CurrentCulture = originalCulture;
    }
}
```

.

```
Thread.CurrentThread.CurrentCulture = new CultureInfo("en-US");

using (new CultureContext("nl-NL"))
{
    // Code in this block uses the "nl-NL" culture
    Console.WriteLine(new DateTime(2016, 12, 25)); // Output: 25-12-2016 00:00:00
}

using (new CultureContext("es-ES"))
{
    // Code in this block uses the "es-ES" culture
    Console.WriteLine(new DateTime(2016, 12, 25)); // Output: 25/12/2016 0:00:00
}

// Reverted back to the original culture
Console.WriteLine(new DateTime(2016, 12, 25)); // Output: 12/25/2016 12:00:00 AM
```

: `CultureContext` .

ASP.NET MVC `BeginForm` .

( ) ( ) .

SSL . .

```
public static class SSLContext
{
    // define the delegate to inject
    public delegate void TunnelRoutine(BinaryReader sslReader, BinaryWriter sslWriter);

    // this allows the routine to be executed under SSL
    public static void ClientTunnel(TcpClient tcpClient, TunnelRoutine routine)
    {
        using (SslStream sslStream = new SslStream(tcpClient.GetStream(), true, _validate))
        {
            sslStream.AuthenticateAsClient(HOSTNAME, null, SslProtocols.Tls, false);

            if (!sslStream.IsAuthenticated)
            {
                throw new SecurityException("SSL tunnel not authenticated");
            }

            if (!sslStream.IsEncrypted)
            {
                throw new SecurityException("SSL tunnel not encrypted");
            }

            using (BinaryReader sslReader = new BinaryReader(sslStream))
            using (BinaryWriter sslWriter = new BinaryWriter(sslStream))
            {
                routine(sslReader, sslWriter);
            }
        }
    }
}
```

SSL SSL . SSL (:).

```
public void ExchangeSymmetricKey(BinaryReader sslReader, BinaryWriter sslWriter)
{
    byte[] bytes = new byte[8];
    (new RNGCryptoServiceProvider()).GetNonZeroBytes(bytes);
    sslWriter.Write(BitConverter.ToUInt64(bytes, 0));
}
```

.

```
SSLContext.ClientTunnel(tcpClient, this.ExchangeSymmetricKey);
```

`try..finally` `using()` `using()` `try..finally`) (`ExchangeSymmetricKey`) . `using()` `using()` .

: https://riptutorial.com/ko/csharp/topic/38/-

83:

Examples

.

.

```
class SmsUtil
{
    public bool SendMessage(string from, string to, string message, int retryCount, object
attachment)
    {
         // Some code
    }
}
```

C # 3.0 :

```
var result = SmsUtil.SendMessage("Mehran", "Maryam", "Hello there!", 12, null);
```

.

```
var result = SmsUtil.SendMessage(
    from: "Mehran",
    to:  "Maryam",
    message "Hello there!",
    retryCount: 12,
    attachment: null);
```

.

:

```
public sealed class SmsUtil
{
    public static bool SendMessage(string from, string to, string message, int retryCount = 5,
object attachment = null)
    {
         // Some code
    }
}
```

`retryCount` :

```
var result = SmsUtil.SendMessage(
                        from       : "Cihan",
                        to         : "Yakar",
                        message    : "Hello there!",
                        attachment : new object());
```

.

.

:

```
public static string Sample(string left, string right)
{
     return string.Join("-",left,right);
}
```

:

```
Console.WriteLine (Sample(left:"A",right:"B"));
Console.WriteLine (Sample(right:"A",left:"B"));
```

:

```
A-B
B-A
```

.

Named Arguments .

```
class Employee
{
    public string Name { get; private set; }

    public string Title { get; set; }

    public Employee(string name = "<No Name>", string title = "<No Title>")
    {
        this.Name = name;
        this.Title = title;
    }
}

var jack = new Employee("Jack", "Associate");   //bad practice in this line
```

.

```
//Evil Code: add optional parameters between existing optional parameters
public Employee(string name = "<No Name>", string department = "intern", string title = "<No
Title>")
{
    this.Name = name;
    this.Department = department;
    this.Title = title;
}

//the below code still compiles, but now "Associate" is an argument of "department"
var jack = new Employee("Jack", "Associate");
```

" " .

```
var jack = new Employee(name: "Jack", title: "Associate");
```

: https://riptutorial.com/ko/csharp/topic/2076/--

84:

: MSDN .

MSDN ,

- .
- .
- .
- .

Ref : MSDN , , . .

MSDN , Optional Argument,

- .
- .
- .
- a.
  - .
  - enum struct .
  - default (valueType) .
- .

Examples

.

```
FindArea(120, 56);
```

(, 120) (, 56). . .

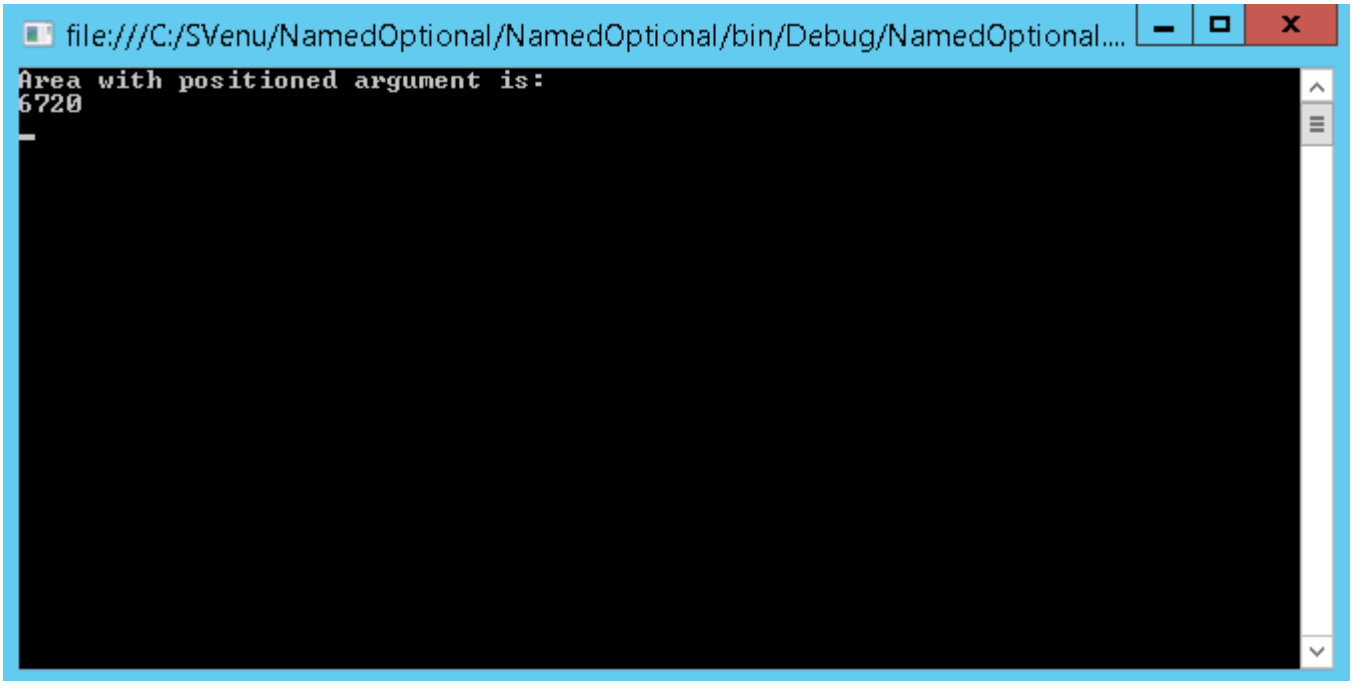
```
private static double FindArea(int length, int width)
       {
           try
           {
               return (length* width);
           }
           catch (Exception)
           {
               throw new NotImplementedException();
           }
       }
```

. ?

```
double area;
Console.WriteLine("Area with positioned argument is: ");
area = FindArea(120, 56);
```

```
Console.WriteLine(area);
Console.Read();
```

.



. .

```
Console.WriteLine("Area with Named argument is: ");
area = FindArea(length: 120, width: 56);
Console.WriteLine(area);
Console.Read();
```

.

```
area = FindArea(length: 120, width: 56);
```

. .

```
Console.WriteLine("Area with Named argument vice versa is: ");
area = FindArea(width: 120, length: 56);
Console.WriteLine(area);
Console.Read();
```

. .

., .

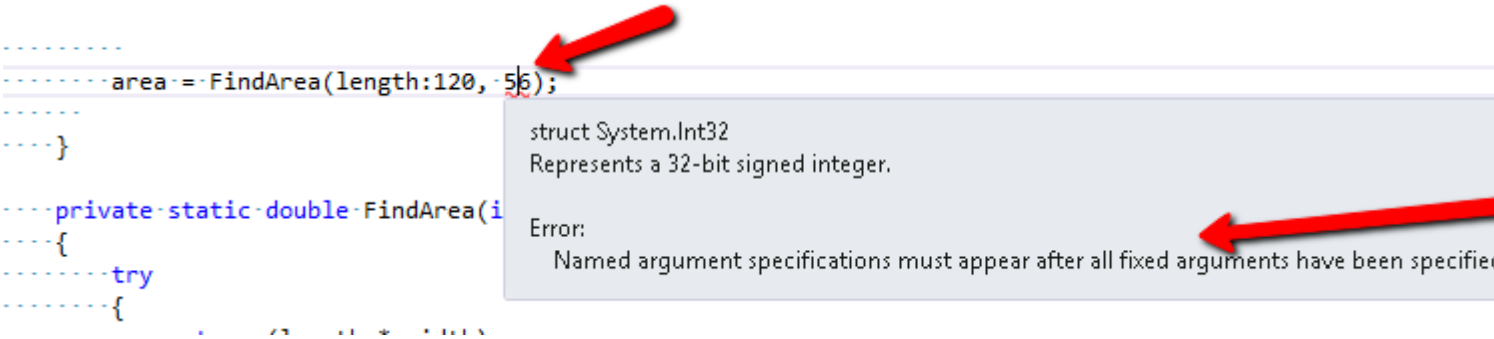
```
Console.WriteLine("Area with Named argument Positional Argument : ");
            area = FindArea(120, width: 56);
            Console.WriteLine(area);
            Console.Read();
```

120 56 .

. .

.

.

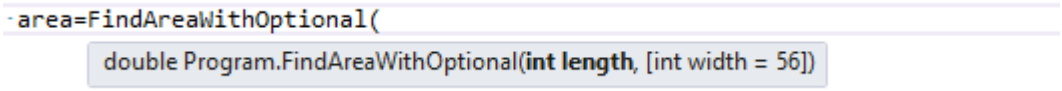


.

.

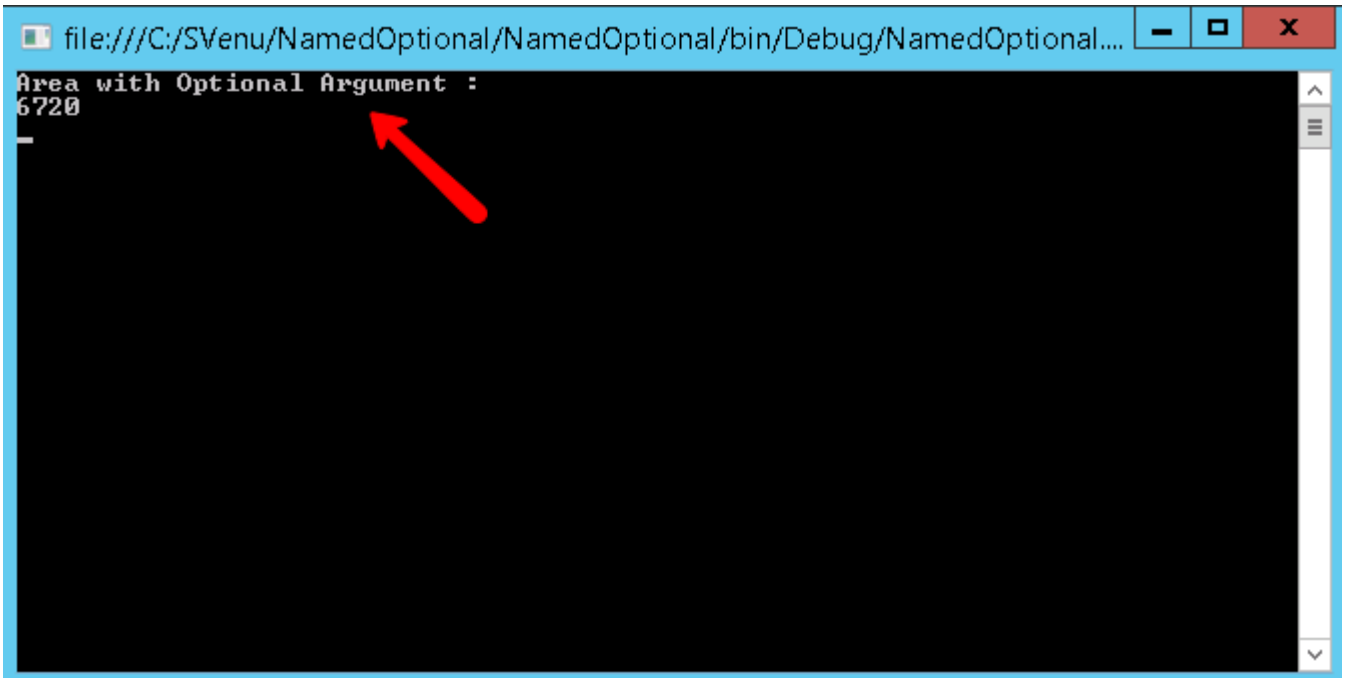
```
private static double FindAreaWithOptional(int length, int width=56)
        {
            try
            {
                return (length * width);
            }
            catch (Exception)
            {
                throw new NotImplementedException();
            }
        }
```

width 56 . IntelliSense .




```
Console.WriteLine("Area with Optional Argument : ");
area = FindAreaWithOptional(120);
Console.WriteLine(area);
Console.Read();
```

.



.

`[Optional]` . . `Optional` "Runtime.InteropServices" .

```
using System.Runtime.InteropServices;
private static double FindAreaWithOptional(int length, [Optional]int width)
   {
       try
       {
           return (length * width);
       }
       catch (Exception)
       {
           throw new NotImplementedException();
       }
   }

area = FindAreaWithOptional(120);  //area=0
```

int 0 0 0.

: https://riptutorial.com/ko/csharp/topic/5220/----

85:

- $ "content {expression} "
- $ "content {expression : format} content"
- $ "content {expression} {{ }} content}" "
- $ "content {expression : format} {{ }} content}" "

`string.Format()` , .

```
var name = "World";
var oldWay = string.Format("Hello, {0}!", name);  // returns "Hello, World"
var newWay = $"Hello, {name}!";                   // returns "Hello, World"
```

Examples

.

```
var StrWithMathExpression = $"1 + 2 = {1 + 2}"; // -> "1 + 2 = 3"

string world = "world";
var StrWithFunctionCall = $"Hello, {world.ToUpper()}!"; // -> "Hello, WORLD!"
```

.NET Fiddle

```
var date = new DateTime(2015, 11, 11);
var str = $"It's {date:MMMM d, yyyy}, make a wish!";
System.Console.WriteLine(str);
```

`DateTime.ToString` DateTime DateTime . .

```
var date = new DateTime(2015, 11, 11);
var str = date.ToString("MMMM d, yyyy");
str = "It's " + str + ", make a wish!";
Console.WriteLine(str);
```

:

2015 11 11 , !

.NET Fiddle

DateTime.ToString

: `MM` , `mm` . .

```
var name = "World";
var str = $"Hello, {name}!";
//str now contains: "Hello, World!";
```

```
$"Hello, {name}!"
```

.

```
string.Format("Hello, {0}!", name);
```

.

```
${value, padding}
```

: .

5 (3 5 .)

```
var number = 42;
var str = $"The answer to life, the universe and everything is {number, 5}.";
//str is "The answer to life, the universe and everything is    42.";
//                                                            ^^^^^
System.Console.WriteLine(str);
```

:

```
The answer to life, the universe and everything is    42.
```

.NET Fiddle

.

```
var number = 42;
var str = $"The answer to life, the universe and everything is ${number, -5}.";
//str is "The answer to life, the universe and everything is 42   .";
//                                                            ^^^^^
System.Console.WriteLine(str);
```

:

```
The answer to life, the universe and everything is 42   .
```

.NET Fiddle

.

```
var number = 42;
var str = $"The answer to life, the universe and everything is ${number, 5:f1}";
//str is "The answer to life, the universe and everything is 42.1 ";
//                                                            ^^^^^
```

.NET Fiddle

.

```
var decimalValue = 120.5;

var asCurrency = $"It costs {decimalValue:C}";
// String value is "It costs $120.50" (depending on your local currency settings)

var withThreeDecimalPlaces = $"Exactly {decimalValue:F3}";
// String value is "Exactly 120.500"

var integerValue = 57;

var prefixedIfNecessary = $"{integerValue:D5}";
// String value is "00057"
```

.NET Fiddle

: https://riptutorial.com/ko/csharp/topic/22/-

86:

+ / `Concat` `StringBuilder` . `Concat` .

Examples

+

```
string s1 = "string1";
string s2 = "string2";

string s3 = s1 + s2; // "string1string2"
```

System.Text.StringBuilder

StringBuilder + . . StringBuilders . .

```
StringBuilder sb = new StringBuilder();
for (int i = 1; i <= 5; i++)
{
    sb.Append(i);
    sb.Append(" ");
}
Console.WriteLine(sb.ToString()); // "1 2 3 4 5 "
```

`Append()` . `StringBuilder` .

```
StringBuilder sb = new StringBuilder();
sb.Append("some string ")
  .Append("another string");
```

String.Join Concat

`String.Join` .

```
string[] value = {"apple", "orange", "grape", "pear"};
string separator = ", ";

string result = String.Join(separator, value, 1, 2);
Console.WriteLine(result);
```

: "orange, grape"

`String.Join(String, String[], Int32, Int32)` .

startIndex count . :

```
string[] value = {"apple", "orange", "grape", "pear"};
```

```
string separator = ", ";
string result = String.Join(separator, value);
Console.WriteLine(result);
```

;

,,,

$

$.

```
var str1 = "text1";
var str2 = " ";
var str3 = "text3";
string result2 = $"{str1}{str2}{str3}"; //"text1 text3"
```

: https://riptutorial.com/ko/csharp/topic/3616/-

87:

- \ '- (0x0027)
- \ "- (0x0022)
- \\ - (0x005C)
- \ 0 - null (0x0000)
- \ a - (0x0007)
- \ b - (0x0008)
- \ f - (0x000C)
- \ n - (0x000A)
- \ r - (0x000D)
- \ t - (0x0009)
- \ v - (0x000B)
- \ u0000 - \ uFFFF -
- \ x0 - \ xFFFF - ( )
- \ U00000000 - \ U0010FFFF - ( )

. .

notEscaped notEscaped2 ('\' 'n') .

```
string escaped = "\n";
string notEscaped = "\\" + "n";
string notEscaped2 = "\\n";

Console.WriteLine(escaped.Length); // 1
Console.WriteLine(notEscaped.Length); // 2
Console.WriteLine(notEscaped2.Length); // 2
```

Examples

```
string sqrt = "\u221A";      // √
string emoji = "\U0001F601"; // 
string text = "\u0022Hello World\u0022"; // "Hello World"
string variableWidth = "\x22Hello World\x22"; // "Hello World"
```

```
char apostrophe = '\'';
```

```
char oneBackslash = '\\';
```

```
// The filename will be c:\myfile.txt in both cases
string filename = "c:\\myfile.txt";
string filename = @"c:\myfile.txt";
```

.

```
string text = "\"Hello World!\", said the quick brown fox.";
string verbatimText = @"""Hello World!"", said the quick brown fox.";
```

.

"!" .

.

```
string text = "Hello\r\nWorld!";
string verbatimText = @"Hello
World!";
```

.

.

```
string s = "\c";
char c = '\c';
```

Unrecognized escape sequence .

string char .

3 .

```
protected abstract IEnumerable<Texte> ObtenirŒuvres();
```

C # Œ . \u#### \U######## . .

```
protected override IEnumerable<Texte> Obtenir\u0152uvres()
{
    // ...
}
```

C # Œ \u0152 \u0152 .

(UTF-8 .)

: https://riptutorial.com/ko/csharp/topic/39/--

88:

Examples

String /

`System.String` .

- `System.String.ToLowerInvariant` String .
- `System.String.ToUpperInvariant` String .

: . .

:

```
string s = "My String";
s = s.ToLowerInvariant(); // "my string"
s = s.ToUpperInvariant(); // "MY STRING"
```

String.ToLower (CultureInfo) String.ToUpper (CultureInfo) .

`System.String.Contains` . true false .

```
string s = "Hello World";
bool stringExists = s.Contains("ello");  //stringExists =true as the string contains the
substring
```

`System.String.IndexOf` .
. -1 .

```
string s = "Hello World";
int location = s.IndexOf("ello"); // location = 1
```

`System.String.LastIndexOf` .

```
string s = "Hello World";
int location = s.LastIndexOf("l"); // location = 9
```

()

`System.String.Trim` .

```
string s = "     String with spaces at both ends      ";
s = s.Trim(); // s = "String with spaces at both ends"
```

:

- . System.String.TrimStart
- . System.String.TrimEnd

.

System.String.Substring .

```
string s ="A portion of word that is retained";
s=str.Substring(26);  //s="retained"

s1 = s.Substring(0,5);  //s="A por"
```

System.String.Replace .

```
string s = "Hello World";
s = s.Replace("World", "Universe"); // s = "Hello Universe"
```

.

```
string s = "Hello World";
s = s.Replace("l", "L"); // s = "HeLLo WorLD"
```

String.Replace .

```
string s = "Hello World";
s = s.Replace("ell", String.Empty); // s = "Ho World"
```

System.String.Split .

```
string sentence = "One Two Three Four";
string[] stringArray = sentence.Split(' ');

foreach (string word in stringArray)
{
    Console.WriteLine(word);
}
```

:

System.String.Join .

```
string[] words = {"One", "Two", "Three", "Four"};
string singleString = String.Join(",", words); // singleString = "One,Two,Three,Four"
```

System.String.Concat + .

```
string first = "Hello ";
string second = "World";

string concat = first + second; // concat = "Hello World"
concat = String.Concat(first, second); // concat = "Hello World"
```

: https://riptutorial.com/ko/csharp/topic/3599/-

89: FormatException

Examples

`string integer` . .

1. `Convert.ToInt16();`
2. `Convert.ToInt32();`
3. `Convert.ToInt64();`
4. `int.Parse();`

`FormatException` throw. (`try..catch`) .

:

, :

```
string inputString = "10.2";
```

1 : `Convert.ToInt32()`

```
int convertedInt = Convert.ToInt32(inputString); // Failed to Convert
// Throws an Exception "Input string was not in a correct format."
```

: - *`Convert.ToInt16();` `Convert.ToInt64();`*

2 : `int.Parse()`

```
int convertedInt = int.Parse(inputString); // Same result "Input string was not in a correct
format.
```

?

`try..catch` .

```
try
{
    string inputString = "10.2";
    int convertedInt = int.Parse(inputString);
}
catch (Exception Ex)
{
    //Display some message, that the conversion has failed.
}
```

`try..catch` , `0` . *(`convertedInt` `0` . catch).* `.TryParse()` .

`.TryParse()` `out` (, `true` , `false`). . :

1 : Boolean .

```
 int convertedInt; // Be the required integer
 bool isSuccessConversion = int.TryParse(inputString, out convertedInt);
```

`isSuccessConversion` Variable `isSuccessConversion` . `convertedInt` `0` (`0`).

2 : `if`

```
if (int.TryParse(inputString, out convertedInt))
{
    // convertedInt will have the converted value
    // Proceed with that
}
else
{
 // Display an error message
}
```

3 : (`0`)

```
int.TryParse(inputString, out convertedInt);
// use the value of convertedInt
// But it will be 0 if not converted
```

FormatException : https://riptutorial.com/ko/csharp/topic/2886/------formatexception-

90:

Examples

N * 2

Windows Forms .

```
var nameList = new BindingList<string>();
ComboBox1.DataSource = nameList;
for(long i = 0; i < 10000; i++ ) {
    nameList.AddRange(new [] {"Alice", "Bob", "Carol" });
}
```

, .

```
var nameList = new BindingList<string>();
ComboBox1.DataSource = nameList;
nameList.RaiseListChangedEvents = false;
for(long i = 0; i < 10000; i++ ) {
    nameList.AddRange(new [] {"Alice", "Bob", "Carol" });
}
nameList.RaiseListChangedEvents = true;
nameList.ResetBindings();
```

```
BindingList<string> listOfUIItems = new BindingList<string>();
listOfUIItems.Add("Alice");
listOfUIItems.Add("Bob");
```

: https://riptutorial.com/ko/csharp/topic/182/---t-

91:

yield , get .

Examples

. .

IEnumerable GetEnumerator() . .

```
int[] numbers = { 1, 2, 3, 4, 5 };

IEnumerator iterator = numbers.GetEnumerator();

while (iterator.MoveNext())
{
    Console.WriteLine(iterator.Current);
}
```

```
1
2
3
4
5
```

foreach .

```
foreach (int number in numbers)
{
    Console.WriteLine(number);
}
```

. C #, , yield .

return return ., yield . . yield .

- yield return . . .
- yield break return . . return .

.

```
IEnumerable<int> Fibonacci(int count)
{
    int prev = 1;
    int curr = 1;

    for (int i = 0; i < count; i++)
    {
        yield return prev;
        int temp = prev + curr;
```

```
        prev = curr;
        curr = temp;
    }
}
```

. 10 .

```
void Main()
{
    foreach (int term in Fibonacci(10))
    {
        Console.WriteLine(term);
    }
}
```

```
1
1
2
3
5
8
13
21
34
55
```

: https://riptutorial.com/ko/csharp/topic/4243/

92:

C # . . , REST .

: MS . https://msdn.microsoft.com/en-us/library/ff647790.aspx .

, ( ). , . , , .

:

(C #)

.NET Framework

Examples

System.Type

:

```
var theString = "hello";
var theType = theString.GetType();
```

:

```
var theType = typeof(string);
```

```
using System;
using System.Reflection;
using System.Linq;

public class Program
{
  public static void Main()
  {
    var members = typeof(object)
                    .GetMembers(BindingFlags.Public |
                                BindingFlags.Static |
                                BindingFlags.Instance);

    foreach (var member in members)
    {
      bool inherited = member.DeclaringType.Equals( typeof(object).Name );
      Console.WriteLine($"{member.Name} is a {member.MemberType}, " +
                        $"it has {(inherited ? "":"not")} been inherited.");
    }
  }
}
```

( ) :

```
GetType is a Method, it has not been inherited.
GetHashCode is a Method, it has not been inherited.
ToString is a Method, it has not been inherited.
Equals is a Method, it has not been inherited.
Equals is a Method, it has not been inherited.
ReferenceEquals is a Method, it has not been inherited.
.ctor is a Constructor, it has not been inherited.
```

BindingFlags GetMembers() . .

GetMembers , GetMembers .

.

```
using System;

public class Program
{
    public static void Main()
    {
        var theString = "hello";
        var method = theString
                    .GetType()
                    .GetMethod("Substring",
                               new[] {typeof(int), typeof(int)}); //The types of the method
arguments
         var result = method.Invoke(theString, new object[] {0, 4});
         Console.WriteLine(result);
    }
}
```

:

.

```
var method = typeof(Math).GetMethod("Exp");
var result = method.Invoke(null, new object[] {2});//Pass null as the first argument (no need
for an instance)
Console.WriteLine(result); //You'll get e^2
```

:

7.38905609893065

:

```
PropertyInfo prop = myInstance.GetType().GetProperty("myProperty");
// get the value myInstance.myProperty
object value = prop.GetValue(myInstance);

int newValue = 1;
// set the value myInstance.myProperty to newValue
prop.setValue(myInstance, newValue);
```

Backing (Backing .NET Framework "k__BackingField").

```
// get backing field info
FieldInfo fieldInfo = myInstance.GetType()
    .GetField("<myProperty>k__BackingField", BindingFlags.Instance | BindingFlags.NonPublic);

int newValue = 1;
// set the value of myInstance.myProperty backing field to newValue
fieldInfo.SetValue(myInstance, newValue);
```

MyAttribute

```
var props = t.GetProperties(BindingFlags.NonPublic | BindingFlags.Public |
            BindingFlags.Instance).Where(
            prop => Attribute.IsDefined(prop, typeof(MyAttribute)));
```

```
var attributes = typeof(t).GetProperty("Name").GetCustomAttributes(false);
```

- MyAttribute

```
static IEnumerable<Type> GetTypesWithAttribute(Assembly assembly) {
    foreach(Type type in assembly.GetTypes()) {
        if (type.GetCustomAttributes(typeof(MyAttribute), true).Length > 0) {
            yield return type;
        }
    }
}
```

```
public static class AttributeExtensions
{

        /// <summary>
        /// Returns the value of a member attribute for any member in a class.
        ///     (a member is a Field, Property, Method, etc...)
        /// <remarks>
        /// If there is more than one member of the same name in the class, it will return the
first one (this applies to overloaded methods)
        /// </remarks>
        /// <example>
        /// Read System.ComponentModel Description Attribute from method 'MyMethodName' in
class 'MyClass':
        ///     var Attribute = typeof(MyClass).GetAttribute("MyMethodName",
(DescriptionAttribute d) => d.Description);
        /// </example>
        /// <param name="type">The class that contains the member as a type</param>
        /// <param name="MemberName">Name of the member in the class</param>
        /// <param name="valueSelector">Attribute type and property to get (will return first
instance if there are multiple attributes of the same type)</param>
        /// <param name="inherit">true to search this member's inheritance chain to find the
attributes; otherwise, false. This parameter is ignored for properties and events</param>
        /// </summary>
        public static TValue GetAttribute<TAttribute, TValue>(this Type type, string
MemberName, Func<TAttribute, TValue> valueSelector, bool inherit = false) where TAttribute :
Attribute
        {
            var att =
```

```
type.GetMember(MemberName).FirstOrDefault().GetCustomAttributes(typeof(TAttribute),
inherit).FirstOrDefault() as TAttribute;
            if (att != null)
            {
                return valueSelector(att);
            }
            return default(TValue);
        }
    }
```

```
//Read System.ComponentModel Description Attribute from method 'MyMethodName' in class
'MyClass'
var Attribute = typeof(MyClass).GetAttribute("MyMethodName", (DescriptionAttribute d) =>
d.Description);
```

.

```
Type type = obj.GetType();
//To restrict return properties. If all properties are required don't provide flag.
BindingFlags flags = BindingFlags.Public | BindingFlags.Instance;
PropertyInfo[] properties = type.GetProperties(flags);

foreach (PropertyInfo property in properties)
{
    Console.WriteLine("Name: " + property.Name + ", Value: " + property.GetValue(obj, null));
}
```

.

, List<T> .

```
var myList = new List<int>();
ShowGenericArguments(myList);
```

ShowGenericArguments .

```
public void ShowGenericArguments(object o)
```

o . Reflection . o .

```
public void ShowGenericArguments(object o)
{
    if (o == null) return;

    Type t = o.GetType();
    if (!t.IsGenericType) return;
    ...
```

Type.IsGenericType true false .

. List<> . *generic* . List<int> .

Type IsConstructedGenericType IsGenericTypeDefinition IsConstructedGenericType .

```
typeof(List<>).IsGenericType // true
typeof(List<>).IsGenericTypeDefinition // true
typeof(List<>).IsConstructedGenericType// false

typeof(List<int>).IsGenericType // true
typeof(List<int>).IsGenericTypeDefinition // false
typeof(List<int>).IsConstructedGenericType// true
```

Type GetGenericArguments() .

```
public void ShowGenericArguments(object o)
{
    if (o == null) return;
    Type t = o.GetType();
    if (!t.IsConstructedGenericType) return;

    foreach(Type genericTypeArgument in t.GetGenericArguments())
        Console.WriteLine(genericTypeArgument.Name);
}
```

(ShowGenericArguments(myList)) .

```
Int32
```

.

. .

```
public class Sample
{
    public void GenericMethod<T>()
    {
        // ...
    }

    public static void StaticMethod<T>()
    {
        //...
    }
}
```

GenericMethod string .

```
Sample sample = new Sample();//or you can get an instance via reflection

MethodInfo method = typeof(Sample).GetMethod("GenericMethod");
MethodInfo generic = method.MakeGenericMethod(typeof(string));
generic.Invoke(sample, null);//Since there are no arguments, we are passing null
```

. null.

```
MethodInfo method = typeof(Sample).GetMethod("StaticMethod");
MethodInfo generic = method.MakeGenericMethod(typeof(string));
generic.Invoke(null, null);
```

.

```
var baseType = typeof(List<>);
var genericType = baseType.MakeGenericType(typeof(String));
var instance = Activator.CreateInstance(genericType);
var method = genericType.GetMethod("GetHashCode");
var result = method.Invoke(instance, new object[] { });
```

(:)

(: `plugins` `plugins`)

```
interface IPlugin
{
    string PluginDescription { get; }
    void DoWork();
}
```

dll .

```
class HelloPlugin : IPlugin
{
    public string PluginDescription => "A plugin that says Hello";
    public void DoWork()
    {
        Console.WriteLine("Hello");
    }
}
```

dll `IPlugin` .

```
    public IEnumerable<IPlugin> InstantiatePlugins(string directory)
    {
        var pluginAssemblyNames = Directory.GetFiles(directory, "*.addin.dll").Select(name =>
new FileInfo(name).FullName).ToArray();
        //load the assemblies into the current AppDomain, so we can instantiate the types
later
        foreach (var fileName in pluginAssemblyNames)
            AppDomain.CurrentDomain.Load(File.ReadAllBytes(fileName));
        var assemblies = pluginAssemblyNames.Select(System.Reflection.Assembly.LoadFile);
        var typesInAssembly = assemblies.SelectMany(asm => asm.GetTypes());
        var pluginTypes = typesInAssembly.Where(type => typeof
(IPlugin).IsAssignableFrom(type));
        return pluginTypes.Select(Activator.CreateInstance).Cast<IPlugin>();
    }
```

`Activator` .

`Activator.CreateInstance()` .NET 3.5 `Activator` () 1 , 2 , 3 ... .

Activator

```
Type type = typeof(BigInteger);
object result = Activator.CreateInstance(type); //Requires parameterless constructor.
Console.WriteLine(result); //Output: 0
result = Activator.CreateInstance(type, 123); //Requires a constructor which can receive an
'int' compatible argument.
Console.WriteLine(result); //Output: 123
```

Activator.CreateInstance .

```
// With a constructor such as MyClass(int, int, string)
Activator.CreateInstance(typeof(MyClass), new object[] { 1, 2, "Hello World" });

Type type = typeof(someObject);
var instance = Activator.CreateInstance(type);
```

MakeGenericType (List<>) (: List<string>).

```
// generic List with no parameters
Type openType = typeof(List<>);

// To create a List<string>
Type[] tArgs = { typeof(string) };
Type target = openType.MakeGenericType(tArgs);

// Create an instance - Activator.CreateInstance will call the default constructor.
// This is equivalent to calling new List<string>().
List<string> result = (List<string>)Activator.CreateInstance(target);
```

List<> typeof .

Activator

new ( )

```
T GetInstance<T>() where T : new()
{
    T instance = new T();
    return instance;
}
```

Invoke

```
// Get the instance of the desired constructor (here it takes a string as a parameter).
ConstructorInfo c = typeof(T).GetConstructor(new[] { typeof(string) });
// Don't forget to check if such constructor exists
if (c == null)
    throw new InvalidOperationException(string.Format("A constructor for type '{0}' was not
found.", typeof(T)));
T instance = (T)c.Invoke(new object[] { "test" });
```

. MSDN :

0 , ,, . , , . , , , .

```
public class GenericFactory<TKey, TType>
    {
       private readonly Dictionary<TKey, Func<object[], TType>> _registeredTypes; //
dictionary, that holds constructor functions.
       private object _locker = new object(); // object for locking dictionary, to guarantee
thread safety

        public GenericFactory()
        {
            _registeredTypes = new Dictionary<TKey, Func<object[], TType>>();
        }

        /// <summary>
        /// Find and register suitable constructor for type
        /// </summary>
        /// <typeparam name="TType"></typeparam>
        /// <param name="key">Key for this constructor</param>
        /// <param name="parameters">Parameters</param>
        public void Register(TKey key, params Type[] parameters)
        {
            ConstructorInfo ci = typeof(TType).GetConstructor(BindingFlags.Public |
BindingFlags.Instance, null, CallingConventions.HasThis, parameters, new ParameterModifier[] {
}); // Get the instance of ctor.
            if (ci == null)
                throw new InvalidOperationException(string.Format("Constructor for type '{0}'
was not found.", typeof(TType)));

            Func<object[], TType> ctor;

            lock (_locker)
            {
                if (!_registeredTypes.TryGetValue(key, out ctor)) // check if such ctor
already been registered
                {
                    var pExp = Expression.Parameter(typeof(object[]), "arguments"); // create
parameter Expression
                    var ctorParams = ci.GetParameters(); // get parameter info from
constructor

                    var argExpressions = new Expression[ctorParams.Length]; // array that will
contains parameter expessions
                    for (var i = 0; i < parameters.Length; i++)
                    {

                        var indexedAcccess = Expression.ArrayIndex(pExp,
Expression.Constant(i));

                        if (!parameters[i].IsClass && !parameters[i].IsInterface) // check if
parameter is a value type
                        {
                            var localVariable = Expression.Variable(parameters[i],
"localVariable"); // if so - we should create local variable that will store paraameter value

                            var block = Expression.Block(new[] { localVariable },
                                    Expression.IfThenElse(Expression.Equal(indexedAcccess,
Expression.Constant(null)),
                                        Expression.Assign(localVariable,
Expression.Default(parameters[i])),
                                        Expression.Assign(localVariable,
```

```
Expression.Convert(indexedAcccess, parameters[i]))
                                    ),
                                    localVariable
                                );

                            argExpressions[i] = block;

                        }
                        else
                            argExpressions[i] = Expression.Convert(indexedAcccess,
parameters[i]);
                    }
                    var newExpr = Expression.New(ci, argExpressions); // create expression
that represents call to specified ctor with the specified arguments.

                    _registeredTypes.Add(key, Expression.Lambda(newExpr, new[] { pExp
}).Compile() as Func<object[], TType>); // compile expression to create delegate, and add
fucntion to dictionary
                }
            }
        }

        /// <summary>
        /// Returns instance of registered type by key.
        /// </summary>
        /// <typeparam name="TType"></typeparam>
        /// <param name="key"></param>
        /// <param name="args"></param>
        /// <returns></returns>
        public TType Create(TKey key, params object[] args)
        {
            Func<object[], TType> foo;
            if (_registeredTypes.TryGetValue(key, out foo))
            {
                return (TType)foo(args);
            }

            throw new ArgumentException("No type registered for this key.");
        }
    }
```

:

```
 public class TestClass
 {
        public TestClass(string parameter)
        {
            Console.Write(parameter);
        }
 }


public void TestMethod()
{
       var factory = new GenericFactory<string, TestClass>();
       factory.Register("key", typeof(string));
       TestClass newInstance = factory.Create("key", "testParameter");
}
```

FormatterServices.GetUninitializedObject

```
T instance = (T)FormatterServices.GetUninitializedObject(typeof(T));
```

FormatterServices.GetUninitializedObject . .

.

. .

```
typeof(KnownType).Assembly.GetType(typeName);
```

- typeName ( ) KnownType .

:

```
Type t = null;
foreach (Assembly ass in AppDomain.CurrentDomain.GetAssemblies())
{
    if (ass.FullName.StartsWith("System."))
        continue;
    t = ass.GetType(typeName);
    if (t != null)
        break;
}
```

. CLR .

.

```
Type.GetType(fullyQualifiedName);
```

, (, MethodInfo.Invoke) . Delegate.CreateDelegate . . , :

```
// Get a MethodInfo for the Math.Max(int, int) method...
var maxMethod = typeof(Math).GetMethod("Max", new Type[] { typeof(int), typeof(int) });
// Now get a strongly-typed delegate for Math.Max(int, int)...
var stronglyTypedDelegate = (Func<int, int, int>)Delegate.CreateDelegate(typeof(Func<int, int,
int>), null, maxMethod);
// Invoke the Math.Max(int, int) method using the strongly-typed delegate...
Console.WriteLine("Max of 3 and 5 is: {0}", stronglyTypedDelegate(3, 5));
```

. MyIntProperty int MyClass getter ('target' MyClass).

```
// Get a MethodInfo for the MyClass.MyIntProperty getter...
var theProperty = typeof(MyClass).GetProperty("MyIntProperty");
var theGetter = theProperty.GetGetMethod();
// Now get a strongly-typed delegate for MyIntProperty that can be executed against any
MyClass instance...
var stronglyTypedGetter = (Func<MyClass, int>)Delegate.CreateDelegate(typeof(Func<MyClass,
int>), theGetter);
// Invoke the MyIntProperty getter against MyClass instance 'target'...
```

```
Console.WriteLine("target.MyIntProperty is: {0}", stronglyTypedGetter(target));
```

... .

```
// Get a MethodInfo for the MyClass.MyIntProperty setter...
var theProperty = typeof(MyClass).GetProperty("MyIntProperty");
var theSetter = theProperty.GetSetMethod();
// Now get a strongly-typed delegate for MyIntProperty that can be executed against any
MyClass instance...
var stronglyTypedSetter = (Action<MyClass, int>)Delegate.CreateDelegate(typeof(Action<MyClass,
int>), theSetter);
// Set MyIntProperty to 5...
stronglyTypedSetter(target, 5);
```

: https://riptutorial.com/ko/csharp/topic/28/

93:

- **:**

  <type> [] <name>;

- **2 :**

  <type> [,] <> = <> [<>, <>];

- **:**

  <type> [] <> = <> [<>];

- **:**

  <> [<>] = <> [<>];

- **:**

  <> = <> [<>];

- **:**

  <> = <> [] {<>, <>, <>, ...};

- **2 :**

  <name> = <type> [,] {{<value>, <value>}, {<value>, <value>}, ...};

- **i :**

  <> [i]

- **:**

  <>.

C # *nullable* .

`.Add()` `.Remove()` . `List` `ArrayList` .

Examples

```
string[] strings = new[] {"foo", "bar"};
object[] objects = strings; // implicit conversion from string[] to object[]
```

. .

```
string[] strings = new[] {"Foo"};
object[] objects = strings;

objects[0] = new object(); // runtime exception, object is not string
string str = strings[0];   // would have been bad if above assignment had succeeded
```

```
int[] arr = new int[] { 0, 10, 20, 30};

// Get
Console.WriteLine(arr[2]); // 20

// Set
arr[2] = 100;

// Get the updated value
Console.WriteLine(arr[2]); // 100
```

([]) . 10

```
int[] arr = new int[10];
```

C # 0 . 0-9. :

```
int[] arr = new int[3] {7,9,4};
Console.WriteLine(arr[0]); //outputs 7
Console.WriteLine(arr[1]); //outputs 9
```

, 0 . . , , () .

.

```
int[] arr = null;   // OK, declares a null reference to an array.
int first = arr[0]; // Throws System.NullReferenceException because there is no actual array.
```

.

```
int[] arr = new int[] { 24, 2, 13, 47, 45 };
```

`new int[]` . (`new`).

```
int[] arr = { 24, 2, 13, 47, 45 };  // OK
int[] arr1;
arr1 = { 24, 2, 13, 47, 45 };       // Won't compile
```

`var` .

```
// same as int[]
var arr = new [] { 1, 2, 3 };
// same as string[]
var arr = new [] { "one", "two", "three" };
// same as double[]
var arr = new [] { 1.0, 2.0, 3.0 };
```

```
int[] arr = new int[] {1, 6, 3, 3, 9};

for (int i = 0; i < arr.Length; i++)
{
    Console.WriteLine(arr[i]);
}
```

foreach :

```
foreach (int element in arr)
{
    Console.WriteLine(element);
}
```

https://msdn.microsoft.com/en-ca/library/y31yhkeb.aspx

```
unsafe
{
    int length = arr.Length;
    fixed (int* p = arr)
    {
        int* pInt = p;
        while (length-- > 0)
        {
            Console.WriteLine(*pInt);
            pInt++;// move pointer to next element
        }
    }
}
```

:

1
6

9

. 10 10 2 .

```
int[,] arr = new int[10, 10];
```

:

```
int[,,] arr = new int[10, 10, 10];
```

.

```
int[,] arr = new int[4, 2] { {1, 1}, {2, 2}, {3, 3}, {4, 4} };

// Access a member of the multi-dimensional array:
Console.Out.WriteLine(arr[3, 1]);  // 4
```

( ) . . .

. .

8 .

```
int[][] a = new int[8][];
```

[] . .

```
for (int i = 0; i < a.length; i++)
{
    a[i] = new int[10];
}
```

I

. a .

```
for (int i = 0; i < a[2].length; i++)
{
    Console.WriteLine(a[2][i]);
}
```

:

```
a[<row_number>][<column_number>]
```

:

```
a[<row_number>][<column_number>] = <value>
```

: () ( ) . .

---

1 int 5 3 . C# :

```
int[,,][,,,,][] arr = new int[8, 10, 12][,,,,][];
```

CLR , , arr :

```
    arr.GetType().ToString() == "System.Int32[][,,,,][,,]"
```

:

```
    typeof(int[,,][,,,,][]).ToString() == "System.Int32[][,,,,][,,]"
```

```
public static class ArrayHelpers
```

```
{
    public static bool Contains<T>(this T[] array, T[] candidate)
    {
        if (IsEmptyLocate(array, candidate))
            return false;

        if (candidate.Length > array.Length)
            return false;

        for (int a = 0; a <= array.Length - candidate.Length; a++)
        {
            if (array[a].Equals(candidate[0]))
            {
                int i = 0;
                for (; i < candidate.Length; i++)
                {
                    if (false == array[a + i].Equals(candidate[i]))
                        break;
                }
                if (i == candidate.Length)
                    return true;
            }
        }
        return false;
    }

    static bool IsEmptyLocate<T>(T[] array, T[] candidate)
    {
        return array == null
            || candidate == null
            || array.Length == 0
            || candidate.Length == 0
            || candidate.Length > array.Length;
    }
}
```

/// Sample

```
byte[] EndOfStream = Encoding.ASCII.GetBytes("---3141592---");
byte[] FakeReceivedFromStream = Encoding.ASCII.GetBytes("Hello, world!!!---3141592---");
if (FakeReceivedFromStream.Contains(EndOfStream))
{
    Console.WriteLine("Message received");
}
```

:

```
int[] arr = new int[10];
```

`0` 10 (`int`).

`System.Linq` `Enumerable.Repeat` .

1. **"true"** 10 `bool` ,

```
bool[] booleanArray = Enumerable.Repeat(true, 10).ToArray();
```

2. **"100"** 5 `int` ,

```
int[] intArray = Enumerable.Repeat(100, 5).ToArray();
```

3. **"C #"** 5 `string`

```
string[] strArray = Enumerable.Repeat("C#", 5).ToArray();
```

Array `Array.Copy()` 0 `Array.Copy()` :

```
var sourceArray = new int[] { 11, 12, 3, 5, 2, 9, 28, 17 };
var destinationArray= new int[3];
Array.Copy(sourceArray, destinationArray, 3);

// destinationArray will have 11,12 and 3
```

0 `CopyTo()` :

```
var sourceArray = new int[] { 11, 12, 7 };
var destinationArray = new int[6];
sourceArray.CopyTo(destinationArray, 2);

// destinationArray will have 0, 0, 11, 12, 7 and 0
```

`Clone` .

```
var sourceArray = new int[] { 11, 12, 7 };
var destinationArray = (int)sourceArray.Clone();

//destinationArray will be created and will have 11,12,17.
```

`CopyTo` `Clone` ., .

LINQ . 1 100 .

`Enumerable.Range` .

.

```
Enumerable.Range(int start, int count)
```

`count` .

:

```
int[] sequence = Enumerable.Range(1, 100).ToArray();
```

1 - 100 (`[1, 2, 3, ..., 98, 99, 100]`) .

Range IEnumerable<int> LINQ .

```
int[] squares = Enumerable.Range(2, 10).Select(x => x * x).ToArray();
```

4 : [4, 9, 16, ..., 100, 121] 10 .

LINQ IEnumerable IEnumerable .

SequenceEqual true , false .

```
int[] arr1 = { 3, 5, 7 };
int[] arr2 = { 3, 5, 7 };
bool result = arr1.SequenceEqual(arr2);
Console.WriteLine("Arrays equal? {0}", result);
```

.

```
Arrays equal? True
```

IEnumerable <>

IList (ICollection IEnumerable).

1 IList<> IReadOnlyList<> ( ) ., .

```
int[] arr1 = { 3, 5, 7 };
IEnumerable<int> enumerableIntegers = arr1; //Allowed because arrays implement IEnumerable<T>
List<int> listOfIntegers = new List<int>();
listOfIntegers.AddRange(arr1); //You can pass in a reference to an array to populate a List.
```

listOfIntegers List<int> 3, 5 7 List<int> .

IEnumerable<> LINQ arr1.Select(i => 10 * i) : arr1.Select(i => 10 * i) .

: https://riptutorial.com/ko/csharp/topic/1429/

94: LINQ (PLINQ)

- ParallelEnumerable.Aggregate (func)
- ParallelEnumerable.Aggregate (seed, func)
- ParallelEnumerable.Aggregate (seed, updateAccumulatorFunc, combineAccumulatorsFunc, resultSelector)
- ParallelEnumerable.Aggregate (seedFactory, updateAccumulatorFunc, combineAccumulatorsFunc, resultSelector)
- ParallelEnumerable.All ()
- ParallelEnumerable.Any ()
- ParallelEnumerable.Any ()
- ParallelEnumerable.AsEnumerable ()
- ParallelEnumerable.AsOrdered ()
- ParallelEnumerable.AsParallel ()
- ParallelEnumerable.AsSequential ()
- Asynchronous ()
- ParallelEnumerable.Average ()
- ParallelEnumerable.Cast ()
- ParallelEnumerable.Concat (second)
- ParallelEnumerable.Contains (value)
- ParallelEnumerable.Contains (value, comparer)
- ParallelEnumerable.Count ()
- ParallelEnumerable.Count ( )
- ParallelEnumerable.DefaultIfEmpty ()
- ParallelEnumerable.DefaultIfEmpty (defaultValue)
- ParallelEnumerable.Distinct ()
- ParallelEnumerable.Distinct ( )
- ParallelEnumerable.ElementAt (index)
- ParallelEnumerable.ElementAtOrDefault (index)
- ParallelEnumerable.Empty ()
- ParallelEnumerable.Except ()
- ParallelEnumerable.Except (second, comparer)
- ParallelEnumerable.First ()
- ParallelEnumerable.First ()
- ParallelEnumerable.FirstOrDefault ()
- ParallelEnumerable.FirstOrDefault ()
- ParallelEnumerable.ForAll ()
- ParallelEnumerable.GroupBy (keySelector)
- ParallelEnumerable.GroupBy (keySelector, comparer)
- ParallelEnumerable.GroupBy (keySelector, elementSelector)
- ParallelEnumerable.GroupBy (keySelector, elementSelector, comparer)
- ParallelEnumerable.GroupBy (keySelector, resultSelector)
- ParallelEnumerable.GroupBy (keySelector, resultSelector, comparer)
- ParallelEnumerable.GroupBy (keySelector, elementSelector, ruleSelector)
- ParallelEnumerable.GroupBy (keySelector, elementSelector, ruleSelector, comparer)

- GroupJoin (inner, outerKeySelector, innerKeySelector, resultSelector)
- GroupJoin (inner, outerKeySelector, , ,)
- ParallelEnumerable.Intersect ()
- ParallelEnumerable.Intersect (second, comparer)
- ParallelEnumerable.Join (inner, outerKeySelector, innerKeySelector, resultSelector)
- ParallelEnumerable.Join (inner, outerKeySelector, , ,)
- ParallelEnumerable.Last ()
- ParallelEnumerable.Last ()
- ParallelEnumerable.LastOrDefault ()
- ParallelEnumerable.LastOrDefault ()
- ParallelEnumerable.LongCount ()
- ParallelEnumerable.LongCount ()
- ParallelEnumerable.Max ()
- ParallelEnumerable.Max ()
- ParallelEnumerable.Min ()
- ParallelEnumerable.Min ()
- ParallelEnumerable.OfType ()
- ParallelEnumerable.OrderBy (keySelector)
- ParallelEnumerable.OrderBy (keySelector, comparer)
- ParallelEnumerable.OrderByDescending (keySelector)
- ParallelEnumerable.OrderByDescending (keySelector, comparer)
- ParallelEnumerable.Range (,)
- ParallelEnumerable.Repeat (,)
- ParallelEnumerable.Reverse ()
- ParallelEnumerable.Select (selector)
- ParallelEnumerable.SelectMany (selector)
- ParallelEnumerable.SelectMany (collectionSelector, resultSelector)
- ParallelEnumerable.SequenceEqual (second)
- ParallelEnumerable.SequenceEqual (,)
- ParallelEnumerable.Single ()
- ParallelEnumerable.Single ()
- ParallelEnumerable.SingleOrDefault ()
- ParallelEnumerable.SingleOrDefault ( )
- ParallelEnumerable.Skip (count)
- ParallelEnumerable.SkipWhile ()
- ParallelEnumerable.Sum ()
- ParallelEnumerable.Sum ()
- ParallelEnumerable.Take (count)
- ParallelEnumerable.TakeWhile ( )
- ParallelEnumerable.ThenBy (keySelector)
- ParallelEnumerable.ThenBy (keySelector, comparer)
- ParallelEnumerable.ThenByDescending (keySelector)
- ParallelEnumerable.ThenByDescending (keySelector, comparer)
- ParallelEnumerable.ToArray ()
- ParallelEnumerable.ToDictionary (keySelector)
- ParallelEnumerable.ToDictionary (keySelector, comparer)

- ParallelEnumerable.ToDictionary (elementSelector)
- ParallelEnumerable.ToDictionary (elementSelector, comparer)
- ParallelEnumerable.ToList ()
- ParallelEnumerable.ToLookup (keySelector)
- ParallelEnumerable.ToLookup (keySelector, comparer)
- ParallelEnumerable.ToLookup (keySelector, elementSelector)
- ParallelEnumerable.ToLookup (keySelector, elementSelector, comparer)
- ParallelEnumerable.Union ()
- ParallelEnumerable.Union (second, comparer)
- ParallelEnumerable. Where ()
- ParallelEnumerable.WithCancellation (cancellationToken)
- ParallelEnumerable.WithDegreeOfParallelism (degreeOfParallelism)
- ParallelEnumerable.WithExecutionMode (executionMode)
- ParallelEnumerable.WithMergeOptions (mergeOptions)
- ParallelEnumerable.Zip (second, resultSelector)

Examples

PLINQ 1 10,000 . .

```
var sequence = Enumerable.Range(1, 10000);
var evenNumbers = sequence.AsParallel()
                          .Where(x => x % 2 == 0)
                          .ToList();

// evenNumbers = { 4, 26, 28, 30, ... }
// Order will vary with different runs
```

WithDegreeOfParallelism

.

```
var sequence = Enumerable.Range(1, 10000);
var evenNumbers = sequence.AsParallel()
                          .WithDegreeOfParallelism(4)
                          .Where(x => x % 2 == 0);
```

AsOrdered

PLINQ 1 10,000 . `AsOrdered` .

```
var sequence = Enumerable.Range(1, 10000);
var evenNumbers = sequence.AsParallel()
                          .AsOrdered()
                          .Where(x => x % 2 == 0)
                          .ToList();

// evenNumbers = { 2, 4, 6, 8, ..., 10000 }
```

AsUnordered

. AsUnordered .

```
var sequence = Enumerable.Range(1, 10000).Select(x => -1 * x); // -1, -2, ...
var evenNumbers = sequence.AsParallel()
                          .OrderBy(x => x)
                          .Take(5000)
                          .AsUnordered()
                          .Where(x => x % 2 == 0) // This line won't be affected by ordering
                          .ToList();
```

LINQ (PLINQ) : https://riptutorial.com/ko/csharp/topic/3569/-linq--plinq-

95:

. . partial . .

- public **partial** class MyPartialClass {}
- .
- partial .
- . public / protected / private .
- abstract .
- sealed .
- .
- .

Examples

( ) . . .

```
using System;

namespace PartialClassAndMethods
{
    public partial class PartialClass
    {
        public void ExampleMethod() {
            Console.WriteLine("Method call from the first declaration.");
        }
    }

    public partial class PartialClass
    {
        public void AnotherExampleMethod()
        {
            Console.WriteLine("Method call from the second declaration.");
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            PartialClass partial = new PartialClass();
            partial.ExampleMethod(); // outputs "Method call from the first declaration."
            partial.AnotherExampleMethod(); // outputs "Method call from the second
declaration."
        }
    }
```

```
}
```

(-) .

```
using System;

namespace PartialClassAndMethods
{
    public partial class PartialClass // Auto-generated
    {
        partial void PartialMethod();
    }

    public partial class PartialClass // Human-written
    {
        public void PartialMethod()
        {
            Console.WriteLine("Partial method called.");
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            PartialClass partial = new PartialClass();
            partial.PartialMethod(); // outputs "Partial method called."
        }
    }
}
```

.

```
// PartialClass1.cs
public partial class PartialClass : BaseClass {}

// PartialClass2.cs
public partial class PartialClass {}
```

. IDE .

```
// PartialClass1.cs
public partial class PartialClass : BaseClass {}

// PartialClass2.cs
public partial class PartialClass : BaseClass {} // base class here is redundant
```

.

```
// PartialClass1.cs
public partial class PartialClass : BaseClass {} // compiler error

// PartialClass2.cs
public partial class PartialClass : OtherBaseClass {} // compiler error
```

: https://riptutorial.com/ko/csharp/topic/3674/---

96:

Examples

System.String

C # (.NET) System.String . `string` .

System.String . , .

substring, Remove, Replace, `+` .

-

```
string str = "mystring";
string newString = str.Substring(3);
Console.WriteLine(newString);
Console.WriteLine(str);
```

`string mystring .`

`. string .`

"Hello" "world" .

```
string myString = "hello";
myString += " world";
```

`string . .`

`string StringBuilder`

```
StringBuilder myStringBuilder = new StringBuilder("hello");
myStringBuilder.append(" world");
```

`StringBuilder .`

: https://riptutorial.com/ko/csharp/topic/1863/

97: / , ,

.

```
static void Main()
{
    new Program().ProcessDataAsync();
    Console.ReadLine();
}
```

Examples

ASP.NET

ASP.NET . `HttpContext.Current` . .

.

`await` .

```
public async Task<ActionResult> Index()
{
    // Execution on the initially assigned thread
    var products = await dbContext.Products.ToListAsync();

    // Execution resumes on a "random" thread from the pool
    // Execution continues using the original request context.
    return View(products);
}
```

. . .

`async` . `IndexSync()` .

```
public async Task<ActionResult> Index()
{
    // Execution on the initially assigned thread
    List<Product> products = await dbContext.Products.ToListAsync();

    // Execution resumes on a "random" thread from the pool
    return View(products);
}

public ActionResult IndexSync()
{
    Task<ActionResult> task = Index();

    // Block waiting for the result synchronously
    ActionResult result = Task.Result;

    return result;
}
```

`db.Products.ToListAsync()` `db.Products.ToListAsync()` (ASP.NET) .

. `await` .

`Task.Result` `Task.Wait()` ( ) . , .

.

ConfigureAwait

.

`ConfigureAwait(false)` .

```
public async Task<ActionResult> Index()
{
    // Execution on the initially assigned thread
    List<Product> products = await dbContext.Products.ToListAsync().ConfigureAwait(false);

    // Execution resumes on a "random" thread from the pool without the original request
context
    return View(products);
}

public ActionResult IndexSync()
{
    Task<ActionResult> task = Index();

    // Block waiting for the result synchronously
    ActionResult result = Task.Result;

    return result;
}
```

. , ( ).

ASP.NET `await someTask.ConfigureAwait(false);` `await someTask.ConfigureAwait(false);` `await someTask.ConfigureAwait(false);` (: `HttpContext.Current.User` . . `HttpContext.Current` null. :

```
public async Task<ActionResult> Index()
{
    // Contains information about the user sending the request
    var user = System.Web.HttpContext.Current.User;

    using (var client = new HttpClient())
    {
        await client.GetAsync("http://google.com").ConfigureAwait(false);
    }

    // Null Reference Exception, Current is null
    var user2 = System.Web.HttpContext.Current.User;

    return View();
}
```

`ConfigureAwait(true)` (ConfigureAwait) `user` `user2` .

ConfigureAwait(false) .

/

async / await .

.

```
public async Task ProcessDataAsync()
{
    // Start the time intensive method
    Task<int> task = TimeintensiveMethod(@"PATH_TO_SOME_FILE");

    // Control returns here before TimeintensiveMethod returns
    Console.WriteLine("You can read this while TimeintensiveMethod is still running.");

    // Wait for TimeintensiveMethod to complete and get its result
    int x = await task;
    Console.WriteLine("Count: " + x);
}

private async Task<int> TimeintensiveMethod(object file)
{
    Console.WriteLine("Start TimeintensiveMethod.");

    // Do some time intensive calculations...
    using (StreamReader reader = new StreamReader(file.ToString()))
    {
        string s = await reader.ReadToEndAsync();

        for (int i = 0; i < 10000; i++)
            s.GetHashCode();
    }
    Console.WriteLine("End TimeintensiveMethod.");

    // return something as a "result"
    return new Random().Next(100);
}
```

BackgroundWorker

BackgroundWorker .

:

1. BackgroundWorker DoWork .
2. RunWorkerAsync . DoWork DoWorkEventArgs RunWorkerAsync .

DoWork BackgroundWorker . .

- DoWork RunWorkerCompleted .
- ProgressChanged ReportProgress .

```
public void ProcessDataAsync()
{
```

```
    // Start the time intensive method
    BackgroundWorker bw = new BackgroundWorker();
    bw.DoWork += BwDoWork;
    bw.RunWorkerCompleted += BwRunWorkerCompleted;
    bw.RunWorkerAsync(@"PATH_TO_SOME_FILE");

    // Control returns here before TimeintensiveMethod returns
    Console.WriteLine("You can read this while TimeintensiveMethod is still running.");
}

// Method that will be called after BwDoWork exits
private void BwRunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)
{
    // we can access possible return values of our Method via the Parameter e
    Console.WriteLine("Count: " + e.Result);
}

// execution of our time intensive Method
private void BwDoWork(object sender, DoWorkEventArgs e)
{
    e.Result = TimeintensiveMethod(e.Argument);
}

private int TimeintensiveMethod(object file)
{
    Console.WriteLine("Start TimeintensiveMethod.");

    // Do some time intensive calculations...
    using (StreamReader reader = new StreamReader(file.ToString()))
    {
        string s = reader.ReadToEnd();

       for (int i = 0; i < 10000; i++)
            s.GetHashCode();
    }
    Console.WriteLine("End TimeintensiveMethod.");

    // return something as a "result"
    return new Random().Next(100);
}
```

Task Task .

Task.Run() .

```
public void ProcessDataAsync()
{
    // Start the time intensive method
    Task<int> t = Task.Run(() => TimeintensiveMethod(@"PATH_TO_SOME_FILE"));

    // Control returns here before TimeintensiveMethod returns
    Console.WriteLine("You can read this while TimeintensiveMethod is still running.");

    Console.WriteLine("Count: " + t.Result);
}

private int TimeintensiveMethod(object file)
{
    Console.WriteLine("Start TimeintensiveMethod.");
```

```
    // Do some time intensive calculations...
    using (StreamReader reader = new StreamReader(file.ToString()))
    {
        string s = reader.ReadToEnd();

        for (int i = 0; i < 10000; i++)
            s.GetHashCode();
    }
    Console.WriteLine("End TimeintensiveMethod.");

    // return something as a "result"
    return new Random().Next(100);
}
```

Thread .

```
public async void ProcessDataAsync()
{
    // Start the time intensive method
    Thread t = new Thread(TimeintensiveMethod);

    // Control returns here before TimeintensiveMethod returns
    Console.WriteLine("You can read this while TimeintensiveMethod is still running.");
}

private void TimeintensiveMethod()
{
    Console.WriteLine("Start TimeintensiveMethod.");

    // Do some time intensive calculations...
    using (StreamReader reader = new StreamReader(@"PATH_TO_SOME_FILE"))
    {
        string v = reader.ReadToEnd();

        for (int i = 0; i < 10000; i++)
            v.GetHashCode();
    }
    Console.WriteLine("End TimeintensiveMethod.");
}
```

Thread void TimeIntensiveMethod .

Thread .

```
int ret;
Thread t= new Thread(() =>
{
    Console.WriteLine("Start TimeintensiveMethod.");

    // Do some time intensive calculations...
    using (StreamReader reader = new StreamReader(file))
    {
        string s = reader.ReadToEnd();

        for (int i = 0; i < 10000; i++)
            s.GetHashCode();
    }
```

```
    Console.WriteLine("End TimeintensiveMethod.");

    // return something to demonstrate the coolness of await-async
    ret = new Random().Next(100);
});

t.Start();
t.Join(1000);
Console.Writeline("Count: " + ret);
```

" "

(:) . .

```
public static class TaskExtensions
{
    public static async void RunAndForget(
        this Task task, Action<Exception> onException = null)
    {
        try
        {
            await task;
        }
        catch (Exception ex)
        {
            onException?.Invoke(ex);
        }
    }
}
```

. async / await .

:

```
var task = Task.FromResult(0); // Or any other task from e.g. external lib.
task.RunAndForget(
    e =>
    {
        // Something went wrong, handle it.
    });
```

/ , , : https://riptutorial.com/ko/csharp/topic/3824/----------

98:

Examples

async / await

.

```
async Task Foo()
{
    Bar();
    await Baz();
    Qux();
}
```

.

```
Task Foo()
{
    Bar();
    Task t = Baz();
    var context = SynchronizationContext.Current;
    t.ContinueWith(task) =>
    {
        if (context == null)
            Qux();
        else
            context.Post((obj) => Qux(), null);
    }, TaskScheduler.Current);

    return t;
}
```

`async` / `await` . , UI, .

.

`ConfigureAwait` .

```
async Task() Foo()
{
    await Task.Run(() => Console.WriteLine("Test"));
}

. . .

Foo().ConfigureAwait(false);
```

ConfigureAwait SynchronizationContext . flowContext false SynchronizationContext .

SynchronizationContext .

SynchronizationContext ?

.

```
private void button1_Click(object sender, EventArgs e)
{
    label1.Text = RunTooLong();
}
```

`RunTooLong` UI `RunTooLong` . .

.

```
private void button1_Click(object sender, EventArgs e)
{
    Task.Run(() => label1.Text = RunTooLong());
}
```

UI UI .

```
private void button1_Click(object sender, EventArgs e)
{
    Task.Run(() =>
    {
        var label1Text = RunTooLong();

        if (label1.InvokeRequired)
            lable1.BeginInvoke((Action) delegate() { label1.Text = label1Text; });
        else
            label1.Text = label1Text;
    });
}
```

. `SynchronizationContext.Post` .

```
private void button1_Click(object sender, EventArgs e)
{
    Task.Run(() =>
    {
        var label1Text = RunTooLong();
        SynchronizationContext.Current.Post((obj) =>
        {
            label1.Text = label1    Text);
        }, null);
    });
}
```

: https://riptutorial.com/ko/csharp/topic/7381/----

99:

C # async I / O (: , ...) . await await .

async void , Task Task<T> .

Task void . Task<T> T .

async void Task Task<T> . async void ed await . async void .

async / await async . (:) MoveNext() .

Examples

```
public async Task<JobResult> GetDataFromWebAsync()
{
  var nextJob = await _database.GetNextJobAsync();
  var response = await _httpClient.GetAsync(nextJob.Uri);
  var pageContents = await response.Content.ReadAsStringAsync();
  return await _database.SaveJobResultAsync(pageContents);
}
```

await . . . "". .

/ /

6.0

C # 6.0 catch finally await .

```
try {
   var client = new AsyncClient();
   await client.DoSomething();
} catch (MyException ex) {
   await client.LogExceptionAsync();
   throw;
} finally {
   await client.CloseAsync();
}
```

5.0 6.0

C # 6.0 . 6.0 Null Null .

```
AsynClient client;
MyException caughtException;
try {
     client = new AsyncClient();
     await client.DoSomething();
} catch (MyException ex) {
     caughtException = ex;
```

```
}

if (client != null) {
    if (caughtException != null) {
       await client.LogExceptionAsync();
    }
    await client.CloseAsync();
    if (caughtException != null) throw caughtException;
}
```

`async` (: `Task.Run`) try / catch throw . . Visual Studio " " .

4.5 Web.config .

web.config system.web.httpRuntime async 4.5 .

```
<httpRuntime targetFramework="4.5" />
```

Async Await 4.5 ASP.NET . / . HttpContext null . WebApi HttpContext.Current .

.

```
public async Task RunConcurrentTasks()
{
    var firstTask = DoSomethingAsync();
    var secondTask = DoSomethingElseAsync();

    await firstTask;
    await secondTask;
}
```

`Task.WhenAll` `Task` . .

```
public async Task RunConcurrentTasks()
{
    var firstTask = DoSomethingAsync();
    var secondTask = DoSomethingElseAsync();

    await Task.WhenAll(firstTask, secondTask);
}
```

.

```
List<Task> tasks = new List<Task>();
while (something) {
    // do stuff
    Task someAsyncTask = someAsyncMethod();
    tasks.Add(someAsyncTask);
}

await Task.WhenAll(tasks);
```

Task.WhenAll . .

```
var task1 = SomeOpAsync();
var task2 = SomeOtherOpAsync();

await Task.WhenAll(task1, task2);

var result = await task2;
```

Task.WhenAny , Task.WhenAll , .

```
public async Task RunConcurrentTasksWhenAny()
{
    var firstTask = TaskOperation("#firstTask executed");
    var secondTask = TaskOperation("#secondTask executed");
    var thirdTask = TaskOperation("#thirdTask executed");
    await Task.WhenAny(firstTask, secondTask, thirdTask);
}
```

Task RunConcurrentTasksWhenAny firstTask , secondTask , thirdTask .

async await async .

await async .

. async async await .

await ; .

: (void) .

, 'suspends' . . , await . . await . .

, :

```
public async Task<TResult> DoIt()
{
    // do something and acquire someTask of type Task<TSomeResult>
    var awaitedResult = await someTask;
    // ... do something more and produce result of type TResult
    return result;
}
```

.

```
public Task<TResult> DoIt()
{
    // ...
    return someTask.ContinueWith(task => {
        var result = ((Task<TSomeResult>)task).Result;
        return DoIt_Continuation(result);
    });
}

private TResult DoIt_Continuation(TSomeResult awaitedResult)
{
    // ...
```

```
}
```

.

```
await Task.Run(() => YourSyncMethod());
```

UI UI .

. **( ) .** async - await . . " .

: DoIt_Continuation ?

await . , UI WinForms WPF . Task.ConfigureAwait() .

```
await Task.Run(() => YourSyncMethod()).ConfigureAwait(continueOnCapturedContext: false);
```

await .

- .
- .
- catch / handling .

Task .

```
public async Task<User> GetUserAsync(int id)
{
    var lookupKey = "Users" + id;

    return await dataStore.GetByKeyAsync(lookupKey);
}
```

GetByKeyAsync GetUserAsync (Task<User>) .

```
public Task<User> GetUserAsync(int id)
{
    var lookupKey = "Users" + id;

    return dataStore.GetByKeyAsync(lookupKey);
}
```

async . GetByKeyAsync Task , ed await .

: Task Task Task throw .

```
public Task SaveAsync()
{
    try {
        return dataStore.SaveChangesAsync();
    }
    catch(Exception ex)
    {
        // this will never be called
```

```
        logger.LogException(ex);
    }
}

// Some other code calling SaveAsync()

// If exception happens, it will be thrown here, not inside SaveAsync()
await SaveAsync();
```

.

.

. async " " . Windows Forms .

```
private async Task<bool> TryThis()
{
    Trace.TraceInformation("Starting TryThis");
    await Task.Run(() =>
    {
        Trace.TraceInformation("In TryThis task");
        for (int i = 0; i < 100; i++)
        {
            // This runs successfully - the loop runs to completion
            Trace.TraceInformation("For loop " + i);
            System.Threading.Thread.Sleep(10);
        }
    });

    // This never happens due to the deadlock
    Trace.TraceInformation("About to return");
    return true;
}

// Button click event handler
private void button1_Click(object sender, EventArgs e)
{
    // .Result causes this to block on the asynchronous call
    bool result = TryThis().Result;
    // Never actually gets here
    Trace.TraceInformation("Done with result");
}
```

. TryThis() "" .

.

```
private async void button1_Click(object sender, EventArgs e)
{
  bool result = await TryThis();
  Trace.TraceInformation("Done with result");
}
```

: async void async void .

/ .

.

```
public async Task MethodA()
{
     await MethodB();
     // Do other work
}

public async Task MethodB()
{
     await MethodC();
     // Do other work
}

public async Task MethodC()
{
     // Or await some other async work
     await Task.Delay(100);
}
```

```
public void MethodA()
{
     MethodB();
     // Do other work
}

public void MethodB()
{
     MethodC();
     // Do other work
}

public void MethodC()
{
     Thread.Sleep(100);
}
```

async / await . I / O . , `MethodA()` , `MethodB()` `MethodC()` .

: https://riptutorial.com/ko/csharp/topic/48/--

100:

.

? .

.

PC :

IP = `192.168.1.115`

1234 .

```
Port 1234   192.168.1.115
```

IP. IP . IP .

```
 _connectingSocket.Connect(new IPEndPoint(IPAddress.Parse("10.10.10.10"), 1234));
```

. `10.10.10.10:1234` : `10.10.10.10:1234` .

IP `192.168.1.178` .

:

. .

. .

Examples

(/) .

. .

```
class Listener
{
    public Socket ListenerSocket; //This is the socket that will listen to any incoming
connections
    public short Port = 1234; // on this port we will listen

    public Listener()
    {
        ListenerSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
    }
 }
```

Listener . SocketType.Stream Tcp . .

.

.

1. ListenerSocket.Bind ();

   IPEndPoint . .

2. ListenerSocket.Listen (10);

   backlog .

3. ListenerSocket.BeginAccept ();

   . AcceptCallBack .

```
    public void StartListening()
    {
        try
        {
                MessageBox.Show($"Listening started port:{Port} protocol type:
{ProtocolType.Tcp}");
                ListenerSocket.Bind(new IPEndPoint(IPAddress.Any, Port));
                ListenerSocket.Listen(10);
                ListenerSocket.BeginAccept(AcceptCallback, ListenerSocket);
        }
        catch(Exception ex)
        {
            throw new Exception("listening error" + ex);
        }
    }
```

:

.

1. ListenerSocket.EndAccept ()

   `Listener.BeginAccept()` . . `The EndAccept()` IAsyncResult . . .

2. `ClientController.AddClient()`

   `EndAccept()` (*ClientController* `EndAccept()` .

3. ListenerSocket.BeginAccept ()

   . . int .

```
    public void AcceptCallback(IAsyncResult ar)
    {
        try
        {
            Console.WriteLine($"Accept CallBack port:{Port} protocol type:
{ProtocolType.Tcp}");
            Socket acceptedSocket = ListenerSocket.EndAccept(ar);
```

```
            ClientController.AddClient(acceptedSocket);

            ListenerSocket.BeginAccept(AcceptCallback, ListenerSocket);
        }
        catch (Exception ex)
        {
            throw new Exception("Base Accept error"+ ex);
        }
    }
```

Listening Socket , ?

Socket receive .

```
    public class ReceivePacket
    {
        private byte[] _buffer;
        private Socket _receiveSocket;

        public ReceivePacket(Socket receiveSocket)
        {
           _receiveSocket = receiveSocket;
        }
    }
```

4 (Int32) {Lenght, } . 4 .

BeginReceive () . `ReceiveCallback` .

```
    public void StartReceiving()
    {
        try
        {
            _buffer = new byte[4];
            _receiveSocket.BeginReceive(_buffer, 0, _buffer.Length, SocketFlags.None,
ReceiveCallback, null);
        }
        catch {}
    }

    private void ReceiveCallback(IAsyncResult AR)
    {
        try
        {
            // if bytes are less than 1 takes place when a client disconnect from the server.
            // So we run the Disconnect function on the current client
            if (_receiveSocket.EndReceive(AR) > 1)
            {
                // Convert the first 4 bytes (int 32) that we received and convert it to an
Int32 (this is the size for the coming data).
                _buffer = new byte[BitConverter.ToInt32(_buffer, 0)];
                // Next receive this data into the buffer with size that we did receive before
                _receiveSocket.Receive(_buffer, _buffer.Length, SocketFlags.None);
                // When we received everything its onto you to convert it into the data that
you've send.
                // For example string, int etc... in this example I only use the
implementation for sending and receiving a string.
```

```
                // Convert the bytes to string and output it in a message box
                string data = Encoding.Default.GetString(_buffer);
                MessageBox.Show(data);
                // Now we have to start all over again with waiting for a data to come from
the socket.
                StartReceiving();
            }
            else
            {
                Disconnect();
            }
        }
        catch
        {
            // if exeption is throw check if socket is connected because than you can
startreive again else Dissconect
            if (!_receiveSocket.Connected)
            {
                Disconnect();
            }
            else
            {
                StartReceiving();
            }
        }
    }

    private void Disconnect()
    {
        // Close connection
        _receiveSocket.Disconnect(true);
        // Next line only apply for the server side receive
        ClientController.RemoveClient(_clientId);
        // Next line only apply on the Client Side receive
        Here you want to run the method TryToConnect()
    }
```

. . .

```
Listener listener = new Listener();
listener.StartListening();
```

```
    class Client
    {
        public Socket _socket { get; set; }
        public ReceivePacket Receive { get; set; }
        public int Id { get; set; }

        public Client(Socket socket, int id)
        {
            Receive = new ReceivePacket(socket, id);
            Receive.StartReceiving();
            _socket = socket;
            Id = id;
        }
    }

     static class ClientController
     {
```

```
        public static List<Client> Clients = new List<Client>();

        public static void AddClient(Socket socket)
        {
            Clients.Add(new Client(socket,Clients.Count));
        }

        public static void RemoveClient(int id)
        {
            Clients.RemoveAt(Clients.FindIndex(x => x.Id == id));
        }
    }
```

. . Connector :

```
class Connector
{
    private Socket _connectingSocket;
}
```

TryToConnect ().

.

1. .
2. .
3. 1 . DOS XD .
4. Connect () . . Connect CallBack .
5. 1234 PC .

```
 public void TryToConnect()
 {
     _connectingSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);

      while (!_connectingSocket.Connected)
      {
          Thread.Sleep(1000);

          try
          {
              _connectingSocket.Connect(new IPEndPoint(IPAddress.Parse("127.0.0.1"),
1234));
          }
          catch { }
      }
      SetupForReceiveing();
     }
 }

 private void SetupForReceiveing()
 {
```

```
    // View Client Class bottom of Client Example
     Client.SetClient(_connectingSocket);
     Client.StartReceiving();
  }
```

. .

```
public class SendPacket
{
    private Socket _sendSocked;

    public SendPacket(Socket sendSocket)
    {
        _sendSocked = sendSocket;
    }

    public void Send(string data)
    {
        try
        {
            /* what hapends here:
                 1. Create a list of bytes
                 2. Add the length of the string to the list.
                    So if this message arrives at the server we can easily read the length of
the coming message.
                 3. Add the message(string) bytes
            */

            var fullPacket = new List<byte>();
            fullPacket.AddRange(BitConverter.GetBytes(data.Length));
            fullPacket.AddRange(Encoding.Default.GetBytes(data));

            /* Send the message to the server we are currently connected to.
            Or package stucture is {length of data 4 bytes (int32), actual data}*/
            _sendSocked.Send(fullPacket.ToArray());
        }
        catch (Exception ex)
        {
            throw new Exception();
        }
    }
```

.

```
    private void ConnectClick(object sender, EventArgs e)
    {
        Connector tpp = new Connector();
        tpp.TryToConnect();
    }

    private void SendClick(object sender, EventArgs e)
    {
        Client.SendString("Test data from client");
    }
```

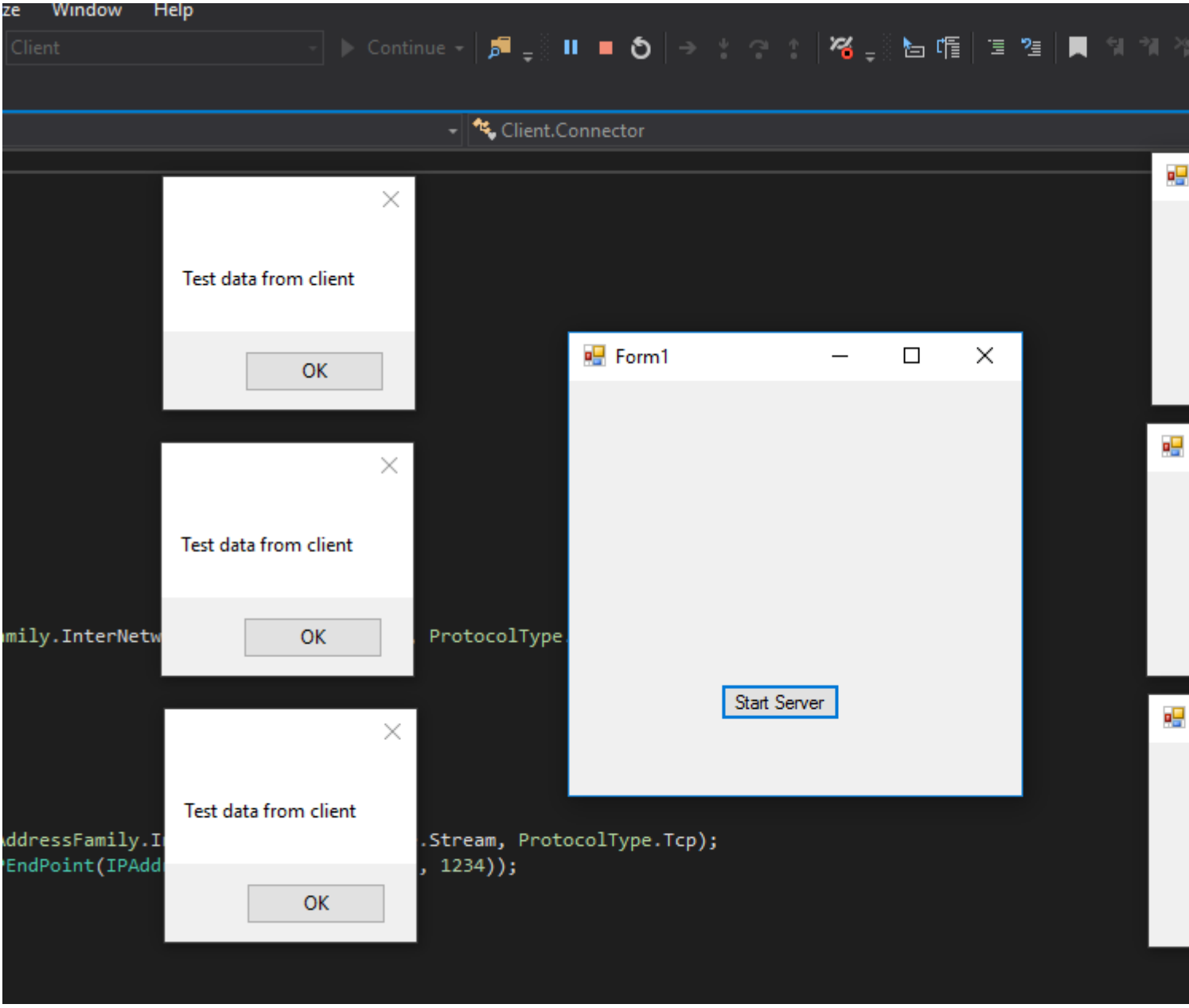
```
    public static void SetClient(Socket socket)
    {
```

```
        Id = 1;
        Socket = socket;
        Receive = new ReceivePacket(socket, Id);
        SendPacket = new SendPacket(socket);
    }
```

.

. . . . .

:



: https://riptutorial.com/ko/csharp/topic/9638/-

101:

PInvoke .

Examples

```
public class NetworkConnection : IDisposable
    {
        string _networkName;

        public NetworkConnection(string networkName,
            NetworkCredential credentials)
        {
            _networkName = networkName;

            var netResource = new NetResource()
            {
                Scope = ResourceScope.GlobalNetwork,
                ResourceType = ResourceType.Disk,
                DisplayType = ResourceDisplaytype.Share,
                RemoteName = networkName
            };

            var userName = string.IsNullOrEmpty(credentials.Domain)
                ? credentials.UserName
                : string.Format(@"{0}\{1}", credentials.Domain, credentials.UserName);

            var result = WNetAddConnection2(
                netResource,
                credentials.Password,
                userName,
                0);

            if (result != 0)
            {
                throw new Win32Exception(result);
            }
        }

    ~NetworkConnection()
    {
        Dispose(false);
    }

    public void Dispose()
    {
        Dispose(true);
        GC.SuppressFinalize(this);
    }

    protected virtual void Dispose(bool disposing)
    {
        WNetCancelConnection2(_networkName, 0, true);
    }

    [DllImport("mpr.dll")]
    private static extern int WNetAddConnection2(NetResource netResource,
```

```
        string password, string username, int flags);

    [DllImport("mpr.dll")]
    private static extern int WNetCancelConnection2(string name, int flags,
        bool force);
}

[StructLayout(LayoutKind.Sequential)]
public class NetResource
{
    public ResourceScope Scope;
    public ResourceType ResourceType;
    public ResourceDisplaytype DisplayType;
    public int Usage;
    public string LocalName;
    public string RemoteName;
    public string Comment;
    public string Provider;
}

public enum ResourceScope : int
{
    Connected = 1,
    GlobalNetwork,
    Remembered,
    Recent,
    Context
};

public enum ResourceType : int
{
    Any = 0,
    Disk = 1,
    Print = 2,
    Reserved = 8,
}

public enum ResourceDisplaytype : int
{
    Generic = 0x0,
    Domain = 0x01,
    Server = 0x02,
    Share = 0x03,
    File = 0x04,
    Group = 0x05,
    Network = 0x06,
    Root = 0x07,
    Shareadmin = 0x08,
    Directory = 0x09,
    Tree = 0x0a,
    Ndscontainer = 0x0b
}
```

: https://riptutorial.com/ko/csharp/topic/9627/------

102:

C # Win32 API

Windows Win32 API . API . .

Windows API . API pinvoke .

Examples

C ++ DLL

C ++ DLL . "myDLL.dll" C ++ `add` .

```
extern "C" __declspec(dllexport) int __stdcall add(int a, int b)
{
    return a + b;
}
```

C # .

```
class Program
{
    // This line will import the C++ method.
    // The name specified in the DllImport attribute must be the DLL name.
    // The names of parameters are unimportant, but the types must be correct.
    [DllImport("myDLL.dll")]
    private static extern int add(int left, int right);

    static void Main(string[] args)
    {
        //The extern method can be called just as any other C# method.
        Console.WriteLine(add(1, 2));
    }
}
```

`extern "C" __stdcall` C ++ .

---

extern C # DLL . DLL , stackoverflow .

com

```
using System;
using System.Runtime.InteropServices;

namespace ComLibrary
{
    [ComVisible(true)]
    public interface IMainType
```

```
    {
        int GetInt();

        void StartTime();

        int StopTime();
    }

    [ComVisible(true)]
    [ClassInterface(ClassInterfaceType.None)]
    public class MainType : IMainType
    {
        private Stopwatch stopWatch;

        public int GetInt()
        {
            return 0;
        }

        public void StartTime()
        {
            stopWatch= new Stopwatch();
            stopWatch.Start();
        }

        public int StopTime()
        {
            return (int)stopWatch.ElapsedMilliseconds;
        }
    }
}
```

C ++

C ++ . . `int add(int a, int b)` `add` C # (`int add(int a, int b)` `?add@@YAHHH@Z` , `_add@8` .

.

- `extern "C"` C C :

```
extern "C" __declspec(dllexport) int __stdcall add(int a, int b)
```

```
[DllImport("myDLL.dll")]
```

(`_add@8`), `StdCall` + `extern "C"` C # .

- `myDLL.def` :

```
EXPORTS
  add
```

```
int __stdcall add(int a, int b)
```

```
[DllImport("myDLL.dll")]
```

`add` .

- . DLL .

```
__declspec(dllexport) int __stdcall add(int a, int b)
```

```
[DllImport("myDLL.dll", EntryPoint = "?add@@YGHHH@Z")]
```

. , . C ++ `Cdecl` C # Windows API `StdCall` . .

- C ++ `StdCall` :

```
extern "C" __declspec(dllexport) int __stdcall add(int a, int b)
```

```
[DllImport("myDLL.dll")]
```

- C # `Cdecl` :

```
extern "C" __declspec(dllexport) int /*__cdecl*/ add(int a, int b)
```

```
[DllImport("myDLL.dll", CallingConvention = CallingConvention.Cdecl)]
```

`Cdecl` `Cdecl` .

```
__declspec(dllexport) int add(int a, int b)
```

```
[DllImport("myDLL.dll", CallingConvention = CallingConvention.Cdecl,
           EntryPoint = "?add@@YAHHH@Z")]
```

- **thiscall** (**__thiscall**) .
- **thiscall** (**__thiscall**) .

DLL

`DllImport` dll . dll `LoadLibrary()` , `GetProcAddress()` `FreeLibrary()` Windows API . .

`GetProcAddress()` `Marshal.GetDelegateForFunctionPointer()` .

`myDLL.dll` `myDLL.dll` .

```
class Program
{
    // import necessary API as shown in other examples
    [DllImport("kernel32.dll", SetLastError = true)]
    public static extern IntPtr LoadLibrary(string lib);
```

```
    [DllImport("kernel32.dll", SetLastError = true)]
    public static extern void FreeLibrary(IntPtr module);
    [DllImport("kernel32.dll", SetLastError = true)]
    public static extern IntPtr GetProcAddress(IntPtr module, string proc);

    // declare a delegate with the required signature
    private delegate int AddDelegate(int a, int b);

    private static void Main()
    {
        // load the dll
        IntPtr module = LoadLibrary("myDLL.dll");
        if (module == IntPtr.Zero) // error handling
        {
            Console.WriteLine($"Could not load library: {Marshal.GetLastWin32Error()}");
            return;
        }

        // get a "pointer" to the method
        IntPtr method = GetProcAddress(module, "add");
        if (method == IntPtr.Zero) // error handling
        {
            Console.WriteLine($"Could not load method: {Marshal.GetLastWin32Error()}");
            FreeLibrary(module);  // unload library
            return;
        }

        // convert "pointer" to delegate
        AddDelegate add = (AddDelegate)Marshal.GetDelegateForFunctionPointer(method,
typeof(AddDelegate));

        // use function
        int result = add(750, 300);

        // unload library
        FreeLibrary(module);
    }
}
```

Win32

interop **GetLastError** API API .

DllImport SetLastError

SetLastError = true

SetLastError (Win32 API) .

SetLastError = false

SetLastError (Win32 API) .

- SetLastError false ().
- Marshal.GetLastWin32Error .

:

```
[DllImport("kernel32.dll", SetLastError=true)]
public static extern IntPtr OpenMutex(uint access, bool handle, string lpName);
```

GetLastError **ERROR_FILE_NOT_FOUND** .

```
var lastErrorCode = Marshal.GetLastWin32Error();

if (lastErrorCode == (uint)ERROR_FILE_NOT_FOUND)
{
    //Deal with error
}
```

.

https://msdn.microsoft.com/en-us/library/windows/desktop/ms681382(v=vs.85).aspx

GetLastError API

GetLastError API .

```
[DllImport("coredll.dll", SetLastError=true)]
static extern Int32 GetLastError();
```

- Win32 API **Marshal.GetLastWin32Error** .

.

SetLastError Win32 CLR **SetLastError** Win32 . . **GetLastError** .

SetLastError = true Win32 CLR .

GC (Garbage Collector) .

GC . **GC** (defragmintation) .

GC / **GC** Inerop ( ) . .

.

GC .

Gc

Gc.Alloc .

```
GCHandle handle = GCHandle.Alloc(yourObject, GCHandleType.Pinned);
```

- **GCHandle** **GC** .

:

```
[DllImport("kernel32.dll", SetLastError = true)]
public static extern void EnterCriticalSection(IntPtr ptr);

[DllImport("kernel32.dll", SetLastError = true)]
public static extern void LeaveCriticalSection(IntPtr ptr);

public void EnterCriticalSection(CRITICAL_SECTION section)
{
    try
    {
        GCHandle handle = GCHandle.Alloc(section, GCHandleType.Pinned);
        EnterCriticalSection(handle.AddrOfPinnedObject());
        //Do Some Critical Work
        LeaveCriticalSection(handle.AddrOfPinnedObject());
    }
    finaly
    {
        handle.Free()
    }
}
```

- ( ) **GcHandle** .
- **GcHandle** . (: finaly).

Marshal **PtrToStructure** .

PtrToStructure .

PtrToStructure :

```
public static T PtrToStructure<T>(IntPtr ptr);
```

T - .

ptr - .

:

```
NATIVE_STRUCT result = Marshal.PtrToStructure<NATIVE_STRUCT>(ptr);
```

- .

```
 T Read<T>(byte[] buffer)
    {
        T result = default(T);

        var gch = GCHandle.Alloc(buffer, GCHandleType.Pinned);

        try
        {
            result = Marshal.PtrToStructure<T>(gch.AddrOfPinnedObject());
        }
        finally
```

```
        {
            gch.Free();
        }

        return result;
    }
```

: https://riptutorial.com/ko/csharp/topic/3278/-

103:

. .

/ (Destructors / Finalizers) . C # / .

C # C ++ Finalizers . `Object.Finalize()` .

. .

(, p / Invoke ( )) . IDisposable, Dispose `using` .

(: ?)

Examples

:

```
public class Animal
{
}
```

.

```
public class Animal
{
    public Animal() {}
}
```

. :

```
public class Animal
{
    public Animal(string name) {}
}
```

`Animal` .

```
// This is valid
var myAnimal = new Animal("Fluffy");
// This fails to compile
var unnamedAnimal = new Animal();
```

.

'Animal' 0 .

.

```
public class Animal
{

    public Animal() {} //Equivalent to a default constructor.
    public Animal(string name) {}
}
```

. , Creature :

```
public class Creature
{
    public Creature(Genus genus) {}
}
```

Animal class Animal : Creature {} .

```
public class Animal
{
    public string Name { get; set; }

    public Animal() : this("Dog")
    {
    }

    public Animal(string name)
    {
        Name = name;
    }
}

var dog = new Animal();      // dog.Name will be set to "Dog" by default.
var cat = new Animal("Cat"); // cat.Name is "Cat", the empty constructor is not called.
```

. . .

- .,
-

:

```
class Animal
{
    // * A static constructor is executed only once,
    //   when a class is first accessed.
    // * A static constructor cannot have any access modifiers
    // * A static constructor cannot have any parameters
    static Animal()
    {
        Console.WriteLine("Animal initialized");
    }

    // Instance constructor, this is executed every time the class is created
    public Animal()
    {
        Console.WriteLine("Animal created");
    }
```

```
    public static void Yawn()
    {
        Console.WriteLine("Yawn!");
    }
}

var turtle = new Animal();
var giraffe = new Animal();
```

:

x . .

```
Animal.Yawn();
```

.

!

.

:

```
public class SessionManager
{
    public static SessionManager Instance;

    static SessionManager()
    {
        Instance = new SessionManager();
    }
}
```

. , Mammal Animal , Animal (A) Mammal .

.

```
public class Animal
{
    public Animal() { Console.WriteLine("An unknown animal gets born."); }
    public Animal(string name) { Console.WriteLine(name + " gets born"); }
}

public class Mammal : Animal
{
    public Mammal(string name)
    {
        Console.WriteLine(name + " is a mammal.");
    }
```

```
}
```

```
new Mammal("George the Cat")  Mammal
```

.
George the Cat .

. : base(args)

```
public class Mammal : Animal
{
    public Mammal(string name) : base(name)
    {
        Console.WriteLine(name + " is a mammal.");
    }
}
```

```
new Mammal("George the Cat")   .
```

.
George the Cat .

. , .

```
class TheBaseClass
{
    ~TheBaseClass()
    {
        Console.WriteLine("Base class finalized!");
    }
}

class TheDerivedClass : TheBaseClass
{
    ~TheDerivedClass()
    {
        Console.WriteLine("Derived class finalized!");
    }
}

//Don't assign to a variable
//to make the object unreachable
new TheDerivedClass();

//Just to make the example work;
//this is otherwise NOT recommended!
GC.Collect();

//Derived class finalized!
//Base class finalized!
```

```
public class SingletonClass
{
    public static SingletonClass Instance { get; } = new SingletonClass();
```

```
    private SingletonClass()
    {
        // Put custom constructor code here
    }
}
```

private `SingletonClass` . `SingletonClass` static `SingletonClass.Instance` .

`Instance` C # . .NET `Instance` . .

`Singleton` AppDomain .

`Instance` . , . . JIT `Instance` ( ) . .

.

`RuntimeHelpers` .

```
using System.Runtime.CompilerServices;
// ...
RuntimeHelpers.RunClassConstructor(typeof(Foo).TypeHandle);
```

: (:) .

: UI .

C # C ++ (OK, C ++). :

```
abstract class Base
{
    protected Base()
    {
        _obj = CreateAnother();
    }

    protected virtual AnotherBase CreateAnother()
    {
        return new AnotherBase();
    }

    private readonly AnotherBase _obj;
}

sealed class Derived : Base
{
    public Derived() { }

    protected override AnotherBase CreateAnother()
    {
        return new AnotherDerived();
    }
}

var test = new Derived();
// test._obj is AnotherDerived
```

C ++ . !

: ( ). (StyleCop). .

generic generic .

```
class Animal<T>
{
    static Animal()
    {
        Console.WriteLine(typeof(T).FullName);
    }

    public static void Yawn() { }
}

Animal<Object>.Yawn();
Animal<String>.Yawn();
```

.

System.Object
System.String

See also ?

throw . AppDomain . `TypeInitializationException` .

```
public class Animal
{
    static Animal()
    {
        Console.WriteLine("Static ctor");
        throw new Exception();
    }

    public static void Yawn() {}
}

try
{
    Animal.Yawn();
}
catch (Exception e)
{
    Console.WriteLine(e.ToString());
}

try
{
    Animal.Yawn();
}
catch (Exception e)
{
    Console.WriteLine(e.ToString());
}
```

.

ctor

System.TypeInitializationException : 'Animal' . ---> System.Exception : 'System.Exception' .

[...]

System.TypeInitializationException : 'Animal' . ---> System.Exception : 'System.Exception' .

.

?

```
public class TestClass
{
    public int TestProperty { get; set; } = 2;

    public TestClass()
    {
        if (TestProperty == 1)
        {
            Console.WriteLine("Shall this be executed?");
        }

        if (TestProperty == 2)
        {
            Console.WriteLine("Or shall this be executed");
        }
    }
}

var testInstance = new TestClass() { TestProperty = 1 };
```

, `TestProperty 1` ' ?

:

```
var testInstance = new TestClass() {TestProperty = 1};
```

. C # 6.0 ' .

```
public class TestClass
{
    public int TestProperty { get; set; } = 2;

    public TestClass()
    {
    }
}
```

.

.

```
public class TestClass
{
    public int TestProperty { get; set; } = 2;

    public TestClass()
    {
        if (TestProperty == 1)
        {
            Console.WriteLine("Shall this be executed?");
        }

        if (TestProperty == 2)
        {
            Console.WriteLine("Or shall this be executed");
        }
    }
}

static void Main(string[] args)
{
    var testInstance = new TestClass() { TestProperty = 1 };
    Console.WriteLine(testInstance.TestProperty); //resulting in 1
}
```

:

```
"Or shall this be executed"
"1"
```

:

`TestProperty  2  TestClass`

```
"Or shall this be executed"
```

`new TestClass() { TestProperty = 1 }  TestProperty  1`
`Console.WriteLine(testInstance.TestProperty)   TestProperty  .`

```
"1"
```

: https://riptutorial.com/ko/csharp/topic/25/---

104:

- (a + b) //
- (a + b) //
- {c = a + b; c + = 5; } //
- {c = a + b; c + = 5; } //

Examples

C # . . .

```
short m = 32767;
short n = 32767;
int result1 =  checked((short)(m + n));   //will throw an OverflowException
int result2 =  unchecked((short)(m + n)); // will return -2
```

.

() .

```
short m = 32767;
short n = 32767;
checked
{
    int result1 = (short)(m + n); //will throw an OverflowException
}
unchecked
{
    int result2 = (short)(m + n); // will return -2
}
```

: https://riptutorial.com/ko/csharp/topic/2394/---

105:

. , . :

- . . .
- . .

vs (`get`) (`set`) . .Net . .Net . .

Examples

```
public class Person
{
    //Id property can be read by other classes, but only set by the Person class
    public int Id {get; private set;}
    //Name property can be retrieved or assigned
    public string Name {get; set;}

    private DateTime dob;
    //Date of Birth property is stored in a private variable, but retrieved or assigned
through the public property.
    public DateTime DOB
    {
        get { return this.dob; }
        set { this.dob = value; }
    }
    //Age property can only be retrieved; it's value is derived from the date of birth
    public int Age
    {
        get
        {
            int offset = HasHadBirthdayThisYear() ? 0 : -1;
            return DateTime.UtcNow.Year - this.dob.Year + offset;
        }
    }

    //this is not a property but a method; though it could be rewritten as a property if
desired.
    private bool HasHadBirthdayThisYear()
    {
        bool hasHadBirthdayThisYear = true;
        DateTime today = DateTime.UtcNow;
        if (today.Month > this.dob.Month)
        {
            hasHadBirthdayThisYear = true;
        }
        else
        {
            if (today.Month == this.dob.Month)
            {
                hasHadBirthdayThisYear = today.Day > this.dob.Day;
            }
            else
            {
                hasHadBirthdayThisYear = false;
```

```
            }
        }
        return hasHadBirthdayThisYear;
    }
}
```

Get

getter .

```
string name;
public string Name
{
    get { return this.name; }
}
```

setter .

```
string name;
public string Name
{
    set { this.name = value; }
}
```

```
class Program
{
    public static void Main(string[] args)
    {
        Person aPerson = new Person("Ann Xena Sample", new DateTime(1984, 10, 22));
        //example of accessing properties (Id, Name & DOB)
        Console.WriteLine("Id is:  \t{0}\nName is:\t'{1}'.\nDOB is: \t{2:yyyy-MM-dd}.\nAge is:
\t{3}", aPerson.Id, aPerson.Name, aPerson.DOB, aPerson.GetAgeInYears());
        //example of setting properties

        aPerson.Name = "   Hans Trimmer  ";
        aPerson.DOB = new DateTime(1961, 11, 11);
        //aPerson.Id = 5; //this won't compile as Id's SET method is private; so only
accessible within the Person class.
        //aPerson.DOB = DateTime.UtcNow.AddYears(1); //this would throw a runtime error as
there's validation to ensure the DOB is in past.

        //see how our changes above take effect; note that the Name has been trimmed
        Console.WriteLine("Id is:  \t{0}\nName is:\t'{1}'.\nDOB is: \t{2:yyyy-MM-dd}.\nAge is:
\t{3}", aPerson.Id, aPerson.Name, aPerson.DOB, aPerson.GetAgeInYears());

        Console.WriteLine("Press any key to continue");
        Console.Read();
    }
}

public class Person
{
    private static int nextId = 0;
    private string name;
    private DateTime dob; //dates are held in UTC; i.e. we disregard timezones
    public Person(string name, DateTime dob)
```

```
    {
        this.Id = ++Person.nextId;
        this.Name = name;
        this.DOB = dob;
    }
    public int Id
    {
        get;
        private set;
    }
    public string Name
    {
        get { return this.name; }
        set
        {
            if (string.IsNullOrWhiteSpace(value)) throw new InvalidNameException(value);
            this.name = value.Trim();
        }
    }
    public DateTime DOB
    {
        get { return this.dob; }
        set
        {
            if (value < DateTime.UtcNow.AddYears(-200) || value > DateTime.UtcNow) throw new
InvalidDobException(value);
            this.dob = value;
        }
    }
    public int GetAgeInYears()
    {
        DateTime today = DateTime.UtcNow;
        int offset = HasHadBirthdayThisYear() ? 0 : -1;
        return today.Year - this.dob.Year + offset;
    }
    private bool HasHadBirthdayThisYear()
    {
        bool hasHadBirthdayThisYear = true;
        DateTime today = DateTime.UtcNow;
        if (today.Month > this.dob.Month)
        {
            hasHadBirthdayThisYear = true;
        }
        else
        {
            if (today.Month == this.dob.Month)
            {
                hasHadBirthdayThisYear = today.Day > this.dob.Day;
            }
            else
            {
                hasHadBirthdayThisYear = false;
            }
        }
        return hasHadBirthdayThisYear;
    }
}

public class InvalidNameException : ApplicationException
{
    const string InvalidNameExceptionMessage = "'{0}' is an invalid name.";
```

```
    public InvalidNameException(string value):
base(string.Format(InvalidNameExceptionMessage,value)){}
}
public class InvalidDobException : ApplicationException
{
    const string InvalidDobExceptionMessage = "'{0:yyyy-MM-dd}' is an invalid DOB.  The date
must not be in the future, or over 200 years in the past.";
    public InvalidDobException(DateTime value):
base(string.Format(InvalidDobExceptionMessage,value)){}
}
```

(C # 6) .

```
public class Name
{
    public string First { get; set; } = "James";
    public string Last { get; set; } = "Smith";
}
```

.

```
  public class Name
  {
      public string First => "James";
      public string Last => "Smith";
  }
```

C # 3 .
getter setter ( ) .

```
public bool IsValid { get; set; }
```

.

.

```
private bool _isValid;
public bool IsValid
{
    get { return _isValid; }
    set { _isValid = value; }
}
```

. .

```
public bool IsValid { get; set { PropertyChanged("IsValid"); } } // Invalid code
```

.

```
public bool IsValid { get; private set; }
```

C # 6 ( ).

```
public bool IsValid { get; }
public bool IsValid { get; } = true;
```

.

, . readonly . .

```
public readonly string SomeProp { get; set; }
```

getter .

```
public string SomeProp { get; }
```

```
public Address
{
    public string ZipCode { get; }
    public string City { get; }
    public string StreetAddress { get; }

    public Address(
        string zipCode,
        string city,
        string streetAddress)
    {
        if (zipCode == null)
            throw new ArgumentNullException(nameof(zipCode));
        if (city == null)
            throw new ArgumentNullException(nameof(city));
        if (streetAddress == null)
            throw new ArgumentNullException(nameof(streetAddress));

        ZipCode = zipCode;
        City = city;
        StreetAddress = streetAddress;
    }
}
```

: https://riptutorial.com/ko/csharp/topic/49/

106:

Examples

```
//1) All attributes should be inherited from System.Attribute
//2) You can customize your attribute usage (e.g. place restrictions) by using
System.AttributeUsage Attribute
//3) You can use this attribute only via reflection in the way it is supposed to be used
//4) MethodMetadataAttribute is just a name. You can use it without "Attribute" postfix - e.g.
[MethodMetadata("This text could be retrieved via reflection")].
//5) You can overload an attribute constructors
[System.AttributeUsage(System.AttributeTargets.Method | System.AttributeTargets.Class)]
public class MethodMetadataAttribute : System.Attribute
{
    //this is custom field given just for an example
    //you can create attribute without any fields
    //even an empty attribute can be used - as marker
    public string Text { get; set; }

    //this constructor could be used as [MethodMetadata]
    public MethodMetadataAttribute ()
    {
    }

    //This constructor could be used as [MethodMetadata("String")]
    public MethodMetadataAttribute (string text)
    {
        Text = text;
    }
}
```

```
[StackDemo(Text = "Hello, World!")]
public class MyClass
{
    [StackDemo("Hello, World!")]
    static void MyMethod()
    {
    }
}
```

GetCustomAttributes . .

```
var attribute = typeof(MyClass).GetCustomAttributes().OfType<MyCustomAttribute>().Single();
```

.

```
foreach(var attribute in typeof(MyClass).GetCustomAttributes()) {
    Console.WriteLine(attribute.GetType());
}
```

System.Reflection.CustomAttributeExtensions GetCustomAttribute MemberInfo .

```
var attribute = (MyCustomAttribute)
typeof(MyClass).GetCustomAttribute(typeof(MyCustomAttribute));
```

GetCustomAttribute .

```
var attribute = typeof(MyClass).GetCustomAttribute<MyCustomAttribute>();
```

inherit . true .

DebuggerDisplay

DebuggerDisplay DebuggerDisplay .

{} . .

```
[DebuggerDisplay("{StringProperty} - {IntProperty}")]
public class AnObject
{
   public int ObjectId { get; set; }
   public string StringProperty { get; set; }
   public int IntProperty { get; set; }
}
```

,nq .

```
[DebuggerDisplay("{StringProperty,nq} - {IntProperty}")]
```

{} .DebuggerDisplay .{} . DebuggerDisplay , C # VB.NET .

DebuggerDisplay .

```
[DebuggerDisplay("{DebuggerDisplay(),nq}")]
public class AnObject
{
   public int ObjectId { get; set; }
   public string StringProperty { get; set; }
   public int IntProperty { get; set; }

   private string DebuggerDisplay()
    {
        return $"{StringProperty} - {IntProperty}"";
    }
}
```

DebuggerDisplay .

DebuggerDisplay #if DEBUG .

```
[DebuggerDisplay("{DebuggerDisplay(),nq}")]
public class AnObject
{
   public int ObjectId { get; set; }
   public string StringProperty { get; set; }
   public int IntProperty { get; set; }

#if DEBUG
   private string DebuggerDisplay()
    {
        return
            $"ObjectId:{this.ObjectId}, StringProperty:{this.StringProperty},
Type:{this.GetType()}";
    }
    #endif
}
```

. .

```
using System.Runtime.CompilerServices;

public void LogException(Exception ex,
                         [CallerMemberName]string callerMemberName = "",
                         [CallerLineNumber]int callerLineNumber = 0,
                         [CallerFilePath]string callerFilePath = "")
{
    //perform logging
}
```

.

```
public void Save(DBContext context)
{
    try
    {
        context.SaveChanges();
    }
    catch (Exception ex)
    {
        LogException(ex);
    }
}
```

LogException .

callerMemberName "Save" .

callerLineNumber LogException .

'callerFilePath' Save .

. . True .

```
using System;
using System.Linq;
using System.Reflection;

namespace InterfaceAttributesDemo {

    [AttributeUsage(AttributeTargets.Interface, Inherited = true)]
    class MyCustomAttribute : Attribute {
        public string Text { get; set; }
    }

    [MyCustomAttribute(Text = "Hello from interface attribute")]
    interface IMyClass {
        void MyMethod();
    }

    class MyClass : IMyClass {
        public void MyMethod() { }
    }

    public class Program {
        public static void Main(string[] args) {
            GetInterfaceAttributeDemo();
        }

        private static void GetInterfaceAttributeDemo() {
            var attribute1 = (MyCustomAttribute)
typeof(MyClass).GetCustomAttribute(typeof(MyCustomAttribute), true);
            Console.WriteLine(attribute1 == null); // True

            var attribute2 =
typeof(MyClass).GetCustomAttributes(true).OfType<MyCustomAttribute>().SingleOrDefault();
            Console.WriteLine(attribute2 == null); // True

            var attribute3 = typeof(MyClass).GetCustomAttribute<MyCustomAttribute>(true);
            Console.WriteLine(attribute3 == null); // True
        }
    }
}
```

.

```
var attribute = typeof(MyClass).GetInterfaces().SelectMany(x =>
x.GetCustomAttributes().OfType<MyCustomAttribute>()).SingleOrDefault();
Console.WriteLine(attribute == null); // False
Console.WriteLine(attribute.Text); // Hello from interface attribute
```

System.Obsolete .

```
[Obsolete("This class is obsolete. Use SomeOtherClass instead.")]
class SomeClass
{
    //
}
```

"This class is obsolete. SomeOtherClass ." .

: https://riptutorial.com/ko/csharp/topic/1062/

107:

.

. .

Examples

C # 6.0 :

getter / setter .

```
public string Foobar { get; set; } = "xyz";
```

```
public string Foobar {
    get { return _foobar; }
    set { _foobar = value; }
}
private string _foobar = "xyz";
```

```
class Example
{
    public string Foobar { get; set; }
    public List<string> Names { get; set; }
    public Example()
    {
        Foobar = "xyz";
        Names = new List<string>(){"carrot","fox","ball"};
    }
}
```

.

```
var redCar = new Car
{
    Wheels = 2,
    Year = 2016,
    Color = Color.Red
};
```

: https://riptutorial.com/ko/csharp/topic/82/--

108:

yield , get . yield IEnumerable IEnumerator ( ) .

- [TYPE]
- yield break

`IEnumerable` , `IEnumerable<T>` , `IEnumerator` `IEnumerator<T>` `yield` (`IEnumerable` `IEnumerator`) .
.

`yield` "" .

`yield break` .

`yield` `IEnumerable<T>` `Task<IEnumerable<T>>` .

- https://msdn.microsoft.com/en-us/library/9k7k7cf0.aspx

Examples

`yield` `IEnumerable` `IEnumerator` ( ) . .

`for` yield return .

```
public static IEnumerable<int> Count(int start, int count)
{
    for (int i = 0; i <= count; i++)
    {
        yield return start + i;
    }
}
```

.

```
foreach (int value in Count(start: 4, count: 10))
{
    Console.WriteLine(value);
}
```

```
4
5
6
...
14
```

.NET Fiddle

`foreach` `Count` iterator . `for` `yield return` .

```
public IEnumerable<User> SelectUsers()
{
    // Execute an SQL query on a database.
    using (IDataReader reader = this.Database.ExecuteReader(CommandType.Text, "SELECT Id, Name
FROM Users"))
    {
        while (reader.Read())
        {
            int id = reader.GetInt32(0);
            string name = reader.GetString(1);
            yield return new User(id, name);
        }
    }
}
```

SQL `IEnumerable<User>` . " `yield` of elements" `IEnumerable<T>` ..

`yield break` `yield` .

```
public static IEnumerable<int> CountUntilAny(int start, HashSet<int> earlyTerminationSet)
{
    int curr = start;

    while (true)
    {
        if (earlyTerminationSet.Contains(curr))
        {
            // we've hit one of the ending values
            yield break;
        }

        yield return curr;

        if (curr == Int32.MaxValue)
        {
            // don't overflow if we get all the way to the end; just stop
            yield break;
        }

        curr++;
    }
}
```

`earlyTerminationSet` `start` .

```
// Iterate from a starting point until you encounter any elements defined as
// terminating elements
var terminatingElements = new HashSet<int>{ 7, 9, 11 };
// This will iterate from 1 until one of the terminating elements is encountered (7)
foreach(var x in CountUntilAny(1,terminatingElements))
{
    // This will write out the results from 1 until 7 (which will trigger terminating)
    Console.WriteLine(x);
}
```

:

1
2

4
5
6

.NET Fiddle

. iterator .

```
public static IEnumerable<int> Count(int start, int count)
{
    // The exception will throw when the method is called, not when the result is iterated
    if (count < 0)
        throw new ArgumentOutOfRangeException(nameof(count));

    return CountCore(start, count);
}

private static IEnumerable<int> CountCore(int start, int count)
{
    // If the exception was thrown here it would be raised during the first MoveNext()
    // call on the IEnumerator, potentially at a point in the code far away from where
    // an incorrect value was passed.
    for (int i = 0; i < count; i++)
    {
        yield return start + i;
    }
}
```

() :

```
// Get the count
var count = Count(1,10);
// Iterate the results
foreach(var x in count)
{
    Console.WriteLine(x);
}
```

:

1
2

4
5
6
7
8
9
10

.NET Fiddle

`yield` `yield` . .

, (,) (`null`) .). . .

C # 7+ `CountCore` `Count` . .

Enumerable Enumerable

```
public IEnumerable<int> F1()
{
    for (int i = 0; i < 3; i++)
        yield return i;

    //return F2(); // Compile Error!!
    foreach (var element in F2())
        yield return element;
}

public int[] F2()
{
    return new[] { 3, 4, 5 };
}
```

`foreach` `yield` .

.

```
private IEnumerable<int> Integers()
{
    var i = 0;
    while(true)
    {
        Console.WriteLine("Inside iterator: " + i);
        yield return i;
        i++;
    }
}

private void PrintNumbers()
{
    var numbers = Integers().Take(3);
    Console.WriteLine("Starting iteration");

    foreach(var number in numbers)
    {
        Console.WriteLine("Inside foreach: " + number);
    }
}
```

.

iterator : 0

foreach : 0
: 1
foreach : 1
: 2
foreach : 2

:

- "Starting iteration" iterator `Integers().Take(3);` (`IEnumerator.MoveNext()`)
- iterator `foreach` .
- iterator `while true` `.Take()` .

Try ... finally

`try...finally` yield `IEnumerator` `try` `Dispose` `finally` .

:

```
private IEnumerable<int> Numbers()
{
    yield return 1;
    try
    {
        yield return 2;
        yield return 3;
    }
    finally
    {
        Console.WriteLine("Finally executed");
    }
}
```

:

```
private void DisposeOutsideTry()
{
    var enumerator = Numbers().GetEnumerator();

    enumerator.MoveNext();
    Console.WriteLine(enumerator.Current);
    enumerator.Dispose();
}
```

.

1

:

```
private void DisposeInsideTry()
{
    var enumerator = Numbers().GetEnumerator();
```

```
    enumerator.MoveNext();
    Console.WriteLine(enumerator.Current);
    enumerator.MoveNext();
    Console.WriteLine(enumerator.Current);
    enumerator.Dispose();
}
```

.

1
2

yield IEnumerator IEnumerable

`IEnumerable<T>` `GetEnumerator()` `IEnumerator<T>` .

`yield` `IEnumerable<T>` , `IEnumerator<T>` . .

`IEnumerable<T>` :

```
public class PrintingEnumerable<T> : IEnumerable<T>
{
    private IEnumerable<T> _wrapped;

    public PrintingEnumerable(IEnumerable<T> wrapped)
    {
        _wrapped = wrapped;
    }

    // This method returns an IEnumerator<T>, rather than an IEnumerable<T>
    // But the yield syntax and usage is identical.
    public IEnumerator<T> GetEnumerator()
    {
        foreach(var item in _wrapped)
        {
            Console.WriteLine("Yielding: " + item);
            yield return item;
        }
    }

    IEnumerator IEnumerable.GetEnumerator()
    {
        return GetEnumerator();
    }
}
```

(, `IEnumerable<T>` .)

`yield` lazy-evaluation . **eager evaluation** .

:

```
IEnumerable<int> myMethod()
{
    for(int i=0; i <= 8675309; i++)
```

```
    {
        yield return i;
    }
}
...
// define the iterator
var it = myMethod.Take(3);
// force its immediate evaluation
// list will contain 0, 1, 2
var list = it.ToList();
```

`ToList` , `ToDictionary` `ToArray` .

:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Numerics; // also add reference to System.Numberics

namespace ConsoleApplication33
{
    class Program
    {
        private static IEnumerable<BigInteger> Fibonacci()
        {
            BigInteger prev = 0;
            BigInteger current = 1;
            while (true)
            {
                yield return current;
                var next = prev + current;
                prev = current;
                current = next;
            }
        }

        static void Main()
        {
            // print Fibonacci numbers from 10001 to 10010
            var numbers = Fibonacci().Skip(10000).Take(10).ToArray();
            Console.WriteLine(string.Join(Environment.NewLine, numbers));
        }
    }
}
```

(IL Disaambler .exe) :

1. C # `IEnumerable<BigInteger>` `IEnumerator<BigInteger>` (ildasm `<Fibonacci>d__0` .
2. . .
3. `bool IEnumerator.MoveNext()` . `MoveNext()` .
    - . `prev` `current` (ildasm `<current>5__2` `<prev>5__1`). (`<>1__state` 1) . (curly brace), `yield return` .
    - `yield return` `yield break` / `}` .
    - `yield return` `Current` . `true` . `MoveNext` .
    - `yield break` / `}` `false` .

10001 468 . current prev . 10000 4MB . .

yield break break . .

( ) (foreach , while ...) ?

yield . () . . . } , . (iterating) return . return yield break ., yield break return . break .

.

```
    /// <summary>
    /// Yields numbers from 0 to 9
    /// </summary>
    /// <returns>{0,1,2,3,4,5,6,7,8,9}</returns>
    public static IEnumerable<int> YieldBreak()
    {
        for (int i = 0; ; i++)
        {
            if (i < 10)
            {
                // Yields a number
                yield return i;
            }
            else
            {
                // Indicates that the iteration has ended, everything
                // from this line on will be ignored
                yield break;
            }
        }
        yield return 10; // This will never get executed
    }
```

```
    /// <summary>
    /// Yields numbers from 0 to 10
    /// </summary>
    /// <returns>{0,1,2,3,4,5,6,7,8,9,10}</returns>
    public static IEnumerable<int> Break()
    {
        for (int i = 0; ; i++)
        {
            if (i < 10)
            {
                // Yields a number
                yield return i;
            }
            else
            {
                // Terminates just the loop
                break;
            }
        }
        // Execution continues
        yield return 10;
    }
```

: https://riptutorial.com/ko/csharp/topic/61/-

109:

. . .

.

.

C # ( ) . . .

CLR . . CPU .

.NET Framework `System.Threading` . System.Threading . , . .

`Thread` `System.Threading` . `AutoResetEvent` , `Interlocked` , `Monitor` , `Mutex` `ThreadPool` .

`System.Threading` `ThreadStart` , `TimerCallback` `WaitCallback` .

`System.Threading` `ThreadPriority` , `ThreadState` `EventResetMode` .

.NET Framework 4 `System.Threading.Tasks.Parallel` `System.Threading.Tasks.Task` , Parallel LINQ (PLINQ), `System.Collections.Concurrent` . `System.Collections.Concurrent` .

Examples

```
class Program
{
    static void Main(string[] args)
    {
        // Create 2 thread objects.  We're using delegates because we need to pass
        // parameters to the threads.
        var thread1 = new Thread(new ThreadStart(() => PerformAction(1)));
        var thread2 = new Thread(new ThreadStart(() => PerformAction(2)));

        // Start the threads running
        thread1.Start();
        // NB: as soon as the above line kicks off the thread, the next line starts;
        // even if thread1 is still processing.
        thread2.Start();

        // Wait for thread1 to complete before continuing
        thread1.Join();
        // Wait for thread2 to complete before continuing
        thread2.Join();

        Console.WriteLine("Done");
        Console.ReadKey();
    }

    // Simple method to help demonstrate the threads running in parallel.
    static void PerformAction(int id)
    {
        var rnd = new Random(id);
        for (int i = 0; i < 100; i++)
```

```
        {
            Console.WriteLine("Thread: {0}: {1}", id, i);
            Thread.Sleep(rnd.Next(0, 1000));
        }
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        // Run 2 Tasks.
        var task1 = Task.Run(() => PerformAction(1));
        var task2 = Task.Run(() => PerformAction(2));

        // Wait (i.e. block this thread) until both Tasks are complete.
        Task.WaitAll(new [] { task1, task2 });

        Console.WriteLine("Done");
        Console.ReadKey();
    }

    // Simple method to help demonstrate the threads running in parallel.
    static void PerformAction(int id)
    {
        var rnd = new Random(id);
        for (int i = 0; i < 100; i++)
        {
            Console.WriteLine("Task: {0}: {1}", id, i);
            Thread.Sleep(rnd.Next(0, 1000));
        }
    }
}
```

```
    private static void explicitTaskParallism()
    {
        Thread.CurrentThread.Name = "Main";

        // Create a task and supply a user delegate by using a lambda expression.
        Task taskA = new Task(() => Console.WriteLine($"Hello from task {nameof(taskA)}."));
        Task taskB = new Task(() => Console.WriteLine($"Hello from task {nameof(taskB)}."));

        // Start the task.
        taskA.Start();
        taskB.Start();

        // Output a message from the calling thread.
        Console.WriteLine("Hello from thread '{0}'.",
                          Thread.CurrentThread.Name);
        taskA.Wait();
        taskB.Wait();
        Console.Read();
    }
```

```
    private static void Main(string[] args)
    {
        var a = new A();
        var b = new B();
```

```
        //implicit task parallelism
        Parallel.Invoke(
            () => a.DoSomeWork(),
            () => b.DoSomeOtherWork()
            );

      }
```

. .

```
using System.Threading;

class MainClass {
    static void Main() {
        var thread = new Thread(Secondary);
        thread.Start();
    }

    static void Secondary() {
        System.Console.WriteLine("Hello World!");
    }
}
```

using System.Threading;

```
class MainClass {
    static void Main() {
        var thread = new Thread(Secondary);
        thread.Start("SecondThread");
    }

    static void Secondary(object threadName) {
        System.Console.WriteLine("Hello World from thread: " + threadName);
    }
}
```

Environment.ProcessorCount .

CLR . , .

```
using System;
using System.Threading;

class MainClass {
    static void Main() {
        for (int i = 0; i < Environment.ProcessorCount; i++) {
            var thread = new Thread(Secondary);
            thread.Start(i);
        }

    }

    static void Secondary(object threadNumber) {
        System.Console.WriteLine("Hello World from thread: " + threadNumber);
    }
}
```

. . .

() .

```
using System.Threading;

class MainClass
{
    static int count { get; set; }

    static void Main()
    {
        for (int i = 1; i <= 2; i++)
        {
            var thread = new Thread(ThreadMethod);
            thread.Start(i);
            Thread.Sleep(500);
        }
    }

    static void ThreadMethod(object threadNumber)
    {
        while (true)
        {
            var temp = count;
            System.Console.WriteLine("Thread " + threadNumber + ": Reading the value of
count.");
            Thread.Sleep(1000);
            count = temp + 1;
            System.Console.WriteLine("Thread " + threadNumber + ": Incrementing the value of
count to:" + count);
            Thread.Sleep(1000);
        }
    }
}
```

1,2,3,4,5 1,1,2,2,3 .

count . .

```
using System.Threading;

class MainClass
{

    static int count { get; set; }
    static readonly object key = new object();

    static void Main()
    {
        for (int i = 1; i <= 2; i++)
        {
            var thread = new Thread(ThreadMethod);
            thread.Start(i);
            Thread.Sleep(500);
        }
    }

    static void ThreadMethod(object threadNumber)
```

```
    {
        while (true)
        {
            lock (key)
            {
                var temp = count;
                System.Console.WriteLine("Thread " + threadNumber + ": Reading the value of
count.");
                Thread.Sleep(1000);
                count = temp + 1;
                System.Console.WriteLine("Thread " + threadNumber + ": Incrementing the value
of count to:" + count);
            }
            Thread.Sleep(1000);
        }
    }
}
```

Parallel.ForEach

foreach foreach .

```
using System;
using System.Threading;
using System.Threading.Tasks;

public class MainClass {

    public static void Main() {
        int[] Numbers = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
        // Single-threaded
        Console.WriteLine("Normal foreach loop: ");
        foreach (var number in Numbers) {
            Console.WriteLine(longCalculation(number));
        }
        // This is the Parallel (Multi-threaded solution)
        Console.WriteLine("Parallel foreach loop: ");
        Parallel.ForEach(Numbers, number => {
            Console.WriteLine(longCalculation(number));
        });
    }

    private static int longCalculation(int number) {
        Thread.Sleep(1000); // Sleep to simulate a long calculation
        return number * number;
    }
}
```

( )

.

Windows Forms GUI GUI . button1 .

```
private void button1_Click(object sender, EventArgs e)
{
```

```
    Thread workerthread= new Thread(dowork);
    workerthread.Start();
    workerthread.Join();
    // Do something after
}

private void dowork()
{
    // Do something before
    textBox1.Invoke(new Action(() => textBox1.Text = "Some Text"));
    // Do something after
}
```

`workerthread.Join()` workerthread . `textBox1.Invoke(invoke_delegate)` GUI invoke_delegate GUI .

.

```
private void dowork()
{
    // Do work
    textBox1.BeginInvoke(new Action(() => textBox1.Text = "Some Text"));
    // Do work that is not dependent on textBox1 being updated first
}
```

. . GUI .

```
private void dowork()
{
    // Do work
    textBox1.BeginInvoke(new Action(() => {
        textBox1.Text = "Some Text";
        // Do work dependent on textBox1 being updated first,
        // start another worker thread or raise an event
    }));
    // Do work that is not dependent on textBox1 being updated first
}
```

GUI workerthread .

```
private void dowork()
{
    // Do work
    Thread workerthread2 = new Thread(() =>
    {
        textBox1.Invoke(new Action(() => textBox1.Text = "Some Text"));
        // Do work dependent on textBox1 being updated first,
        // start another worker thread or raise an event
    });
    workerthread2.Start();
    // Do work that is not dependent on textBox1 being updated first
}
```

. . .

( )

.

thread1 A thread2 B A B .

button1 .

```
private void button_Click(object sender, EventArgs e)
{
    DeadlockWorkers workers = new DeadlockWorkers();
    workers.StartThreads();
    textBox.Text = workers.GetResult();
}

private class DeadlockWorkers
{
    Thread thread1, thread2;

    object resourceA = new object();
    object resourceB = new object();

    string output;

    public void StartThreads()
    {
        thread1 = new Thread(Thread1DoWork);
        thread2 = new Thread(Thread2DoWork);
        thread1.Start();
        thread2.Start();
    }

    public string GetResult()
    {
        thread1.Join();
        thread2.Join();
        return output;
    }

    public void Thread1DoWork()
    {
        Thread.Sleep(100);
        lock (resourceA)
        {
            Thread.Sleep(100);
            lock (resourceB)
            {
                output += "T1#";
            }
        }
    }

    public void Thread2DoWork()
    {
        Thread.Sleep(100);
        lock (resourceB)
        {
            Thread.Sleep(100);
            lock (resourceA)
```

```
            {
                output += "T2#";
            }
        }
    }
}
```

Monitor.TryEnter (lock_object, timeout_in_milliseconds) . Monitor.TryEnter timeout_in_milliseconds lock_object false :

```
private void button_Click(object sender, EventArgs e)
{
    MonitorWorkers workers = new MonitorWorkers();
    workers.StartThreads();
    textBox.Text = workers.GetResult();
}

private class MonitorWorkers
{
    Thread thread1, thread2;

    object resourceA = new object();
    object resourceB = new object();

    string output;

    public void StartThreads()
    {
        thread1 = new Thread(Thread1DoWork);
        thread2 = new Thread(Thread2DoWork);
        thread1.Start();
        thread2.Start();
    }

    public string GetResult()
    {
        thread1.Join();
        thread2.Join();
        return output;
    }

    public void Thread1DoWork()
    {
        bool mustDoWork = true;
        Thread.Sleep(100);
        while (mustDoWork)
        {
            lock (resourceA)
            {
                Thread.Sleep(100);
                if (Monitor.TryEnter(resourceB, 0))
                {
                    output += "T1#";
                    mustDoWork = false;
                    Monitor.Exit(resourceB);
                }
            }
            if (mustDoWork) Thread.Yield();
        }
    }
```

```
    public void Thread2DoWork()
    {
        Thread.Sleep(100);
        lock (resourceB)
        {
            Thread.Sleep(100);
            lock (resourceA)
            {
                output += "T2#";
            }
        }
    }
}
```

2 1 thread2 . thread1 A . . . .

: https://riptutorial.com/ko/csharp/topic/51/

110:

. throw .

Examples

Windows Forms NullReferenceException

throw .

```
private void button1_Click(object sender, EventArgs e)
{
    string msg = null;
    msg.ToCharArray();
}
```

.

```
System.NullReferenceException: "Object reference not set to an instance of an object."
   at WindowsFormsApplication1.Form1.button1_Click(Object sender, EventArgs e) in
F:\WindowsFormsApplication1\WindowsFormsApplication1\Form1.cs:line 29
   at System.Windows.Forms.Control.OnClick(EventArgs e)
   at System.Windows.Forms.Button.OnClick(EventArgs e)
   at System.Windows.Forms.Button.OnMouseUp(MouseEventArgs mevent)
```

.

.

F : \ WindowsFormsApplication1 \ WindowsFormsApplication1 \ Form1.cs
WindowsFormsApplication1.Form1.button1_Click (, EventArgs e) : 29

. : Form1.cs 29 .

.

System.Windows.Forms.Control.OnClick (EventArgs e)

`button1_Click` . `button1_Click` `System.Windows.Forms.Control.OnClick` .

.

System.Windows.Forms.Button.OnClick (EventArgs e)

`System.windows.Forms.Control.OnClick` .

Exception . !

.Net . `System.Windows.Forms` , .

, "stack trace" ?

, , . "" , .
.

.

. .

: https://riptutorial.com/ko/csharp/topic/8923/----

111:

- stopWatch.Start () - .
- stopWatch.Stop () - .
- stopWatch.Elapsed - .

.

Examples

. . .

```
Stopwatch stopWatch = new Stopwatch();
stopWatch.Start();

double d = 0;
for (int i = 0; i < 1000 * 1000 * 1000; i++)
{
    d += 1;
}

stopWatch.Stop();
Console.WriteLine("Time elapsed: {0:hh\\:mm\\:ss\\.fffffff}", stopWatch.Elapsed);
```

Stopwach System.Diagnostics System.Diagnostics using System.Diagnostics; using System.Diagnostics; .

IsHighResolution

- IsHighResolution DateTime .
- .

```
// Display the timer frequency and resolution.
if (Stopwatch.IsHighResolution)
{
    Console.WriteLine("Operations timed using the system's high-resolution performance
counter.");
}
else
{
    Console.WriteLine("Operations timed using the DateTime class.");
}

long frequency = Stopwatch.Frequency;
Console.WriteLine("  Timer frequency in ticks per second = {0}",
    frequency);
long nanosecPerTick = (1000L*1000L*1000L) / frequency;
Console.WriteLine("  Timer is accurate within {0} nanoseconds",
    nanosecPerTick);
}
```

https://dotnetfiddle.net/ckrWUo

Stopwatch . StopWatch IsHighResolution true. IsHighResolution Stopwatch .

/ OS , . Stopwatch DateTime / TimeSpan .

Stopwatch.Frequency ( ) Stopwatch Elapsed ElapsedMilliseconds .

Datetime Dateime .

: https://riptutorial.com/ko/csharp/topic/3676/

112: : Json with C

C # Json .

Examples

Json

```
{
    "id": 89,
    "name": "Aldous Huxley",
    "type": "Author",
    "books":[{
               "name": "Brave New World",
               "date": 1932
             },
             {
               "name": "Eyeless in Gaza",
               "date": 1936
             },
             {
               "name": "The Genius and the Goddess",
               "date": 1955
             }]
}
```

Json .

: Json

C # Json Newtonsoft (.net) . . .

Visual Studio *Tools / Nuget Package Manager / Manage Package to Solution /* "Newtonsoft" . NuGet , .

C

json .

 : . json .

 : json / . . json C # deserialize .

json . Visual Studio *"Edit / Paste Special / JSON "* JSON json .

```
using Newtonsoft.Json;

  class Author
{
    [JsonProperty("id")] // Set the variable below to represent the json attribute
    public int id;       //"id"
```

```
    [JsonProperty("name")]
    public string name;
    [JsonProperty("type")]
    public string type;
    [JsonProperty("books")]
    public Book[] books;

    public Author(int id, string name, string type, Book[] books) {
        this.id = id;
        this.name = name;
        this.type= type;
        this.books = books;
    }
}

 class Book
{
   [JsonProperty("name")]
   public string name;
   [JsonProperty("date")]
   public DateTime date;
}
```

```
 static void Main(string[] args)
    {
        Book[] books = new Book[3];
        Author author = new Author(89,"Aldous Huxley","Author",books);
        string objectDeserialized = JsonConvert.SerializeObject(author);
        //Converting author into json
    }
```

".SerializeObject" . .

json .

```
static void Main(string[] args)
{
    string jsonExample; // Has the previous json
    Author author = JsonConvert.DeserializeObject<Author>(jsonExample);
}
```

".DeserializeObject" ' *jsonExample* ' " " deserialize. json .

.

```
using System.Runtime.Serialization.Formatters.Binary;
using System.Xml.Serialization;

namespace Framework
{
public static class IGUtilities
{
        public static string Serialization(this T obj)
        {
        string data = JsonConvert.SerializeObject(obj);
        return data;
```

```
        }

        public static T Deserialization(this string JsonData)
        {
        T copy = JsonConvert.DeserializeObject(JsonData);
        return copy;
        }

         public static T Clone(this T obj)
        {
            string data = JsonConvert.SerializeObject(obj);
            T copy = JsonConvert.DeserializeObject(data);
            return copy;
        }
}
}
```

: Json with C # : https://riptutorial.com/ko/csharp/topic/9910/---json-with-c-sharp

113:

Examples

```
public class Singleton
{
    private readonly static Singleton instance = new Singleton();
    private Singleton() { }
    public static Singleton Instance => instance;
}
```

`instance` . CLR .

`instance` thread , `readonly` .

Singleton (Double Checked Locking)

`static` .NET . .

```
public sealed class ThreadSafeSingleton
{
   private static volatile ThreadSafeSingleton instance;
   private static object lockObject = new Object();

   private ThreadSafeSingleton()
   {
   }

   public static ThreadSafeSingleton Instance
   {
      get
      {
         if (instance == null)
         {
            lock (lockObject)
            {
               if (instance == null)
               {
                  instance = new ThreadSafeSingleton();
               }
            }
         }

         return instance;
      }
   }
}
```

`if (instance == null)` , . , null thread . . x (NULL) . . .

Singleton (Lazy))

.Net 4.0 thread-safe Singleton .

```
public class LazySingleton
{
    private static readonly Lazy<LazySingleton> _instance =
        new Lazy<LazySingleton>(() => new LazySingleton());

    public static LazySingleton Instance
    {
        get { return _instance.Value; }
    }

    private LazySingleton() { }
}
```

`Lazy<T>` .

.

```
using System;

public class Program
{
    public static void Main()
    {
        var instance = LazySingleton.Instance;
    }
}
```

.NET Fiddle

(.NET 3.5 ,)

.NET 3.5 `Lazy<T>` .

```
public class Singleton
{
    private Singleton() // prevents public instantiation
    {
    }

    public static Singleton Instance
    {
        get
        {
            return Nested.instance;
        }
    }

    private class Nested
    {
        // Explicit static constructor to tell C# compiler
        // not to mark type as beforefieldinit
        static Nested()
        {
        }

        internal static readonly Singleton instance = new Singleton();
    }
```

```
}
```

Jon Skeet .

`Nested` private Singleton `Sigleton` (: public readonly) .

Singleton

`LazySingleton` `LazySingleton.` `IDisposable` null .

```
public class LazySingleton : IDisposable
{
    private static volatile Lazy<LazySingleton> _instance;
    private static volatile int _instanceCount = 0;
    private bool _alreadyDisposed = false;

public static LazySingleton Instance
{
    get
    {
        if (_instance == null)
            _instance = new Lazy<LazySingleton>(() => new LazySingleton());
        _instanceCount++;
        return _instance.Value;
    }
}

private LazySingleton() { }

// Public implementation of Dispose pattern callable by consumers.
public void Dispose()
{
    if (--_instanceCount == 0) // No more references to this object.
    {
       Dispose(true);
       GC.SuppressFinalize(this);
    }
}

// Protected implementation of Dispose pattern.
protected virtual void Dispose(bool disposing)
{
    if (_alreadyDisposed) return;

    if (disposing)
    {
        _instance = null; // Allow GC to dispose of this instance.
        // Free any other managed objects here.
    }

    // Free any unmanaged objects here.
    _alreadyDisposed = true;
}
```

`Dispose()` . . `Dispose()` . .

```
public class Program
```

```
{
    public static void Main()
    {
        using (var instance = LazySingleton.Instance)
        {
            // Do work with instance
        }
    }
}
```

.

: https://riptutorial.com/ko/csharp/topic/1192/--

114:

GUID (UUID) 'Globally Unique Identifier'('Universally Unique Identifier') . 128 .

`Guid` *UUID Universally Unique Identifiers Globally Unique Identifiers* .

128 . pseudorandom , `Guid` (10 ^ 18 `Guid` .

`Guid` . () ID .

Examples

Guid

Guid `ToString` .

```
string myGuidString = myGuid.ToString();
```

`ToString` Guid .

```
var guid = new Guid("7febf16f-651b-43b0-a5e3-0da8da49e90d");

// None           "7febf16f651b43b0a5e30da8da49e90d"
Console.WriteLine(guid.ToString("N"));

// Hyphens        "7febf16f-651b-43b0-a5e3-0da8da49e90d"
Console.WriteLine(guid.ToString("D"));

// Braces         "{7febf16f-651b-43b0-a5e3-0da8da49e90d}"
Console.WriteLine(guid.ToString("B"));

// Parentheses    "(7febf16f-651b-43b0-a5e3-0da8da49e90d)"
Console.WriteLine(guid.ToString("P"));

// Hex            "{0x7febf16f,0x651b,0x43b0{0xa5,0xe3,0x0d,0xa8,0xda,0x49,0xe9,0x0d}}"
Console.WriteLine(guid.ToString("X"));
```

Guid .

- (`00000000-0000-0000-0000-000000000000`) :

```
Guid g = Guid.Empty;
Guid g2 = new Guid();
```

- ( ) Guid :

```
Guid g = Guid.NewGuid();
```

- :

```
Guid g = new Guid("0b214de7-8958-4956-8eed-28f9ba2c47c6");
Guid g2 = new Guid("0b214de7895849568eed28f9ba2c47c6");
Guid g3 = Guid.Parse("0b214de7-8958-4956-8eed-28f9ba2c47c6");
```

nullable GUID

GUID null null .

:

```
Guid? myGuidVar = null;
```

NULL .

: https://riptutorial.com/ko/csharp/topic/1153/

115: (System.Security.Cryptography)

Examples

@jbtule . :

" (AEAD) , .NET . AES256 HMAC256 , MAC , .

Bouncy Castle (nuget) AES256- GCM .

, . 256 `NewKey()` .

. *256* .

: `byte[]` Gist StackOverflow 4 API . "

.NET (AES) - - MAC (HMAC) []

```
/*
 * This work (Modern Encryption of a String C#, by James Tuley),
 * identified by James Tuley, is free of known copyright restrictions.
 * https://gist.github.com/4336842
 * http://creativecommons.org/publicdomain/mark/1.0/
 */

using System;
using System.IO;
using System.Security.Cryptography;
using System.Text;

namespace Encryption
{
  public static class AESThenHMAC
  {
    private static readonly RandomNumberGenerator Random = RandomNumberGenerator.Create();

    //Preconfigured Encryption Parameters
    public static readonly int BlockBitSize = 128;
    public static readonly int KeyBitSize = 256;

    //Preconfigured Password Key Derivation Parameters
    public static readonly int SaltBitSize = 64;
    public static readonly int Iterations = 10000;
    public static readonly int MinPasswordLength = 12;

    /// <summary>
    /// Helper that generates a random key on each call.
    /// </summary>
    /// <returns></returns>
    public static byte[] NewKey()
    {
      var key = new byte[KeyBitSize / 8];
      Random.GetBytes(key);
      return key;
```

```
    }

    /// <summary>
    /// Simple Encryption (AES) then Authentication (HMAC) for a UTF8 Message.
    /// </summary>
    /// <param name="secretMessage">The secret message.</param>
    /// <param name="cryptKey">The crypt key.</param>
    /// <param name="authKey">The auth key.</param>
    /// <param name="nonSecretPayload">(Optional) Non-Secret Payload.</param>
    /// <returns>
    /// Encrypted Message
    /// </returns>
    /// <exception cref="System.ArgumentException">Secret Message
Required!;secretMessage</exception>
    /// <remarks>
    /// Adds overhead of (Optional-Payload + BlockSize(16) + Message-Padded-To-Blocksize +
HMac-Tag(32)) * 1.33 Base64
    /// </remarks>
    public static string SimpleEncrypt(string secretMessage, byte[] cryptKey, byte[] authKey,
                        byte[] nonSecretPayload = null)
    {
      if (string.IsNullOrEmpty(secretMessage))
        throw new ArgumentException("Secret Message Required!", "secretMessage");

      var plainText = Encoding.UTF8.GetBytes(secretMessage);
      var cipherText = SimpleEncrypt(plainText, cryptKey, authKey, nonSecretPayload);
      return Convert.ToBase64String(cipherText);
    }

    /// <summary>
    /// Simple Authentication (HMAC) then Decryption (AES) for a secrets UTF8 Message.
    /// </summary>
    /// <param name="encryptedMessage">The encrypted message.</param>
    /// <param name="cryptKey">The crypt key.</param>
    /// <param name="authKey">The auth key.</param>
    /// <param name="nonSecretPayloadLength">Length of the non secret payload.</param>
    /// <returns>
    /// Decrypted Message
    /// </returns>
    /// <exception cref="System.ArgumentException">Encrypted Message
Required!;encryptedMessage</exception>
    public static string SimpleDecrypt(string encryptedMessage, byte[] cryptKey, byte[]
authKey,
                        int nonSecretPayloadLength = 0)
    {
      if (string.IsNullOrWhiteSpace(encryptedMessage))
        throw new ArgumentException("Encrypted Message Required!", "encryptedMessage");

      var cipherText = Convert.FromBase64String(encryptedMessage);
      var plainText = SimpleDecrypt(cipherText, cryptKey, authKey, nonSecretPayloadLength);
      return plainText == null ? null : Encoding.UTF8.GetString(plainText);
    }

    /// <summary>
    /// Simple Encryption (AES) then Authentication (HMAC) of a UTF8 message
    /// using Keys derived from a Password (PBKDF2).
    /// </summary>
    /// <param name="secretMessage">The secret message.</param>
    /// <param name="password">The password.</param>
    /// <param name="nonSecretPayload">The non secret payload.</param>
    /// <returns>
```

```
    /// Encrypted Message
    /// </returns>
    /// <exception cref="System.ArgumentException">password</exception>
    /// <remarks>
    /// Significantly less secure than using random binary keys.
    /// Adds additional non secret payload for key generation parameters.
    /// </remarks>
    public static string SimpleEncryptWithPassword(string secretMessage, string password,
                            byte[] nonSecretPayload = null)
    {
      if (string.IsNullOrEmpty(secretMessage))
        throw new ArgumentException("Secret Message Required!", "secretMessage");

      var plainText = Encoding.UTF8.GetBytes(secretMessage);
      var cipherText = SimpleEncryptWithPassword(plainText, password, nonSecretPayload);
      return Convert.ToBase64String(cipherText);
    }

    /// <summary>
    /// Simple Authentication (HMAC) and then Descryption (AES) of a UTF8 Message
    /// using keys derived from a password (PBKDF2).
    /// </summary>
    /// <param name="encryptedMessage">The encrypted message.</param>
    /// <param name="password">The password.</param>
    /// <param name="nonSecretPayloadLength">Length of the non secret payload.</param>
    /// <returns>
    /// Decrypted Message
    /// </returns>
    /// <exception cref="System.ArgumentException">Encrypted Message
Required!;encryptedMessage</exception>
    /// <remarks>
    /// Significantly less secure than using random binary keys.
    /// </remarks>
    public static string SimpleDecryptWithPassword(string encryptedMessage, string password,
                            int nonSecretPayloadLength = 0)
    {
      if (string.IsNullOrWhiteSpace(encryptedMessage))
        throw new ArgumentException("Encrypted Message Required!", "encryptedMessage");

      var cipherText = Convert.FromBase64String(encryptedMessage);
      var plainText = SimpleDecryptWithPassword(cipherText, password, nonSecretPayloadLength);
      return plainText == null ? null : Encoding.UTF8.GetString(plainText);
    }

    public static byte[] SimpleEncrypt(byte[] secretMessage, byte[] cryptKey, byte[] authKey,
byte[] nonSecretPayload = null)
    {
      //User Error Checks
      if (cryptKey == null || cryptKey.Length != KeyBitSize / 8)
        throw new ArgumentException(String.Format("Key needs to be {0} bit!", KeyBitSize),
"cryptKey");

      if (authKey == null || authKey.Length != KeyBitSize / 8)
        throw new ArgumentException(String.Format("Key needs to be {0} bit!", KeyBitSize),
"authKey");

      if (secretMessage == null || secretMessage.Length < 1)
        throw new ArgumentException("Secret Message Required!", "secretMessage");

      //non-secret payload optional
      nonSecretPayload = nonSecretPayload ?? new byte[] { };
```

```
      byte[] cipherText;
      byte[] iv;

      using (var aes = new AesManaged
      {
        KeySize = KeyBitSize,
        BlockSize = BlockBitSize,
        Mode = CipherMode.CBC,
        Padding = PaddingMode.PKCS7
      })
      {

        //Use random IV
        aes.GenerateIV();
        iv = aes.IV;

        using (var encrypter = aes.CreateEncryptor(cryptKey, iv))
        using (var cipherStream = new MemoryStream())
        {
          using (var cryptoStream = new CryptoStream(cipherStream, encrypter,
CryptoStreamMode.Write))
          using (var binaryWriter = new BinaryWriter(cryptoStream))
          {
            //Encrypt Data
            binaryWriter.Write(secretMessage);
          }

          cipherText = cipherStream.ToArray();
        }

      }

      //Assemble encrypted message and add authentication
      using (var hmac = new HMACSHA256(authKey))
      using (var encryptedStream = new MemoryStream())
      {
        using (var binaryWriter = new BinaryWriter(encryptedStream))
        {
          //Prepend non-secret payload if any
          binaryWriter.Write(nonSecretPayload);
          //Prepend IV
          binaryWriter.Write(iv);
          //Write Ciphertext
          binaryWriter.Write(cipherText);
          binaryWriter.Flush();

          //Authenticate all data
          var tag = hmac.ComputeHash(encryptedStream.ToArray());
          //Postpend tag
          binaryWriter.Write(tag);
        }
        return encryptedStream.ToArray();
      }

    }

    public static byte[] SimpleDecrypt(byte[] encryptedMessage, byte[] cryptKey, byte[]
authKey, int nonSecretPayloadLength = 0)
    {
```

```
      //Basic Usage Error Checks
      if (cryptKey == null || cryptKey.Length != KeyBitSize / 8)
        throw new ArgumentException(String.Format("CryptKey needs to be {0} bit!",
KeyBitSize), "cryptKey");

      if (authKey == null || authKey.Length != KeyBitSize / 8)
        throw new ArgumentException(String.Format("AuthKey needs to be {0} bit!", KeyBitSize),
"authKey");

      if (encryptedMessage == null || encryptedMessage.Length == 0)
        throw new ArgumentException("Encrypted Message Required!", "encryptedMessage");

      using (var hmac = new HMACSHA256(authKey))
      {
        var sentTag = new byte[hmac.HashSize / 8];
        //Calculate Tag
        var calcTag = hmac.ComputeHash(encryptedMessage, 0, encryptedMessage.Length -
sentTag.Length);
        var ivLength = (BlockBitSize / 8);

        //if message length is to small just return null
        if (encryptedMessage.Length < sentTag.Length + nonSecretPayloadLength + ivLength)
          return null;

        //Grab Sent Tag
        Array.Copy(encryptedMessage, encryptedMessage.Length - sentTag.Length, sentTag, 0,
sentTag.Length);

        //Compare Tag with constant time comparison
        var compare = 0;
        for (var i = 0; i < sentTag.Length; i++)
          compare |= sentTag[i] ^ calcTag[i];

        //if message doesn't authenticate return null
        if (compare != 0)
          return null;

        using (var aes = new AesManaged
        {
          KeySize = KeyBitSize,
          BlockSize = BlockBitSize,
          Mode = CipherMode.CBC,
          Padding = PaddingMode.PKCS7
        })
        {

          //Grab IV from message
          var iv = new byte[ivLength];
          Array.Copy(encryptedMessage, nonSecretPayloadLength, iv, 0, iv.Length);

          using (var decrypter = aes.CreateDecryptor(cryptKey, iv))
          using (var plainTextStream = new MemoryStream())
          {
            using (var decrypterStream = new CryptoStream(plainTextStream, decrypter,
CryptoStreamMode.Write))
            using (var binaryWriter = new BinaryWriter(decrypterStream))
            {
              //Decrypt Cipher Text from Message
              binaryWriter.Write(
                encryptedMessage,
                nonSecretPayloadLength + iv.Length,
```

```
                encryptedMessage.Length - nonSecretPayloadLength - iv.Length - sentTag.Length
              );
            }
            //Return Plain Text
            return plainTextStream.ToArray();
          }
        }
      }
    }

    public static byte[] SimpleEncryptWithPassword(byte[] secretMessage, string password,
byte[] nonSecretPayload = null)
    {
      nonSecretPayload = nonSecretPayload ?? new byte[] {};

      //User Error Checks
      if (string.IsNullOrWhiteSpace(password) || password.Length < MinPasswordLength)
        throw new ArgumentException(String.Format("Must have a password of at least {0}
characters!", MinPasswordLength), "password");

      if (secretMessage == null || secretMessage.Length ==0)
        throw new ArgumentException("Secret Message Required!", "secretMessage");

      var payload = new byte[((SaltBitSize / 8) * 2) + nonSecretPayload.Length];

      Array.Copy(nonSecretPayload, payload, nonSecretPayload.Length);
      int payloadIndex = nonSecretPayload.Length;

      byte[] cryptKey;
      byte[] authKey;
      //Use Random Salt to prevent pre-generated weak password attacks.
      using (var generator = new Rfc2898DeriveBytes(password, SaltBitSize / 8, Iterations))
      {
        var salt = generator.Salt;

        //Generate Keys
        cryptKey = generator.GetBytes(KeyBitSize / 8);

        //Create Non Secret Payload
        Array.Copy(salt, 0, payload, payloadIndex, salt.Length);
        payloadIndex += salt.Length;
      }

      //Deriving separate key, might be less efficient than using HKDF,
      //but now compatible with RNEncryptor which had a very similar wireformat and requires
less code than HKDF.
      using (var generator = new Rfc2898DeriveBytes(password, SaltBitSize / 8, Iterations))
      {
        var salt = generator.Salt;

        //Generate Keys
        authKey = generator.GetBytes(KeyBitSize / 8);

        //Create Rest of Non Secret Payload
        Array.Copy(salt, 0, payload, payloadIndex, salt.Length);
      }

      return SimpleEncrypt(secretMessage, cryptKey, authKey, payload);
    }

    public static byte[] SimpleDecryptWithPassword(byte[] encryptedMessage, string password,
```

```
int nonSecretPayloadLength = 0)
    {
      //User Error Checks
      if (string.IsNullOrWhiteSpace(password) || password.Length < MinPasswordLength)
        throw new ArgumentException(String.Format("Must have a password of at least {0}
characters!", MinPasswordLength), "password");

      if (encryptedMessage == null || encryptedMessage.Length == 0)
        throw new ArgumentException("Encrypted Message Required!", "encryptedMessage");

      var cryptSalt = new byte[SaltBitSize / 8];
      var authSalt = new byte[SaltBitSize / 8];

      //Grab Salt from Non-Secret Payload
      Array.Copy(encryptedMessage, nonSecretPayloadLength, cryptSalt, 0, cryptSalt.Length);
      Array.Copy(encryptedMessage, nonSecretPayloadLength + cryptSalt.Length, authSalt, 0,
authSalt.Length);

      byte[] cryptKey;
      byte[] authKey;

      //Generate crypt key
      using (var generator = new Rfc2898DeriveBytes(password, cryptSalt, Iterations))
      {
        cryptKey = generator.GetBytes(KeyBitSize / 8);
      }
      //Generate auth key
      using (var generator = new Rfc2898DeriveBytes(password, authSalt, Iterations))
      {
        authKey = generator.GetBytes(KeyBitSize / 8);
      }

      return SimpleDecrypt(encryptedMessage, cryptKey, authKey, cryptSalt.Length +
authSalt.Length + nonSecretPayloadLength);
    }
  }
}
```

AES-GCM []

```
/*
 * This work (Modern Encryption of a String C#, by James Tuley),
 * identified by James Tuley, is free of known copyright restrictions.
 * https://gist.github.com/4336842
 * http://creativecommons.org/publicdomain/mark/1.0/
 */

using System;
using System.IO;
using System.Text;
using Org.BouncyCastle.Crypto;
using Org.BouncyCastle.Crypto.Engines;
using Org.BouncyCastle.Crypto.Generators;
using Org.BouncyCastle.Crypto.Modes;
using Org.BouncyCastle.Crypto.Parameters;
using Org.BouncyCastle.Security;
namespace Encryption
{
```

```
  public static class AESGCM
  {
    private static readonly SecureRandom Random = new SecureRandom();

    //Preconfigured Encryption Parameters
    public static readonly int NonceBitSize = 128;
    public static readonly int MacBitSize = 128;
    public static readonly int KeyBitSize = 256;

    //Preconfigured Password Key Derivation Parameters
    public static readonly int SaltBitSize = 128;
    public static readonly int Iterations = 10000;
    public static readonly int MinPasswordLength = 12;


    /// <summary>
    /// Helper that generates a random new key on each call.
    /// </summary>
    /// <returns></returns>
    public static byte[] NewKey()
    {
      var key = new byte[KeyBitSize / 8];
      Random.NextBytes(key);
      return key;
    }

    /// <summary>
    /// Simple Encryption And Authentication (AES-GCM) of a UTF8 string.
    /// </summary>
    /// <param name="secretMessage">The secret message.</param>
    /// <param name="key">The key.</param>
    /// <param name="nonSecretPayload">Optional non-secret payload.</param>
    /// <returns>
    /// Encrypted Message
    /// </returns>
    /// <exception cref="System.ArgumentException">Secret Message
Required!;secretMessage</exception>
    /// <remarks>
    /// Adds overhead of (Optional-Payload + BlockSize(16) + Message +  HMac-Tag(16)) * 1.33
Base64
    /// </remarks>
    public static string SimpleEncrypt(string secretMessage, byte[] key, byte[]
nonSecretPayload = null)
    {
      if (string.IsNullOrEmpty(secretMessage))
        throw new ArgumentException("Secret Message Required!", "secretMessage");

      var plainText = Encoding.UTF8.GetBytes(secretMessage);
      var cipherText = SimpleEncrypt(plainText, key, nonSecretPayload);
      return Convert.ToBase64String(cipherText);
    }


    /// <summary>
    /// Simple Decryption & Authentication (AES-GCM) of a UTF8 Message
    /// </summary>
    /// <param name="encryptedMessage">The encrypted message.</param>
    /// <param name="key">The key.</param>
    /// <param name="nonSecretPayloadLength">Length of the optional non-secret
payload.</param>
    /// <returns>Decrypted Message</returns>
```

```
    public static string SimpleDecrypt(string encryptedMessage, byte[] key, int
nonSecretPayloadLength = 0)
    {
      if (string.IsNullOrEmpty(encryptedMessage))
        throw new ArgumentException("Encrypted Message Required!", "encryptedMessage");

      var cipherText = Convert.FromBase64String(encryptedMessage);
      var plainText = SimpleDecrypt(cipherText, key, nonSecretPayloadLength);
      return plainText == null ? null : Encoding.UTF8.GetString(plainText);
    }

    /// <summary>
    /// Simple Encryption And Authentication (AES-GCM) of a UTF8 String
    /// using key derived from a password (PBKDF2).
    /// </summary>
    /// <param name="secretMessage">The secret message.</param>
    /// <param name="password">The password.</param>
    /// <param name="nonSecretPayload">The non secret payload.</param>
    /// <returns>
    /// Encrypted Message
    /// </returns>
    /// <remarks>
    /// Significantly less secure than using random binary keys.
    /// Adds additional non secret payload for key generation parameters.
    /// </remarks>
    public static string SimpleEncryptWithPassword(string secretMessage, string password,
                            byte[] nonSecretPayload = null)
    {
      if (string.IsNullOrEmpty(secretMessage))
        throw new ArgumentException("Secret Message Required!", "secretMessage");

      var plainText = Encoding.UTF8.GetBytes(secretMessage);
      var cipherText = SimpleEncryptWithPassword(plainText, password, nonSecretPayload);
      return Convert.ToBase64String(cipherText);
    }



    /// <summary>
    /// Simple Decryption and Authentication (AES-GCM) of a UTF8 message
    /// using a key derived from a password (PBKDF2)
    /// </summary>
    /// <param name="encryptedMessage">The encrypted message.</param>
    /// <param name="password">The password.</param>
    /// <param name="nonSecretPayloadLength">Length of the non secret payload.</param>
    /// <returns>
    /// Decrypted Message
    /// </returns>
    /// <exception cref="System.ArgumentException">Encrypted Message
Required!;encryptedMessage</exception>
    /// <remarks>
    /// Significantly less secure than using random binary keys.
    /// </remarks>
    public static string SimpleDecryptWithPassword(string encryptedMessage, string password,
                            int nonSecretPayloadLength = 0)
    {
      if (string.IsNullOrWhiteSpace(encryptedMessage))
        throw new ArgumentException("Encrypted Message Required!", "encryptedMessage");

      var cipherText = Convert.FromBase64String(encryptedMessage);
      var plainText = SimpleDecryptWithPassword(cipherText, password, nonSecretPayloadLength);
      return plainText == null ? null : Encoding.UTF8.GetString(plainText);
```

```
    }

    public static byte[] SimpleEncrypt(byte[] secretMessage, byte[] key, byte[]
nonSecretPayload = null)
    {
      //User Error Checks
      if (key == null || key.Length != KeyBitSize / 8)
        throw new ArgumentException(String.Format("Key needs to be {0} bit!", KeyBitSize),
"key");

      if (secretMessage == null || secretMessage.Length == 0)
        throw new ArgumentException("Secret Message Required!", "secretMessage");

      //Non-secret Payload Optional
      nonSecretPayload = nonSecretPayload ?? new byte[] { };

      //Using random nonce large enough not to repeat
      var nonce = new byte[NonceBitSize / 8];
      Random.NextBytes(nonce, 0, nonce.Length);

      var cipher = new GcmBlockCipher(new AesFastEngine());
      var parameters = new AeadParameters(new KeyParameter(key), MacBitSize, nonce,
nonSecretPayload);
      cipher.Init(true, parameters);

      //Generate Cipher Text With Auth Tag
      var cipherText = new byte[cipher.GetOutputSize(secretMessage.Length)];
      var len = cipher.ProcessBytes(secretMessage, 0, secretMessage.Length, cipherText, 0);
      cipher.DoFinal(cipherText, len);

      //Assemble Message
      using (var combinedStream = new MemoryStream())
      {
        using (var binaryWriter = new BinaryWriter(combinedStream))
        {
          //Prepend Authenticated Payload
          binaryWriter.Write(nonSecretPayload);
          //Prepend Nonce
          binaryWriter.Write(nonce);
          //Write Cipher Text
          binaryWriter.Write(cipherText);
        }
        return combinedStream.ToArray();
      }
    }

    public static byte[] SimpleDecrypt(byte[] encryptedMessage, byte[] key, int
nonSecretPayloadLength = 0)
    {
      //User Error Checks
      if (key == null || key.Length != KeyBitSize / 8)
        throw new ArgumentException(String.Format("Key needs to be {0} bit!", KeyBitSize),
"key");

      if (encryptedMessage == null || encryptedMessage.Length == 0)
        throw new ArgumentException("Encrypted Message Required!", "encryptedMessage");

      using (var cipherStream = new MemoryStream(encryptedMessage))
      using (var cipherReader = new BinaryReader(cipherStream))
      {
        //Grab Payload
```

```
        var nonSecretPayload = cipherReader.ReadBytes(nonSecretPayloadLength);

        //Grab Nonce
        var nonce = cipherReader.ReadBytes(NonceBitSize / 8);

        var cipher = new GcmBlockCipher(new AesFastEngine());
        var parameters = new AeadParameters(new KeyParameter(key), MacBitSize, nonce,
nonSecretPayload);
        cipher.Init(false, parameters);

        //Decrypt Cipher Text
        var cipherText = cipherReader.ReadBytes(encryptedMessage.Length -
nonSecretPayloadLength - nonce.Length);
        var plainText = new byte[cipher.GetOutputSize(cipherText.Length)];

        try
        {
          var len = cipher.ProcessBytes(cipherText, 0, cipherText.Length, plainText, 0);
          cipher.DoFinal(plainText, len);

        }
        catch (InvalidCipherTextException)
        {
          //Return null if it doesn't authenticate
          return null;
        }

        return plainText;
      }

    }

    public static byte[] SimpleEncryptWithPassword(byte[] secretMessage, string password,
byte[] nonSecretPayload = null)
    {
      nonSecretPayload = nonSecretPayload ?? new byte[] {};

      //User Error Checks
      if (string.IsNullOrWhiteSpace(password) || password.Length < MinPasswordLength)
        throw new ArgumentException(String.Format("Must have a password of at least {0}
characters!", MinPasswordLength), "password");

      if (secretMessage == null || secretMessage.Length == 0)
        throw new ArgumentException("Secret Message Required!", "secretMessage");

      var generator = new Pkcs5S2ParametersGenerator();

      //Use Random Salt to minimize pre-generated weak password attacks.
      var salt = new byte[SaltBitSize / 8];
      Random.NextBytes(salt);

      generator.Init(
        PbeParametersGenerator.Pkcs5PasswordToBytes(password.ToCharArray()),
        salt,
        Iterations);

      //Generate Key
      var key = (KeyParameter)generator.GenerateDerivedMacParameters(KeyBitSize);

      //Create Full Non Secret Payload
      var payload = new byte[salt.Length + nonSecretPayload.Length];
```

```
      Array.Copy(nonSecretPayload, payload, nonSecretPayload.Length);
      Array.Copy(salt,0, payload,nonSecretPayload.Length, salt.Length);

      return SimpleEncrypt(secretMessage, key.GetKey(), payload);
    }

    public static byte[] SimpleDecryptWithPassword(byte[] encryptedMessage, string password,
int nonSecretPayloadLength = 0)
    {
      //User Error Checks
      if (string.IsNullOrWhiteSpace(password) || password.Length < MinPasswordLength)
        throw new ArgumentException(String.Format("Must have a password of at least {0}
characters!", MinPasswordLength), "password");

      if (encryptedMessage == null || encryptedMessage.Length == 0)
        throw new ArgumentException("Encrypted Message Required!", "encryptedMessage");

      var generator = new Pkcs5S2ParametersGenerator();

      //Grab Salt from Payload
      var salt = new byte[SaltBitSize / 8];
      Array.Copy(encryptedMessage, nonSecretPayloadLength, salt, 0, salt.Length);

      generator.Init(
        PbeParametersGenerator.Pkcs5PasswordToBytes(password.ToCharArray()),
        salt,
        Iterations);

      //Generate Key
      var key = (KeyParameter)generator.GenerateDerivedMacParameters(KeyBitSize);

      return SimpleDecrypt(encryptedMessage, key.GetKey(), salt.Length +
nonSecretPayloadLength);
    }
  }
}
```

. *System.Security.Cryptography* .

.

:

- .
- .

:

- . .
- .

System.Security.Cryptography .

- AesManaged (AES).
- AesCryptoServiceProvider (AES FIPS 140-2).
- DESCryptoServiceProvider (DES).

- RC2CryptoServiceProvider (Rivest Cipher 2).
- RijndaelManaged (AES). : RijndaelManaged FIPS-197 .
- TripleDES (TripleDES).

.

:

- .
- ( ) .

:

- . . 1,024 RSACryptoServiceProvider 128 .
- .

System.Security.Cryptography .

- DSACryptoServiceProvider ( )
- RSACryptoServiceProvider (RSA)

! ( ) . (> 10k) . ~ 100ms . . DoS .

.

```
private void firstHash(string userName, string userPassword, int numberOfItterations)
{
    Rfc2898DeriveBytes PBKDF2 = new Rfc2898DeriveBytes(userPassword, 8, numberOfItterations);
//Hash the password with a 8 byte salt
    byte[] hashedPassword = PBKDF2.GetBytes(20);    //Returns a 20 byte hash
    byte[] salt = PBKDF2.Salt;
    writeHashToFile(userName, hashedPassword, salt, numberOfItterations); //Store the hashed
password with the salt and number of itterations to check against future password entries
}
```

```
private bool checkPassword(string userName, string userPassword, int numberOfItterations)
{
    byte[] usersHash = getUserHashFromFile(userName);
    byte[] userSalt = getUserSaltFromFile(userName);
    Rfc2898DeriveBytes PBKDF2 = new Rfc2898DeriveBytes(userPassword, userSalt,
numberOfItterations);    //Hash the password with the users salt
    byte[] hashedPassword = PBKDF2.GetBytes(20);    //Returns a 20 byte hash
    bool passwordsMach = comparePasswords(usersHash, hashedPassword);    //Compares byte
arrays
    return passwordsMach;
}
```

AES .

. . salt IV .

```
public static void ProcessFile(string inputPath, string password, bool encryptMode, string
```

```
outputPath)
{
    using (var cypher = new AesManaged())
    using (var fsIn = new FileStream(inputPath, FileMode.Open))
    using (var fsOut = new FileStream(outputPath, FileMode.Create))
    {
        const int saltLength = 256;
        var salt = new byte[saltLength];
        var iv = new byte[cypher.BlockSize / 8];

        if (encryptMode)
        {
            // Generate random salt and IV, then write them to file
            using (var rng = new RNGCryptoServiceProvider())
            {
                rng.GetBytes(salt);
                rng.GetBytes(iv);
            }
            fsOut.Write(salt, 0, salt.Length);
            fsOut.Write(iv, 0, iv.Length);
        }
        else
        {
            // Read the salt and IV from the file
            fsIn.Read(salt, 0, saltLength);
            fsIn.Read(iv, 0, iv.Length);
        }

        // Generate a secure password, based on the password and salt provided
        var pdb = new Rfc2898DeriveBytes(password, salt);
        var key = pdb.GetBytes(cypher.KeySize / 8);

        // Encrypt or decrypt the file
        using (var cryptoTransform = encryptMode
            ? cypher.CreateEncryptor(key, iv)
            : cypher.CreateDecryptor(key, iv))
        using (var cs = new CryptoStream(fsOut, cryptoTransform, CryptoStreamMode.Write))
        {
            fsIn.CopyTo(cs);
        }
    }
}
```

psuedo-random number generator Random () . Crypto RNGCryptoServiceProvider .

Cryptographically Secure , .

```
public static byte[] GenerateRandomData(int length)
{
    var rnd = new byte[length];
    using (var rng = new RNGCryptoServiceProvider())
        rng.GetBytes(rnd);
    return rnd;
}
```

( )

```
public static int GenerateRandomInt(int minVal=0, int maxVal=100)
```

```
{
    var rnd = new byte[4];
    using (var rng = new RNGCryptoServiceProvider())
        rng.GetBytes(rnd);
    var i = Math.Abs(BitConverter.ToInt32(rnd, 0));
    return Convert.ToInt32(i % (maxVal - minVal + 1) + minVal);
}
```

```
public static string GenerateRandomString(int length, string allowableChars=null)
{
    if (string.IsNullOrEmpty(allowableChars))
        allowableChars = @"ABCDEFGHIJKLMNOPQRSTUVWXYZ";

    // Generate random data
    var rnd = new byte[length];
    using (var rng = new RNGCryptoServiceProvider())
        rng.GetBytes(rnd);

    // Generate the output string
    var allowable = allowableChars.ToCharArray();
    var l = allowable.Length;
    var chars = new char[length];
    for (var i = 0; i < length; i++)
        chars[i] = allowable[rnd[i] % l];

    return new string(chars);
}
```

. . . , .

. . .

() ( ) iv () . .

.

```
public static class AsymmetricProvider
{
    #region Key Generation
    public class KeyPair
    {
        public string PublicKey { get; set; }
        public string PrivateKey { get; set; }
    }

    public static KeyPair GenerateNewKeyPair(int keySize = 4096)
    {
        // KeySize is measured in bits. 1024 is the default, 2048 is better, 4096 is more
robust but takes a fair bit longer to generate.
        using (var rsa = new RSACryptoServiceProvider(keySize))
        {
            return new KeyPair {PublicKey = rsa.ToXmlString(false), PrivateKey =
rsa.ToXmlString(true)};
        }
    }

    #endregion
```

```
    #region Asymmetric Data Encryption and Decryption

    public static byte[] EncryptData(byte[] data, string publicKey)
    {
        using (var asymmetricProvider = new RSACryptoServiceProvider())
        {
            asymmetricProvider.FromXmlString(publicKey);
            return asymmetricProvider.Encrypt(data, true);
        }
    }

    public static byte[] DecryptData(byte[] data, string publicKey)
    {
        using (var asymmetricProvider = new RSACryptoServiceProvider())
        {
            asymmetricProvider.FromXmlString(publicKey);
            if (asymmetricProvider.PublicOnly)
                throw new Exception("The key provided is a public key and does not contain the
private key elements required for decryption");
            return asymmetricProvider.Decrypt(data, true);
        }
    }

    public static string EncryptString(string value, string publicKey)
    {
        return Convert.ToBase64String(EncryptData(Encoding.UTF8.GetBytes(value), publicKey));
    }

    public static string DecryptString(string value, string privateKey)
    {
        return Encoding.UTF8.GetString(EncryptData(Convert.FromBase64String(value),
privateKey));
    }

    #endregion

    #region Hybrid File Encryption and Decription

    public static void EncryptFile(string inputFilePath, string outputFilePath, string
publicKey)
    {
        using (var symmetricCypher = new AesManaged())
        {
            // Generate random key and IV for symmetric encryption
            var key = new byte[symmetricCypher.KeySize / 8];
            var iv = new byte[symmetricCypher.BlockSize / 8];
            using (var rng = new RNGCryptoServiceProvider())
            {
                rng.GetBytes(key);
                rng.GetBytes(iv);
            }

            // Encrypt the symmetric key and IV
            var buf = new byte[key.Length + iv.Length];
            Array.Copy(key, buf, key.Length);
            Array.Copy(iv, 0, buf, key.Length, iv.Length);
            buf = EncryptData(buf, publicKey);

            var bufLen = BitConverter.GetBytes(buf.Length);

            // Symmetrically encrypt the data and write it to the file, along with the
```

```
encrypted key and iv
            using (var cypherKey = symmetricCypher.CreateEncryptor(key, iv))
            using (var fsIn = new FileStream(inputFilePath, FileMode.Open))
            using (var fsOut = new FileStream(outputFilePath, FileMode.Create))
            using (var cs = new CryptoStream(fsOut, cypherKey, CryptoStreamMode.Write))
            {
                fsOut.Write(bufLen,0, bufLen.Length);
                fsOut.Write(buf, 0, buf.Length);
                fsIn.CopyTo(cs);
            }
        }
    }

    public static void DecryptFile(string inputFilePath, string outputFilePath, string
privateKey)
    {
        using (var symmetricCypher = new AesManaged())
        using (var fsIn = new FileStream(inputFilePath, FileMode.Open))
        {
            // Determine the length of the encrypted key and IV
            var buf = new byte[sizeof(int)];
            fsIn.Read(buf, 0, buf.Length);
            var bufLen = BitConverter.ToInt32(buf, 0);

            // Read the encrypted key and IV data from the file and decrypt using the
asymmetric algorithm
            buf = new byte[bufLen];
            fsIn.Read(buf, 0, buf.Length);
            buf = DecryptData(buf, privateKey);

            var key = new byte[symmetricCypher.KeySize / 8];
            var iv = new byte[symmetricCypher.BlockSize / 8];
            Array.Copy(buf, key, key.Length);
            Array.Copy(buf, key.Length, iv, 0, iv.Length);

            // Decript the file data using the symmetric algorithm
            using (var cypherKey = symmetricCypher.CreateDecryptor(key, iv))
            using (var fsOut = new FileStream(outputFilePath, FileMode.Create))
            using (var cs = new CryptoStream(fsOut, cypherKey, CryptoStreamMode.Write))
            {
                fsIn.CopyTo(cs);
            }
        }
    }

    #endregion

    #region Key Storage

    public static void WritePublicKey(string publicKeyFilePath, string publicKey)
    {
        File.WriteAllText(publicKeyFilePath, publicKey);
    }
    public static string ReadPublicKey(string publicKeyFilePath)
    {
        return File.ReadAllText(publicKeyFilePath);
    }

    private const string SymmetricSalt = "Stack_Overflow!"; // Change me!

    public static string ReadPrivateKey(string privateKeyFilePath, string password)
```

```
    {
        var salt = Encoding.UTF8.GetBytes(SymmetricSalt);
        var cypherText = File.ReadAllBytes(privateKeyFilePath);

        using (var cypher = new AesManaged())
        {
            var pdb = new Rfc2898DeriveBytes(password, salt);
            var key = pdb.GetBytes(cypher.KeySize / 8);
            var iv = pdb.GetBytes(cypher.BlockSize / 8);

            using (var decryptor = cypher.CreateDecryptor(key, iv))
            using (var msDecrypt = new MemoryStream(cypherText))
            using (var csDecrypt = new CryptoStream(msDecrypt, decryptor,
CryptoStreamMode.Read))
            using (var srDecrypt = new StreamReader(csDecrypt))
            {
                return srDecrypt.ReadToEnd();
            }
        }
    }

    public static void WritePrivateKey(string privateKeyFilePath, string privateKey, string
password)
    {
        var salt = Encoding.UTF8.GetBytes(SymmetricSalt);
        using (var cypher = new AesManaged())
        {
            var pdb = new Rfc2898DeriveBytes(password, salt);
            var key = pdb.GetBytes(cypher.KeySize / 8);
            var iv = pdb.GetBytes(cypher.BlockSize / 8);

            using (var encryptor = cypher.CreateEncryptor(key, iv))
            using (var fsEncrypt = new FileStream(privateKeyFilePath, FileMode.Create))
            using (var csEncrypt = new CryptoStream(fsEncrypt, encryptor,
CryptoStreamMode.Write))
            using (var swEncrypt = new StreamWriter(csEncrypt))
            {
                swEncrypt.Write(privateKey);
            }
        }
    }

    #endregion
}
```

:

```
private static void HybridCryptoTest(string privateKeyPath, string privateKeyPassword, string
inputPath)
{
    // Setup the test
    var publicKeyPath = Path.ChangeExtension(privateKeyPath, ".public");
    var outputPath = Path.Combine(Path.ChangeExtension(inputPath, ".enc"));
    var testPath = Path.Combine(Path.ChangeExtension(inputPath, ".test"));

    if (!File.Exists(privateKeyPath))
    {
        var keys = AsymmetricProvider.GenerateNewKeyPair(2048);
        AsymmetricProvider.WritePublicKey(publicKeyPath, keys.PublicKey);
        AsymmetricProvider.WritePrivateKey(privateKeyPath, keys.PrivateKey,
```

```
privateKeyPassword);
    }

    // Encrypt the file
    var publicKey = AsymmetricProvider.ReadPublicKey(publicKeyPath);
    AsymmetricProvider.EncryptFile(inputPath, outputPath, publicKey);

    // Decrypt it again to compare against the source file
    var privateKey = AsymmetricProvider.ReadPrivateKey(privateKeyPath, privateKeyPassword);
    AsymmetricProvider.DecryptFile(outputPath, testPath, privateKey);

    // Check that the two files match
    var source = File.ReadAllBytes(inputPath);
    var dest = File.ReadAllBytes(testPath);

    if (source.Length != dest.Length)
        throw new Exception("Length does not match");

    if (source.Where((t, i) => t != dest[i]).Any())
        throw new Exception("Data mismatch");
}
```

(System.Security.Cryptography) : https://riptutorial.com/ko/csharp/topic/2988/--system-security-cryptography-

116:

,

- `internal` .
- deault `private`
- getter setter . `private`

setter getter . `public string someProperty {get; private set;}`

Examples

`public` ( ), , .

```
public class Foo()
{
    public string SomeProperty { get; set; }

    public class Baz
    {
        public int Value { get; set; }
    }
}

public class Bar()
{
    public Bar()
    {
        var myInstance = new Foo();
        var someValue = foo.SomeProperty;
        var myNestedInstance = new Foo.Baz();
        var otherValue = myNestedInstance.Value;
    }
}
```

`private` , , .

```
public class Foo()
{
    private string someProperty { get; set; }

    private class Baz
    {
        public string Value { get; set; }
    }

    public void Do()
    {
        var baz = new Baz { Value = 42 };
    }
}

public class Bar()
{
```

```
    public Bar()
    {
        var myInstance = new Foo();

        // Compile Error - not accessible due to private modifier
        var someValue = foo.someProperty;
        // Compile Error - not accessible due to private modifier
        var baz = new Foo.Baz();
    }
}
```

internal ( ), , .

```
internal class Foo
{
    internal string SomeProperty {get; set;}
}

internal class Bar
{
    var myInstance = new Foo();
    internal string SomeField = foo.SomeProperty;

    internal class Baz
    {
        private string blah;
        public int N { get; set; }
    }
}
```

AssemblyInfo.cs .

```
using System.Runtime.CompilerServices;

[assembly:InternalsVisibleTo("MyTests")]
```

`protected` fields, methods .

```
public class Foo()
{
    protected void SomeFooMethod()
    {
        //do something
    }

    protected class Thing
    {
        private string blah;
        public int N { get; set; }
    }
}

public class Bar() : Foo
{
    private void someBarMethod()
    {
        SomeFooMethod(); // inside derived class
```

```
        var thing = new Thing(); // can use nested class
    }
}

public class Baz()
{
    private void someBazMethod()
    {
        var foo = new Foo();
        foo.SomeFooMethod(); //not accessible due to protected modifier
    }
}
```

protected internal ,, .

1

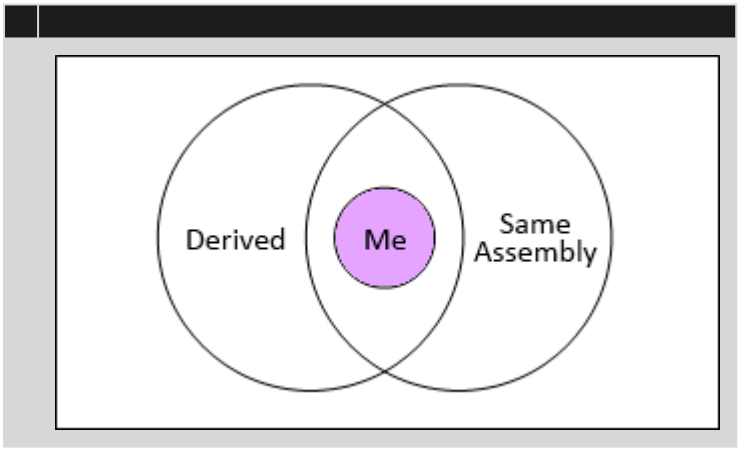
```
public class Foo
{
    public string MyPublicProperty { get; set; }
    protected internal string MyProtectedInternalProperty  { get; set; }

    protected internal class MyProtectedInternalNestedClass
    {
        private string blah;
        public int N { get; set; }
    }
}

public class Bar
{
    void MyMethod1()
    {
        Foo foo = new Foo();
        var myPublicProperty = foo.MyPublicProperty;
        var myProtectedInternalProperty = foo.MyProtectedInternalProperty;
        var myProtectedInternalNestedInstance =
            new Foo.MyProtectedInternalNestedClass();
    }
}
```

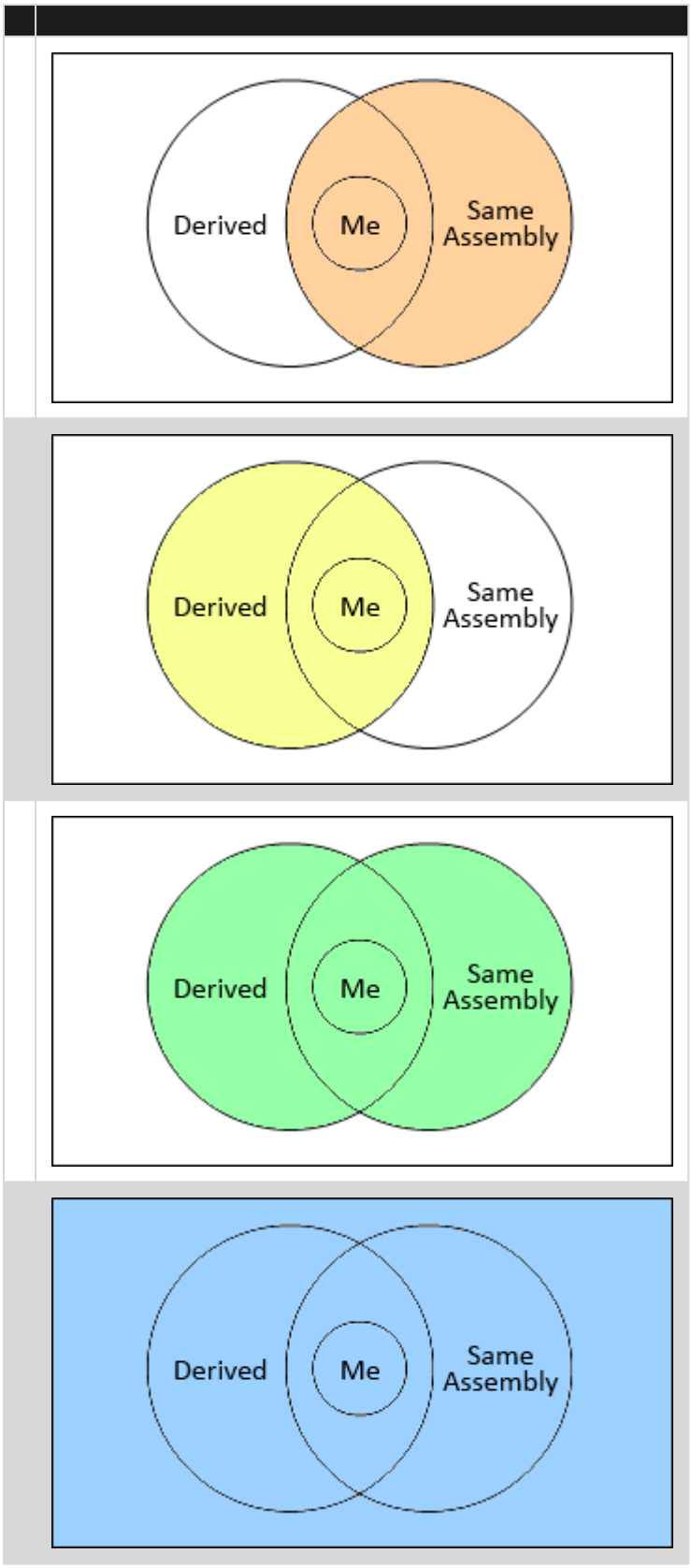
2

```
public class Baz : Foo
{
    void MyMethod1()
    {
        var myPublicProperty = MyPublicProperty;
        var myProtectedInternalProperty = MyProtectedInternalProperty;
        var thing = new MyProtectedInternalNestedClass();
    }

    void MyMethod2()
    {
        Foo foo = new Foo();
        var myPublicProperty = foo.MyPublicProperty;

        // Compile Error
```

```
        var myProtectedInternalProperty = foo.MyProtectedInternalProperty;
        // Compile Error
        var myProtectedInternalNestedInstance =
            new Foo.MyProtectedInternalNestedClass();
    }

}

public class Qux
{
    void MyMethod1()
    {
        Baz baz = new Baz();
        var myPublicProperty = baz.MyPublicProperty;

        // Compile Error
        var myProtectedInternalProperty = baz.MyProtectedInternalProperty;
        // Compile Error
        var myProtectedInternalNestedInstance =
            new Baz.MyProtectedInternalNestedClass();
    }

    void MyMethod2()
    {
        Foo foo = new Foo();
        var myPublicProperty = foo.MyPublicProperty;

        //Compile Error
        var myProtectedInternalProperty = foo.MyProtectedInternalProperty;
        // Compile Error
        var myProtectedInternalNestedInstance =
            new Foo.MyProtectedInternalNestedClass();
    }
}
```

.





.

: https://riptutorial.com/ko/csharp/topic/960/--

117:

Examples

. . .

.

1.
2. ActionFilterAttribute

.

OnActionExecuting - .

OnActionExecuted - .

OnResultExecuting - .

OnResultExecuted - .

.

```
    using System;

    using System.Diagnostics;

    using System.Web.Mvc;



    namespace WebApplication1
    {

        public class MyFirstCustomFilter : ActionFilterAttribute
        {
            public override void OnResultExecuting(ResultExecutingContext filterContext)
            {
                //You may fetch data from database here
                filterContext.Controller.ViewBag.GreetMesssage = "Hello Foo";
                base.OnResultExecuting(filterContext);
            }

            public override void OnActionExecuting(ActionExecutingContext filterContext)
            {
                var controllerName = filterContext.RouteData.Values["controller"];
                var actionName = filterContext.RouteData.Values["action"];
                var message = String.Format("{0} controller:{1} action:{2}",
"onactionexecuting", controllerName, actionName);
                Debug.WriteLine(message, "Action Filter Log");
                base.OnActionExecuting(filterContext);
            }
        }
    }
```

: https://riptutorial.com/ko/csharp/topic/1505/-

118:

C # . . .

"" 2 . Single Responsibility Principle .

2 .

Examples

" "+ " "

.

```
public static object FunctionWithUnknowReturnValues ()
{
    /// anonymous object
    return new { a = 1, b = 2 };
}
```

.

```
/// dynamic object
dynamic x = FunctionWithUnknowReturnValues();

Console.WriteLine(x.a);
Console.WriteLine(x.b);
```

`Tuple<string, MyClass>` `Tuple` .

```
public Tuple<string, MyClass> FunctionWith2ReturnValues ()
{
    return Tuple.Create("abc", new MyClass());
}
```

.

```
Console.WriteLine(x.Item1);
Console.WriteLine(x.Item2);
```

`ref` Argument Reference . `out` `ref` .

: - .

Out Parameter : out , out .

```
static void Main(string[] args)
{
    int a = 2;
```

```
    int b = 3;
    int add = 0;
    int mult= 0;
    AddOrMult(a, b, ref add, ref mult); //AddOrMult(a, b, out add, out mult);
    Console.WriteLine(add); //5
    Console.WriteLine(mult); //6
}

private static void AddOrMult(int a, int b, ref int add, ref int mult) //AddOrMult(int a, int
b, out int add, out int mult)
{
    add = a + b;
    mult = a * b;
}
```

: https://riptutorial.com/ko/csharp/topic/3908/---

119:

C # . (++) . (+, -, *, /) . (? :) C # .

- OperandType operatorSymbol (OperandType operand1)
- public static OperandType operatorSymbol (OperandType operand1, OperandType2 operand2)

operatorSymbol	(: +, -, /, *)
OperandType	.
1	.
2	.
	.

`static methods virtual .`

( ) " ". . ( ) ( ).

- `ab` - .
- `a?.b` - .
- `->` - .
- `f(x)` - .
- `a[x]` - .
- `a?[x]` - .
- `x++` - .
- `x--` - .
- `new` - .
- `default(T)` - `T` .
- `typeof` - `Type` .
- `checked` - .
- `unchecked` - .
- `delegate` - .
- `sizeof` - () .

- `+x` - `x` .
- `-x` - .
- `!x` - .
- `~x` - / .
- `++x` - .
- `--x` - .
- `(T)x` - .

    - `await` - `Task` `await` .
    - `&x` - () `x` .
    - `*x` - .
-
    - `x * y` - .
    - `x / y` - .
    - `x % y` - .
-
    - `x + y` - .
    - `x - y` - .
-
    - `x << y` - .
    - `x >> y` - .
- **/**
    - `x < y` - .
    - `x > y` - .
    - `x <= y` - .
    - `x >= y` - .
    - `is` - .
    - `as` - .
-
    - `x == y` - .
    - `x != y` - .
- **AND**
    - `x & y` - / AND.
- **XOR**
    - `x ^ y` - / XOR.
- **OR**
    - `x | y` - / OR.
- **AND**
    - `x && y` - AND .
- **OR**
    - `x || y` - OR .
- **Null-coalescing**
    - `x ?? y` - `x` . `y` .
-
    - `x ? y : z` - `x` true `y` / . `z` .

- Null-Coalescing
- Null
-

Examples

C # `operator` .

`+` .

`Complex` `Complex` :

```
public struct Complex
{
    public double Real { get; set; }
    public double Imaginary { get; set; }
}
```

`+` . :

```
Complex a = new Complex() { Real = 1, Imaginary = 2 };
Complex b = new Complex() { Real = 4, Imaginary = 8 };
Complex c = a + b;
```

`+` . `operator` .

```
public static Complex operator +(Complex c1, Complex c2)
{
   return new Complex
   {
       Real = c1.Real + c2.Real,
       Imaginary = c1.Imaginary + c2.Imaginary
   };
}
```

`+` , `-` , `*` , `/` . (: `==` `!=`). .

(: `<` `>`).

( ) MSDN - (C #) .

7.0

`operator is` C # 7.0 . .

`Cartesian`

```
public class Cartesian
{
    public int X { get; }
    public int Y { get; }
}
```

operator is Polar

```
public static class Polar
{
    public static bool operator is(Cartesian c, out double R, out double Theta)
    {
        R = Math.Sqrt(c.X*c.X + c.Y*c.Y);
        Theta = Math.Atan2(c.Y, c.X);
        return c.X != 0 || c.Y != 0;
    }
}
```

.

```
var c = Cartesian(3, 4);
if (c is Polar(var R, *))
{
    Console.WriteLine(R);
}
```

(Roslyn Pattern Matching Documentation)

() .

```
"a" == "b"     // Returns false.
"a" == "a"     // Returns true.
1 == 0         // Returns false.
1 == 1         // Returns true.
false == true  // Returns false.
false == false // Returns true.
```

Java , .

. .

```
1 == 1.0              // Returns true because there is an implicit cast from int to double.
new Object() == 1.0   // Will not compile.
MyStruct.AsInt() == 1 // Calls AsInt() on MyStruct and compares the resulting int with 1.
```

Visual Basic.NET .

```
var x = new Object();
var y = new Object();
x == y // Returns false, the operands (objects in this case) have different references.
x == x // Returns true, both operands have the same reference.
```

(=) .

, true true .
true ( ) . , .

.

```
"a" != "b"     // Returns true.
"a" != "a"     // Returns false.
1 != 0         // Returns true.
1 != 1         // Returns false.
false != true  // Returns true.
false != false // Returns false.

var x = new Object();
var y = new Object();
x != y // Returns true, the operands have different references.
x != x // Returns false, both operands have the same reference.
```

equals (==) .

.

```
3 > 5    //Returns false.
1 > 0    //Returns true.
2 > 2    //Return false.

var x = 10;
var y = 15;
x > y    //Returns false.
y > x    //Returns true.
```

.

```
2 < 4     //Returns true.
1 < -3    //Returns false.
2 < 2     //Return false.

var x = 12;
var y = 22;
x < y    //Returns true.
y < x    //Returns false.
```

.

```
7 >= 8    //Returns false.
0 >= 0    //Returns true.
```

.

```
2 <= 4    //Returns true.
1 <= -3    //Returns false.
1 <= 1     //Returns true.
```

.

firstCondition && && secondCondition firstCondition true *secondCondition* ofcource *firstOperand secondOperand* true true . . *NullReferenceException* . .

```
bool hasMoreThanThreeElements = myList != null && mList.Count > 3;
```

mLList ! = null *mList.Count> 3* .

AND

`&&` AND (`&`) .

```
var x = true;
var y = false;

x && x // Returns true.
x && y // Returns false (y is evaluated).
y && x // Returns false (x is not evaluated).
y && y // Returns false (right y is not evaluated).
```

OR

`||` OR (`|`) .

```
var x = true;
var y = false;

x || x // Returns true (right x is not evaluated).
x || y // Returns true (y is not evaluated).
y || x // Returns true (x and y are evaluated).
y || y // Returns false (y and y are evaluated).
```

```
if(object != null && object.Property)
// object.Property is never accessed if object is null, because of the short circuit.
    Action1();
else
    Action2();
```

`int` * .

```
sizeof(bool)    // Returns 1.
sizeof(byte)    // Returns 1.
sizeof(sbyte)   // Returns 1.
sizeof(char)    // Returns 2.
sizeof(short)   // Returns 2.
sizeof(ushort)  // Returns 2.
sizeof(int)     // Returns 4.
sizeof(uint)    // Returns 4.
sizeof(float)   // Returns 4.
sizeof(long)    // Returns 8.
sizeof(ulong)   // Returns 8.
sizeof(double)  // Returns 8.
sizeof(decimal) // Returns 16.
```

* .

`sizeof` .

```
public struct CustomType
{
```

```
    public int value;
}

static void Main()
{
    unsafe
    {
        Console.WriteLine(sizeof(CustomType)); // outputs: 4
    }
}
```

. .

1. object.Equals object.GetHashCode
2. IEquatable<T>.Equals (,)
3. operator == operator != (,)

Equals GetHashCode . Equals . .

Equals == . . == .

.

- .
- : A A (NULL).
- Transitvity : A B B C A C .
- A B A B .
- B C Class2 Class1 : Class1.Equals(A,B) Class2.Equals(A,B) .

```
class Student : IEquatable<Student>
{
    public string Name { get; set; } = "";

    public bool Equals(Student other)
    {
        if (ReferenceEquals(other, null)) return false;
        if (ReferenceEquals(other, this)) return true;
        return string.Equals(Name, other.Name);
    }

    public override bool Equals(object obj)
    {
        if (ReferenceEquals(null, obj)) return false;
        if (ReferenceEquals(this, obj)) return true;

        return Equals(obj as Student);
    }

    public override int GetHashCode()
    {
        return Name?.GetHashCode() ?? 0;
    }

    public static bool operator ==(Student left, Student right)
    {
        return Equals(left, right);
```

```
    }

    public static bool operator !=(Student left, Student right)
    {
        return !Equals(left, right);
    }
}
```

:

```
var now = DateTime.UtcNow;
//accesses member of a class.  In this case the UtcNow property.
```

: Null

```
var zipcode = myEmployee?.Address?.ZipCode;
//returns null if the left operand is null.
//the above is the equivalent of:
var zipcode = (string)null;
if (myEmployee != null && myEmployee.Address != null)
    zipcode = myEmployee.Address.ZipCode;
```

:

```
var age = GetAge(dateOfBirth);
//the above calls the function GetAge passing parameter dateOfBirth.
```

: Aggregate Object Indexing

```
var letters = "letters".ToCharArray();
char letter = letters[1];
Console.WriteLine("Second Letter is {0}",letter);
//in the above example we take the second character from the array
//by calling letters[1]
//NB: Array Indexing starts at 0; i.e. the first letter would be given by letters[0].
```

: Null

```
var letters = null;
char? letter = letters?[1];
Console.WriteLine("Second Letter is {0}",letter);
//in the above example  rather than throwing an error because letters is null
//letter is assigned the value null
```

" "

"exclusive or"(XOR) . ^

bool true .

```
true ^ false   // Returns true
false ^ true   // Returns true
false ^ false  // Returns false
true ^ true    // Returns false
```

. .

-

```
uint value = 15;              // 00001111

uint doubled = value << 1;    // Result = 00011110 = 30
uint shiftFour = value << 4;  // Result = 11110000 = 240
```

```
uint value = 240;             // 11110000

uint halved = value >> 1;     // Result = 01111000 = 120
uint shiftFour = value >> 4;  // Result = 00001111 = 15
```

C # explicit implicit . .

```
public static <implicit/explicit> operator <ResultingType>(<SourceType> myType)
```

. .

implicit explicit :

```
public class BinaryImage
{
    private bool[] _pixels;

    public static implicit operator ColorImage(BinaryImage im)
    {
        return new ColorImage(im);
    }

    public static explicit operator bool[](BinaryImage im)
    {
        return im._pixels;
    }
}
```

:

```
var binaryImage = new BinaryImage();
ColorImage colorImage = binaryImage; // implicit cast, note the lack of type
bool[] pixels = (bool[])binaryImage; // explicit cast, defining the type
```

., .

```
public class BinaryImage
{
    public static explicit operator ColorImage(BinaryImage im)
```

```
    {
        return new ColorImage(im);
    }

    public static explicit operator BinaryImage(ColorImage cm)
    {
        return new BinaryImage(cm);
    }
}
```

, as . explicit implicit .

```
ColorImage cm = myBinaryImage as ColorImage;
```

.

C # = .

:

```
x += y
```

```
x = x + y
```

:

- +=
- -=
- *=
- /=
- %=
- &=
- |=
- ^=
- <<=
- >>=

? :

.

:

```
condition ? expression_if_true : expression_if_false;
```

:

```
string name = "Frank";
Console.WriteLine(name == "Frank" ? "The name is Frank" : "The name is not Frank");
```

. . , .

clamp min max .

```
light.intensity = Clamp(light.intensity, minLight, maxLight);

public static float Clamp(float val, float min, float max)
{
    return (val < min) ? min : (val > max) ? max : val;
}
```

.

```
a ? b ? "a is true, b is true" : "a is true, b is false" : "a is false"

// This is evaluated from left to right and can be more easily seen with parenthesis:

a ? (b ? x : y) : z

// Where the result is x if a && b, y if a && !b, and z if !a
```

.

expression_if_true *expression_if_false* , .

```
condition ? 3 : "Not three"; // Doesn't compile because `int` and `string` lack an implicit
conversion.

condition ? 3.ToString() : "Not three"; // OK because both possible outputs are strings.

condition ? 3 : 3.5; // OK because there is an implicit conversion from `int` to `double`. The
ternary operator will return a `double`.

condition ? 3.5 : 3; // OK because there is an implicit conversion from `int` to `double`. The
ternary operator will return a `double`.
```

.

```
public class Car
{}

public class SportsCar : Car
{}

public class SUV : Car
{}

condition ? new SportsCar() : new Car(); // OK because there is an implicit conversion from
`SportsCar` to `Car`. The ternary operator will return a reference of type `Car`.

condition ? new Car() : new SportsCar(); // OK because there is an implicit conversion from
`SportsCar` to `Car`. The ternary operator will return a reference of type `Car`.

condition ? new SportsCar() : new SUV(); // Doesn't compile because there is no implicit
conversion from `SportsCar` to SUV or `SUV` to `SportsCar`. The compiler is not smart enough
to realize that both of them have an implicit conversion to `Car`.

condition ? new SportsCar() as Car : new SUV() as Car; // OK because both expressions evaluate
```

```
to a reference of type `Car`. The ternary operator will return a reference of type `Car`.
```

System.Type System.Type .

```
System.Type type = typeof(Point)        //System.Drawing.Point
System.Type type = typeof(IDisposable)  //System.IDisposable
System.Type type = typeof(Colors)       //System.Drawing.Color
System.Type type = typeof(List<>)       //System.Collections.Generic.List`1[T]
```

GetType   System.Type .

typeof   .

```
public class Animal {}
public class Dog : Animal {}

var animal = new Dog();

Assert.IsTrue(animal.GetType() == typeof(Animal)); // fail, animal is typeof(Dog)
Assert.IsTrue(animal.GetType() == typeof(Dog));    // pass, animal is typeof(Dog)
Assert.IsTrue(animal is Animal);                   // pass, animal implements Animal
```

(T : struct)

struct , enum   char , int float   . new T() .

```
default(int)            // 0
default(DateTime)       // 0001-01-01 12:00:00 AM
default(char)           // '\0' This is the "null character", not a zero or a line break.
default(Guid)           // 00000000-0000-0000-0000-000000000000
default(MyStruct)       // new MyStruct()

// Note: default of an enum is 0, and not the first *key* in that enum
// so it could potentially fail the Enum.IsDefined test
default(MyEnum)         // (MyEnum)0
```

(T :)

class , interface , . null .

```
default(object)         // null
default(string)         // null
default(MyClass)        // null
default(IDisposable)    // null
default(dynamic)        // null
```

variable , type member   .

```
int counter = 10;
nameof(counter); // Returns "counter"
Client client = new Client();
```

```
nameof(client.Address.PostalCode)); // Returns "PostalCode"
```

`nameof` C # 6.0 . (: `switch` , `case`).). raising , , MVC .

?. (Null)

6.0

C # 6.0 Null `?.` `NullReferenceException` `null` `null` . `null` . . `null` () , `null` nullable . , `Nullable<T>` .

```
var bar = Foo.GetBar()?.Value; // will return null if GetBar() returns null
var baz = Foo.GetBar()?.IntegerValue; // baz will be of type Nullable<int>, i.e. int?
```

. `null` if . Null .

```
event EventHandler<string> RaiseMe;
RaiseMe?.Invoke("Event raised");
```

`X++` `x` `1`

```
var x = 42;
x++;
Console.WriteLine(x); // 43
```

`X--` -

```
var x = 42
x--;
Console.WriteLine(x); // 41
```

`++x` `++x` x `x++` `x++`

```
var x = 42;
Console.WriteLine(++x); // 43
System.out.println(x); // 43
```

```
var x = 42;
Console.WriteLine(x++); // 42
System.out.println(x); // 43
```

for .

```
for(int i = 0; i < 10; i++)
{
}
```

=>

3.0

=> = .

LINQ .

```
string[] words = { "cherry", "apple", "blueberry" };

int shortestWordLength = words.Min((string w) => w.Length); //5
```

LINQ .

```
int shortestWordLength = words.Min(w => w.Length); //also compiles with the same result
```

.

```
(input parameters) => expression
```

=> / / .

```
// expression
(int x, string s) => s.Length > x

// expression
(int x, int y) => x + y

// statement
(string x) => Console.WriteLine(x)

// block
(string x) => {
        x += " says Hello!";
        Console.WriteLine(x);
    }
```

.

```
delegate void TestDelegate(string s);

TestDelegate myDelegate = s => Console.WriteLine(s + " World");

myDelegate("Hello");
```

```
void MyMethod(string s)
{
    Console.WriteLine(s + " World");
}

delegate void TestDelegate(string s);

TestDelegate myDelegate = MyMethod;

myDelegate("Hello");
```

'='

= .

```
int a = 3;     // assigns value 3 to variable a
int b = a = 5; // first assigns value 5 to variable a, then does the same for variable b
Console.WriteLine(a = 3 + 4); // prints 7
```

?? Null-Coalescing

Null-Coalescing `??` null . null , .

```
object foo = null;
object bar = new object();

var c = foo ?? bar;
//c will be bar since foo was null
```

The `??` `if` .

```
//config will be the first non-null returned.
var config = RetrieveConfigOnMachine() ??
             RetrieveConfigFromService() ??
             new DefaultConfiguration();
```

: https://riptutorial.com/ko/csharp/topic/18/

120:

enum byte, sbyte, short, ushort, int, uint, long, ulong . int .

Weekday : byte {Monday = 1, Tuesday = 2, Wednesday = 3, Thursday = 4, Friday = 5}

P / , . enum int int .

- enum {Red, Green, Blue} // Enum
- enum : byte {, , } //
- enum Colors {Red = 23, Green = 45, Blue = 12} //
- Colors.Red // .
- int value = (int) Colors.Red // enum int .
- Colors color = (Colors) intValue // int enum .

Enum (" ") .

( ) . , . .

Examples

```
enum MyEnum
{
    One,
    Two,
    Three
}

foreach(MyEnum e in Enum.GetValues(typeof(MyEnum)))
    Console.WriteLine(e);
```

.

```
One
Two
Three
```

`FlagsAttribute` `ToString()` .

```
[Flags]
enum MyEnum
{
    //None = 0, can be used but not combined in bitwise operations
    FlagA = 1,
    FlagB = 2,
    FlagC = 4,
    FlagD = 8
    //you must use powers of two or combinations of powers of two
    //for bitwise operations to work
}
```

```
var twoFlags = MyEnum.FlagA | MyEnum.FlagB;

// This will enumerate all the flags in the variable: "FlagA, FlagB".
Console.WriteLine(twoFlags);
```

`FlagsAttribute` 2 ( ) . `UInt64` 64 ( ) . `enum` `Int32` `int` . 32 . . 32 .

```
public enum BigEnum : ulong
{
    BigValue = 1 << 63
}
```

"" .

```
[Flags]
enum FlagsEnum
{
    None = 0,
    Option1 = 1,
    Option2 = 2,
    Option3 = 4,

    Default = Option1 | Option3,
    All = Option1 | Option2 | Option3,
}
```

2 10 (<<)

```
[Flags]
enum FlagsEnum
{
    None = 0,
    Option1 = 1 << 0,
    Option2 = 1 << 1,
    Option3 = 1 << 2,

    Default = Option1 | Option3,
    All = Option1 | Option2 | Option3,
}
```

C # 7.0 .

enum `HasFlag` . .

```
[Flags]
enum MyEnum
{
    One = 1,
    Two = 2,
    Three = 4
}
```

`value`

```
var value = MyEnum.One | MyEnum.Two;
```

HasFlag .

```
if(value.HasFlag(MyEnum.One))
    Console.WriteLine("Enum has One");

if(value.HasFlag(MyEnum.Two))
    Console.WriteLine("Enum has Two");

if(value.HasFlag(MyEnum.Three))
    Console.WriteLine("Enum has Three");
```

enum .

```
var type = typeof(MyEnum);
var names = Enum.GetNames(type);

foreach (var name in names)
{
    var item = (MyEnum)Enum.Parse(type, name);

    if (value.HasFlag(item))
        Console.WriteLine("Enum has " + name);
}
```

```
foreach(MyEnum flagToCheck in Enum.GetValues(typeof(MyEnum)))
{
    if(value.HasFlag(flagToCheck))
    {
         Console.WriteLine("Enum has " + flagToCheck);
    }
}
```

.

```
Enum has One
Enum has Two
```

enum .

flags enum .

```
[Flags]
enum FlagsEnum
{
    Option1 = 1,
    Option2 = 2,
    Option3 = 4,
    Option2And3 = Option2 | Option3;

    Default = Option1 | Option3,
}
```

`Default` OR . AND .

```
var value = FlagsEnum.Default;

bool isOption2And3Set = (value & FlagsEnum.Option2And3) == FlagsEnum.Option2And3;

Assert.True(isOption2And3Set);
```

```
public enum DayOfWeek
{
    Sunday,
    Monday,
    Tuesday,
    Wednesday,
    Thursday,
    Friday,
    Saturday
}


// Enum to string
string thursday = DayOfWeek.Thursday.ToString(); // "Thursday"

string seventhDay = Enum.GetName(typeof(DayOfWeek), 6); // "Saturday"

string monday = Enum.GetName(typeof(DayOfWeek), DayOfWeek.Monday); // "Monday"


// String to enum (.NET 4.0+ only - see below for alternative syntax for earlier .NET
versions)
DayOfWeek tuesday;
Enum.TryParse("Tuesday", out tuesday); // DayOfWeek.Tuesday

DayOfWeek sunday;
bool matchFound1 = Enum.TryParse("SUNDAY", out sunday); // Returns false (case-sensitive
match)

DayOfWeek wednesday;
bool matchFound2 = Enum.TryParse("WEDNESDAY", true, out wednesday); // Returns true;
DayOfWeek.Wednesday (case-insensitive match)


// String to enum (all .NET versions)
DayOfWeek friday = (DayOfWeek)Enum.Parse(typeof(DayOfWeek), "Friday"); // DayOfWeek.Friday

DayOfWeek caturday = (DayOfWeek)Enum.Parse(typeof(DayOfWeek), "Caturady"); // Thows
ArgumentException

// All names of an enum type as strings
string[] weekdays = Enum.GetNames(typeof(DayOfWeek));
```

== ZERO

0 . 0 0.

```
public class Program
{
```

```
    enum EnumExample
    {
        one = 1,
        two = 2
    }

    public void Main()
    {
        var e = default(EnumExample);

        if (e == EnumExample.one)
            Console.WriteLine("defaults to one");
        else
            Console.WriteLine("Unknown");
    }
}
```

: https://dotnetfiddle.net/l5Rwie

MSDN :

( ) .

. 0 . , .

```
public enum Day
{
    Monday,
    Tuesday,
    Wednesday,
    Thursday,
    Friday,
    Saturday,
    Sunday
}
```

:

```
// Define variables with values corresponding to specific days
Day myFavoriteDay = Day.Friday;
Day myLeastFavoriteDay = Day.Monday;

// Get the int that corresponds to myFavoriteDay
// Friday is number 4
int myFavoriteDayIndex = (int)myFavoriteDay;

// Get the day that represents number 5
Day dayFive = (Day)5;
```

enum int byte , sbyte , short , ushort , uint , long ulong . int .

```
public enum Day : byte
{
    // same as before
}
```

. .

```
Enum.GetUnderlyingType(typeof(Days)));
```

:

```
System.Byte
```

: .NET

enum

FlagsAttribute . enum enum .

1 : [Flags]

```
[Flags]
enum Colors
{
    Red=1,
    Blue=2,
    Green=4,
    Yellow=8
}

var color = Colors.Red | Colors.Blue;
Console.WriteLine(color.ToString());
```

, .

2 : [Flags]

```
enum Colors
{
    Red=1,
    Blue=2,
    Green=4,
    Yellow=8
}
var color = Colors.Red | Colors.Blue;
Console.WriteLine(color.ToString());
```

3

<<

(<<) enum 2 1 .

enum .

```
[Flags]
public enum MyEnum
```

```
{
    None  = 0,
    Flag1 = 1 << 0,
    Flag2 = 1 << 1,
    Flag3 = 1 << 2,
    Flag4 = 1 << 3,
    Flag5 = 1 << 4,
    ...
    Flag31 = 1 << 30
}
```

`MyEnum Flag30 = 1073741822` (11111111111111111111111111110) `MyEnum` .

enum

enum enum . `System.ComponentModel.DescriptionAttribute` .

:

```
public enum PossibleResults
{
    [Description("Success")]
    OK = 1,
    [Description("File not found")]
    FileNotFound = 2,
    [Description("Access denied")]
    AccessDenied = 3
}
```

enum .

```
public static string GetDescriptionAttribute(PossibleResults result)
{
        return
((DescriptionAttribute)Attribute.GetCustomAttribute((result.GetType().GetField(result.ToString())),
typeof(DescriptionAttribute))).Description;
}

static void Main(string[] args)
{
    PossibleResults result = PossibleResults.FileNotFound;
    Console.WriteLine(result); // Prints "FileNotFound"
    Console.WriteLine(GetDescriptionAttribute(result)); // Prints "File not found"
}
```

enum .

```
static class EnumExtensions
{
    public static string GetDescription(this Enum enumValue)
    {
        return
((DescriptionAttribute)Attribute.GetCustomAttribute((enumValue.GetType().GetField(enumValue.ToString())
typeof(DescriptionAttribute))).Description;
    }
}
```

: `Console.WriteLine(result.GetDescription());`

enum

enum-instance .

```
[Flags]
public enum MyEnum
{
    Flag1 = 1 << 0,
    Flag2 = 1 << 1,
    Flag3 = 1 << 2
}

var value = MyEnum.Flag1;

// set additional value
value |= MyEnum.Flag2;  //value is now Flag1, Flag2
value |= MyEnum.Flag3;  //value is now Flag1, Flag2, Flag3

// remove flag
value &= ~MyEnum.Flag2; //value is now Flag1, Flag3
```

Enum .

.

`DaysOfWeek` 7 `int` .

```
public enum DaysOfWeek
{
    Monday = 1,
    Tuesday = 2,
    Wednesday = 3,
    Thursday = 4,
    Friday = 5,
    Saturday = 6,
    Sunday = 7
}

DaysOfWeek d = (DaysOfWeek)31;
Console.WriteLine(d); // prints 31

DaysOFWeek s = DaysOfWeek.Sunday;
s++; // No error
```

.

enum `Enum.IsDefined` . ,

```
DaysOfWeek d = (DaysOfWeek)31;
Console.WriteLine(Enum.IsDefined(typeof(DaysOfWeek),d)); // prints False
```

: https://riptutorial.com/ko/csharp/topic/931/-

121:

Examples

```
try
{
    /* code that could throw an exception */
}
catch (Exception ex)
{
    /* handle the exception */
}
```

.
.

```
try
{
    /* code to open a file */
}
catch (System.IO.FileNotFoundException)
{
    /* code to handle the file being not found */
}
catch (System.IO.UnauthorizedAccessException)
{
    /* code to handle not being allowed access to the file */
}
catch (System.IO.IOException)
{
    /* code to handle IOException or it's descendant other than the previous two */
}
catch (System.Exception)
{
    /* code to handle other errors */
}
```

. . catch .

throw . C # .

```
Exception ex = new Exception();

// constructor with an overload that takes a message string
Exception ex = new Exception("Error message");
```

throw .

```
try
{
    throw new Exception("Error");
}
```

```
catch (Exception ex)
{
    Console.Write(ex.Message); // Logs 'Error' to the output window
}
```

: catch " " .

```
void DoSomething()
{
    int b=1; int c=5;
    try
    {
        var a = 1;
        b = a - 1;
        c = a / b;
        a = a / c;
    }
    catch (DivideByZeroException dEx) when (b==0)
    {
        // we're throwing the same kind of exception
        throw new DivideByZeroException("Cannot divide by b because it is zero", dEx);
    }
    catch (DivideByZeroException dEx) when (c==0)
    {
        // we're throwing the same kind of exception
        throw new DivideByZeroException("Cannot divide by c because it is zero", dEx);
    }
}

void Main()
{
    try
    {
        DoSomething();
    }
    catch (Exception ex)
    {
        // Logs full error information (incl. inner exception)
        Console.Write(ex.ToString());
    }
}
```

(`ex.InnerException`).

.

System.DivideByZeroException : 0 b . ---> System.DivideByZeroException : 0 .
C UserQuery.g__DoSomething0_0 () : [...] \ LINQPadQuery.cs : line 36
--- ---
C UserQuery.g__DoSomething0_0 () : [...] \ LINQPadQuery.cs : line 42
UserQuery.Main () C : [...] \ LINQPadQuery.cs : 55

LinqPad ( ). .

c = a / b;

`DoSomething()`

.

.NET Fiddle .

```
try
{
    /* code that could throw an exception */
}
catch (Exception)
{
    /* handle the exception */
}
finally
{
    /* Code that will be executed, regardless if an exception was thrown / caught or not */
}
```

`try / catch / finally` .
:

```
FileStream f = null;

try
{
    f = File.OpenRead("file.txt");
    /* process the file here */
}
finally
{
    f?.Close(); // f may be null, so use the null conditional operator.
}
```

try `catch` `finally` . catch . finally .

`FileStream` (`OpenRead`) `using` `IDisposable` .

`try` `return` `finally` . .

- StackOverflow .
- `Environment.FailFast`
- .

WCF IErrorHandler

WCF IErrorHandler . WCF . XML JSON .

IErrorHandler :

```
using System.ServiceModel.Channels;
using System.ServiceModel.Dispatcher;
using System.Runtime.Serialization.Json;
using System.ServiceModel;
using System.ServiceModel.Web;
```

```
namespace BehaviorsAndInspectors
{
    public class ErrorHandler : IErrorHandler
    {

        public bool HandleError(Exception ex)
        {
            // Log exceptions here

            return true;

        } // end

        public void ProvideFault(Exception ex, MessageVersion version, ref Message fault)
        {
            // Get the outgoing response portion of the current context
            var response = WebOperationContext.Current.OutgoingResponse;

            // Set the default http status code
            response.StatusCode = HttpStatusCode.InternalServerError;

            // Add ContentType header that specifies we are using JSON
            response.ContentType = new MediaTypeHeaderValue("application/json").ToString();

            // Create the fault message that is returned (note the ref parameter) with
BaseDataResponseContract
            fault = Message.CreateMessage(
                version,
                string.Empty,
                new CustomReturnType { ErrorMessage = "An unhandled exception occurred!" },
                new DataContractJsonSerializer(typeof(BaseDataResponseContract), new
List<Type> { typeof(BaseDataResponseContract) }));

            if (ex.GetType() == typeof(VariousExceptionTypes))
            {
                 // You might want to catch different types of exceptions here and process
them differently
            }

            // Tell WCF to use JSON encoding rather than default XML
            var webBodyFormatMessageProperty = new
WebBodyFormatMessageProperty(WebContentFormat.Json);
            fault.Properties.Add(WebBodyFormatMessageProperty.Name,
webBodyFormatMessageProperty);

        } // end

    } // end class

} // end namespace
```

. IEndpointBehavior, IContractBehavior IOperationBehavior .

:

```
using System;
using System.Collections.ObjectModel;
using System.ServiceModel;
using System.ServiceModel.Channels;
using System.ServiceModel.Configuration;
```

```
using System.ServiceModel.Description;
using System.ServiceModel.Dispatcher;

namespace BehaviorsAndInspectors
{
    public class ErrorHandlerExtension : BehaviorExtensionElement, IServiceBehavior
    {
        public override Type BehaviorType
        {
            get { return GetType(); }
        }

        protected override object CreateBehavior()
        {
            return this;
        }

        private IErrorHandler GetInstance()
        {
            return new ErrorHandler();
        }

        void IServiceBehavior.AddBindingParameters(ServiceDescription serviceDescription,
ServiceHostBase serviceHostBase, Collection<ServiceEndpoint> endpoints,
BindingParameterCollection bindingParameters) { } // end

        void IServiceBehavior.ApplyDispatchBehavior(ServiceDescription serviceDescription,
ServiceHostBase serviceHostBase)
        {
            var errorHandlerInstance = GetInstance();

            foreach (ChannelDispatcher dispatcher in serviceHostBase.ChannelDispatchers)
            {
                dispatcher.ErrorHandlers.Add(errorHandlerInstance);
            }
        }

        void IServiceBehavior.Validate(ServiceDescription serviceDescription, ServiceHostBase
serviceHostBase) { } // end

    } // end class

} // end namespace
```

Web.config :

```
...
<system.serviceModel>

    <services>
      <service name="WebServices.MyService">
        <endpoint binding="webHttpBinding" contract="WebServices.IMyService" />
      </service>
    </services>

    <extensions>
      <behaviorExtensions>
        <!-- This extension if for the WCF Error Handling-->
        <add name="ErrorHandlerBehavior"
type="WebServices.BehaviorsAndInspectors.ErrorHandlerExtensionBehavior, WebServices,
```

```
Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
      </behaviorExtensions>
    </extensions>

    <behaviors>
      <serviceBehaviors>
        <behavior>
          <serviceMetadata httpGetEnabled="true"/>
          <serviceDebug includeExceptionDetailInFaults="true"/>
          <ErrorHandlerBehavior />
        </behavior>
      </serviceBehaviors>
    </behaviors>

    ....
</system.serviceModel>
...
```

.

https://msdn.microsoft.com/en-us/library/system.servicemodel.dispatcher.ierrorhandler(v=vs.100).aspx

http://www.brainthud.com/cards/5218/25441/which-four-behavior-interfaces-exist-for-interacting-with-a-service-or-client-description-what-methods-do-they-the-

:

HTTP 401 IErrorHandler

IErrorHandler WCF .. ?

- HTTP JSON WCF ?

HttpClient Content-Type ?

throw . .

.

Exception .

```
public class ParserException : Exception
{
    public ParserException() :
      base("The parsing went wrong and we have no additional information.") { }
}
```

.

```
public class ParserException : Exception
{
    public ParserException(string fileName, int lineNumber) :
```

```
      base($"Parser error in {fileName}:{lineNumber}")
    {
      FileName = fileName;
      LineNumber = lineNumber;
    }
    public string FileName {get; private set;}
    public int LineNumber {get; private set;}
}
```

, catch(ParserException x) catch(ParserException x) , catch(ParserException x) .

.

. FormatException . ' **InnerException** ' .

```
//new constructor:
ParserException(string msg, Exception inner) : base(msg, inner) {
}
```

AppDomain . AppDomain . Visual Studio Exception .

```
[Serializable]
public class ParserException : Exception
{
    // Constructor without arguments allows throwing your exception without
    // providing any information, including error message. Should be included
    // if your exception is meaningful without any additional details. Should
    // set message by calling base constructor (default message is not helpful).
    public ParserException()
        : base("Parser failure.")
    {}

    // Constructor with message argument allows overriding default error message.
    // Should be included if users can provide more helpful messages than
    // generic automatically generated messages.
    public ParserException(string message)
        : base(message)
    {}

    // Constructor for serialization support. If your exception contains custom
    // properties, read their values here.
    protected ParserException(SerializationInfo info, StreamingContext context)
        : base(info, context)
    {}
}
```

ParserException

```
try
{
    Process.StartRun(fileName)
```

```
}
catch (ParserException ex)
{
    Console.WriteLine($"{ex.Message} in ${ex.FileName}:${ex.LineNumber}");
}
catch (PostProcessException x)
{
    ...
}
```

catch wrapping . .

```
try
{
    int foo = int.Parse(token);
}
catch (FormatException ex)
{
    //Assuming you added this constructor
    throw new ParserException(
      $"Failed to read {token} as number.",
      FileName,
      LineNumber,
      ex);
}
```

InnerException .

. .

.

```
try
{
  // ...
}
catch (SomeStandardException ex)
{
  // ...
  throw new MyCustomException(someMessage);
}
```

. , . ., .

```
try
{
  // ...
}
catch (IOException ex)
{
  // ...
  throw new StorageServiceException(@"The Storage Service encountered a problem saving
your data. Please consult the inner exception for technical details.
```

```
If you are not able to resolve the problem, please call 555-555-1234 for technical
assistance.", ex);
}
```

---

throw.

```
try
{
    ...
}
catch (Exception ex)
{
    ...
    throw;
}
```

. ! catch rethrow .

```
try
{
    ...
}
catch (Exception ex)
{
    ...
    throw ex;
}
```

---

if-then while . Baseball Exception Handling .

.

```
try
{
    while (AccountManager.HasMoreAccounts())
    {
        account = AccountManager.GetNextAccount();
        if (account.Name == userName)
        {
            //We found it
            throw new AccountFoundException(account);
        }
    }
}
catch (AccountFoundException found)
{
    Console.Write("Here are your account details: " + found.Account.Details.ToString());
}
```

.

```
Account found = null;
```

```
while (AccountManager.HasMoreAccounts() && (found==null))
{
    account = AccountManager.GetNextAccount();
    if (account.Name == userName)
    {
        //We found it
        found = account;
    }
}
Console.Write("Here are your account details: " + found.Details.ToString());
```

()

(!). .

```
try
{
     var f = File.Open(myfile);
     // do something
}
catch (Exception x)
{
     // Assume file not found
     Console.Write("Could not open file");
     // but maybe the error was a NullReferenceException because of a bug in the file handling
code?
}
```

:

```
try
{
     var f = File.Open(myfile);
     // do something which should normally not throw exceptions
}
catch (IOException)
{
     Console.Write("File not found");
}
// Unfortunatelly, this one does not derive from the above, so declare separatelly
catch (UnauthorizedAccessException)
{
     Console.Write("Insufficient rights");
}
```

. .

. .

```
public void DoSomething(String s)
{
     if (s == null)
         throw new ArgumentNullException(nameof(s));
     // Implementation goes here
}
```

```
try
{
     DoSomething(myString);
}
catch(ArgumentNullException x)
{
    // if this happens, we have a programming error and we should check
    // why myString was null in the first place.
}
```

/

. AggregateExceptions . . AggregateExceptions .

. . . , Parallel throw . .

```
    public void Run()
    {
        try
        {
            this.SillyMethod(1, 2);
        }
        catch (AggregateException ex)
        {
            Console.WriteLine(ex.Message);
            foreach (Exception innerException in ex.InnerExceptions)
            {
                Console.WriteLine(innerException.Message);
            }
        }
    }

    private void SillyMethod(int input1, int input2)
    {
        var exceptions = new List<Exception>();

        if (input1 == 1)
        {
            exceptions.Add(new ArgumentException("I do not like ones"));
        }
        if (input2 == 2)
        {
            exceptions.Add(new ArgumentException("I do not like twos"));
        }
        if (exceptions.Any())
        {
            throw new AggregateException("Funny stuff happended during execution",
exceptions);
        }
    }
```

catch

/ `try catch` .

.

```
try
{
//some code here
    try
    {
        //some thing which throws an exception. For Eg : divide by 0
    }
    catch (DivideByZeroException dzEx)
    {
        //handle here only this exception
        //throw from here will be passed on to the parent catch block
    }
    finally
    {
        //any thing to do after it is done.
    }
 //resume from here & proceed as normal;
}
catch(Exception e)
{
    //handle here
}
```

: catch

	.
( ) .	
`throw` .	Re-throw exception - `throw new ArgumentNullException() throw ex`
	.
/ .	/ .
.	.

▪

`. . if-else` `if-else` .

Mr. Bad Practices .

```
// This is a snippet example for DO NOT
object myObject;
void DoingSomethingWithMyObject()
{
    Console.WriteLine(myObject.ToString());
}
```

`Console.WriteLine(myObject.ToString());` `Console.WriteLine(myObject.ToString());`
NullReferenceException throw. Mr. Bad Practice `myObject` null catch & handle

NullReferenceException :

```
// This is a snippet example for DO NOT
object myObject;
void DoingSomethingWithMyObject()
{
    try
    {
        Console.WriteLine(myObject.ToString());
    }
    catch(NullReferenceException ex)
    {
        // Hmmm, if I create a new instance of object and assign it to myObject:
        myObject = new object();
        // Nice, now I can continue to work with myObject
        DoSomethingElseWithMyObject();
    }
}
```

myObject null ? ? Console.WriteLine(myObject.ToString());
Console.WriteLine(myObject.ToString()); ? try...catch ?

Mr. Best Practices ? ?

```
// This is a snippet example for DO
object myObject;
void DoingSomethingWithMyObject()
{
    if(myObject == null)
        myObject = new object();

    // When execution reaches this point, we are sure that myObject is not null
    DoSomethingElseWithMyObject();
}
```

Mr. Best Practices .

throw .

. . . .

.

```
try
{
    //Some code that might throw an exception
}
catch(Exception ex)
{
    //empty catch block, bad practice
}
```

. . .

```
try
{
    //Some code that might throw an exception
}
catch(NullException ex)
{
    LogManager.Log(ex.ToString());
}
```

▪

, . catch. , , , 30 . , " " . .

```
try
{
    //Try to save the data to the main database.
}
catch(SqlException ex)
{
    //Try to save the data to the alternative database.
}
//If anything other than a SqlException is thrown, there is nothing we can do here. Let the
exception bubble up to a level where it can be handled.
```

AppDomain.UnhandledException . . . .

```
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    private static void Main(string[] args)
    {
        AppDomain.CurrentDomain.UnhandledException += new
UnhandledExceptionEventHandler(UnhandledException);
    }
```

Application.ThreadException Windows Forms Windows Forms . ThreadException . .

```
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    private static void Main(string[] args)
    {
        AppDomain.CurrentDomain.UnhandledException += new
UnhandledExceptionEventHandler(UnhandledException);
        Application.ThreadException += new ThreadExceptionEventHandler(ThreadException);
    }
```

```
static void UnhandledException(object sender, UnhandledExceptionEventArgs e)
    {
        Exception ex = (Exception)e.ExceptionObject;
        // your code
```

```
    }

static void ThreadException(object sender, ThreadExceptionEventArgs e)
    {
        Exception ex = e.Exception;
        // your code
    }
```

.

throw .

```
public void WalkInto(Destination destination)
{
    if (destination.Name == "Mordor")
    {
        throw new InvalidOperationException("One does not simply walk into Mordor.");
    }
    // ... Implement your normal walking code here.
}
```

: https://riptutorial.com/ko/csharp/topic/40/-

122:

`nameof` , .

. , IDE .

- ()

Examples

:

`nameof` , . . , IDE .

```
var myString = "String Contents";
Console.WriteLine(nameof(myString));
```

?

myString

"myString" . .

`nameof` . :

```
string greeting = "Hello!";
Object mailMessageBody = greeting;

Console.WriteLine(nameof(greeting)); // Returns "greeting"
Console.WriteLine(nameof(mailMessageBody)); // Returns "mailMessageBody", NOT "greeting"!
```

```
public void DoSomething(int paramValue)
{
    Console.WriteLine(nameof(paramValue));
}

...

int myValue = 10;
DoSomething(myValue);
```

paramValue

PropertyChanged

```
public class Person : INotifyPropertyChanged
{
    private string _address;

    public event PropertyChangedEventHandler PropertyChanged;
```

```
    private void OnPropertyChanged(string propertyName)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }

    public string Address
    {
        get { return _address; }
        set
        {
            if (_address == value)
            {
                return;
            }

            _address = value;
            OnPropertyChanged(nameof(Address));
        }
    }
}

...

var person = new Person();
person.PropertyChanged += (s,e) => Console.WriteLine(e.PropertyName);

person.Address = "123 Fake Street";
```

PropertyChanged

```
public class BugReport : INotifyPropertyChanged
{
    public string Title { ... }
    public BugStatus Status { ... }
}

...

private void BugReport_PropertyChanged(object sender, PropertyChangedEventArgs e)
{
    var bugReport = (BugReport)sender;

    switch (e.PropertyName)
    {
        case nameof(bugReport.Title):
            Console.WriteLine("{0} changed to {1}", e.PropertyName, bugReport.Title);
            break;

        case nameof(bugReport.Status):
            Console.WriteLine("{0} changed to {1}", e.PropertyName, bugReport.Status);
            break;
    }
}

...

var report = new BugReport();
report.PropertyChanged += BugReport_PropertyChanged;
```

```
report.Title = "Everything is on fire and broken";
report.Status = BugStatus.ShowStopper;
```

. .

ShowStopper .

.

```
public class SomeClass<TItem>
{
    public void PrintTypeName()
    {
        Console.WriteLine(nameof(TItem));
    }
}

...

var myClass = new SomeClass<int>();
myClass.PrintTypeName();

Console.WriteLine(nameof(SomeClass<int>));
```

T

SomeClass

```
Console.WriteLine(nameof(CompanyNamespace.MyNamespace));
Console.WriteLine(nameof(MyClass));
Console.WriteLine(nameof(MyClass.MyNestedClass));
Console.WriteLine(nameof(MyNamespace.MyClass.MyNestedClass.MyStaticProperty));
```

MyNamespace

MyNestedClass

MyStaticProperty

Guard

```
public class Order
{
    public OrderLine AddOrderLine(OrderLine orderLine)
    {
        if (orderLine == null) throw new ArgumentNullException(nameof(orderLine));
        ...
    }
}
```

```
public class Order
{
    public OrderLine AddOrderLine(OrderLine orderLine)
    {
```

```
        if (orderLine == null) throw new ArgumentNullException("orderLine");
        ...
    }
}
```

nameof .

MVC

:

```
@Html.ActionLink("Log in", "UserController", "LogIn")
```

.

```
@Html.ActionLink("Log in", @typeof(UserController), @nameof(UserController.LogIn))
```

UserController.LogIn UserController.SignIn , . .

: https://riptutorial.com/ko/csharp/topic/80/

123:

. . C # .

- C # . , .

- . .

Examples

MSDN

```
class Digit
{
    public Digit(double d) { val = d; }
    public double val;

    // User-defined conversion from Digit to double
    public static implicit operator double(Digit d)
    {
        Console.WriteLine("Digit to double implict conversion called");
        return d.val;
    }
    //  User-defined conversion from double to Digit
    public static implicit operator Digit(double d)
    {
        Console.WriteLine("double to Digit implict conversion called");
        return new Digit(d);
    }
}

class Program
{
    static void Main(string[] args)
    {
        Digit dig = new Digit(7);
        //This call invokes the implicit "double" operator
        double num = dig;
        //This call invokes the implicit "Digit" operator
        Digit dig2 = 12;
        Console.WriteLine("num = {0} dig2 = {1}", num, dig2.val);
        Console.ReadLine();
    }
}
```

:

implict
Digit implict conversion double
num = 7 dig2 = 12

.NET Fiddle

```
using System;
namespace TypeConversionApplication
{
   class ExplicitConversion
   {
      static void Main(string[] args)
      {
         double d = 5673.74;
         int i;

         // cast double to int.
         i = (int)d;
         Console.WriteLine(i);
         Console.ReadKey();
      }
   }
}
```

: https://riptutorial.com/ko/csharp/topic/3489/-

124:

Examples

( ) .

.

// .

```
public class Program
{
    // This is the entry point of my program.
    public static void Main()
    {
        // Prints a message to the console. - This is a comment!
        System.Console.WriteLine("Hello, World!");

        // System.Console.WriteLine("Hello, World again!"); // You can even comment out code.
        System.Console.ReadLine();
    }
}
```

/* */ .

```
public class Program
{
    public static void Main()
    {
        /*
            This is a multi line comment
            it will be ignored by the compiler.
        */
        System.Console.WriteLine("Hello, World!");

        // It's also possible to make an inline comment with /* */
        // although it's rarely used in practice
        System.Console.WriteLine(/* Inline comment */ "Hello, World!");

        System.Console.ReadLine();
    }
}
```

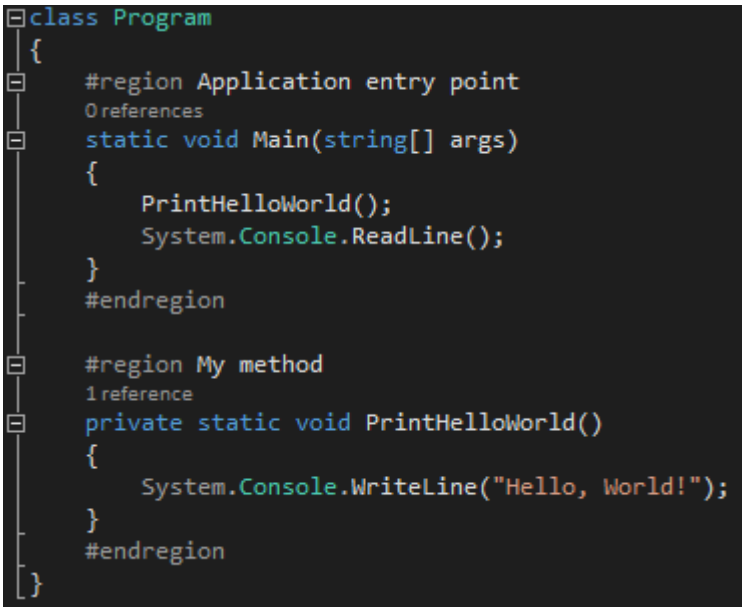
.

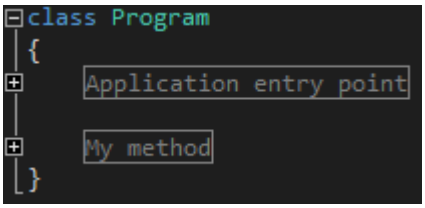
: StyleCop SA1124 DoNotUseRegions . C # .

.

```
class Program
{
    #region Application entry point
    static void Main(string[] args)
    {
        PrintHelloWorld();
        System.Console.ReadLine();
    }
    #endregion

    #region My method
    private static void PrintHelloWorld()
    {
        System.Console.WriteLine("Hello, World!");
    }
    #endregion
}
```

IDE + - .



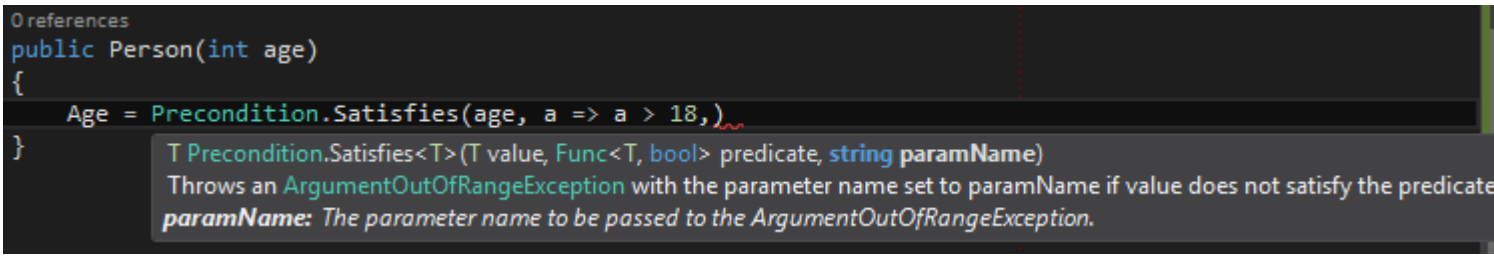


XML API .

```
/// <summary>
/// A helper class for validating method arguments.
/// </summary>
public static class Precondition
{
    /// <summary>
    ///     Throws an <see cref="ArgumentOutOfRangeException"/> with the parameter
    ///     name set to <c>paramName</c> if <c>value</c> does not satisfy the
    ///     <c>predicate</c> specified.
    /// </summary>
    /// <typeparam name="T">
```

```
    ///     The type of the argument checked
    /// </typeparam>
    /// <param name="value">
    ///     The argument to be checked
    /// </param>
    /// <param name="predicate">
    ///     The predicate the value is required to satisfy
    /// </param>
    /// <param name="paramName">
    ///     The parameter name to be passed to the
    ///     <see cref="ArgumentOutOfRangeException"/>.
    /// </param>
    /// <returns>The value specified</returns>
    public static T Satisfies<T>(T value, Func<T, bool> predicate, string paramName)
    {
        if (!predicate(value))
            throw new ArgumentOutOfRangeException(paramName);

        return value;
    }
}
```

IntelliSense .



: https://riptutorial.com/ko/csharp/topic/5346/--

125:

Wikipedia .

. (). (injection) () .

** 5 . (John Munsch) () 5 . . . . " ." . . . (5) () .

** MEF DLL . ILogger MEF . ILogger . **

Examples

MEF

```
public interface ILogger
{
    void Log(string message);
}

[Export(typeof(ILogger))]
[ExportMetadata("Name", "Console")]
public class ConsoleLogger:ILogger
{
    public void Log(string message)
    {
        Console.WriteLine(message);
    }
}

[Export(typeof(ILogger))]
[ExportMetadata("Name", "File")]
public class FileLogger:ILogger
{
    public void Log(string message)
    {
        //Write the message to file
    }
}

public class User
{
    private readonly ILogger logger;
    public User(ILogger logger)
    {
        this.logger = logger;
    }
    public void LogUser(string message)
    {
        logger.Log(message)  ;
    }
}

public interface ILoggerMetaData
{
    string Name { get; }
```

```
}

internal class Program
{
    private CompositionContainer _container;

    [ImportMany]
    private IEnumerable<Lazy<ILogger, ILoggerMetaData>> _loggers;

    private static void Main()
    {
        ComposeLoggers();
        Lazy<ILogger, ILoggerMetaData> loggerNameAndLoggerMapping = _ loggers.First((n) =>
((n.Metadata.Name.ToUpper() =="Console"));
        ILogger logger= loggerNameAndLoggerMapping.Value
        var user = new User(logger);
        user.LogUser("user name");
    }

    private void ComposeLoggers()
    {
        //An aggregate catalog that combines multiple catalogs
        var catalog = new AggregateCatalog();
        string loggersDllDirectory =Path.Combine(Utilities.GetApplicationDirectory(),
"Loggers");
        if (!Directory.Exists(loggersDllDirectory ))
        {
            Directory.CreateDirectory(loggersDllDirectory );
        }
        //Adds all the parts found in the same assembly as the PluginManager class
        catalog.Catalogs.Add(new AssemblyCatalog(typeof(Program).Assembly));
        catalog.Catalogs.Add(new DirectoryCatalog(loggersDllDirectory ));

        //Create the CompositionContainer with the parts in the catalog
        _container = new CompositionContainer(catalog);

        //Fill the imports of this object
        try
        {
            this._container.ComposeParts(this);
        }
        catch (CompositionException compositionException)
        {
            throw new CompositionException(compositionException.Message);
        }
    }
}
```

C # ASP.NET Unity

depedency injection ? . AnimalController .

```
public class AnimalController()
{
    private SantaAndHisReindeer _SantaAndHisReindeer = new SantaAndHisReindeer();

    public AnimalController(){
            Console.WriteLine("");
    }
```

```
}
```

AnimalController _SantaAndHisReindeer . .

Depedency Injection .

Unity DI , :) NuGet ( ) .

Visual Studio Tools -> NuGet Package Manager -> -> -> -> .

.

App-Data UnityConfig.cs UnityMvcActivator.cs

UnityConfig - RegisterTypes .

```
namespace Vegan.WebUi.App_Start
{

public class UnityConfig
{
    #region Unity Container
    private static Lazy<IUnityContainer> container = new Lazy<IUnityContainer>(() =>
    {
        var container = new UnityContainer();
        RegisterTypes(container);
        return container;
    });

    /// <summary>
    /// Gets the configured Unity container.
    /// </summary>
    public static IUnityContainer GetConfiguredContainer()
    {
        return container.Value;
    }
    #endregion

    /// <summary>Registers the type mappings with the Unity container.</summary>
    /// <param name="container">The unity container to configure.</param>
    /// <remarks>There is no need to register concrete types such as controllers or API
controllers (unless you want to
    /// change the defaults), as Unity allows resolving a concrete type even if it was not
previously registered.</remarks>
    public static void RegisterTypes(IUnityContainer container)
    {
        // NOTE: To load from web.config uncomment the line below. Make sure to add a
Microsoft.Practices.Unity.Configuration to the using statements.
        // container.LoadConfiguration();

        // TODO: Register your types here
        // container.RegisterType<IProductRepository, ProductRepository>();

        container.RegisterType<ISanta, SantaAndHisReindeer>();

     }
 }
}
```

UnityMvcActivator -> ASP.NET MVC Unity .

```
using System.Linq;
using System.Web.Mvc;
using Microsoft.Practices.Unity.Mvc;

[assembly:
WebActivatorEx.PreApplicationStartMethod(typeof(Vegan.WebUi.App_Start.UnityWebActivator),
"Start")]
[assembly:
WebActivatorEx.ApplicationShutdownMethod(typeof(Vegan.WebUi.App_Start.UnityWebActivator),
"Shutdown")]

namespace Vegan.WebUi.App_Start
{
/// <summary>Provides the bootstrapping for integrating Unity with ASP.NET MVC.</summary>
public static class UnityWebActivator
{
    /// <summary>Integrates Unity when the application starts.</summary>
    public static void Start()
    {
        var container = UnityConfig.GetConfiguredContainer();


FilterProviders.Providers.Remove(FilterProviders.Providers.OfType<FilterAttributeFilterProvider>().Firs

        FilterProviders.Providers.Add(new UnityFilterAttributeFilterProvider(container));

        DependencyResolver.SetResolver(new UnityDependencyResolver(container));

        // TODO: Uncomment if you want to use PerRequestLifetimeManager
        //
Microsoft.Web.Infrastructure.DynamicModuleHelper.DynamicModuleUtility.RegisterModule(typeof(UnityPerRec

    }

    /// <summary>Disposes the Unity container when the application is shut down.</summary>
    public static void Shutdown()
    {
        var container = UnityConfig.GetConfiguredContainer();
        container.Dispose();
    }
}
}
```

SantAndHisReindeer :)

```
 public class AnimalController()
    {
        private readonly SantaAndHisReindeer _SantaAndHisReindeer;

        public AnimalController(SantaAndHisReindeer SantaAndHisReindeer){

                _SantAndHisReindeer = SantaAndHisReindeer;
        }
    }
```

.

Global.asax.cs UnityWebActivator.Start () . Unity .

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Optimization;
using System.Web.Routing;
using Vegan.WebUi.App_Start;

namespace Vegan.WebUi
{
    public class MvcApplication : System.Web.HttpApplication
    {
        protected void Application_Start()
        {
            AreaRegistration.RegisterAllAreas();
            FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
            RouteConfig.RegisterRoutes(RouteTable.Routes);
            BundleConfig.RegisterBundles(BundleTable.Bundles);
            UnityWebActivator.Start();
        }
    }
}
```

: https://riptutorial.com/ko/csharp/topic/5766/-

126:

C # . .

.NET docs.microsoft.com/dotnet/standard/design-guidelines .

, HorizontalAlignment AlignmentHorizontal .

`CanScrollHorizontally ScrollableX` (X) .

, .

(: `string strName` .

C # .

. . , .

Examples

.

. Pascal case . : `BackColor`

. : `backColor`

. : `IO`

("_") ("-") . .

.

		carName
		AppDomain
		ErrorLevel
		ValueChanged
		WebException
		RedValue
		IDisposable

		ToString
		System.Drawing
		typeName
		BackColor

MSDN .

, . IComponent ICustomAttributeProvider IPersistable .

I , (Pascal) .

.

```
public interface IServiceProvider
public interface IFormatable
```

camelCase _camelCaseWithLeadingUnderscore .

```
public class Rational
{
    private readonly int numerator;
    private readonly int denominator;

    public Rational(int numerator, int denominator)
    {
        // "this" keyword is required to refer to the class-scope field
        this.numerator = numerator;
        this.denominator = denominator;
    }
}
```

```
public class Rational
{
    private readonly int _numerator;
    private readonly int _denominator;

    public Rational(int numerator, int denominator)
    {
        // Names are unique, so "this" keyword is not required
        _numerator = numerator;
        _denominator = denominator;
    }
}
```

.

```
<Company>.(<Product>|<Technology>)[.<Feature>][.<Subnamespace>].
```

:

```
Fabrikam.Math
Litware.Security
```

.

Enum .

```
public enum Volume
{
   Low,
   Medium,
   High
}
```

.

```
[Flags]
public enum MyColors
{
    Yellow = 1,
    Green = 2,
    Red = 4,
    Blue = 8
}
```

: *FlagsAttribute* .

"

```
public enum VolumeEnum // Incorrect
```

```
public enum Color
{
    ColorBlue, // Remove Color, unnecessary
    ColorGreen,
}
```

"

"-Exception" .

.

```
public class MyCustomException : Exception
public class FooException : Exception
```

: https://riptutorial.com/ko/csharp/topic/2330/--

127:

.

() . , Click .

. Button Clicked .

EventArgsT	EventArgs .
EventName	.
	.
SenderObject	.
EventArguments	EventArgsT .

:

- `null` . null . `NullReferenceException` .

6.0

- null (: `EventName`) (: `eventName`) . .

:

```
    if(Changed != null)      // Changed has 1 subscriber at this point
                             // In another thread, that one subscriber decided to unsubscribe
        Changed(this, args); // `Changed` is now null, `NullReferenceException` is thrown.
```

:

```
    // Cache the "Changed" event as a local. If it is not null, then use
    // the LOCAL variable (handler) to raise the event, NOT the event itself.
    var handler = Changed;
    if(handler != null)
        handler(this, args);
```

6.0

- `if` null null (?) . `EventName?.Invoke(SenderObject, new EventArgsT());`
- Action <> / .

Examples

`class` `struct` .

```
public class MyClass
{
    // Declares the event for MyClass
    public event EventHandler MyEvent;

    // Raises the MyEvent event
    public void RaiseEvent()
    {
        OnMyEvent();
    }
}
```

`add` `set` . C # . .

6.0

```
private void OnMyEvent()
{
    EventName?.Invoke(this, EventArgs.Empty);
}
```

6.0

```
private void OnMyEvent()
{
    // Use a local for EventName, because another thread can modify the
    // public EventName between when we check it for null, and when we
    // raise the event.
    var eventName = EventName;

    // If eventName == null, then it means there are no event-subscribers,
    // and therefore, we cannot raise the event.
    if(eventName != null)
        eventName(this, EventArgs.Empty);

}
```

. / .

6.0 C # `EventName?.Invoke` . . `NullReferenceException` . C # 6.0 C # 6 .

:

```
public event EventHandler<EventArgsT> EventName;
```

:

```
public void HandlerName(object sender, EventArgsT args) { /* Handler logic */ }
```

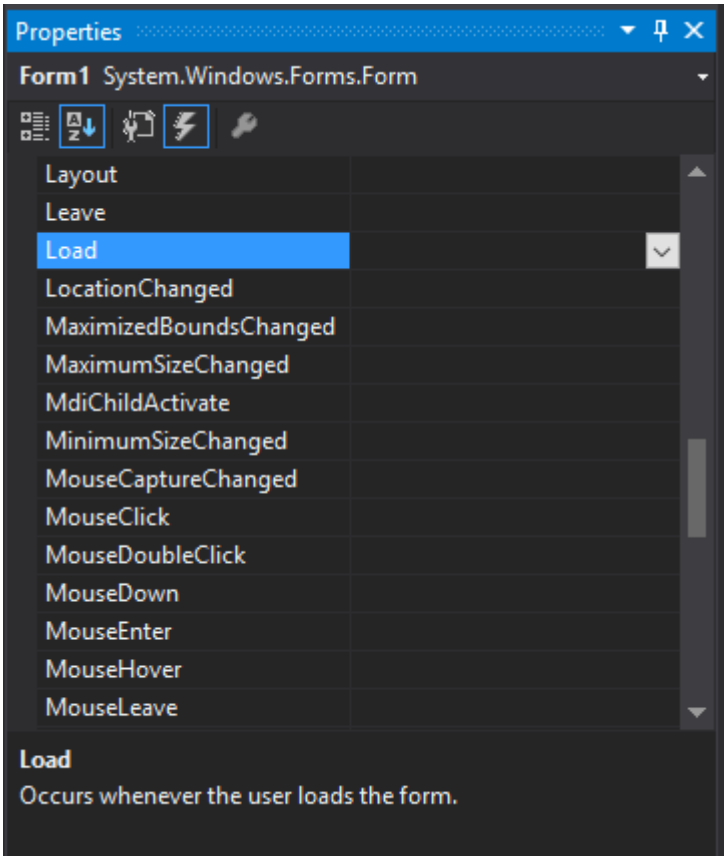
:

:

```
EventName += HandlerName;
```

:

1. (Lightening bolt).
2. .



3. Visual Studio .

```
private void Form1_Load(object sender, EventArgs e)
{

}
```

:

```
EventName(SenderObject, EventArguments);
```

:

```
public event EventHandler<EventArgsType> EventName;
```

=> :

```
EventName += (obj, eventArgs) => { /* Handler logic */ };
```

:

```
EventName += delegate(object obj, EventArgsType eventArgs) { /* Handler Logic */ };
```

.

```
EventName += delegate { /* Handler Logic */ }
```

:

```
EventName?.Invoke(SenderObject, EventArguments);
```

EventHandler EventHandler<T> . :

```
//Declaring an event
public event Action<Param1Type, Param2Type, ...> EventName;
```

EventHandler .

```
//Adding a named event handler
public void HandlerName(Param1Type parameter1, Param2Type parameter2, ...) {
    /* Handler logic */
}
EventName += HandlerName;

//Adding an anonymous event handler
EventName += (parameter1, parameter2, ...) => { /* Handler Logic */ };

//Invoking the event
EventName(parameter1, parameter2, ...);
```

( ).

```
public event EventHandler Event1, Event2, Event3;
```

EventHandler (Event1 , Event2 Event3) .
: C # (v5.0 §13.2.3) .

EventArg

. MouseDown MouseUp MouseEventArgs Location Buttons .

arg :

- EventArgs .
- EventArgs .

Price PriceChangingEventArgs . CurrentPrice NewPrice . Price .

PriceChangingEventArgs

```
public class PriceChangingEventArgs : EventArgs
{
    public PriceChangingEventArgs(int currentPrice, int newPrice)
    {
        this.CurrentPrice = currentPrice;
        this.NewPrice = newPrice;
    }

    public int CurrentPrice { get; private set; }
    public int NewPrice { get; private set; }
}
```

```
public class Product
{
    public event EventHandler<PriceChangingEventArgs> PriceChanging;

    int price;
    public int Price
    {
        get { return price; }
        set
        {
            var e = new PriceChangingEventArgs(price, value);
            OnPriceChanging(e);
            price = value;
        }
    }

    protected void OnPriceChanging(PriceChangingEventArgs e)
    {
        var handler = PriceChanging;
        if (handler != null)
            handler(this, e);
    }
}
```

. .

NewPrice .

```
public int NewPrice { get; set; }
```

e.NewPrice OnPriceChanging Price .

```
int price;
public int Price
{
    get { return price; }
    set
    {
        var e = new PriceChangingEventArgs(price, value);
        OnPriceChanging(e);
        price = e.NewPrice;
    }
}
```

, FormClosing (A) Form .

.

- `CancelEventArgs` arg .
- `EventHandler<T>` cancel arg .

`Price` `PriceChangingEventArgs` . `Value` . `Price` . `Price` .

PriceChangingEventArgs

```
public class PriceChangingEventArgs : CancelEventArgs
{
    int value;
    public int Value
    {
        get { return value; }
    }
    public PriceChangingEventArgs(int value)
    {
        this.value = value;
    }
}
```

```
public class Product
{
    int price;
    public int Price
    {
        get { return price; }
        set
        {
            var e = new PriceChangingEventArgs(value);
            OnPriceChanging(e);
            if (!e.Cancel)
                price = value;
        }
    }

    public event EventHandler<PriceChangingEventArgs> PropertyChanging;
    protected void OnPriceChanging(PriceChangingEventArgs e)
    {
        var handler = PropertyChanging;
        if (handler != null)
            PropertyChanging(this, e);
    }
}
```

. .NET Framework . `EventHandlerList` .

```
public class SampleClass
{
    // Define the delegate collection.
    protected EventHandlerList eventDelegates = new EventHandlerList();

    // Define a unique key for each event.
    static readonly object someEventKey = new object();

    // Define the SomeEvent event property.
```

```
    public event EventHandler SomeEvent
    {
        add
        {
            // Add the input delegate to the collection.
            eventDelegates.AddHandler(someEventKey, value);
        }
        remove
        {
            // Remove the input delegate from the collection.
            eventDelegates.RemoveHandler(someEventKey, value);
        }
    }

    // Raise the event with the delegate specified by someEventKey
    protected void OnSomeEvent(EventArgs e)
    {
        var handler = (EventHandler)eventDelegates[someEventKey];
        if (handler != null)
            handler(this, e);
    }
}
```

WinForms GUI . .

`EventHandlerList` .

: https://riptutorial.com/ko/csharp/topic/64/

128:

.NET C # . .NET . . , .

Examples

`[Serializable]` .

```
[Serializable]
public class Vector
{
    public int X;
    public int Y;
    public int Z;

    [NonSerialized]
    public decimal DontSerializeThis;

    [OptionalField]
    public string Name;
}
```

`[NonSerialized]` . `X` , `Y` , `Z` `Name` serialize.

`[NonSerialized]` `[OptionalField]` . `X` , `Y` `Z` . `DontSerializeThis` `default(decimal)` (0) . `Name` , . , `default(string)` (null). `[OptionalField]` .

`[NonSerialized]` (: `int` 0, `string` null, `bool` false). (,). `[OnDeserializing]` (BEFORE BEFORE `[OnDeserializing]`) `[OnDeserialized]` (`[OnDeserialized]`) `[OnSerializing]` `[OnSerialized]` .

Vector "Rating" 1 . , 0.

```
[Serializable]
public class Vector
{
    public int X;
    public int Y;
    public int Z;

    [NonSerialized]
    public decimal Rating = 1M;

    public Vector()
    {
        Rating = 1M;
    }

    public Vector(decimal initialRating)
    {
        Rating = initialRating;
    }
}
```

1 .

```
[OnDeserializing]
void OnDeserializing(StreamingContext context)
{
    Rating = 1M;
}
```

deserialize .

```
[OnDeserialized]
void OnDeserialized(StreamingContext context)
{
    Rating = 1 + ((X+Y+Z)/3);
}
```

[OnSerializing] [OnSerialized] .

ISerializable

,

ISerializable .

```
[Serializable]
public class Item : ISerializable
{
    private string _name;

    public string Name
    {
        get { return _name; }
        set { _name = value; }
    }

    public Item ()
    {

    }

    protected Item (SerializationInfo info, StreamingContext context)
    {
        _name = (string)info.GetValue("_name", typeof(string));
    }

    public void GetObjectData(SerializationInfo info, StreamingContext context)
    {
        info.AddValue("_name", _name, typeof(string));
    }
}
```

```
info.AddValue("_name", _name, typeof(string));
```

deserialize .

```
_name = (string)info.GetValue("_name", typeof(string));
```

(ISerializationSurrogate)

.

.

ISerializationSurrogate

```
public class ItemSurrogate : ISerializationSurrogate
{
    public void GetObjectData(object obj, SerializationInfo info, StreamingContext context)
    {
        var item = (Item)obj;
        info.AddValue("_name", item.Name);
    }

    public object SetObjectData(object obj, SerializationInfo info, StreamingContext context,
ISurrogateSelector selector)
    {
        var item = (Item)obj;
        item.Name = (string)info.GetValue("_name", typeof(string));
        return item;
    }
}
```

SurrogateSelector IFormatter IFormatter

```
var surrogateSelector = new SurrogateSelector();
surrogateSelector.AddSurrogate(typeof(Item), new StreamingContext(StreamingContextStates.All),
new ItemSurrogate());
var binaryFormatter = new BinaryFormatter
{
    SurrogateSelector = surrogateSelector
};
```

.

```
//this class is not serializable
public class Item
{
    private string _name;

    public string Name
    {
        get { return _name; }
        set { _name = value; }
    }
}
```

```
using System;
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
```

```
namespace BinarySerializationExample
{
    class Item
    {
        private string _name;

        public string Name
        {
            get { return _name; }
            set { _name = value; }
        }
    }

    class ItemSurrogate : ISerializationSurrogate
    {
        public void GetObjectData(object obj, SerializationInfo info, StreamingContext
context)
        {
            var item = (Item)obj;
            info.AddValue("_name", item.Name);
        }

        public object SetObjectData(object obj, SerializationInfo info, StreamingContext
context, ISurrogateSelector selector)
        {
            var item = (Item)obj;
            item.Name = (string)info.GetValue("_name", typeof(string));
            return item;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            var item = new Item
            {
                Name = "Orange"
            };

            var bytes = SerializeData(item);
            var deserializedData = (Item)DeserializeData(bytes);
        }

        private static byte[] SerializeData(object obj)
        {
            var surrogateSelector = new SurrogateSelector();
            surrogateSelector.AddSurrogate(typeof(Item), new
StreamingContext(StreamingContextStates.All), new ItemSurrogate());

            var binaryFormatter = new BinaryFormatter
            {
                SurrogateSelector = surrogateSelector
            };

            using (var memoryStream = new MemoryStream())
            {
                binaryFormatter.Serialize(memoryStream, obj);
                return memoryStream.ToArray();
            }
```

```
        }

        private static object DeserializeData(byte[] bytes)
        {
            var surrogateSelector = new SurrogateSelector();
            surrogateSelector.AddSurrogate(typeof(Item), new
StreamingContext(StreamingContextStates.All), new ItemSurrogate());

            var binaryFormatter = new BinaryFormatter
            {
                SurrogateSelector = surrogateSelector
            };

            using (var memoryStream = new MemoryStream(bytes))
                return binaryFormatter.Deserialize(memoryStream);
        }
    }
}
```

SerializationBinder

```
class MyBinder : SerializationBinder
{
    public override Type BindToType(string assemblyName, string typeName)
    {
        if (typeName.Equals("BinarySerializationExample.Item"))
            return typeof(Item);
        return null;
    }
}
```

.

BinaryFormatter .

```
object DeserializeData(byte[] bytes)
{
    var binaryFormatter = new BinaryFormatter();
    binaryFormatter.Binder = new MyBinder();

    using (var memoryStream = new MemoryStream(bytes))
        return binaryFormatter.Deserialize(memoryStream);
}
```

```
using System;
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;

namespace BinarySerializationExample
{
    class MyBinder : SerializationBinder
    {
        public override Type BindToType(string assemblyName, string typeName)
        {
            if (typeName.Equals("BinarySerializationExample.Item"))
                return typeof(Item);
```

```
            return null;
        }
    }

    [Serializable]
    public class Item
    {
        private string _name;

        public string Name
        {
            get { return _name; }
            set { _name = value; }
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            var item = new Item
            {
                Name = "Orange"
            };

            var bytes = SerializeData(item);
            var deserializedData = (Item)DeserializeData(bytes);
        }

        private static byte[] SerializeData(object obj)
        {
            var binaryFormatter = new BinaryFormatter();
            using (var memoryStream = new MemoryStream())
            {
                binaryFormatter.Serialize(memoryStream, obj);
                return memoryStream.ToArray();
            }
        }

        private static object DeserializeData(byte[] bytes)
        {
            var binaryFormatter = new BinaryFormatter
            {
                Binder = new MyBinder()
            };

            using (var memoryStream = new MemoryStream(bytes))
                return binaryFormatter.Deserialize(memoryStream);
        }
    }
}
```

.

.

1

```
[Serializable]
```

```
class Data
{
    [OptionalField]
    private int _version;

    public int Version
    {
        get { return _version; }
        set { _version = value; }
    }
}
```

. .

.

2

```
[Serializable]
class NewItem
{
    [OptionalField]
    private string _name;

    public string Name
    {
        get { return _name; }
        set { _name = value; }
    }
}

[Serializable]
class Data
{
    [OptionalField]
    private int _version;

    public int Version
    {
        get { return _version; }
        set { _version = value; }
    }

    [OptionalField]
    private List<NewItem> _newItems;

    public List<NewItem> NewItems
    {
        get { return _newItems; }
        set { _newItems = value; }
    }
}
```

```
private static byte[] SerializeData(object obj)
{
    var binaryFormatter = new BinaryFormatter();
    using (var memoryStream = new MemoryStream())
    {
```

```
        binaryFormatter.Serialize(memoryStream, obj);
        return memoryStream.ToArray();
    }
}

private static object DeserializeData(byte[] bytes)
{
    var binaryFormatter = new BinaryFormatter();
    using (var memoryStream = new MemoryStream(bytes))
        return binaryFormatter.Deserialize(memoryStream);
}
```

v2 v1 ?

.

```
System.Runtime.Serialization.SerializationException was unhandled
Message=The ObjectManager found an invalid number of fixups. This usually indicates a problem
in the Formatter.Source=mscorlib
StackTrace:
   at System.Runtime.Serialization.ObjectManager.DoFixups()
   at System.Runtime.Serialization.Formatters.Binary.ObjectReader.Deserialize(HeaderHandler
handler, __BinaryParser serParser, Boolean fCheck, Boolean isCrossAppDomain,
IMethodCallMessage methodCallMessage)
   at System.Runtime.Serialization.Formatters.Binary.BinaryFormatter.Deserialize(Stream
serializationStream, HeaderHandler handler, Boolean fCheck, Boolean isCrossAppDomain,
IMethodCallMessage methodCallMessage)
   at System.Runtime.Serialization.Formatters.Binary.BinaryFormatter.Deserialize(Stream
serializationStream)
   at Microsoft.Samples.TestV1.Main(String[] args) in c:\Users\andrew\Documents\Visual Studio
2013\Projects\vts\CS\V1 Application\TestV1Part2\TestV1Part2.cs:line 29
   at System.AppDomain._nExecuteAssembly(Assembly assembly, String[] args)
   at Microsoft.VisualStudio.HostingProcess.HostProc.RunUsersAssembly()
   at System.Threading.ExecutionContext.Run(ExecutionContext executionContext, ContextCallback
callback, Object state)
   at System.Threading.ThreadHelper.ThreadStart()
```

?

ObjectManager . .

ObjectManager . .

. "IncorrectNumberOfFixups" .

" . .

```
A Note:
Similar code will work correctly if you do not use arrays with new classes
```

?

- ( ) . keyvalue ()
- ISerializable .

: https://riptutorial.com/ko/csharp/topic/4120/-

129:

Examples

(`var`).

```
var anon = new { Foo = 1, Bar = 2 };
// anon.Foo == 1
// anon.Bar == 2
```

/ .

```
int foo = 1;
int bar = 2;
var anon2 = new { foo, bar };
// anon2.foo == 1
// anon2.bar == 2
```

. , .

```
string foo = "some string";
var anon3 = new { foo.Length };
// anon3.Length == 11
var anon4 = new { foo.Length <= 10 ? "short string" : "long string" };
// compiler error - Invalid anonymous type member declarator.
var anon5 = new { Description = foo.Length <= 10 ? "short string" : "long string" };
// OK
```

vs

.

```
var anon = new { Value = 1 };
Console.WriteLine(anon.Id); // compile time error
```

`dynamic` .

```
dynamic val = "foo";
Console.WriteLine(val.Id); // compiles, but throws runtime error
```

.

```
void Log<T>(T obj) {
    // ...
}
Log(new { Value = 10 });
```

LINQ .

```
var products = new[] {
    new { Amount = 10, Id = 0 },
    new { Amount = 20, Id = 1 },
    new { Amount = 15, Id = 2 }
};
var idsByAmount = products.OrderBy(x => x.Amount).Select(x => x.Id);
// idsByAmount: 0, 2, 1
```

. .

```
var anon = new { Foo = 1, Bar = 2 };
var anon2 = new { Foo = 5, Bar = 10 };
List<T> CreateList<T>(params T[] items) {
    return new List<T>(items);
}

var list1 = CreateList(anon, anon2);
```

`List<T>` `ToList` LINQ `List<T>` .

```
var list2 = new[] {anon, anon2}.ToList();
```

`Equals` . `a.Prop.Equals(b.Prop)`) .

```
var anon = new { Foo = 1, Bar = 2 };
var anon2 = new { Foo = 1, Bar = 2 };
var anon3 = new { Foo = 5, Bar = 10 };
var anon3 = new { Foo = 5, Bar = 10 };
var anon4 = new { Bar = 2, Foo = 1 };
// anon.Equals(anon2) == true
// anon.Equals(anon3) == false
// anon.Equals(anon4) == false (anon and anon4 have different types, see below)
```

.

```
var anon = new { Foo = 1, Bar = 2 };
var anon2 = new { Foo = 7, Bar = 1 };
var anon3 = new { Bar = 1, Foo = 3 };
var anon4 = new { Fa = 1, Bar = 2 };
// anon and anon2 have the same type
// anon and anon3 have diferent types (Bar and Foo appear in different orders)
// anon and anon4 have different types (property names are different)
```

.

```
var arr = new[] {
    new { Id = 0 },
    new { Id = 1 }
};
```

: https://riptutorial.com/ko/csharp/topic/765/-

130:

- public this [IndexType index] {get {...} set {...}}

.

- , .
- .
- .
- .
- .

Examples

```
class Foo
{
    private string[] cities = new[] { "Paris", "London", "Berlin" };

    public string this[int index]
    {
        get {
            return cities[index];
        }
        set {
            cities[index] = value;
        }
    }
}
```

:

```
    var foo = new Foo();

    // access a value
    string berlin = foo[2];

    // assign a value
    foo[0] = "Rome";
```

2

```
interface ITable {
    // an indexer can be declared in an interface
    object this[int x, int y] { get; set; }
}

class DataTable : ITable
{
    private object[,] cells = new object[10, 10];

    /// <summary>
```

```
    /// implementation of the indexer declared in the interface
    /// </summary>
    /// <param name="x">X-Index</param>
    /// <param name="y">Y-Index</param>
    /// <returns>Content of this cell</returns>
    public object this[int x, int y]
    {
        get
        {
            return cells[x, y];
        }
        set
        {
            cells[x, y] = value;
        }
    }
}
```

SparseArray .

. O (1) , 100 . SparseArray

```
class SparseArray
    {
        Dictionary<int, string> array = new Dictionary<int, string>();

        public string this[int i]
        {
            get
            {
                if(!array.ContainsKey(i))
                {
                    return null;
                }
                return array[i];
            }
            set
            {
                if(!array.ContainsKey(i))
                    array.Add(i, value);
            }
        }
    }
```

: https://riptutorial.com/ko/csharp/topic/1660/

131:

Examples

. interface " : InterfaceName . .
: InterfaceName, ISecondInterface

```
public interface INoiseMaker
{
    string MakeNoise();
}

public class Cat : INoiseMaker
{
    public string MakeNoise()
    {
        return "Nyan";
    }
}

public class Dog : INoiseMaker
{
    public string MakeNoise()
    {
        return "Woof";
    }
}
```

INoiseMaker cat dog string MakeNoise() string MakeNoise() string MakeNoise() .

```
public interface IAnimal
{
    string Name { get; set; }
}

public interface INoiseMaker
{
    string MakeNoise();
}

public class Cat : IAnimal, INoiseMaker
{
    public Cat()
    {
        Name = "Cat";
    }

    public string Name { get; set; }

    public string MakeNoise()
    {
        return "Nyan";
    }
}
```

.).

```
interface IChauffeur
{
    string Drive();
}

interface IGolfPlayer
{
    string Drive();
}

class GolfingChauffeur : IChauffeur, IGolfPlayer
{
    public string Drive()
    {
        return "Vroom!";
    }

    string IGolfPlayer.Drive()
    {
        return "Took a swing...";
    }
}


GolfingChauffeur obj = new GolfingChauffeur();
IChauffeur chauffeur = obj;
IGolfPlayer golfer = obj;

Console.WriteLine(obj.Drive()); // Vroom!
Console.WriteLine(chauffeur.Drive()); // Vroom!
Console.WriteLine(golfer.Drive()); // Took a swing...
```

.

```
public class Golfer : IGolfPlayer
{
    string IGolfPlayer.Drive()
    {
        return "Swinging hard...";
    }
    public void Swing()
    {
        Drive(); // Compiler error: No such method
    }
}
```

.

.

```
public class ProGolfer : IGolfPlayer
{
    string IGolfPlayer.Swear() // Error
    {
        return "The ball is in the pit";
    }
```

```
}
```

.

:

. .

:

. IGolfPlayer.Drive Drive .

. .

IShape . .

```
public interface IShape
{
    double ComputeArea();
}
```

: Rectangle Circle

```
public class Rectangle : IShape
{
    private double length;
    private double width;

    public Rectangle(double length, double width)
    {
        this.length = length;
        this.width = width;
    }

    public double ComputeArea()
    {
        return length * width;
    }
}

public class Circle : IShape
{
    private double radius;

    public Circle(double radius)
    {
        this.radius = radius;
    }

    public double ComputeArea()
    {
        return Math.Pow(radius, 2.0) * Math.PI;
```

```
    }
}
```

, . IShape .

```
private static void Main(string[] args)
{
    var shapes = new List<IShape>() { new Rectangle(5, 10), new Circle(5) };
    ComputeArea(shapes);

    Console.ReadKey();
}

private static void ComputeArea(IEnumerable<IShape> shapes)
{
    foreach (shape in shapes)
    {
        Console.WriteLine("Area: {0:N}, shape.ComputeArea());
    }
}

// Output:
// Area : 50.00
// Area : 78.54
```

"" ., .

:

- .
- .
- *( )
- ""
- 1 "" .

```
public interface ICanDoThis{
    void TheThingICanDo();
    int SomeValueProperty { get; set; }
}
```

:

- "|" .
- ";" .
- .

```
public class MyClass : ICanDoThis {
    public void TheThingICanDo(){
        // do the thing
    }

    public int SomeValueProperty { get; set; }
    public int SomeValueNotImplemtingAnything { get; set; }
}
```

.

```
ICanDoThis obj = new MyClass();

// ok
obj.TheThingICanDo();

// ok
obj.SomeValueProperty = 5;

// Error, this member doesn't exist in the interface
obj.SomeValueNotImplemtingAnything = 5;

// in order to access the property in the class you must "down cast" it
((MyClass)obj).SomeValueNotImplemtingAnything = 5; // ok
```

WinForms WPF UI . ? :

```
public class MyTextBlock : TextBlock {
    public void SetText(string str){
        this.Text = str;
    }
}

public class MyButton : Button {
    public void SetText(string str){
        this.Content = str;
    }
}
```

"" . .

```
public interface ITextControl{
    void SetText(string str);
}

public class MyTextBlock : TextBlock, ITextControl {
    public void SetText(string str){
        this.Text = str;
    }
}

public class MyButton : Button, ITextControl {
    public void SetText(string str){
        this.Content = str;
    }

    public int Clicks { get; set; }
}
```

MyButton MyTextBlock .

```
var controls = new List<ITextControls>{
    new MyTextBlock(),
    new MyButton()
};
```

```
foreach(var ctrl in controls){
    ctrl.SetText("This text will be applied to both controls despite them being different");


    // Compiler Error, no such member in interface
    ctrl.Clicks = 0;

    // Runtime Error because 1 class is in fact not a button which makes this cast invalid
    ((MyButton)ctrl).Clicks = 0;


    /* the solution is to check the type first.
    This is usually considered bad practice since
    it's a symptom of poor abstraction */
    var button = ctrl as MyButton;
    if(button != null)
        button.Clicks = 0; // no errors


}
```

""

? !

```
public interface IMessageService {
    void OnMessageRecieve();
    void SendMessage();
    string Result { get; set; }
    int Encoding { get; set; }
    // yadda yadda
}
```

.

```
public class MyObjectWithMessages : IMessageService {
     public void OnMessageRecieve(){

     }

     public void SendMessage(){

     }

     public string Result { get; set; }
     public int Encoding { get; set; }
}
```

.

```
var obj = new MyObjectWithMessages();

// why would i want to call this function?
obj.OnMessageRecieve();
```

: . .

.

```
public class MyObjectWithMessages : IMessageService{
    void IMessageService.OnMessageRecieve() {

    }

    void IMessageService.SendMessage() {

    }

    string IMessageService.Result { get; set; }
    int IMessageService.Encoding { get; set; }
}
```

.

```
var obj = new MyObjectWithMessages();

/* error member does not exist on type MyObjectWithMessages.
 * We've succesfully made it "private" */
obj.OnMessageRecieve();
```

, .

```
((IMessageService)obj).OnMessageRecieve();
```

IComparable

. IComparable IComparable<T> .

. , ID .

```
public class Item {

    public string name; // though public variables are generally bad practice,
    public int idNumber; // to keep this example simple we will use them instead
    public decimal price; // of a property.

    // body omitted for brevity

}
```

Item List<Item> , ID . , List<T> Sort() . Item List<T> List<T> . IComparable .

CompareTo CompareTo "" , 0 " " .

```
Item apple = new Item();
apple.idNumber = 15;
Item banana = new Item();
banana.idNumber = 4;
Item cow = new Item();
cow.idNumber = 15;
Item diamond = new Item();
```

```
diamond.idNumber = 18;

Console.WriteLine(apple.CompareTo(banana)); // 11
Console.WriteLine(apple.CompareTo(cow)); // 0
Console.WriteLine(apple.CompareTo(diamond)); // -3
```

Item .

```
public class Item : IComparable<Item> {

    private string name;
    private int idNumber;
    private decimal price;

    public int CompareTo(Item otherItem) {

        return (this.idNumber - otherItem.idNumber);

    }

    // rest of code omitted for brevity

}
```

CompareTo ID ?

List<Item> Sort() List Item CompareTo . List<T> Item Item .

: https://riptutorial.com/ko/csharp/topic/2208/

132:

ExpressionBuilder . .

```
 private static readonly MethodInfo ContainsMethod = typeof(string).GetMethod("Contains",
new[] { typeof(string) });
 private static readonly MethodInfo StartsWithMethod = typeof(string).GetMethod("StartsWith",
new[] { typeof(string) });
 private static readonly MethodInfo EndsWithMethod = typeof(string).GetMethod("EndsWith",
new[] { typeof(string) });
```

GetExpression :

- Expression GetExpression<T>
- BinaryExpression GetExpression<T>
- ConstantExpression GetConstant

.

Examples

QueryFilter

.

```
public class QueryFilter
{
    public string PropertyName { get; set; }
    public string Value { get; set; }
    public Operator Operator { get; set; }

    // In the query {a => a.Name.Equals("Pedro")}
    // Property name to filter - propertyName = "Name"
    // Filter value - value = "Pedro"
    // Operation to perform - operation = enum Operator.Equals
    public QueryFilter(string propertyName, string value, Operator operatorValue)
    {
        PropertyName = propertyName;
        Value = value;
        Operator = operatorValue;
    }
}
```

:

```
 public enum Operator
{
    Contains,
    GreaterThan,
    GreaterThanOrEqual,
    LessThan,
    LessThanOrEqualTo,
```

```
    StartsWith,
    EndsWith,
    Equals,
    NotEqual
}
```

GetExpression

```
public static Expression<Func<T, bool>> GetExpression<T>(IList<QueryFilter> filters)
{
    Expression exp = null;

    // Represents a named parameter expression. {parm => parm.Name.Equals()}, it is the param
part
    // To create a ParameterExpression need the type of the entity that the query is against
an a name
    // The type is possible to find with the generic T and the name is fixed parm
    ParameterExpression param = Expression.Parameter(typeof(T), "parm");

    // It is good parctice never trust in the client, so it is wise to validate.
    if (filters.Count == 0)
        return null;

    // The expression creation differ if there is one, two or more filters.
    if (filters.Count != 1)
    {
        if (filters.Count == 2)
            // It is result from direct call.
            // For simplicity sake the private overloads will be explained in another example.
            exp = GetExpression<T>(param, filters[0], filters[1]);
        else
        {
            // As there is no method for more than two filters,
            // I iterate through all the filters and put I in the query two at a time
            while (filters.Count > 0)
            {
                // Retreive the first two filters
                var f1 = filters[0];
                var f2 = filters[1];

                // To build a expression with a conditional AND operation that evaluates
                // the second operand only if the first operand evaluates to true.
                // It needed to use the BinaryExpression a Expression derived class
                // That has the AndAlso method that join two expression together
                exp = exp == null ? GetExpression<T>(param, filters[0], filters[1]) :
Expression.AndAlso(exp, GetExpression<T>(param, filters[0], filters[1]));

                // Remove the two just used filters, for the method in the next iteration
finds the next filters
                filters.Remove(f1);
                filters.Remove(f2);

                // If it is that last filter, add the last one and remove it
                if (filters.Count == 1)
                {
                    exp = Expression.AndAlso(exp, GetExpression<T>(param, filters[0]));
                    filters.RemoveAt(0);
                }
            }
```

```
        }
    }
    else
        // It is result from direct call.
        exp = GetExpression<T>(param, filters[0]);

           // converts the Expression into Lambda and retuns the query
    return Expression.Lambda<Func<T, bool>>(exp, param);
}
```

GetExpression

:

. .

```
private static Expression GetExpression<T>(ParameterExpression param, QueryFilter queryFilter)
{
    // Represents accessing a field or property, so here we are accessing for example:
    // the property "Name" of the entity
    MemberExpression member = Expression.Property(param, queryFilter.PropertyName);

    //Represents an expression that has a constant value, so here we are accessing for
example:
    // the values of the Property "Name".
    // Also for clarity sake the GetConstant will be explained in another example.
    ConstantExpression constant = GetConstant(member.Type, queryFilter.Value);

    // With these two, now I can build the expression
    // every operator has it one way to call, so the switch will do.
    switch (queryFilter.Operator)
    {
        case Operator.Equals:
            return Expression.Equal(member, constant);

        case Operator.Contains:
            return Expression.Call(member, ContainsMethod, constant);

        case Operator.GreaterThan:
            return Expression.GreaterThan(member, constant);

        case Operator.GreaterThanOrEqual:
            return Expression.GreaterThanOrEqual(member, constant);

        case Operator.LessThan:
            return Expression.LessThan(member, constant);

        case Operator.LessThanOrEqualTo:
            return Expression.LessThanOrEqual(member, constant);

        case Operator.StartsWith:
            return Expression.Call(member, StartsWithMethod, constant);

        case Operator.EndsWith:
            return Expression.Call(member, EndsWithMethod, constant);
    }
```

```
    return null;
}
```

BinaryExpresion .

```
private static BinaryExpression GetExpression<T>(ParameterExpression param, QueryFilter
filter1, QueryFilter filter2)
{
    // Built two separated expression and join them after.
    Expression result1 = GetExpression<T>(param, filter1);
    Expression result2 = GetExpression<T>(param, filter2);
    return Expression.AndAlso(result1, result2);
}
```

ConstantExpression

ConstantExpression MemberExpression . ConstantExpression .

```
private static ConstantExpression GetConstant(Type type, string value)
{
    // Discover the type, convert it, and create ConstantExpression
    ConstantExpression constant = null;
    if (type == typeof(int))
    {
        int num;
        int.TryParse(value, out num);
        constant = Expression.Constant(num);
    }
    else if(type == typeof(string))
    {
        constant = Expression.Constant(value);
    }
    else if (type == typeof(DateTime))
    {
        DateTime date;
        DateTime.TryParse(value, out date);
        constant = Expression.Constant(date);
    }
    else if (type == typeof(bool))
    {
        bool flag;
        if (bool.TryParse(value, out flag))
        {
            flag = true;
        }
        constant = Expression.Constant(flag);
    }
    else if (type == typeof(decimal))
    {
        decimal number;
        decimal.TryParse(value, out number);
        constant = Expression.Constant(number);
    }
```

```
    return constant;
}
```

= List (); QueryFilter = QueryFilter ("", "", Operator.StartsWith); filters.Add ();

```
 Expression<Func<Food, bool>> query = ExpressionBuilder.GetExpression<Food>(filters);
```

Food "Burger" .

:

```
query = {parm => a.parm.StartsWith("Burger")}


Expression<Func<T, bool>> GetExpression<T>(IList<QueryFilter> filters)
```

: https://riptutorial.com/ko/csharp/topic/6721/---

133: C # (csc.exe)

Examples

C

C # C # . C # (csc.exe) . `%WINDIR%\Microsoft.NET\Framework64\v4.0.30319\csc.exe`

: .NET Framework .

C # (: C #) [ ] .).

1. `Windows Key` + `R` .
2. `notepad` `Enter` .
3. .
4. `ConsoleApp.cs` , ... → , `ConsoleApp.cs` , '' `All Files` .
5. `Save` `Save`

1. `Windows` + `R` .
2. .

```
%WINDIR%\Microsoft.NET\Framework64\v4.0.30319\csc.exe /t:exe
/out:"C:\Users\yourUserName\Documents\ConsoleApp.exe"
"C:\Users\yourUserName\Documents\ConsoleApp.cs"
```

`ConsoleApp.cs` . (`ConsoleApp.exe`) . `ConsoleApp.exe` .

! . .

```
using System;

namespace ConsoleApp
{
    class Program
    {
        private static string input = String.Empty;

        static void Main(string[] args)
        {
            goto DisplayGreeting;

            DisplayGreeting:
            {
                Console.WriteLine("Hello! What is your name?");

                input = Console.ReadLine();
```

```
                if (input.Length >= 1)
                {
                    Console.WriteLine(
                        "Hello, " +
                        input +
                        ", enter 'Exit' at any time to exit this app.");

                    goto AwaitFurtherInstruction;
                }
                else
                {
                    goto DisplayGreeting;
                }
            }

            AwaitFurtherInstruction:
            {
                input = Console.ReadLine();

                if(input.ToLower() == "exit")
                {
                    input = String.Empty;

                    Environment.Exit(0);
                }
                else
                {
                    goto AwaitFurtherInstruction;
                }
            }
        }
    }
}
```

C # (csc.exe) : https://riptutorial.com/ko/csharp/topic/6676/---c-sharp---csc-exe------

134:

Examples

```
string helloWorld = "hello world, how is it going?";
string[] parts1 = helloWorld.Split(',');

//parts1: ["hello world", " how is it going?"]

string[] parts2 = helloWorld.Split(' ');

//parts2: ["hello", "world,", "how", "is", "it", "going?"]
```

```
string helloWorld = "Hello World!";
string world = helloWorld.Substring(6); //world = "World!"
string hello = helloWorld.Substring(0,5); // hello = "Hello"
```

Substring Substring ( ).

.

```
string HelloWorld = "Hello World";
HelloWorld.StartsWith("Hello"); // true
HelloWorld.StartsWith("Foo"); // false
```

System.String.Contains . true false .

```
string s = "Hello World";
bool stringExists = s.Contains("ello");  //stringExists =true as the string contains the
substring
```

/ .

String.Trim()

```
string x = "   Hello World!    ";
string y = x.Trim(); // "Hello World!"

string q = "{(Hi!*";
string r = q.Trim( '(', '*', '{' ); // "Hi!"
```

String.TrimStart() String.TrimEnd()

```
string q = "{(Hi*";
string r = q.TrimStart( '{' ); // "(Hi*"
string s = q.TrimEnd( '*' );   // "{(Hi"
```

String.Format()

.

```
String.Format("Hello {0} Foo {1}", "World", "Bar") //Hello World Foo Bar
```

```
var parts = new[] { "Foo", "Bar", "Fizz", "Buzz"};
var joined = string.Join(", ", parts);

//joined = "Foo, Bar, Fizz, Buzz"
```

.

```
string s = "Foo";
string paddedLeft = s.PadLeft(5);        // paddedLeft = "  Foo" (pads with spaces by default)
string paddedRight = s.PadRight(6, '+'); // paddedRight = "Foo+++"
string noPadded = s.PadLeft(2);          // noPadded = "Foo" (original string is never
shortened)
```

Array .

String.Join / String.Join . . . Array .

char array :

```
string delimiter=",";
char[] charArray = new[] { 'a', 'b', 'c' };
string inputString = String.Join(delimiter, charArray);
```

: a,b,c "" delimiter abc .

List of char :

```
string delimiter = "|";
List<char> charList = new List<char>() { 'a', 'b', 'c' };
string inputString = String.Join(delimiter, charList);
```

: a|b|c

List of Strings :

```
string delimiter = " ";
List<string> stringList = new List<string>() { "Ram", "is", "a","boy" };
string inputString = String.Join(delimiter, stringList);
```

: Ram is a boy

array of strings :

```
string delimiter = "_";
string[] stringArray = new [] { "Ram", "is", "a","boy" };
string inputString = String.Join(delimiter, stringArray);
```

: `Ram_is_a_boy`

ToString

`String.Format` , `.ToString` . ToString . .

:

```
int intValue = 10;
string zeroPaddedInteger = intValue.ToString("000"); // Output will be "010"
string customFormat = intValue.ToString("Input value is 0"); // output will be   "Input value
is 10"
```

:

```
double doubleValue = 10.456;
string roundedDouble = doubleValue.ToString("0.00"); // output 10.46
string integerPart = doubleValue.ToString("00");    // output 10
string customFormat = doubleValue.ToString("Input value is 0.0");  // Input value is 10.5
```

ToString DateTime

```
DateTime currentDate = DateTime.Now; //  {7/21/2016 7:23:15 PM}
string dateTimeString = currentDate.ToString("dd-MM-yyyy HH:mm:ss"); // "21-07-2016 19:23:15"
string dateOnlyString = currentDate.ToString("dd-MM-yyyy"); // "21-07-2016"
string dateWithMonthInWords = currentDate.ToString("dd-MMMM-yyyy HH:mm:ss"); // "21-July-2016
19:23:15"
```

x

Visual Basic , Left, Right Mid . C # `Substring()` . .

```
   public static class StringExtensions
   {
      /// <summary>
      /// VB Left function
      /// </summary>
      /// <param name="stringparam"></param>
      /// <param name="numchars"></param>
      /// <returns>Left-most numchars characters</returns>
      public static string Left( this string stringparam, int numchars )
      {
         // Handle possible Null or numeric stringparam being passed
         stringparam += string.Empty;

         // Handle possible negative numchars being passed
         numchars = Math.Abs( numchars );

         // Validate numchars parameter
         if (numchars > stringparam.Length)
            numchars = stringparam.Length;

         return stringparam.Substring( 0, numchars );
      }
```

```
/// <summary>
/// VB Right function
/// </summary>
/// <param name="stringparam"></param>
/// <param name="numchars"></param>
/// <returns>Right-most numchars characters</returns>
public static string Right( this string stringparam, int numchars )
{
   // Handle possible Null or numeric stringparam being passed
   stringparam += string.Empty;

   // Handle possible negative numchars being passed
   numchars = Math.Abs( numchars );

   // Validate numchars parameter
   if (numchars > stringparam.Length)
      numchars = stringparam.Length;

   return stringparam.Substring( stringparam.Length - numchars );
}

/// <summary>
/// VB Mid function - to end of string
/// </summary>
/// <param name="stringparam"></param>
/// <param name="startIndex">VB-Style startindex, 1st char startindex = 1</param>
/// <returns>Balance of string beginning at startindex character</returns>
public static string Mid( this string stringparam, int startindex )
{
   // Handle possible Null or numeric stringparam being passed
   stringparam += string.Empty;

   // Handle possible negative startindex being passed
   startindex = Math.Abs( startindex );

   // Validate numchars parameter
   if (startindex > stringparam.Length)
      startindex = stringparam.Length;

   // C# strings are zero-based, convert passed startindex
   return stringparam.Substring( startindex - 1 );
}

/// <summary>
/// VB Mid function - for number of characters
/// </summary>
/// <param name="stringparam"></param>
/// <param name="startIndex">VB-Style startindex, 1st char startindex = 1</param>
/// <param name="numchars">number of characters to return</param>
/// <returns>Balance of string beginning at startindex character</returns>
public static string Mid( this string stringparam, int startindex, int numchars)
{
   // Handle possible Null or numeric stringparam being passed
   stringparam += string.Empty;

   // Handle possible negative startindex being passed
   startindex = Math.Abs( startindex );

   // Handle possible negative numchars being passed
   numchars = Math.Abs( numchars );
```

```
        // Validate numchars parameter
        if (startindex > stringparam.Length)
           startindex = stringparam.Length;

        // C# strings are zero-based, convert passed startindex
        return stringparam.Substring( startindex - 1, numchars );

      }
   }
```

.

```
string myLongString = "Hello World!";
string myShortString = myLongString.Right(6);  // "World!"
string myLeftString = myLongString.Left(5);    // "Hello"
string myMidString1 = myLongString.Left(4);    // "lo World"
string myMidString2 = myLongString.Left(2,3);    // "ell"
```

String.IsNullOrEmpty () String.IsNullOrWhiteSpace () .

```
string nullString = null;
string emptyString = "";
string whitespaceString = "    ";
string tabString = "\t";
string newlineString = "\n";
string nonEmptyString = "abc123";

bool result;

result = String.IsNullOrEmpty(nullString);            // true
result = String.IsNullOrEmpty(emptyString);           // true
result = String.IsNullOrEmpty(whitespaceString);      // false
result = String.IsNullOrEmpty(tabString);             // false
result = String.IsNullOrEmpty(newlineString);         // false
result = String.IsNullOrEmpty(nonEmptyString);        // false

result = String.IsNullOrWhiteSpace(nullString);       // true
result = String.IsNullOrWhiteSpace(emptyString);      // true
result = String.IsNullOrWhiteSpace(tabString);        // true
result = String.IsNullOrWhiteSpace(newlineString);    // true
result = String.IsNullOrWhiteSpace(whitespaceString); // true
result = String.IsNullOrWhiteSpace(nonEmptyString);   // false
```

Substring . .

```
string s = "hello";
char c = s[1]; //Returns 'e'
```

string Substring Substring , char .

.

```
string s = "hello";
foreach (char c in s)
```

```
    Console.WriteLine(c);
/********* This will print each character on a new line:
h
e
l
l
o
**********/
```

2 , 8 16

1. 10 2 **2 .**

```
Int32 Number = 15;
Console.WriteLine(Convert.ToString(Number, 2));  //OUTPUT : 1111
```

2. 10 8 **8 .**

```
int Number = 15;
Console.WriteLine(Convert.ToString(Number, 8));  //OUTPUT : 17
```

3. 10 16 **16**

```
var Number = 15;
Console.WriteLine(Convert.ToString(Number, 16));  //OUTPUT : f
```

```
string str = "this--is--a--complete--sentence";
string[] tokens = str.Split(new[] { "--" }, StringSplitOptions.None);
```

:

["this", "is", "a", "complete", "sentence"]

.

```
char[] a = s.ToCharArray();
System.Array.Reverse(a);
string r = new string(a);
```

.
NULL .

Glyph / GraphemeCluster ( ) .

"" .

:

.

. . .

grapheme . , a ä graphemes (: ä . a diaresis)). (:)
.

( ) , . , . , , 2 . OTF GSUB GPOS . .

C # CodePoint.

, Les Misé rables .

```
string s = "Les Mise\u0301rables";
```

.

e R .
string.reverse.reverse .

GraphemeCluster .
.

```
    private static System.Collections.Generic.List<string> GraphemeClusters(string s)
    {
        System.Collections.Generic.List<string> ls = new
System.Collections.Generic.List<string>();

        System.Globalization.TextElementEnumerator enumerator =
System.Globalization.StringInfo.GetTextElementEnumerator(s);
        while (enumerator.MoveNext())
        {
            ls.Add((string)enumerator.Current);
        }

        return ls;
    }


    // this
    private static string ReverseGraphemeClusters(string s)
    {
        if(string.IsNullOrEmpty(s) || s.Length == 1)
             return s;

        System.Collections.Generic.List<string> ls = GraphemeClusters(s);
        ls.Reverse();

        return string.Join("", ls.ToArray());
    }

    public static void TestMe()
    {
        string s = "Les Mise\u0301rables";
        // s = "noël";
        string r = ReverseGraphemeClusters(s);

        // This would be wrong:
```

```
        // char[] a = s.ToCharArray();
        // System.Array.Reverse(a);
        // string r = new string(a);

        System.Console.WriteLine(r);
    }
```

-, - / / (/ /) .

System.String.Replace .

```
string s = "Hello World";
 s = s.Replace("World", "Universe"); // s = "Hello Universe"
```

.

String.Empty .

```
string s = "Hello World";
s = s.Replace("ell", String.Empty); // s = "Ho World"
```

String /

System.String .

- System.String.ToLowerInvariant String .
- System.String.ToUpperInvariant String .

: . .

:

```
string s = "My String";
s = s.ToLowerInvariant(); // "my string"
s = s.ToUpperInvariant(); // "MY STRING"
```

String.ToLower (CultureInfo) String.ToUpper (CultureInfo) .

System.String.Join .

```
string[] words = {"One", "Two", "Three", "Four"};
string singleString = String.Join(",", words); // singleString = "One,Two,Three,Four"
```

System.String.Concat + .

```
string first = "Hello ";
string second = "World";

string concat = first + second; // concat = "Hello World"
concat = String.Concat(first, second); // concat = "Hello World"
```

C # 6 .

```
string concat = $"{first},{second}";
```

: https://riptutorial.com/ko/csharp/topic/73/--

135:

Examples

Parallel.ForEach

Parallel.ForEach URL ping .

```
static void Main()
{
    string [] urls =
    {
        "www.stackoverflow.com",
        "www.google.net",
        "www.facebook.com",
        "www.twitter.com"
    };

    System.Threading.Tasks.Parallel.ForEach(urls, url =>
    {
        var ping = new System.Net.NetworkInformation.Ping();

        var result = ping.Send(url);

        if (result.Status == System.Net.NetworkInformation.IPStatus.Success)
        {
            Console.WriteLine(string.Format("{0} is online", url));
        }
    });
}
```

Parallel.For

Parallel.For URL ping .

```
static void Main()
{
    string [] urls =
    {
        "www.stackoverflow.com",
        "www.google.net",
        "www.facebook.com",
        "www.twitter.com"
    };

    System.Threading.Tasks.Parallel.For(0, urls.Length, i =>
    {
        var ping = new System.Net.NetworkInformation.Ping();

        var result = ping.Send(urls[i]);

        if (result.Status == System.Net.NetworkInformation.IPStatus.Success)
        {
            Console.WriteLine(string.Format("{0} is online", urls[i]));
```

```
        }
    });
}
```

Parallel.Invoke

( )

```
static void Main()
{
    string [] urls =
    {
        "www.stackoverflow.com",
        "www.google.net",
        "www.facebook.com",
        "www.twitter.com"
    };

    System.Threading.Tasks.Parallel.Invoke(
        () => PingUrl(urls[0]),
        () => PingUrl(urls[1]),
        () => PingUrl(urls[2]),
        () => PingUrl(urls[3])
    );
}

void PingUrl(string url)
{
    var ping = new System.Net.NetworkInformation.Ping();

    var result = ping.Send(url);

    if (result.Status == System.Net.NetworkInformation.IPStatus.Success)
    {
        Console.WriteLine(string.Format("{0} is online", url));
    }
}
```

```
public class Foo
{
    private const int TASK_ITERATION_DELAY_MS = 1000;
    private CancellationTokenSource _cts;

    public Foo()
    {
        this._cts = new CancellationTokenSource();
    }

    public void StartExecution()
    {
        Task.Factory.StartNew(this.OwnCodeCancelableTask_EveryNSeconds, this._cts.Token);
    }

    public void CancelExecution()
    {
        this._cts.Cancel();
    }
```

```
    /// <summary>
    /// "Infinite" loop that runs every N seconds. Good for checking for a heartbeat or
updates.
    /// </summary>
    /// <param name="taskState">The cancellation token from our _cts field, passed in the
StartNew call</param>
    private async void OwnCodeCancelableTask_EveryNSeconds(object taskState)
    {
        var token = (CancellationToken)taskState;

        while (!token.IsCancellationRequested)
        {
            Console.WriteLine("Do the work that needs to happen every N seconds in this
loop");

            // Passing token here allows the Delay to be cancelled if your task gets
cancelled.
            await Task.Delay(TASK_ITERATION_DELAY_MS, token);
        }
    }
}
```

CancellationTokenSource

```
public class Foo
{
    private CancellationTokenSource _cts;

    public Foo()
    {
        this._cts = new CancellationTokenSource();
    }

    public void StartExecution()
    {
        Task.Factory.StartNew(this.OwnCodeCancelableTask, this._cts.Token);
    }

    public void CancelExecution()
    {
        this._cts.Cancel();
    }

    /// <summary>
    /// "Infinite" loop with no delays. Writing to a database while pulling from a buffer for
example.
    /// </summary>
    /// <param name="taskState">The cancellation token from our _cts field, passed in the
StartNew call</param>
    private void OwnCodeCancelableTask(object taskState)
    {
        var token = (CancellationToken) taskState; //Our cancellation token passed from
StartNew();

        while ( !token.IsCancellationRequested )
        {
            Console.WriteLine("Do your task work in this loop");
        }
    }
```

```
}
```

PingUrl

```
    static void Main(string[] args)
    {
        string url = "www.stackoverflow.com";
        var pingTask = PingUrlAsync(url);
        Console.WriteLine($"Waiting for response from {url}");
        Task.WaitAll(pingTask);
        Console.WriteLine(pingTask.Result);
    }

    static async Task<string> PingUrlAsync(string url)
    {
        string response = string.Empty;
        var ping = new System.Net.NetworkInformation.Ping();

        var result = await ping.SendPingAsync(url);

        await Task.Delay(5000); //simulate slow internet

        if (result.Status == System.Net.NetworkInformation.IPStatus.Success)
        {
            response = $"{url} is online";
        }

        return response;
    }
```

: https://riptutorial.com/ko/csharp/topic/1010/--

136: (TPL)

Examples

JoinBlock

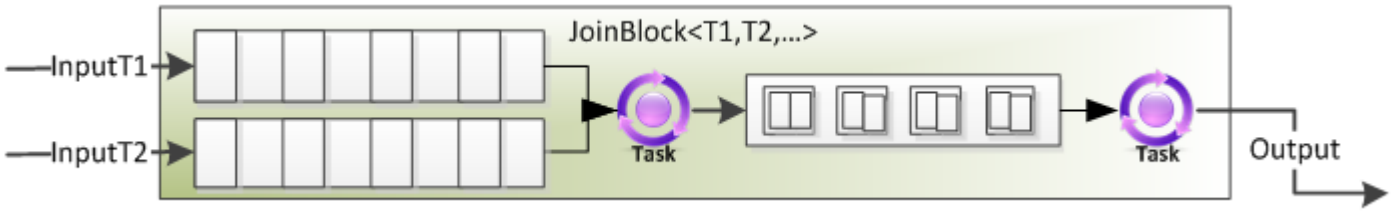
(2-3)

BatchBlock JoinBlock <T1, T2, ...> . JoinBlock <T1, T2, ...> .

JoinBlock <string, double, int> ISourceBlock <Tuple <string, double, int >>.

BatchBlock JoinBlock <T1, T2, ...>

- .
- non-greedy . 2 . .




```
var throttle = new JoinBlock<ExpensiveObject, Request>();
for(int i=0; i<10; i++)
{
    requestProcessor.Target1.Post(new ExpensiveObject());
}

var processor = new Transform<Tuple<ExpensiveObject, Request>, ExpensiveObject>(pair =>
{
    var resource = pair.Item1;
    var request = pair.Item2;

    request.ProcessWith(resource);

    return resource;
});

throttle.LinkTo(processor);
processor.LinkTo(throttle.Target1);
```

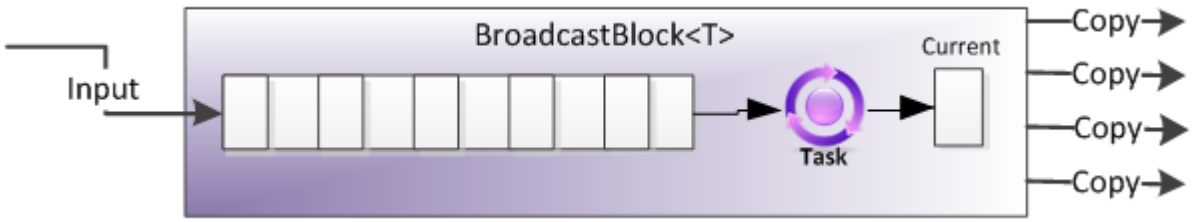
TPL Stephen Toub

BroadcastBlock

( )

BufferBlock , BroadcastBlock , "" .

BufferBlock BroadcastBlock . (FIFO). .



Throttled Producer /

```
var ui = TaskScheduler.FromCurrentSynchronizationContext();
var bb = new BroadcastBlock<ImageData>(i => i);

var saveToDiskBlock = new ActionBlock<ImageData>(item =>
    item.Image.Save(item.Path)
);

var showInUiBlock = new ActionBlock<ImageData>(item =>
    imagePanel.AddImage(item.Image),
    new DataflowBlockOptions { TaskScheduler =
TaskScheduler.FromCurrentSynchronizationContext() }
);

bb.LinkTo(saveToDiskBlock);
bb.LinkTo(showInUiBlock);
```

```
public class MyAgent
{
    public ISourceBlock<string> Status { get; private set; }

    public MyAgent()
    {
        Status = new BroadcastBlock<string>();
        Run();
    }

    private void Run()
    {
        Status.Post("Starting");
        Status.Post("Doing cool stuff");
        …
        Status.Post("Done");
    }
}
```

TPL Stephen Toub

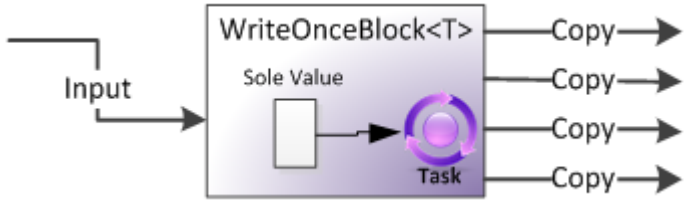
WriteOnceBlock

(Readonly : .)

BufferBlock TPL Dataflow WriteOnceBlock .

.

WriteOnceBlock C # readonly . , .



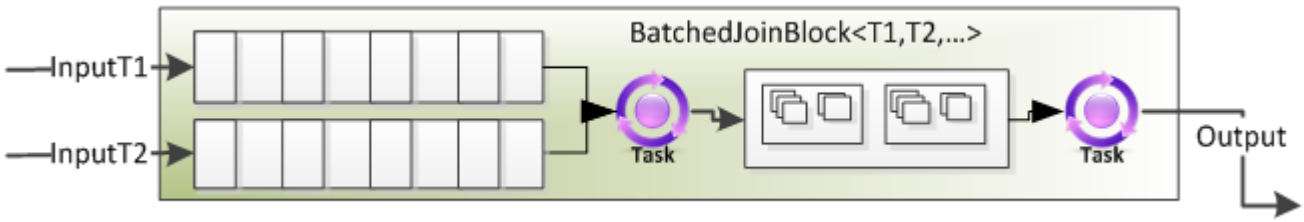

```
public static async void SplitIntoBlocks(this Task<T> task,
    out IPropagatorBlock<T> result,
    out IPropagatorBlock<Exception> exception)
{
    result = new WriteOnceBlock<T>(i => i);
    exception = new WriteOnceBlock<Exception>(i => i);

    try
    {
        result.Post(await task);
    }
    catch(Exception ex)
    {
        exception.Post(ex);
    }
}
```

TPL Stephen Toub

BatchedJoinBlock

(2-3 .)

BatchedJoinBlock <T1, T2, ...> BatchBlock JoinBlock <T1, T2, ...> .
JoinBlock <T1, T2, ...> BatchBlock N BatchedJoinBlock <T1, T2, ...> N .



I

N / .

```
var batchedJoin = new BatchedJoinBlock<string, Exception>(10);

for (int i=0; i<10; i++)
{
    Task.Factory.StartNew(() => {
        try { batchedJoin.Target1.Post(DoWork()); }
        catch(Exception ex) { batchJoin.Target2.Post(ex); }
    });
```

```
}

var results = await batchedJoin.ReceiveAsync();

foreach(string s in results.Item1)
{
    Console.WriteLine(s);
}

foreach(Exception e in results.Item2)
{
    Console.WriteLine(e);
}
```

TPL Stephen Toub

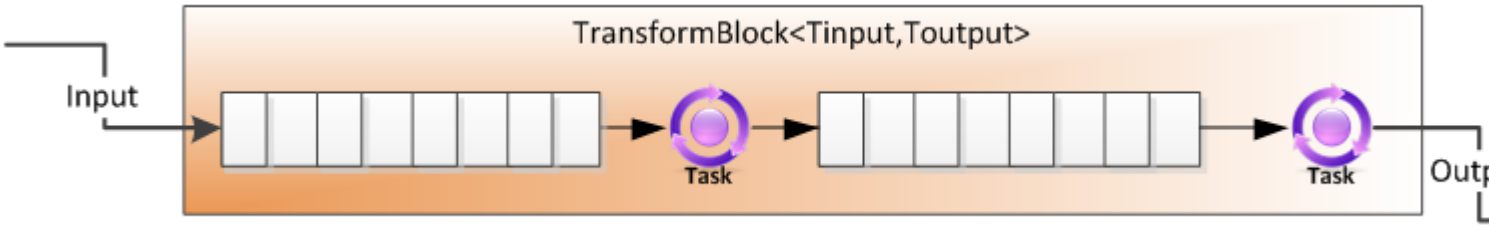
TransformBlock

(,)

ActionBlock , TransformBlock <TInput, TOutput> . **ActionBlock , .** Func <TInput, TOutput> , Func <TInput, Task> . . LINQ Select () . Select .

TransformBlock <TInput, TOutput> MaxDegreeOfParallelism 1 .).

. .

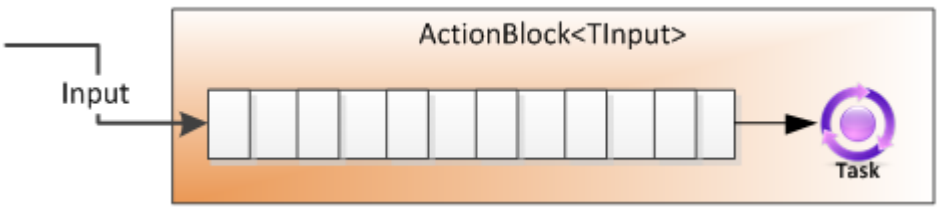



```
var compressor = new TransformBlock<byte[], byte[]>(input => Compress(input));
var encryptor = new TransformBlock<byte[], byte[]>(input => Encrypt(input));

compressor.LinkTo(Encryptor);
```

TPL Stephen Toub

()

. ActionBlock ActionBlock "" . ActionBlock .



```
var ab = new ActionBlock<TInput>(i =>
{
    Compute(i);
});
…
ab.Post(1);
ab.Post(2);
ab.Post(3);
```

5

```
var downloader = new ActionBlock<string>(async url =>
{
    byte [] imageData = await DownloadAsync(url);
    Process(imageData);
}, new DataflowBlockOptions { MaxDegreeOfParallelism = 5 });

downloader.Post("http://website.com/path/to/images");
downloader.Post("http://another-website.com/path/to/images");
```

TPL Stephen Toub

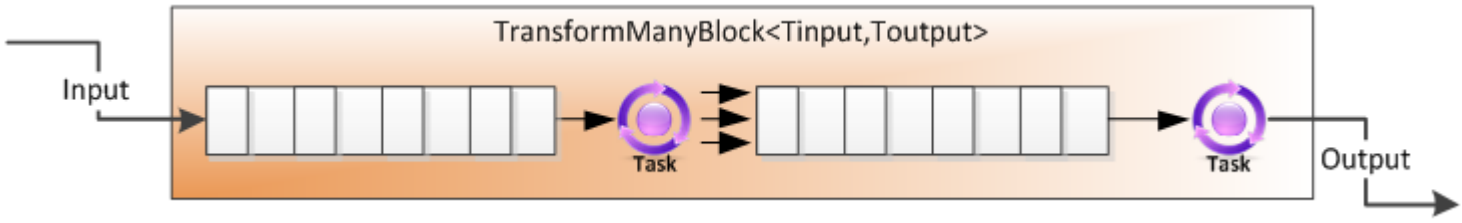
TransformManyBlock

(SelectMany, 1-m : LINQ SelectMany "".)

TransformManyBlock <TInput, TOutput> TransformBlock <TInput, TOutput> .
TransformBlock <TInput, TOutput> TransformManyBlock <TInput, TOutput> (0) .
ActionBlock TransformBlock <TInput, TOutput> .

Func <TInput, IEnumerable> Func <TInput, Task <IEnumerable >> . ActionBlock
TransformBlock <TInput, TOutput> TransformManyBlock <TInput, TOutput> .

.




```
var downloader = new TransformManyBlock<string, string>(async url =>
{
    Console.WriteLine("Downloading " + url);
    try
    {
        return ParseLinks(await DownloadContents(url));
    }
    catch{}

    return Enumerable.Empty<string>();
});
downloader.LinkTo(downloader);
```

Enumerable

```
var expanded = new TransformManyBlock<T[], T>(array => array);
```

1 0 1

```
public IPropagatorBlock<T> CreateFilteredBuffer<T>(Predicate<T> filter)
{
    return new TransformManyBlock<T, T>(item =>
        filter(item) ? new [] { item } : Enumerable.Empty<T>());
}
```

TPL Stephen Toub

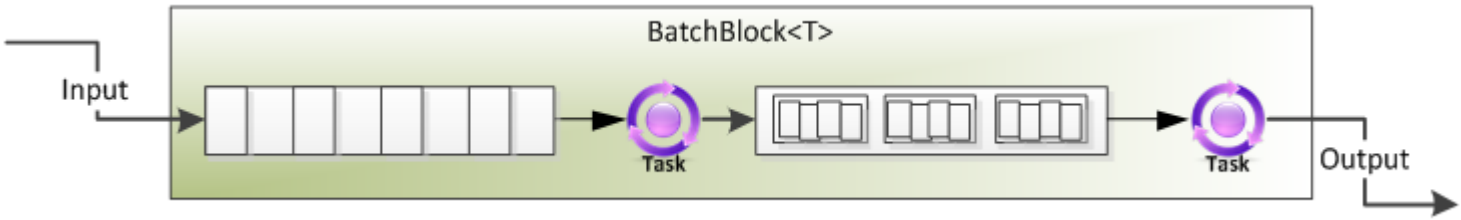
BatchBlock

( )

BatchBlock N . .

BatchBlock .

- , .
- ◦ . BatchBlock N 1 , 1 N , .



100

```
var batchRequests = new BatchBlock<Request>(batchSize:100);
var sendToDb = new ActionBlock<Request[]>(reqs => SubmitToDatabase(reqs));

batchRequests.LinkTo(sendToDb);
```

```
var batch = new BatchBlock<T>(batchSize:Int32.MaxValue);
new Timer(() => { batch.TriggerBatch(); }).Change(1000, 1000);
```

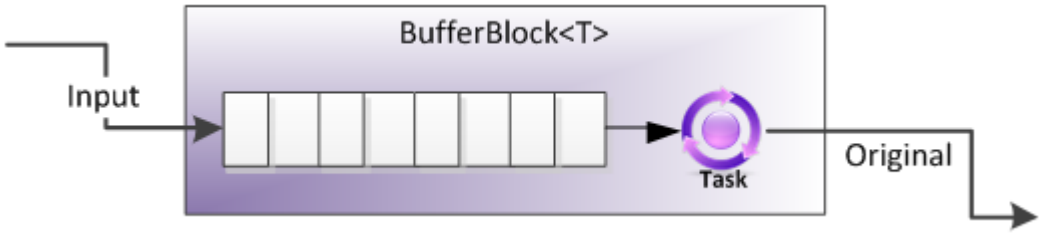
TPL Stephen Toub

BufferBlock

(FIFO :)

, BufferBlock T .

T "" . (FIFO) .
"" T (FIFO).



Throttled Producer /

```
// Hand-off through a bounded BufferBlock<T>
private static BufferBlock<int> _Buffer = new BufferBlock<int>(
    new DataflowBlockOptions { BoundedCapacity = 10 });

// Producer
private static async void Producer()
{
    while(true)
    {
        await _Buffer.SendAsync(Produce());
    }
}

// Consumer
private static async Task Consumer()
{
    while(true)
    {
        Process(await _Buffer.ReceiveAsync());
    }
}

// Start the Producer and Consumer
private static async Task Run()
{
    await Task.WhenAll(Producer(), Consumer());
}
```

TPL Stephen Toub

(TPL) : https://riptutorial.com/ko/csharp/topic/3110/----tpl----

137:

- (obj) {}

`lock` . (:). .

MSDN

lock , .

lock critical . .

.

C # 5.0 `lock` .

```
bool lockTaken = false;
try
{
    System.Threading.Monitor.Enter(refObject, ref lockTaken);
    // code
}
finally
{
    if (lockTaken)
        System.Threading.Monitor.Exit(refObject);
}
```

C # 4.0 `lock` .

```
System.Threading.Monitor.Enter(refObject);
try
{
    // code
}
finally
{
     System.Threading.Monitor.Exit(refObject);
}
```

Examples

`lock` .

`ReserveRoom` . `lock` . `lock` .

```
public class Hotel
{
    private readonly object _roomLock = new object();

    public void ReserveRoom(int roomNumber)
```

```
    {
        // lock keyword ensures that only one thread executes critical section at once
        // in this case, reserves a hotel room of given number
        // preventing double bookings
        lock (_roomLock)
        {
            // reserve room logic goes here
        }
    }
}
```

`lock` -ed , .

.

lock

. . try finally .

```
lock(locker)
{
    throw new Exception();
}
```

C # 5.0 .

`lock`

```
lock (x) ...
```

`x` .

```
bool __lockWasTaken = false;
try {
    System.Threading.Monitor.Enter(x, ref __lockWasTaken);
    ...
}
finally {
    if (__lockWasTaken) System.Threading.Monitor.Exit(x);
}
```

`x` .

.

```
lock(locker)
{
    return 5;
}
```

SO .

Object

C # inbuilt `lock` . `object` .

```
public class ThreadSafe {
  private static readonly object locker = new object();


  public void SomeThreadSafeMethod() {
    lock (locker) {
      // Only one thread can be here at a time.
    }
  }
}
```

NB . `Type` (`typeof(ThreadSafe)`) `Type` , (. `ThreadSafe` AppDomain `Type` .

/

`lock` . `lock` .

```
List<string> stringList = new List<string>();

public void AddToListNotThreadSafe(string something)
{
    // DO NOT do this, as each call to this method
    // will lock on a different instance of an Object.
    // This provides no thread safety, it only degrades performance.
    var localLock = new Object();
    lock(localLock)
    {
        stringList.Add(something);
    }
}

// Define object that can be used for thread safety in the AddToList method
readonly object classLock = new object();

public void AddToList(List<string> stringList, string something)
{
    // USE THE classLock instance field to achieve a
    // thread-safe lock before adding to stringList
    lock(classLock)
    {
        stringList.Add(something);
    }
}
```

.

`lock(obj)` `obj.ToString()` `obj.ToString()` .

```
object obj = new Object();

public void SomeMethod()
{
     lock(obj)
    {
       //do dangerous stuff
    }
 }

 //Meanwhile on other tread
 public void SomeOtherMethod()
 {
   var objInString = obj.ToString(); //this does not block
 }
```

.

```
public abstract class Base
{
    protected readonly object padlock;
    protected readonly List<string> list;

    public Base()
    {
        this.padlock = new object();
        this.list = new List<string>();
    }

    public abstract void Do();
}

public class Derived1 : Base
{
    public override void Do()
    {
        lock (this.padlock)
        {
            this.list.Add("Derived1");
        }
    }
}

public class Derived2 : Base
{
    public override void Do()
    {
        this.list.Add("Derived2"); // OOPS! I forgot to lock!
    }
}
```

.

```
public abstract class Base
{
    private readonly object padlock; // This is now private
    protected readonly List<string> list;
```

```
    public Base()
    {
        this.padlock = new object();
        this.list = new List<string>();
    }

    public void Do()
    {
        lock (this.padlock) {
            this.DoInternal();
        }
    }

    protected abstract void DoInternal();
}

public class Derived1 : Base
{
    protected override void DoInternal()
    {
        this.list.Add("Derived1"); // Yay! No need to lock
    }
}
```

ValueType .

`object` . boxed IncInSync .

```
public int Count { get; private set; }

private readonly int counterLock = 1;

public void Inc()
{
    IncInSync(counterLock);
}

private void IncInSync(object monitorResource)
{
    lock (monitorResource)
    {
        Count++;
    }
}
```

`Inc` .

```
BulemicCounter.Inc:
IL_0000:  nop
IL_0001:  ldarg.0
IL_0002:  ldarg.0
IL_0003:  ldfld       UserQuery+BulemicCounter.counterLock
IL_0008:  box         System.Int32**
IL_000D:  call        UserQuery+BulemicCounter.IncInSync
IL_0012:  nop
```

```
IL_0013:  ret
```

Boxed ValueType .

```
private readonly object counterLock = 1;
```

boxing .

```
IL_0001:  ldc.i4.1
IL_0002:  box         System.Int32
IL_0007:  stfld       UserQuery+BulemicCounter.counterLock
```

List Dictionary .

```
public class Cache
{
    private readonly object padlock;
    private readonly Dictionary<string, object> values;

    public WordStats()
    {
        this.padlock = new object();
        this.values = new Dictionary<string, object>();
    }

    public void Add(string key, object value)
    {
        lock (this.padlock)
        {
            this.values.Add(key, value);
        }
    }

    /* rest of class omitted */
}
```

values . .

ConcurrentDictionary .

```
public class Cache
{
    private readonly ConcurrentDictionary<string, object> values;

    public WordStats()
    {
        this.values = new ConcurrentDictionary<string, object>();
    }

    public void Add(string key, object value)
    {
        this.values.Add(key, value);
    }
```

```
    /* rest of class omitted */
}
```

.

: https://riptutorial.com/ko/csharp/topic/1495/-

138:

. **StackOverflow** . " " `for` , `while` `foreach` , .

- .

  - .
    - ,
  - . ( ) .
    - . *n* !

:

- https://www.cs.umd.edu/class/fall2002/cmsc214/Tutorial/recursion2.html
- https://en.wikipedia.org/wiki/Recursion#In_computer_science

Examples

.

. . .

```
/// <summary>
/// Create an object structure the code can recursively describe
/// </summary>
public class Root
{
    public string Name { get; set; }
    public ChildOne Child { get; set; }
}
public class ChildOne
{
    public string ChildOneName { get; set; }
    public ChildTwo Child { get; set; }
}
public class ChildTwo
{
    public string ChildTwoName { get; set; }
}
/// <summary>
/// The console application with the recursive function DescribeTypeOfObject
/// </summary>
public class Program
{
    static void Main(string[] args)
    {
        // point A, we call the function with type 'Root'
        DescribeTypeOfObject(typeof(Root));
        Console.WriteLine("Press a key to exit");
        Console.ReadKey();
    }

    static void DescribeTypeOfObject(Type type)
```

```
    {
        // get all properties of this type
        Console.WriteLine($"Describing type {type.Name}");
        PropertyInfo[] propertyInfos = type.GetProperties();
        foreach (PropertyInfo pi in propertyInfos)
        {
            Console.WriteLine($"Has property {pi.Name} of type {pi.PropertyType.Name}");
            // is a custom class type? describe it too
            if (pi.PropertyType.IsClass && !pi.PropertyType.FullName.StartsWith("System."))
            {
                // point B, we call the function type this property
                DescribeTypeOfObject(pi.PropertyType);
            }
        }
        // done with all properties
        // we return to the point where we were called
        // point A for the first call
        // point B for all properties of type custom class
    }
}
```

.

.

.

```
public int Factorial(int number)
{
    return number == 0 ? 1 : n * Factorial(number - 1);
}
```

`number` `number` .

:

, `Factorial(4)`

1. `number (4) == 1` ?
2. ? return `4 * Factorial(number-1)` (3)
3. `Factorial(3)` .
4. `Factorial(1)` `number (1) == 1` 1 .
5. `4 * 3 * 2 * 1` "" 24 .

. .

. .

```
internal class Program
{
    internal const int RootLevel = 0;
    internal const char Tab = '\t';

    internal static void Main()
    {
```

```
        Console.WriteLine("Enter the path of the root directory:");
        var rootDirectorypath = Console.ReadLine();

        Console.WriteLine(
            $"Getting directory tree of '{rootDirectorypath}'");

        PrintDirectoryTree(rootDirectorypath);
        Console.WriteLine("Press 'Enter' to quit...");
        Console.ReadLine();
    }

    internal static void PrintDirectoryTree(string rootDirectoryPath)
    {
        try
        {
            if (!Directory.Exists(rootDirectoryPath))
            {
                throw new DirectoryNotFoundException(
                    $"Directory '{rootDirectoryPath}' not found.");
            }

            var rootDirectory = new DirectoryInfo(rootDirectoryPath);
            PrintDirectoryTree(rootDirectory, RootLevel);
        }
        catch (DirectoryNotFoundException e)
        {
            Console.WriteLine(e.Message);
        }
    }

    private static void PrintDirectoryTree(
        DirectoryInfo directory, int currentLevel)
    {
        var indentation = string.Empty;
        for (var i = RootLevel; i < currentLevel; i++)
        {
            indentation += Tab;
        }

        Console.WriteLine($"{indentation}-{directory.Name}");
        var nextLevel = currentLevel + 1;
        try
        {
            foreach (var subDirectory in directory.GetDirectories())
            {
                PrintDirectoryTree(subDirectory, nextLevel);
            }
        }
        catch (UnauthorizedAccessException e)
        {
            Console.WriteLine($"{indentation}-{e.Message}");
        }
    }
}
```

. .

`Main` :

main . `PrintDirectoryTree` .

PrintDirectoryTree(string) :

. . DirectoryNotFoundException . DirectoryNotFoundException catch . DirectoryInfo rootDirectory PrintDirectoryTree rootDirectory RootLevel 0) .

PrintDirectoryTree(DirectoryInfo, int) :

. DirectoryInfo .DirectoryInfo , . , .

```
-Root
    -Child 1
    -Child 2
        -Grandchild 2.1
    -Child 3
```

, , 1 . : . . .

Throw UnauthorizedAccessException . .

.

```
internal static void PrintDirectoryTree(string directoryName)
{
    try
    {
        if (!Directory.Exists(directoryName)) return;
        Console.WriteLine(directoryName);
        foreach (var d in Directory.GetDirectories(directoryName))
        {
            PrintDirectoryTree(d);
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
}
```

.DirectoryInfo .

.

i> 0 F (n) = F (n-2) + F (n-1) ,

```
// Returns the i'th Fibonacci number
public int fib(int i) {
    if(i <= 2) {
        // Base case of the recursive function.
        // i is either 1 or 2, whose associated Fibonacci sequence numbers are 1 and 1.
        return 1;
    }
    // Recursive case. Return the sum of the two previous Fibonacci numbers.
    // This works because the definition of the Fibonacci sequence specifies
    // that the sum of two adjacent elements equals the next element.
    return  fib(i - 2) + fib(i - 1);
```

```
}

fib(10); // Returns 55
```

(:! !) 1 . , 9! = 9 x 8! = 9 x 8 x 7! = 9 × 8 × 7 × 6 × 5 × 4 × 3 × 2 × 1.

:

```
long Factorial(long x)
{
    if (x < 1)
    {
        throw new OutOfRangeException("Factorial can only be used with positive numbers.");
    }

    if (x == 1)
    {
        return 1;
    } else {
        return x * Factorial(x - 1);
    }
}
```

PowerOf

. n e, e.

:

- 2 ² = 2x2
- 2³ = 2x2x2 2³ = 2² x 2
  ( ). .
- ◦ 0 0 0 . 0 = 0 x 0 x 0
  ◦ 0 1 .

:

```
public int CalcPowerOf(int b, int e) {
    if (b == 0) { return 0; } // when base is 0, it doesn't matter, it will always return 0
    if (e == 0) { return 1; } // math rule, exponent 0 always returns 1
    return b * CalcPowerOf(b, e - 1); // actual recursive logic, where we split the problem,
aka: 2³ = 2 * 2² etc..
}
```

xUnit :
. xUnit .

```
    [Theory]
    [MemberData(nameof(PowerOfTestData))]
    public void PowerOfTest(int @base, int exponent, int expected) {
        Assert.Equal(expected, CalcPowerOf(@base, exponent));
    }
```

```
    public static IEnumerable<object[]> PowerOfTestData() {
        yield return new object[] { 0, 0, 0 };
        yield return new object[] { 0, 1, 0 };
        yield return new object[] { 2, 0, 1 };
        yield return new object[] { 2, 1, 2 };
        yield return new object[] { 2, 2, 4 };
        yield return new object[] { 5, 2, 25 };
        yield return new object[] { 5, 3, 125 };
        yield return new object[] { 5, 4, 625 };
}
```

: https://riptutorial.com/ko/csharp/topic/2470/

139:

- #define *[symbol]* // .
- #undef *[symbol]* // .
- #warning *[ ]* // . #if .
- #error *[ ]* // . #if .
- #line *[ ] ( )* // ( ). T4 .
- #pragma warning [disable | restore] *[ ]* // / .
- #pragma checksum " *[filename]* " " *[guid]* " " *[checksum]* "// .
- #region *[region name]* // .
- #endregion // .
- #if *[condition]* // .
- #else / #if .
- #elif *[condition]* // #if .
- #endif // #if .

. . , , . . .

 . . .

(`#if` , `#elif`) . :

- `==` `!=` . true () false ( ) .
- `&&` , `||` , `!`
- `()`

:

```
#if !DEBUG && (SOME_SYMBOL || SOME_OTHER_SYMBOL) && RELEASE == true
Console.WriteLine("OK!");
#endif
```

"OK" . `DEBUG` `SOME_SYMBOL` `SOME_OTHER_SYMBOL` `RELEASE` .

: . `#if` .

: MSDN C #

Examples

.

```
// Compile with /d:A or /d:B to see the difference
string SomeFunction()
{
#if A
    return "A";
#elif B
```

```
    return "B";
#else
    return "C";
#endif
}
```

.

```
void SomeFunc()
{
    try
    {
        SomeRiskyMethod();
    }
    catch (ArgumentException ex)
    {
        #if DEBUG
        log.Error("SomeFunc", ex);
        #endif

        HandleException(ex);
    }
}
```

```
#warning      #error  #error   .
```

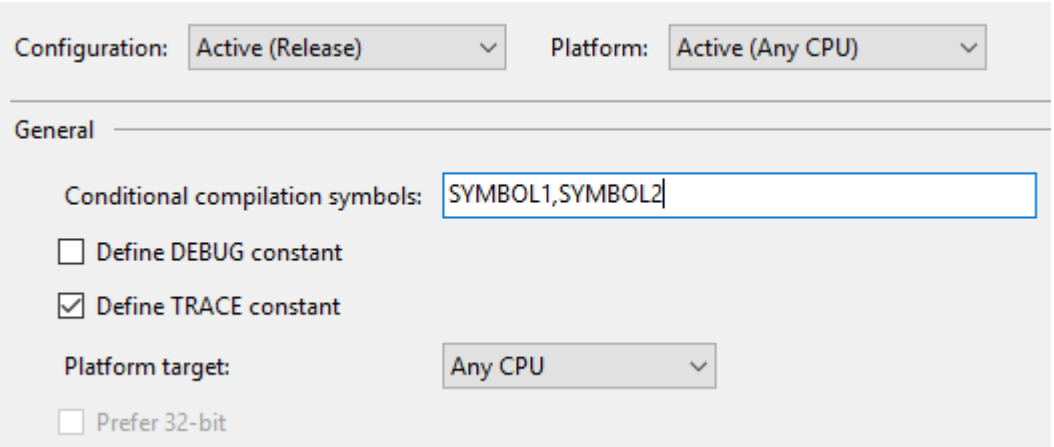
```
#if SOME_SYMBOL
#error This is a compiler Error.
#elif SOME_OTHER_SYMBOL
#warning This is a compiler Warning.
#endif
```

.

. .

```
#define MYSYMBOL
```

Visual Studio > > .



(*DEBUG TRACE* .)

C # `csc.exe /define:[name]` .

`#undefine` .

Visual Studio `DEBUG` ( ).

```
public void DoBusinessLogic()
{
    try
    {
        AuthenticateUser();
        LoadAccount();
        ProcessAccount();
        FinalizeTransaction();
    }
    catch (Exception ex)
    {
#if DEBUG
        System.Diagnostics.Trace.WriteLine("Unhandled exception!");
        System.Diagnostics.Trace.WriteLine(ex);
        throw;
#else
        LoggingFramework.LogError(ex);
        DisplayFriendlyErrorMessage();
#endif
    }
}
```

(, `DEBUG`), , throw. (`DEBUG`) .

`#region #endregion` .

```
#region Event Handlers

public void Button_Click(object s, EventArgs e)
{
    // ...
}

public void DropDown_SelectedIndexChanged(object s, EventArgs e)
{
    // ...
}

#endregion
```

IDE (: Visual Studio) .

`#line` .

```
void Test()
{
    #line 42 "Answer"
    #line filename "SomeFile.cs"
    int life; // compiler warning CS0168 in "SomeFile.cs" at Line 42
```

```
    #line default
    // compiler warnings reset to default
}
```

Pragma Checksum

`#pragma checksum` (PDB) .

```
#pragma checksum "MyCode.cs" "{00000000-0000-0000-0000-000000000000}" "{0123456789A}"
```

`System.Diagnostics` `Conditional` .

```
#define EXAMPLE_A

using System.Diagnostics;
class Program
{
    static void Main()
    {
        ExampleA(); // This method will be called
        ExampleB(); // This method will not be called
    }

    [Conditional("EXAMPLE_A")]
    static void ExampleA() {...}

    [Conditional("EXAMPLE_B")]
    static void ExampleB() {...}
}
```

`#pragma warning disable` `#pragma warning disable` `#pragma warning restore` .

```
#pragma warning disable CS0168

// Will not generate the "unused variable" compiler warning since it was disabled
var x = 5;

#pragma warning restore CS0168

// Will generate a compiler warning since the warning was just restored
var y = 8;
```

.

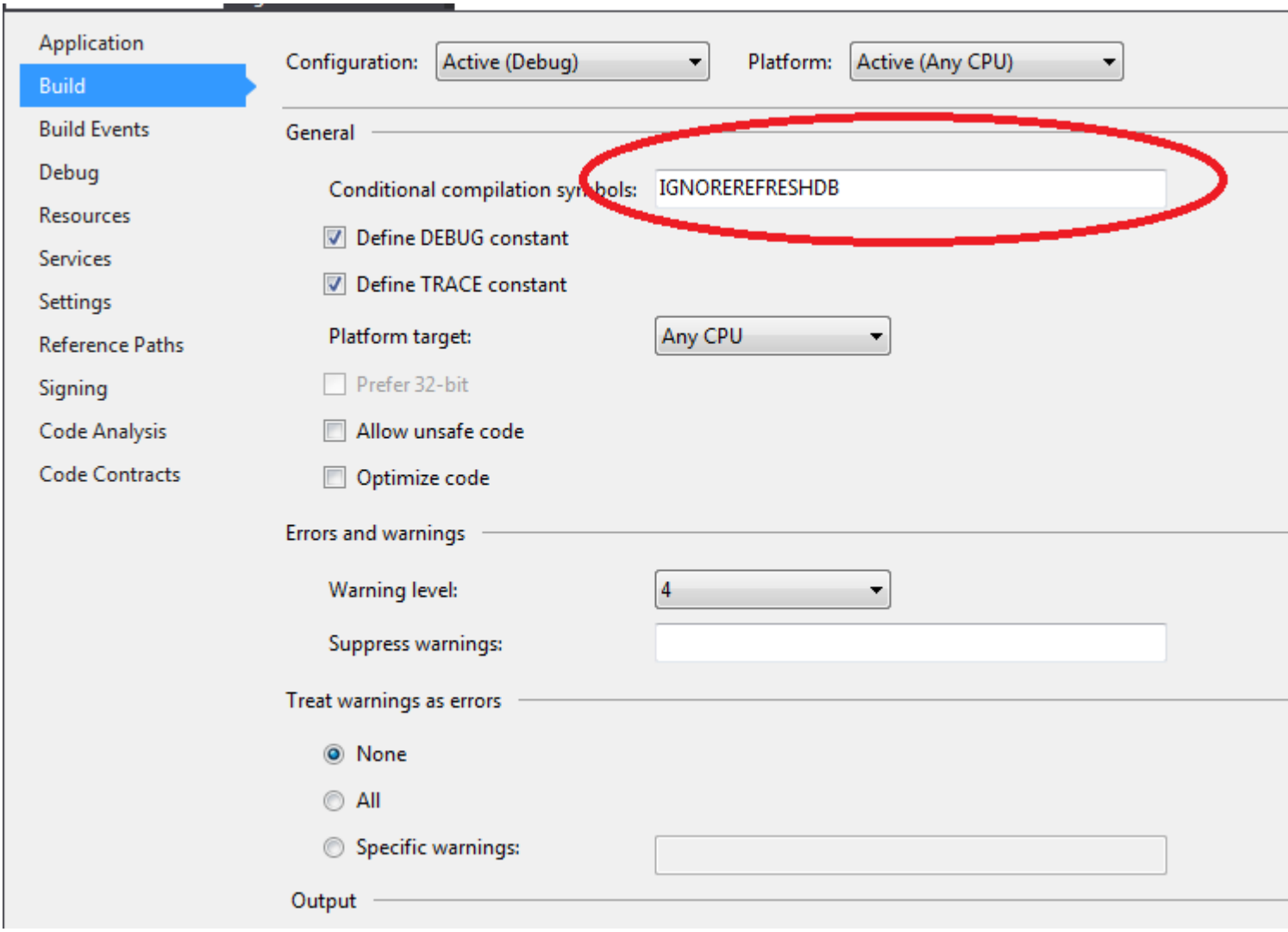
```
#pragma warning disable CS0168, CS0219
```

`CS` ( ).

```
#pragma warning disable 0168, 0219, CS0414
```

.

Solution Explorer -> □□□ □□□ □□□ -> Properties -> Build -> Conditional compilation symbols .



:

```
public void Init()
{
    #if !IGNOREREFRESHDB
    // will skip code here
     db.Initialize();
    #endif
}
```

: https://riptutorial.com/ko/csharp/topic/755/-

140:

- `new Regex(pattern);` // .
- `Regex.Match(input);` // *Match* .
- `Regex.Matches(input);` // *MatchCollection* .

	`string` . msdn .
RegexOptions *[ ]*	`Singleline` `Multiline` . `Multiline-Mode` `SingleLine-Mode` `Multiline-Mode` `NewLine` (\ n) (.) . : msdn
[ ]	. .

```
using System.Text.RegularExpressions;
```

- :
- Regex101 : .

. *`XmlDocument`* *xml* .

Regex . *20* *`right way`* .

Examples

`using System.Text.RegularExpressions;`

```
string pattern = ":(.*?):";
string lookup = "--:text in here:--";

// Instanciate your regex object and pass a pattern to it
Regex rgxLookup = new Regex(pattern, RegexOptions.Singleline, TimeSpan.FromSeconds(1));
// Get the match from your regex-object
Match mLookup = rgxLookup.Match(lookup);

// The group-index 0 always covers the full pattern.
// Matches inside parentheses will be accessed through the index 1 and above.
string found = mLookup.Groups[1].Value;
```

:

```
found = "text in here"
```

`using System.Text.RegularExpressions;`

```
List<string> found = new List<string>();
string pattern = ":(.*?):";
```

```
string lookup = "--:text in here:--:another one:-:third one:---!123:fourth:";

// Instanciate your regex object and pass a pattern to it
Regex rgxLookup = new Regex(pattern, RegexOptions.Singleline, TimeSpan.FromSeconds(1));
MatchCollection mLookup = rgxLookup.Matches(lookup);

foreach(Match match in mLookup)
{
    found.Add(match.Groups[1].Value);
}
```

:

```
found = new List<string>() { "text in here", "another one", "third one", "fourth" }
```

: https://riptutorial.com/ko/csharp/topic/3774/-

141:

Examples

static .

1. .
2. .

```
public class Foo
{
    public Foo{
        Counter++;
        NonStaticCounter++;
    }

    public static int Counter { get; set; }
    public int NonStaticCounter { get; set; }
}

public class Program
{
    static void Main(string[] args)
    {
        //Create an instance
        var foo1 = new Foo();
        Console.WriteLine(foo1.NonStaticCounter); //this will print "1"

        //Notice this next call doesn't access the instance but calls by the class name.
        Console.WriteLine(Foo.Counter); //this will also print "1"

        //Create a second instance
        var foo2 = new Foo();

        Console.WriteLine(foo2.NonStaticCounter); //this will print "1"

        Console.WriteLine(Foo.Counter); //this will now print "2"
        //The static property incremented on both instances and can persist for the whole
class

    }
}
```

"" .

1. ( ).
2. .
3. `static sealed` .

```
public static class Foo
{
    //Notice there is no constructor as this cannot be an instance
```

```
    public static int Counter { get; set; }
    public static int GetCount()
    {
        return Counter;
    }
}

public class Program
{
    static void Main(string[] args)
    {
        Foo.Counter++;
        Console.WriteLine(Foo.GetCount()); //this will print 1

        //var foo1 = new Foo();
        //this line would break the code as the Foo class does not have a constructor
    }
}
```

static .

```
void Main()
{
    Console.WriteLine("Static classes are lazily initialized");
    Console.WriteLine("The static constructor is only invoked when the class is first
accessed");
    Foo.SayHi();

    Console.WriteLine("Reflecting on a type won't trigger its static .ctor");
    var barType = typeof(Bar);

    Console.WriteLine("However, you can manually trigger it with
System.Runtime.CompilerServices.RuntimeHelpers");
    RuntimeHelpers.RunClassConstructor(barType.TypeHandle);
}

// Define other methods and classes here
public static class Foo
{
    static Foo()
    {
        Console.WriteLine("static Foo.ctor");
    }
    public static void SayHi()
    {
        Console.WriteLine("Foo: Hi");
    }
}
public static class Bar
{
    static Bar()
    {
        Console.WriteLine("static Bar.ctor");
    }
}
```

: https://riptutorial.com/ko/csharp/topic/1653/-

142:

- `public void SomeMethod <T> () { }`
- `public void SomeMethod<T, V>() { }`
- `public T SomeMethod<T>(IEnumerable<T> sequence) { ... }`
- `public void SomeMethod<T>() where T : new() { }`
- `public void SomeMethod<T, V>() where T : new() where V : struct { }`
- `public void SomeMethod<T>() where T: IDisposable { }`
- `public void SomeMethod<T>() where T: Foo { }`
- `public class MyClass<T> { public T Data {get; set; } }`

T, V

C # . C # CIL .

C # . . generic generic .

. . `Dictionary<TKey, Tvalue> Dictionary`2` .

Examples

()

:

```
class MyGenericClass<T1, T2, T3, ...>
{
    // Do something with the type parameters.
}
```

:

```
var x = new MyGenericClass<int, char, bool>();
```

( ) :

```
void AnotherMethod(MyGenericClass<float, byte, char> arg) { ... }
```

()

:

```
void MyGenericMethod<T1, T2, T3>(T1 a, T2 b, T3 c)
{
    // Do something with the type parameters.
}
```

:

genric .

```
int x =10;
int y =20;
string z = "test";
MyGenericMethod(x,y,z);
```

arguemnts .

```
MyGenericMethod<int, int, string>(x,y,z);
```

()

:

```
interface IMyGenericInterface<T1, T2, T3, ...> { ... }
```

() :

```
class ClassA<T1, T2, T3> : IMyGenericInterface<T1, T2, T3> { ... }

class ClassB<T1, T2> : IMyGenericInterface<T1, T2, int> { ... }

class ClassC<T1> : IMyGenericInterface<T1, char, int> { ... }

class ClassD : IMyGenericInterface<bool, char, int> { ... }
```

( ) :

```
void SomeMethod(IMyGenericInterface<int, char, bool> arg) { ... }
```

()

, . .

. . () - .

```
void M<T>(T obj)
{
}
```

.

```
M<object>(new object());
M(new object());
```

.

```
M<string>("");
M("");
```

.

```
M<object>("");
M((object) "");
M("" as object);
```

.

. X . . .

```
void X<T1, T2>(T1 obj)
{
}
```

.

```
X("");
```

(object) .

```
X<object>("");
```

.

```
X<string, object>("");
```

( )

.

```
interface IType;
interface IAnotherType;

// T must be a subtype of IType
interface IGeneric<T>
    where T : IType
{
}

// T must be a subtype of IType
class Generic<T>
    where T : IType
{
}

class NonGeneric
{
    // T must be a subtype of IType
```

```
    public void DoSomething<T>(T arg)
        where T : IType
    {
    }
}

// Valid definitions and expressions:
class Type : IType { }
class Sub : IGeneric<Type> { }
class Sub : Generic<Type> { }
new NonGeneric().DoSomething(new Type());

// Invalid definitions and expressions:
class AnotherType : IAnotherType { }
class Sub : IGeneric<AnotherType> { }
class Sub : Generic<AnotherType> { }
new NonGeneric().DoSomething(new AnotherType());
```

:

```
class Generic<T, T1>
    where T : IType
    where T1 : Base, new()
{
}
```

. .

```
class A { /* ... */ }
class B { /* ... */ }

interface I1 { }
interface I2 { }

class Generic<T>
    where T : A, I1, I2
{
}

class Generic2<T>
    where T : A, B //Compilation error
{
}
```

.

```
class Generic<T>
    where T : A, I1
{
}

class Generic2<T>
    where T : I1, A //Compilation error
{
}
```

. .

( )

`class` `struct` . (: `new()`) .

```
// TRef must be a reference type, the use of Int32, Single, etc. is invalid.
// Interfaces are valid, as they are reference types
class AcceptsRefType<TRef>
    where TRef : class
{
    // TStruct must be a value type.
    public void AcceptStruct<TStruct>()
        where TStruct : struct
    {
    }

    // If multiple constraints are used along with class/struct
    // then the class or struct constraint MUST be specified first
    public void Foo<TComparableClass>()
        where TComparableClass : class, IComparable
    {
    }
}
```

(new-keyword)

`new()` () .

```
class Foo
{
    public Foo () { }
}

class Bar
{
    public Bar (string s) { ... }
}

class Factory<T>
    where T : new()
{
    public T Create()
    {
        return new T();
    }
}

Foo f = new Factory<Foo>().Create(); // Valid.
Bar b = new Factory<Bar>().Create(); // Invalid, Bar does not define a default/empty
constructor.
```

`Create()` .

'Bar' 'Factory' 'T' public .

.

()

.

```
class Tuple<T1,T2>
{
   public Tuple(T1 value1, T2 value2)
   {
   }
}

var x = new Tuple(2, "two");              // This WON'T work...
var y = new Tuple<int, string>(2, "two"); // even though the explicit form will.
```

.

'Tuple <T1, T2>' 2 .

.

```
static class Tuple
{
    public static Tuple<T1, T2> Create<T1, T2>(T1 value1, T2 value2)
    {
         return new Tuple<T1, T2>(value1, value2);
    }
}

var x = Tuple.Create(2, "two");  // This WILL work...
```

typeof .

```
class NameGetter<T>
{
    public string GetTypeName()
    {
        return typeof(T).Name;
    }
}
```

. .

.

```
public void SomeMethod<T, V>()
{
   // No code for simplicity
}

SomeMethod(); // doesn't compile
SomeMethod<int, bool>(); // compiles
```

.

```
public K SomeMethod<K, V>(V input)
{
    return default(K);
}

int num1 = SomeMethod(3); // doesn't compile
int num2 = SomeMethod<int>("3"); // doesn't compile
int num3 = SomeMethod<int, string>("3"); // compiles.
```

.

Animal Eat generic TFood .

```
public interface IFood
{
    void EatenBy(Animal animal);
}

public class Grass: IFood
{
    public void EatenBy(Animal animal)
    {
        Console.WriteLine("Grass was eaten by: {0}", animal.Name);
    }
}

public class Animal
{
    public string Name { get; set; }

    public void Eat<TFood>(TFood food)
        where TFood : IFood
    {
        food.EatenBy(this);
    }
}

public class Carnivore : Animal
{
    public Carnivore()
    {
        Name = "Carnivore";
    }
}

public class Herbivore : Animal, IFood
{
    public Herbivore()
    {
        Name = "Herbivore";
    }

    public void EatenBy(Animal animal)
    {
        Console.WriteLine("Herbivore was eaten by: {0}", animal.Name);
    }
}
```

Eat .

```
var grass = new Grass();
var sheep = new Herbivore();
var lion = new Carnivore();

sheep.Eat(grass);
//Output: Grass was eaten by: Herbivore

lion.Eat(sheep);
//Output: Herbivore was eaten by: Carnivore
```

:

```
sheep.Eat(lion);
```

IFood . : " 'Carnivore' 'Animal.Eat (TFood)' 'TFood' ' Carnivore "IFood '. "

`IEnumerable<T>` `IEnumerable<T1>` ? `T` `T1` . `IEnumerable` `T` *(covariant)* `IEnumerable` `T` .

```
class Animal { /* ... */ }
class Dog : Animal { /* ... */ }

IEnumerable<Dog> dogs = Enumerable.Empty<Dog>();
IEnumerable<Animal> animals = dogs;  // IEnumerable<Dog> is a subtype of IEnumerable<Animal>
// dogs = animals;  // Compilation error - IEnumerable<Animal> is not a subtype of
IEnumerable<Dog>
```

(covariant) .

`IEnumerable` `T` . `Dog` `Animal` .

k `out` g . g ..

```
interface IEnumerable<out T> { /* ... */ }
```

(covariant) .

```
interface Bad<out T>
{
    void SetT(T t);  // type error
}
```

.

```
using NUnit.Framework;

namespace ToyStore
{
   enum Taste { Bitter, Sweet };

   interface IWidget
   {
```

```
      int Weight { get; }
   }

   interface IFactory<out TWidget>
       where TWidget : IWidget
   {
      TWidget Create();
   }

   class Toy : IWidget
   {
      public int Weight { get; set; }
      public Taste Taste { get; set; }
   }

   class ToyFactory : IFactory<Toy>
   {
      public const int StandardWeight = 100;
      public const Taste StandardTaste = Taste.Sweet;

      public Toy Create() { return new Toy { Weight = StandardWeight, Taste = StandardTaste };
}
   }

   [TestFixture]
   public class GivenAToyFactory
   {
      [Test]
      public static void WhenUsingToyFactoryToMakeWidgets()
      {
         var toyFactory = new ToyFactory();

         //// Without out keyword, note the verbose explicit cast:
         // IFactory<IWidget> rustBeltFactory = (IFactory<IWidget>)toyFactory;

         // covariance: concrete being assigned to abstract (shiny and new)
         IFactory<IWidget> widgetFactory = toyFactory;
         IWidget anotherToy = widgetFactory.Create();
         Assert.That(anotherToy.Weight, Is.EqualTo(ToyFactory.StandardWeight)); // abstract
contract
         Assert.That(((Toy)anotherToy).Taste, Is.EqualTo(ToyFactory.StandardTaste)); //
concrete contract
      }
   }
}
```

`IComparer<T>` `IComparer<T1>` ? `T1` `T` . `IComparer` `T` `IComparer` `T` .

```
class Animal { /* ... */ }
class Dog : Animal { /* ... */ }

IComparer<Animal> animalComparer = /* ... */;
IComparer<Dog> dogComparer = animalComparer;  // IComparer<Animal> is a subtype of
IComparer<Dog>
// animalComparer = dogComparer;  // Compilation error - IComparer<Dog> is not a subtype of
IComparer<Animal>
```

.

IComparer T . Animal Dog .

contravariant in . .

```
interface IComparer<in T> { /* ... */ }
```

.

```
interface Bad<in T>
{
    T GetT();  // type error
}
```

IList<T> IList<T1> . IList type .

```
class Animal { /* ... */ }
class Dog : Animal { /* ... */ }

IList<Dog> dogs = new List<Dog>();
IList<Animal> animals = dogs;  // type error
```

.

IList (covariant) .

```
IList<Animal> animals = new List<Dog>();  // supposing this were allowed...
animals.Add(new Giraffe());  // ... then this would also be allowed, which is bad!
```

IList .

```
IList<Dog> dogs = new List<Animal> { new Dog(), new Giraffe() };  // if this were allowed...
Dog dog = dogs[1];  // ... then this would be allowed, which is bad!
```

in out .

```
interface IList<T> { /* ... */ }
```

.

```
interface IEnumerable<out T>
{
    // ...
}
interface IComparer<in T>
{
    // ...
}
```

```
class BadClass<in T1, out T2>  // not allowed
{
}
```

```
struct BadStruct<in T1, out T2>  // not allowed
{
}
```

.

```
class MyClass
{
    public T Bad<out T, in T1>(T1 t1)  // not allowed
    {
        // ...
    }
}
```

.

```
interface IFoo<in T1, out T2, T3>
//  T1 : Contravariant type
//  T2 : Covariant type
//  T3 : Invariant type
{
    // ...
}

IFoo<Animal, Dog, int> foo1 = /* ... */;
IFoo<Dog, Animal, int> foo2 = foo1;
// IFoo<Animal, Dog, int> is a subtype of IFoo<Dog, Animal, int>
```

.

```
delegate void Action<in T>(T t);    // T is an input
delegate T Func<out T>();           // T is an output
delegate T2 Func<in T1, out T2>();  // T1 is an input, T2 is an output
```

( ) D B Liskov Substitution Principle .

- D B .
- D B .

.

```
Func<object, string> original = SomeMethod;
Func<object, object> d1 = original;
Func<string, string> d2 = original;
Func<string, object> d3 = original;
```

. T T .

```
interface IReturnCovariant<out T>
{
    IEnumerable<T> GetTs();
}
```

contravariant . T T .

```
interface IReturnContravariant<in T>
{
    IComparer<T> GetTComparer();
}
```

. T T .

```
interface IAcceptCovariant<in T>
{
    void ProcessTs(IEnumerable<T> ts);
}
```

contravariant . T T .

```
interface IAcceptContravariant<out T>
{
    void CompareTs(IComparer<T> tComparer);
}
```

.

EqualityComparer<TType>.Default .

```
public void Foo<TBar>(TBar arg1, TBar arg2)
{
    var comparer = EqualityComparer<TBar>.Default;
    if (comparer.Equals(arg1,arg2)
    {
        ...
    }
}
```

TBar IEquatale<TBar> IEquatable<TBar>.Equals(TBar other) Object.Equals() . boxing / unboxing .

```
    /// <summary>
    /// Converts a data type to another data type.
    /// </summary>
    public static class Cast
    {
        /// <summary>
        /// Converts input to Type of default value or given as typeparam T
        /// </summary>
        /// <typeparam name="T">typeparam is the type in which value will be returned, it
could be any type eg. int, string, bool, decimal etc.</typeparam>
        /// <param name="input">Input that need to be converted to specified type</param>
        /// <param name="defaultValue">defaultValue will be returned in case of value is null
or any exception occures</param>
        /// <returns>Input is converted in Type of default value or given as typeparam T and
returned</returns>
        public static T To<T>(object input, T defaultValue)
        {
```

```
            var result = defaultValue;
            try
            {
                if (input == null || input == DBNull.Value) return result;
                if (typeof (T).IsEnum)
                {
                    result = (T) Enum.ToObject(typeof (T), To(input,
Convert.ToInt32(defaultValue)));
                }
                else
                {
                    result = (T) Convert.ChangeType(input, typeof (T));
                }
            }
            catch (Exception ex)
            {
                Tracer.Current.LogException(ex);
            }

            return result;
        }

        /// <summary>
        /// Converts input to Type of typeparam T
        /// </summary>
        /// <typeparam name="T">typeparam is the type in which value will be returned, it
could be any type eg. int, string, bool, decimal etc.</typeparam>
        /// <param name="input">Input that need to be converted to specified type</param>
        /// <returns>Input is converted in Type of default value or given as typeparam T and
returned</returns>
        public static T To<T>(object input)
        {
            return To(input, default(T));
        }



    }
```

:

```
std.Name = Cast.To<string>(drConnection["Name"]);
std.Age = Cast.To<int>(drConnection["Age"]);
std.IsPassed = Cast.To<bool>(drConnection["IsPassed"]);


// Casting type using default value
//Following both ways are correct
// Way 1 (In following style input is converted into type of default value)
std.Name = Cast.To(drConnection["Name"], "");
std.Marks = Cast.To(drConnection["Marks"], 0);
// Way 2
std.Name = Cast.To<string>(drConnection["Name"], "");
std.Marks = Cast.To<int>(drConnection["Marks"], 0);
```

```
    /// <summary>
    /// Read configuration values from app.config and convert to specified types
    /// </summary>
    public static class ConfigurationReader
```

```
    {
        /// <summary>
        /// Get value from AppSettings by key, convert to Type of default value or typeparam T
and return
        /// </summary>
        /// <typeparam name="T">typeparam is the type in which value will be returned, it
could be any type eg. int, string, bool, decimal etc.</typeparam>
        /// <param name="strKey">key to find value from AppSettings</param>
        /// <param name="defaultValue">defaultValue will be returned in case of value is null
or any exception occures</param>
        /// <returns>AppSettings value against key is returned in Type of default value or
given as typeparam T</returns>
        public static T GetConfigKeyValue<T>(string strKey, T defaultValue)
        {
            var result = defaultValue;
            try
            {
                if (ConfigurationManager.AppSettings[strKey] != null)
                    result = (T)Convert.ChangeType(ConfigurationManager.AppSettings[strKey],
typeof(T));
            }
            catch (Exception ex)
            {
                Tracer.Current.LogException(ex);
            }

            return result;
        }
        /// <summary>
        /// Get value from AppSettings by key, convert to Type of default value or typeparam T
and return
        /// </summary>
        /// <typeparam name="T">typeparam is the type in which value will be returned, it
could be any type eg. int, string, bool, decimal etc.</typeparam>
        /// <param name="strKey">key to find value from AppSettings</param>
        /// <returns>AppSettings value against key is returned in Type given as typeparam
T</returns>
        public static T GetConfigKeyValue<T>(string strKey)
        {
            return GetConfigKeyValue(strKey, default(T));
        }

    }
```

:

```
var timeOut = ConfigurationReader.GetConfigKeyValue("RequestTimeout", 2000);
var url = ConfigurationReader.GetConfigKeyValue("URL", "www.someurl.com");
var enabled = ConfigurationReader.GetConfigKeyValue("IsEnabled", false);
```

: https://riptutorial.com/ko/csharp/topic/27/

143:

Examples

If-Else Statement

`decision branch` . C # ( ) `If-Else` .

100 int ( ) .

```
static void PrintPassOrFail(int score)
{
    if (score >= 50) // If score is greater or equal to 50
    {
        Console.WriteLine("Pass!");
    }
    else // If score is not greater or equal to 50
    {
        Console.WriteLine("Fail!");
    }
}
```

`If score >= 50` (`score >= 50`) . `boolean` . `true` `if { }` .

`PrintPassOrFail(60);` `PrintPassOrFail(60);` , ***Pass!*** Console Print . 60 50 .

`PrintPassOrFail(30);` , ***Fail!*** . 30 50 `If else { }` .

100 . 100 0 **If-Else If-Else Statement** .

If-Else If-Else Statement

If-Else Statement `Else If` . `Else If` **If-Else If-Else** `If` `If` . **If-Else** .

If-Else Statement , 100 . . **If-Else Statement** .

```
static void PrintPassOrFail(int score)
{
    if (score > 100) // If score is greater than 100
    {
        Console.WriteLine("Error: score is greater than 100!");
    }
    else if (score < 0) // Else If score is less than 0
    {
        Console.WriteLine("Error: score is less than 0!");
    }
    else if (score >= 50) // Else if score is greater or equal to 50
    {
        Console.WriteLine("Pass!");
    }
    else // If none above, then score must be between 0 and 49
    {
```

```
        Console.WriteLine("Fail!");
    }
}
```

. .

, `PrintPassOFail(110);` `PrintPassOFail(110);` , ***Error : score 100 !*** ; `PrintPassOrFail(-20);` `PrintPassOrFail(-20);` `PrintPassOrFail(-20);` ***Error : score 0 .*** .

switch . / , .

`switch` `if...else if... else..` .

.

```
switch(expression) {
   case constant-expression:
      statement(s);
      break;
   case constant-expression:
      statement(s);
      break;

   // you can have any number of case statements
   default : // Optional
      statement(s);
      break;
}
```

switch sevaral .

- switch .
- case . . switch .
- switch . .
- `break` . . break `return` , `throw` `goto case` .

.

```
char grade = 'B';

switch (grade)
{
    case 'A':
        Console.WriteLine("Excellent!");
        break;
    case 'B':
    case 'C':
        Console.WriteLine("Well done");
        break;
    case 'D':
        Console.WriteLine("You passed");
        break;
    case 'F':
        Console.WriteLine("Better try again");
        break;
```

```
    default:
        Console.WriteLine("Invalid grade");
        break;
}
```

```
if (conditionA && conditionB && conditionC) //...
```

~ .

```
bool conditions = conditionA && conditionB && conditionC;
if (conditions) // ...
```

, "if" .

true false true .

```
if (conditionA == true && conditionB == false && conditionC == true) // ...
```

.

```
if (conditionA && !conditionB && conditionC)
```

: https://riptutorial.com/ko/csharp/topic/3144/

144:

. "-1" (-1) Convert.ToInt32() Int32.Parse() . .

Examples

:

```
public interface IMyInterface1
{
    string GetName();
}

public interface IMyInterface2
{
    string GetName();
}

public class MyClass : IMyInterface1, IMyInterface2
{
    string IMyInterface1.GetName()
    {
        return "IMyInterface1";
    }

    string IMyInterface2.GetName()
    {
        return "IMyInterface2";
    }
}
```

:

```
    MyClass obj = new MyClass();

    IMyInterface1 myClass1 = (IMyInterface1)obj;
    IMyInterface2 myClass2 = (IMyInterface2)obj;

    Console.WriteLine("I am : {0}", myClass1.GetName());
    Console.WriteLine("I am : {0}", myClass2.GetName());

    // Outputs :
    // I am : IMyInterface1
    // I am : IMyInterface2
```

.

```
object value = -1;
int number = (int) value;
Console.WriteLine(Math.Abs(number));
```

value Math.Abs() , Math.Abs() object .

value int value InvalidCastException .

(`as ')

as . , null .

```
object value = "-1";
int? number = value as int?;
if(number != null)
{
    Console.WriteLine(Math.Abs(number.Value));
}
```

null , as null 3 null null .

.

```
int number = -1;
object value = number;
Console.WriteLine(value);
```

int object S . , -1 object Console.WriteLine() .

```
Console.WriteLine(-1);
```

is .

```
if(value is int)
{
   Console.WriteLine(value + "is an int");
}
```

.

```
double value = -1.1;
int number = (int) value;
```

. double -1.1 -1 .

"" .

```
object value = -1.1;
int number = (int) value; // throws InvalidCastException
```

C # . . JavaScript .

```
public class JsExpression
{
    private readonly string expression;
    public JsExpression(string rawExpression)
```

```
    {
        this.expression = rawExpression;
    }
    public override string ToString()
    {
        return this.expression;
    }
    public JsExpression IsEqualTo(JsExpression other)
    {
        return new JsExpression("(" + this + " == " + other + ")");
    }
}
```

JavaScript JsExpression .

```
JsExpression intExpression = new JsExpression("-1");
JsExpression doubleExpression = new JsExpression("-1.0");
Console.WriteLine(intExpression.IsEqualTo(doubleExpression)); // (-1 == -1.0)
```

JsExpression .

```
public static explicit operator JsExpression(int value)
{
    return new JsExpression(value.ToString());
}
public static explicit operator JsExpression(double value)
{
    return new JsExpression(value.ToString());
}

// Usage:
JsExpression intExpression = (JsExpression)(-1);
JsExpression doubleExpression = (JsExpression)(-1.0);
Console.WriteLine(intExpression.IsEqualTo(doubleExpression)); // (-1 == -1.0)
```

, .

```
public static implicit operator JsExpression(int value)
{
    return new JsExpression(value.ToString());
}
public static implicit operator JsExpression(double value)
{
    return new JsExpression(value.ToString());
}

// Usage:
JsExpression intExpression = -1;
Console.WriteLine(intExpression.IsEqualTo(-1.0)); // (-1 == -1.0)
```

LINQ

.

```
interface IThing {  }
```

```
class Thing : IThing {  }
```

LINQ `Enumerable.Cast<>()` `Enumerable.OfType<>()` `IEnumerable<>` .

```
IEnumerable<IThing> things = new IThing[] {new Thing()};
IEnumerable<Thing> things2 = things.Cast<Thing>();
IEnumerable<Thing> things3 = things.OfType<Thing>();
```

`things2` `Cast<>()` `things` `Thing` . (), `InvalidCastException` Throw.

`things3` `OfType<>()` ., .

.

```
double[] doubles = new[]{1,2,3}.Cast<double>().ToArray(); // Throws InvalidCastException
```

`.Select()` .

```
double[] doubles = new[]{1,2,3}.Select(i => (double)i).ToArray();
```

: https://riptutorial.com/ko/csharp/topic/2690/

145:

`using` ( ) .

`using` (, `IDisposable`) Using Statement .

Examples

```
using System;
using BasicStuff = System;
using Sayer = System.Console;
using static System.Console;  //From C# 6

class Program
{
    public static void Main()
    {
        System.Console.WriteLine("Ignoring usings and specifying full type name");
        Console.WriteLine("Thanks to the 'using System' directive");
        BasicStuff.Console.WriteLine("Namespace aliasing");
        Sayer.WriteLine("Type aliasing");
        WriteLine("Thanks to the 'using static' directive (from C# 6)");
    }
}
```

```
using System.Text;
//allows you to access classes within this namespace such as StringBuilder
//without prefixing them with the namespace.  i.e:

//...
var sb = new StringBuilder();
//instead of
var sb = new System.Text.StringBuilder();
```

```
using st = System.Text;
//allows you to access classes within this namespace such as StringBuilder
//prefixing them with only the defined alias and not the full namespace.  i.e:

//...
var sb = new st.StringBuilder();
//instead of
var sb = new System.Text.StringBuilder();
```

6.0

. .

```
using static System.Console;

// ...

string GetName()
```

```
{
    WriteLine("Enter your name.");
    return ReadLine();
}
```

.

```
using static System.Math;

namespace Geometry
{
    public class Circle
    {
        public double Radius { get; set; };

        public double Area => PI * Pow(Radius, 2);
    }
}
```

(: System.Random UnityEngine.Random) Random .

:

```
using UnityEngine;
using System;

Random rnd = new Random();
```

Random . .

```
using UnityEngine;
using System;
using Random = System.Random;

Random rnd = new Random();
```

.

```
using UnityEngine;
using System;
using Random = System.Random;

Random rnd = new Random();
int unityRandom = UnityEngine.Random.Range(0,100);
```

rnd System.Random unityRandom UnityEngine.Random .

using . .

:

```
using <identifier> = <namespace-or-type-name>;
```

:

```
using NewType = Dictionary<string, Dictionary<string,int>>;
NewType multiDictionary = new NewType();
//Use instances as you are using the original one
multiDictionary.Add("test", new Dictionary<string,int>());
```

: https://riptutorial.com/ko/csharp/topic/52/-

146:

Examples

Debug.WriteLine

Listeners .

```
public static void Main(string[] args)
{
    Debug.WriteLine("Hello");
}
```

Visual Studio Xamarin Studio . TraceListenerCollection .

TraceListeners

Debug.Listeners TextWriterTraceListener .

```
public static void Main(string[] args)
{
    TextWriterTraceListener myWriter = new TextWriterTraceListener(@"debug.txt");
    Debug.Listeners.Add(myWriter);
    Debug.WriteLine("Hello");

    myWriter.Flush();
}
```

ConsoleTraceListener .

```
public static void Main(string[] args)
{
    ConsoleTraceListener myWriter = new ConsoleTraceListener();
    Debug.Listeners.Add(myWriter);
    Debug.WriteLine("Hello");
}
```

: https://riptutorial.com/ko/csharp/topic/2147/

147:

, , . , , . , . . .

Examples

.

.

- , .
- , :

Singleton `CreateInstance()` `GetInstance()` (C #, `Instance` `GetInstance()` . `Instance` .

Singleton . . .

`new` `private.` `private.`

C # .

```
class Singleton
{
    // Because the _instance member is made private, the only way to get the single
    // instance is via the static Instance property below. This can also be similarly
    // achieved with a GetInstance() method instead of the property.
    private static Singleton _instance = null;

    // Making the constructor private prevents other instances from being
    // created via something like Singleton s = new Singleton(), protecting
    // against unintentional misuse.
    private Singleton()
    {
    }

    public static Singleton Instance
    {
        get
        {
            // The first call will create the one and only instance.
            if (_instance == null)
            {
                _instance = new Singleton();
            }

            // Every call afterwards will return the single instance created above.
            return _instance;
        }
    }
}
```

Instance Singleton .

```
class Program
{
    static void Main(string[] args)
    {
        Singleton s1 = Singleton.Instance;
        Singleton s2 = Singleton.Instance;

        // Both Singleton objects above should now reference the same Singleton instance.
        if (Object.ReferenceEquals(s1, s2))
        {
            Console.WriteLine("Singleton is working");
        }
        else
        {
            // Otherwise, the Singleton Instance property is returning something
            // other than the unique, single instance when called.
            Console.WriteLine("Singleton is broken");
        }
    }
}
```

: thread .

Singleton Implementation .

. .

" ?"

C # `static` . , .

C # Singleton Pattern vs. Static Class .

. . Factory Method DP .

., . .

`IDevice` .

```
public interface IDevice
{
    int Measure();
    void TurnOff();
    void TurnOn();
}
```

. `IDevice` .

```
public class AmMeter : IDevice
{
    private Random r = null;
    public AmMeter()
    {
        r = new Random();
    }
    public int Measure() { return r.Next(-25, 60); }
```

```
    public void TurnOff() { Console.WriteLine("AmMeter flashes lights saying good bye!"); }
    public void TurnOn() { Console.WriteLine("AmMeter turns on..."); }
}
public class OhmMeter : IDevice
{
    private Random r = null;
    public OhmMeter()
    {
        r = new Random();
    }
    public int Measure() { return r.Next(0, 1000000); }
    public void TurnOff() { Console.WriteLine("OhmMeter flashes lights saying good bye!"); }
    public void TurnOn() { Console.WriteLine("OhmMeter turns on..."); }
}
public class VoltMeter : IDevice
{
    private Random r = null;
    public VoltMeter()
    {
        r = new Random();
    }
    public int Measure() { return r.Next(-230, 230); }
    public void TurnOff() { Console.WriteLine("VoltMeter flashes lights saying good bye!"); }
    public void TurnOn() { Console.WriteLine("VoltMeter turns on..."); }
}
```

factory . static `DeviceFactory` `DeviceFactory` :

```
public enum Device
{
    AM,
    VOLT,
    OHM
}
public class DeviceFactory
{
    public static IDevice CreateDevice(Device d)
    {
        switch(d)
        {
            case Device.AM: return new AmMeter();
            case Device.VOLT: return new VoltMeter();
            case Device.OHM: return new OhmMeter();
            default: return new AmMeter();
        }
    }
}
```

! .

```
public class Program
{
    static void Main(string[] args)
    {
        IDevice device = DeviceFactory.CreateDevice(Device.AM);
        device.TurnOn();
        Console.WriteLine(device.Measure());
        Console.WriteLine(device.Measure());
        Console.WriteLine(device.Measure());
```

```
        Console.WriteLine(device.Measure());
        Console.WriteLine(device.Measure());
        device.TurnOff();
        Console.WriteLine();

        device = DeviceFactory.CreateDevice(Device.VOLT);
        device.TurnOn();
        Console.WriteLine(device.Measure());
        Console.WriteLine(device.Measure());
        Console.WriteLine(device.Measure());
        Console.WriteLine(device.Measure());
        Console.WriteLine(device.Measure());
        device.TurnOff();
        Console.WriteLine();

        device = DeviceFactory.CreateDevice(Device.OHM);
        device.TurnOn();
        Console.WriteLine(device.Measure());
        Console.WriteLine(device.Measure());
        Console.WriteLine(device.Measure());
        Console.WriteLine(device.Measure());
        Console.WriteLine(device.Measure());
        device.TurnOff();
        Console.WriteLine();
    }
}
```

.

AmMeter ...

36

6

33

43

24

AmMeter !

VoltMeter ...

102

-61

85

138

36

VoltMeter !

OhmMeter ...

723828

368536

685412

800266

578595

OhmMeter !

.

Builder . Shop VehicleBuilders .

```
using System;
using System.Collections.Generic;

namespace GangOfFour.Builder
{
  /// <summary>
  /// MainApp startup class for Real-World
  /// Builder Design Pattern.
  /// </summary>
  public class MainApp
  {
    /// <summary>
    /// Entry point into console application.
    /// </summary>
    public static void Main()
    {
      VehicleBuilder builder;

      // Create shop with vehicle builders
      Shop shop = new Shop();

      // Construct and display vehicles
      builder = new ScooterBuilder();
      shop.Construct(builder);
      builder.Vehicle.Show();

      builder = new CarBuilder();
      shop.Construct(builder);
      builder.Vehicle.Show();

      builder = new MotorCycleBuilder();
      shop.Construct(builder);
      builder.Vehicle.Show();

      // Wait for user
      Console.ReadKey();
    }
  }

  /// <summary>
```

```
/// The 'Director' class
/// </summary>
class Shop
{
  // Builder uses a complex series of steps
  public void Construct(VehicleBuilder vehicleBuilder)
  {
    vehicleBuilder.BuildFrame();
    vehicleBuilder.BuildEngine();
    vehicleBuilder.BuildWheels();
    vehicleBuilder.BuildDoors();
  }
}

/// <summary>
/// The 'Builder' abstract class
/// </summary>
abstract class VehicleBuilder
{
  protected Vehicle vehicle;

  // Gets vehicle instance
  public Vehicle Vehicle
  {
    get { return vehicle; }
  }

  // Abstract build methods
  public abstract void BuildFrame();
  public abstract void BuildEngine();
  public abstract void BuildWheels();
  public abstract void BuildDoors();
}

/// <summary>
/// The 'ConcreteBuilder1' class
/// </summary>
class MotorCycleBuilder : VehicleBuilder
{
  public MotorCycleBuilder()
  {
    vehicle = new Vehicle("MotorCycle");
  }

  public override void BuildFrame()
  {
    vehicle["frame"] = "MotorCycle Frame";
  }

  public override void BuildEngine()
  {
    vehicle["engine"] = "500 cc";
  }

  public override void BuildWheels()
  {
    vehicle["wheels"] = "2";
  }

  public override void BuildDoors()
  {
```

```
    vehicle["doors"] = "0";
  }
}


/// <summary>
/// The 'ConcreteBuilder2' class
/// </summary>
class CarBuilder : VehicleBuilder
{
  public CarBuilder()
  {
    vehicle = new Vehicle("Car");
  }

  public override void BuildFrame()
  {
    vehicle["frame"] = "Car Frame";
  }

  public override void BuildEngine()
  {
    vehicle["engine"] = "2500 cc";
  }

  public override void BuildWheels()
  {
    vehicle["wheels"] = "4";
  }

  public override void BuildDoors()
  {
    vehicle["doors"] = "4";
  }
}

/// <summary>
/// The 'ConcreteBuilder3' class
/// </summary>
class ScooterBuilder : VehicleBuilder
{
  public ScooterBuilder()
  {
    vehicle = new Vehicle("Scooter");
  }

  public override void BuildFrame()
  {
    vehicle["frame"] = "Scooter Frame";
  }

  public override void BuildEngine()
  {
    vehicle["engine"] = "50 cc";
  }

  public override void BuildWheels()
  {
    vehicle["wheels"] = "2";
  }
```

```
    public override void BuildDoors()
    {
      vehicle["doors"] = "0";
    }
  }

  /// <summary>
  /// The 'Product' class
  /// </summary>
  class Vehicle
  {
    private string _vehicleType;
    private Dictionary<string,string> _parts =
      new Dictionary<string,string>();

    // Constructor
    public Vehicle(string vehicleType)
    {
      this._vehicleType = vehicleType;
    }

    // Indexer
    public string this[string key]
    {
      get { return _parts[key]; }
      set { _parts[key] = value; }
    }

    public void Show()
    {
      Console.WriteLine("\n----------------------------");
      Console.WriteLine("Vehicle Type: {0}", _vehicleType);
      Console.WriteLine(" Frame : {0}", _parts["frame"]);
      Console.WriteLine(" Engine : {0}", _parts["engine"]);
      Console.WriteLine(" #Wheels: {0}", _parts["wheels"]);
      Console.WriteLine(" #Doors : {0}", _parts["doors"]);
    }
  }
}
```

: :
:
: 2
#Doors : 0

:
:
: 2500 cc
: 4
#Doors : 4

: MotorCycle
:
: 500cc

: 2
#Doors : 0

.

Color .

```
using System;
using System.Collections.Generic;

namespace GangOfFour.Prototype
{
  /// <summary>
  /// MainApp startup class for Real-World
  /// Prototype Design Pattern.
  /// </summary>
  class MainApp
  {
    /// <summary>
    /// Entry point into console application.
    /// </summary>
    static void Main()
    {
      ColorManager colormanager = new ColorManager();

      // Initialize with standard colors
      colormanager["red"] = new Color(255, 0, 0);
      colormanager["green"] = new Color(0, 255, 0);
      colormanager["blue"] = new Color(0, 0, 255);

      // User adds personalized colors
      colormanager["angry"] = new Color(255, 54, 0);
      colormanager["peace"] = new Color(128, 211, 128);
      colormanager["flame"] = new Color(211, 34, 20);

      // User clones selected colors
      Color color1 = colormanager["red"].Clone() as Color;
      Color color2 = colormanager["peace"].Clone() as Color;
      Color color3 = colormanager["flame"].Clone() as Color;

      // Wait for user
      Console.ReadKey();
    }
  }

  /// <summary>
  /// The 'Prototype' abstract class
  /// </summary>
  abstract class ColorPrototype
  {
    public abstract ColorPrototype Clone();
  }

  /// <summary>
  /// The 'ConcretePrototype' class
  /// </summary>
  class Color : ColorPrototype
  {
    private int _red;
    private int _green;
```

```
    private int _blue;

    // Constructor
    public Color(int red, int green, int blue)
    {
      this._red = red;
      this._green = green;
      this._blue = blue;
    }

    // Create a shallow copy
    public override ColorPrototype Clone()
    {
      Console.WriteLine(
        "Cloning color RGB: {0,3},{1,3},{2,3}",
        _red, _green, _blue);

      return this.MemberwiseClone() as ColorPrototype;
    }
  }

  /// <summary>
  /// Prototype manager
  /// </summary>
  class ColorManager
  {
    private Dictionary<string, ColorPrototype> _colors =
      new Dictionary<string, ColorPrototype>();

    // Indexer
    public ColorPrototype this[string key]
    {
      get { return _colors[key]; }
      set { _colors.Add(key, value); }
    }
  }
}
```

:

RGB : 255, 0, 0

RGB : 128,211,128

RGB : 211, 34, 20

.

. .

```
using System;

namespace GangOfFour.AbstractFactory
{
  /// <summary>
  /// MainApp startup class for Real-World
  /// Abstract Factory Design Pattern.
  /// </summary>
```

```
class MainApp
{
  /// <summary>
  /// Entry point into console application.
  /// </summary>
  public static void Main()
  {
    // Create and run the African animal world
    ContinentFactory africa = new AfricaFactory();
    AnimalWorld world = new AnimalWorld(africa);
    world.RunFoodChain();

    // Create and run the American animal world
    ContinentFactory america = new AmericaFactory();
    world = new AnimalWorld(america);
    world.RunFoodChain();

    // Wait for user input
    Console.ReadKey();
  }
}


/// <summary>
/// The 'AbstractFactory' abstract class
/// </summary>
abstract class ContinentFactory
{
  public abstract Herbivore CreateHerbivore();
  public abstract Carnivore CreateCarnivore();
}

/// <summary>
/// The 'ConcreteFactory1' class
/// </summary>
class AfricaFactory : ContinentFactory
{
  public override Herbivore CreateHerbivore()
  {
    return new Wildebeest();
  }
  public override Carnivore CreateCarnivore()
  {
    return new Lion();
  }
}

/// <summary>
/// The 'ConcreteFactory2' class
/// </summary>
class AmericaFactory : ContinentFactory
{
  public override Herbivore CreateHerbivore()
  {
    return new Bison();
  }
  public override Carnivore CreateCarnivore()
  {
    return new Wolf();
  }
}
```

```
/// <summary>
/// The 'AbstractProductA' abstract class
/// </summary>
abstract class Herbivore
{
}

/// <summary>
/// The 'AbstractProductB' abstract class
/// </summary>
abstract class Carnivore
{
  public abstract void Eat(Herbivore h);
}

/// <summary>
/// The 'ProductA1' class
/// </summary>
class Wildebeest : Herbivore
{
}

/// <summary>
/// The 'ProductB1' class
/// </summary>
class Lion : Carnivore
{
  public override void Eat(Herbivore h)
  {
    // Eat Wildebeest
    Console.WriteLine(this.GetType().Name +
      " eats " + h.GetType().Name);
  }
}

/// <summary>
/// The 'ProductA2' class
/// </summary>
class Bison : Herbivore
{
}

/// <summary>
/// The 'ProductB2' class
/// </summary>
class Wolf : Carnivore
{
  public override void Eat(Herbivore h)
  {
    // Eat Bison
    Console.WriteLine(this.GetType().Name +
      " eats " + h.GetType().Name);
  }
}

/// <summary>
/// The 'Client' class
/// </summary>
class AnimalWorld
{
```

```
    private Herbivore _herbivore;
    private Carnivore _carnivore;

    // Constructor
    public AnimalWorld(ContinentFactory factory)
    {
      _carnivore = factory.CreateCarnivore();
      _herbivore = factory.CreateHerbivore();
    }

    public void RunFoodChain()
    {
      _carnivore.Eat(_herbivore);
    }
  }
}
```

:

Wildebeest .

.

: https://riptutorial.com/ko/csharp/topic/6654/---

148:

- @" . \n \n . @"
- @""""".""

@ .

```
var combinedString = @"\t means a tab" + @" and \n means a newline";
```

Examples

```
var multiLine = @"This is a

multiline paragraph";
```

:

.NET Fiddle

.

```
var multilineWithDoubleQuotes = @"I went to a city named

                        ""San Diego""

                    during summer vacation.";
```

.NET Fiddle

2 3 / . .

" "" 2 .

```
var str = @"""I don't think so,"" he said.";
Console.WriteLine(str);
```

:

  " ".

.NET Fiddle

C # 6 .

```
Console.WriteLine($@"Testing \n 1 2 {5 - 2}
New line");
```

:

\n 1 2 3

.NET Fiddle

. .

.

. ( )

("" "). , @ .

```
var filename = @"c:\temp\newfile.txt"
```

:

c : \ temp \ newfile.txt

( ) :

```
var filename = "c:\temp\newfile.txt"
```

:

```
c:    emp
ewfile.txt
```

. (\t \n .)

.NET Fiddle

: https://riptutorial.com/ko/csharp/topic/16/--

149:

Examples

MemoryCache

```
//Get instance of cache
using System.Runtime.Caching;

var cache = MemoryCache.Default;

//Check if cache contains an item with
cache.Contains("CacheKey");

//get item from cache
var item = cache.Get("CacheKey");

//get item from cache or add item if not existing
object list = MemoryCache.Default.AddOrGetExisting("CacheKey", "object to be stored",
DateTime.Now.AddHours(12));

//note if item not existing the item is added by this method
//but the method returns null
```

: https://riptutorial.com/ko/csharp/topic/4383/

150:

System.Collections.IEnumerable Add . .

Examples

.

```
var stringList = new List<string>
{
    "foo",
    "bar",
};
```

Add() . .

```
var temp = new List<string>();
temp.Add("foo");
temp.Add("bar");
var stringList = temp;
```

.

Add() .

```
var numberDictionary = new Dictionary<int, string>
{
    { 1, "One" },
    { 2, "Two" },
};
```

.

```
var temp = new Dictionary<int, string>();
temp.Add(1, "One");
temp.Add(2, "Two");
var numberDictionarynumberDictionary = temp;
```

C # 6

C # 6 .

:

```
var dict = new Dictionary<string, int>
{
    ["key1"] = 1,
    ["key2"] = 50
```

```
};
```

.

```
var dict = new Dictionary<string, int>();
dict["key1"] = 1;
dict["key2"] = 50
```

C # 6 .

```
var dict = new Dictionary<string, int>
{
    { "key1", 1 },
    { "key2", 50 }
};
```

?

```
var dict = new Dictionary<string, int>();
dict.Add("key1", 1);
dict.Add("key2", 50);
```

Add() ., , .

```
public class IndexableClass
{
    public int this[int index]
    {
        set
        {
            Console.WriteLine("{0} was assigned to index {1}", value, index);
        }
    }
}

var foo = new IndexableClass
{
    [0] = 10,
    [1] = 20
}
```

.

```
10 was assigned to index 0
20 was assigned to index 1
```

IEnumerable Add . C # 6 IEnumerable Add .

```
class Program
{
    static void Main()
    {
        var col = new MyCollection {
            "foo",
```

```
            { "bar", 3 },
            "baz",
            123.45d,
        };
    }
}

class MyCollection : IEnumerable
{
    private IList list = new ArrayList();

    public void Add(string item)
    {
        list.Add(item)
    }

    public void Add(string item, int count)
    {
        for(int i=0;i< count;i++) {
            list.Add(item);
        }
    }

    public IEnumerator GetEnumerator()
    {
        return list.GetEnumerator();
    }
}

static class MyCollectionExtensions
{
    public static void Add(this MyCollection @this, double value) =>
        @this.Add(value.ToString());
}
```

Collection Initializer

.

```
public class LotteryTicket : IEnumerable{
    public int[] LuckyNumbers;
    public string UserName;

    public void Add(string userName, params int[] luckyNumbers){
        UserName = userName;
        Lottery = luckyNumbers;
    }
}
```

.

```
var Tickets = new List<LotteryTicket>{
    {"Mr Cool"  , 35663, 35732, 12312, 75685},
    {"Bruce"    , 26874, 66677, 24546, 36483, 46768, 24632, 24527},
    {"John Cena", 25446, 83356, 65536, 23783, 24567, 89337}
}
```

```
public class Tag
{
    public IList<string> Synonyms { get; set; }
}
```

`Synonyms` `.Tag` `Synonyms` .

```
Tag t = new Tag
{
    Synonyms = new List<string> {"c#", "c-sharp"}
};
```

. (`Synonyms` (`Synonyms` private setter) :

```
public class Tag
{
    public Tag()
    {
        Synonyms = new List<string>();
    }

    public IList<string> Synonyms { get; private set; }
}
```

`Tag` .

```
Tag t = new Tag
{
    Synonyms = {"c#", "c-sharp"}
};
```

initializers `Add()` . . `Add()` .

: https://riptutorial.com/ko/csharp/topic/21/--

151:

1. Contract.Requires (, userMessage)

   Contract.Requires (, userMessage)

   Contract.Result <T>

   Contract.Ensures ()

   Contract.Invariants ()

.NET System.Diagnostics Contracts Contract by Design .NET 4.0 . API , . ,
.

?

. . -

-
- . .
- .

. , .

Examples

```
namespace CodeContractsDemo
{
    using System;
    using System.Collections.Generic;
    using System.Diagnostics.Contracts;

    public class PaymentProcessor
    {
        private List<Payment> _payments = new List<Payment>();

        public void Add(Payment payment)
        {
            Contract.Requires(payment != null);
            Contract.Requires(!string.IsNullOrEmpty(payment.Name));
            Contract.Requires(payment.Date <= DateTime.Now);
            Contract.Requires(payment.Amount > 0);

            this._payments.Add(payment);
        }
    }
}
```

```
public double GetPaymentsTotal(string name)
```

```
{
    Contract.Ensures(Contract.Result<double>() >= 0);

    double total = 0.0;

    foreach (var payment in this._payments) {
        if (string.Equals(payment.Name, name)) {
            total += payment.Amount;
        }
    }

    return total;
}
```

```
namespace CodeContractsDemo
{
    using System;
    using System.Diagnostics.Contracts;

    public class Point
    {
        public int X { get; set; }
        public int Y { get; set; }

        public Point()
        {
        }

        public Point(int x, int y)
        {
            this.X = x;
            this.Y = y;
        }

        public void Set(int x, int y)
        {
            this.X = x;
            this.Y = y;
        }

        public void Test(int x, int y)
        {
            for (int dx = -x; dx <= x; dx++) {
                this.X = dx;
                Console.WriteLine("Current X = {0}", this.X);
            }

            for (int dy = -y; dy <= y; dy++) {
                this.Y = dy;
                Console.WriteLine("Current Y = {0}", this.Y);
            }

            Console.WriteLine("X = {0}", this.X);
            Console.WriteLine("Y = {0}", this.Y);
        }

        [ContractInvariantMethod]
        private void ValidateCoordinates()
        {
            Contract.Invariant(this.X >= 0);
```

```
            Contract.Invariant(this.Y >= 0);
        }
    }
}
```

```
[ContractClass(typeof(ValidationContract))]
interface IValidation
{
    string CustomerID{get;set;}
    string Password{get;set;}
}

[ContractClassFor(typeof(IValidation))]
sealed class ValidationContract:IValidation
{
    string IValidation.CustomerID
    {
        [Pure]
        get
        {
            return Contract.Result<string>();
        }
        set
        {
            Contract.Requires<ArgumentNullException>(!string.IsNullOrEmpty(value), "Customer
ID cannot be null!!");
        }
    }

    string IValidation.Password
    {
        [Pure]
        get
        {
            return Contract.Result<string>();
        }
        set
        {
            Contract.Requires<ArgumentNullException>(!string.IsNullOrEmpty(value), "Password
cannot be null!!");
        }
    }
}

class Validation:IValidation
{
    public string GetCustomerPassword(string customerID)
    {
        Contract.Requires(!string.IsNullOrEmpty(customerID),"Customer ID cannot be Null");
        Contract.Requires<ArgumentNullException>(!string.IsNullOrEmpty(customerID),
"Exception!!");
        Contract.Ensures(Contract.Result<string>() != null);
        string password="AAA@1234";
        if (customerID!=null)
        {
            return password;
        }
        else
        {
            return null;
```

```
        }

    }

    private string m_custID, m_PWD;

    public string CustomerID
    {
        get
        {
            return m_custID;
        }
        set
        {
            m_custID = value;
        }
    }

    public string Password
    {
        get
        {
            return m_PWD;
        }
        set
        {
            m_PWD = value;
        }
    }
}
```

`[ContractClass]` `IValidation` . . `ValidationContract` `Contract.Requires<T>` null . `T` .

get `[Pure]` . `IValidation` .

: https://riptutorial.com/ko/csharp/topic/4241/-

152:

Examples

true.

., ., Debug.Assert true true .

Debug.Assert DEBUG . RELEASE . .

, .

```
var systemData = RetrieveSystemConfiguration();
Debug.Assert(systemData != null);
```

RetrieveSystemConfiguration () null assert .

.

```
UserData user = RetrieveUserData();
Debug.Assert(user != null);
Debug.Assert(user.Age > 0);
int year = DateTime.Today.Year - user.Age;
```

RetrieveUserData () . Age ( ) .

assert .

```
string input = Console.ReadLine();
int age = Convert.ToInt32(input);
Debug.Assert(age > 16);
Console.WriteLine("Great, you are over 16");
```

Assert . . . .

: https://riptutorial.com/ko/csharp/topic/4349/---

153:

. @ . @if if .

C # ""( ) . @ (, ,) .

- abstract
- as
- base
- bool
- break
- byte
- case
- catch
- char
- checked
- class
- const
- continue
- decimal
- default
- delegate
- do
- double
- else
- enum
- event
- explicit
- extern
- false
- finally
- fixed
- float
- for
- foreach
- goto
- if
- implicit
- in
- int
- interface
- internal
- is
- lock
- long
- namespace
- new
- null
- object
- operator
- out
- override
- params

- private
- protected
- public
- readonly
- ref
- return
- sbyte
- sealed
- short
- sizeof
- stackalloc
- static
- string
- struct
- switch
- this
- throw
- true
- try
- typeof
- uint
- ulong
- unchecked
- unsafe
- ushort
- using ()
- using ()
- virtual
- void
- volatile
- when
- while

C # . . , @ .

- add
- alias
- ascending
- async
- await
- descending
- dynamic
- from
- get
- global
- group
- into
- join
- let
- nameof
- orderby
- partial
- remove
- select
- set

- value
- var
- where
- yield

Examples

stackalloc

stackalloc . .

```
//Allocate 1024 bytes. This returns a pointer to the first byte.
byte* ptr = stackalloc byte[1024];

//Assign some values...
ptr[0] = 109;
ptr[1] = 13;
ptr[2] = 232;
...
```

.

C # . . .

```
//Allocate 1 byte
byte* ptr = stackalloc byte[1];

//Unpredictable results...
ptr[10] = 1;
ptr[-1] = 2;
```

. , stackalloc .

```
unsafe IntPtr Leak() {
    //Allocate some memory on the stack
    var ptr = stackalloc byte[1024];

    //Return a pointer to that memory (this exits the scope of "Leak")
    return new IntPtr(ptr);
}

unsafe void Bad() {
     //ptr is now an invalid pointer, using it in any way will have
     //unpredictable results. This is exactly the same as accessing beyond
     //the bounds of the pointer.
     var ptr = Leak();
}
```

stackalloc . .

```
byte* ptr;
...
ptr = stackalloc byte[1024];
```

:

stackalloc (interop) . .

- .
- .
- CPU .

volatile . volatile . volatile .

volatile . .

```
public class Example
{
    public int x;

    public void DoStuff()
    {
        x = 5;

        // the compiler will optimize this to y = 15
        var y = x + 10;

        /* the value of x will always be the current value, but y will always be "15" */
        Debug.WriteLine("x = " + x + ", y = " + y);
    }
}
```

, x = 5 y = x + 10 y 15 , y = 15 . x public x . . x volatile .

```
public class Example
{
    public volatile int x;

    public void DoStuff()
    {
        x = 5;

        // the compiler no longer optimizes this statement
        var y = x + 10;

        /* the value of x and y will always be the correct values */
        Debug.WriteLine("x = " + x + ", y = " + y);
    }
}
```

x . x .

volatile class struct . .

```
public void MyMethod()
{
    volatile int x;
}
```

volatile .

-
- sbyte , byte , short , ushort , int , uint , char , float bool
- byte , sbyte , short , ushort , int uint .
- IntPtr UIntPtr

:

- volatile lock .
- volatile .
- volatile 32 64 . Interlocked.Read Interlocked.Exchange .

fixed . arround, . .

- fixed .

```
var myStr = "Hello world!";

fixed (char* ptr = myStr)
{
    // myStr is now fixed (won't be [re]moved by the Garbage Collector).
    // We can now do something with ptr.
}
```

.

```
unsafe struct Example
{
    public fixed byte SomeField[8];
    public fixed char AnotherField[64];
}
```

fixed struct ( ).

, , , , nullable (int?) default(TheType) null .

```
class MyClass {}
Debug.Assert(default(MyClass) == null);
Debug.Assert(default(string) == null);
```

default(TheType) new TheType() .

```
struct Coordinates
{
    public int X { get; set; }
    public int Y { get; set; }
}

struct MyStruct
{
    public string Name { get; set; }
```

```
    public Coordinates Location { get; set; }
    public Coordinates? SecondLocation { get; set; }
    public TimeSpan Duration { get; set; }
}

var defaultStruct = default(MyStruct);
Debug.Assert(defaultStruct.Equals(new MyStruct()));
Debug.Assert(defaultStruct.Location.Equals(new Coordinates()));
Debug.Assert(defaultStruct.Location.X == 0);
Debug.Assert(defaultStruct.Location.Y == 0);
Debug.Assert(defaultStruct.SecondLocation == null);
Debug.Assert(defaultStruct.Name == null);
Debug.Assert(defaultStruct.Duration == TimeSpan.Zero);
```

default(T) T T :

```
public T GetResourceOrDefault<T>(string resourceName)
{
   if (ResourceExists(resourceName))
   {
      return (T)GetResource(resourceName);
   }
   else
   {
      return default(T);
   }
}
```

readonly . readonly .

readonly const . const . readonly . readonly .

readonly .

```
class Person
{
    readonly string _name;
    readonly string _surname = "Surname";

    Person(string name)
    {
        _name = name;
    }
    void ChangeName()
    {
        _name = "another name"; // Compile error
        _surname = "another surname"; // Compile error
    }
}
```

: . . .

: .

```
public class Car
{
```

```
    public double Speed {get; set;}
}

//In code

private readonly Car car = new Car();

private void SomeMethod()
{
    car.Speed = 100;
}
```

as . , as (), InvalidCastException null .

expression as type expression is type ? (type)expression : (type)null expression is type ? (type)expression : (type)null as , (NULL) , . . .

expression ( ) .

as . - is , . / 'as' . is .

, .

as null NullReferenceException .

```
object something = "Hello";
Console.WriteLine(something as string);        //Hello
Console.Writeline(something as Nullable<int>); //null
Console.WriteLine(something as int?);          //null

//This does NOT compile:
//destination type must be a reference type (or a nullable value type)
Console.WriteLine(something as int);
```

.NET Fiddle

as :

```
Console.WriteLine(something is string ? (string)something : (string)null);
```

Equals .

```
class MyCustomClass
{

    public override bool Equals(object obj)
    {
        MyCustomClass customObject = obj as MyCustomClass;

        // if it is null it may be really null
        // or it may be of a different type
        if (Object.ReferenceEquals(null, customObject))
        {
            // If it is null then it is not equal to this instance.
            return false;
```

```
        }

        // Other equality controls specific to class
    }

}
```

~.

., BaseInterface BaseInterface   BaseInterface .

```
interface BaseInterface {}
class BaseClass : BaseInterface {}
class DerivedClass : BaseClass {}

var d = new DerivedClass();
Console.WriteLine(d is DerivedClass);  // True
Console.WriteLine(d is BaseClass);     // True
Console.WriteLine(d is BaseInterface); // True
Console.WriteLine(d is object);        // True
Console.WriteLine(d is string);        // False

var b = new BaseClass();
Console.WriteLine(b is DerivedClass);  // False
Console.WriteLine(b is BaseClass);     // True
Console.WriteLine(b is BaseInterface); // True
Console.WriteLine(b is object);        // True
Console.WriteLine(b is string);        // False
```

, as '

```
interface BaseInterface {}
class BaseClass : BaseInterface {}
class DerivedClass : BaseClass {}

var d = new DerivedClass();
Console.WriteLine(d is DerivedClass);  // True - valid use of 'is'
Console.WriteLine(d is BaseClass);     // True - valid use of 'is'

if(d is BaseClass){
    var castedD = (BaseClass)d;
    castedD.Method(); // valid, but not best practice
}

var asD = d as BaseClass;

if(asD!=null){
    asD.Method(); //prefered method since you incur only one unboxing penalty
}
```

C # 7 pattern matching is . C # 7 :

7.0

```
if(d is BaseClass asD ){
    asD.Method();
```

```
}
```

, Type .

```
Type type = typeof(string);
Console.WriteLine(type.FullName); //System.String
Console.WriteLine("Hello".GetType() == type); //True
Console.WriteLine("Hello".GetType() == typeof(string)); //True
```

const

const . readonly .

.

```
const double c = 299792458;  // Speed of light

double CalculateEnergy(double mass)
{
    return mass * c * c;
}
```

c return mass * 299792458 * 299792458 .

c . .

```
const double c = 299792458;  // Speed of light

c = 500;  //compile-time error
```

.

```
private const double c = 299792458;
public const double c = 299792458;
internal const double c = 299792458;
```

const static . static .

.

```
double CalculateEnergy(double mass)
{
    const c = 299792458;
    return mass * c * c;
}
```

private public .

---

const . sbyte , byte , short , ushort , int , uint , long , ulong , char , float , double , decimal , bool enum . (: TimeSpan Guid) const .

string . null .

---

const switch case , , .

---

const . , A public const int MaxRetries = 3; , B A MaxRetries 5 ( ) B B A).

, , , const . const static readonly .

namespace . C # .

```
namespace StackOverflow
{
    namespace Documentation
    {
        namespace CSharp.Keywords
        {
            public class Program
            {
                public static void Main()
                {
                    Console.WriteLine(typeof(Program).Namespace);
                    //StackOverflow.Documentation.CSharp.Keywords
                }
            }
        }
    }
}
```

C # . .

```
namespace StackOverflow.Documentation.CSharp.Keywords
{
    public class Program
    {
        public static void Main()
        {
            Console.WriteLine(typeof(Program).Namespace);
            //StackOverflow.Documentation.CSharp.Keywords
        }
    }
}
```

try, catch, finally, throw

try , catch , finally throw .

```
var processor = new InputProcessor();

// The code within the try block will be executed. If an exception occurs during execution of
// this code, execution will pass to the catch block corresponding to the exception type.
try
{
    processor.Process(input);
```

```
}
// If a FormatException is thrown during the try block, then this catch block
// will be executed.
catch (FormatException ex)
{
    // Throw is a keyword that will manually throw an exception, triggering any catch block
that is
    // waiting for that exception type.
    throw new InvalidOperationException("Invalid input", ex);
}
// catch can be used to catch all or any specific exceptions. This catch block,
// with no type specified, catches any exception that hasn't already been caught
// in a prior catch block.
catch
{
    LogUnexpectedException();
    throw; // Re-throws the original exception.
}
// The finally block is executed after all try-catch blocks have been; either after the try
has
// succeeded in running all commands or after all exceptions have been caught.
finally
{
    processor.Dispose();
}
```

: `return try finally` . :

```
try
{
    connection.Open();
    return connection.Get(query);
}
finally
{
    connection.Close();
}
```

`connection.Close()` `connection.Get(query)` .

(for, foreach, do, while).

```
for (var i = 0; i < 10; i++)
{
    if (i < 5)
    {
        continue;
    }
    Console.WriteLine(i);
}
```

:

5
6
7

8
9

.NET Fiddle□□□ □□□ □□

```
var stuff = new [] {"a", "b", null, "c", "d"};

foreach (var s in stuff)
{
    if (s == null)
    {
        continue;
    }
    Console.WriteLine(s);
}
```

:

.NET Fiddle

ref out . , .

```
int x = 5;
ChangeX(ref x);
// The value of x could be different now
```

(ref) .

```
Address a = new Address();
ChangeFieldInAddress(a);
// a will be the same instance as before, even if it is modified
CreateANewInstance(ref a);
// a could be an entirely new instance now
```

out ref ref out .

out out .

```
int number = 1;
Console.WriteLine("Before AddByRef: " + number); // number = 1
AddOneByRef(ref number);
Console.WriteLine("After AddByRef: " + number);  // number = 2
SetByOut(out number);
Console.WriteLine("After SetByOut: " + number);  // number = 34

void AddOneByRef(ref int value)
{
    value++;
}
```

```
void SetByOut(out int value)
{
    value = 34;
}
```

.NET Fiddle□□□ □□□ □□

out (ref) :

```
void PrintByOut(out int value)
{
    Console.WriteLine("Hello!");
}
```

out . , out (covariant) .

generic . . . - MSDN

```
//if we have an interface like this
interface ICovariant<out R> { }

//and two variables like
ICovariant<Object> iobj = new Sample<Object>();
ICovariant<String> istr = new Sample<String>();

// then the following statement is valid
// without the out keyword this would have thrown error
iobj = istr; // implicit conversion occurs here
```

,

checked unchecked . checked unchecked "" .

checked ( ) throw. unchecked . .

checked . .

checked checked unchecked .

unchecked . " (wrap around)" .

```
byte Checksum(byte[] data) {
    byte result = 0;
    for (int i = 0; i < data.Length; i++) {
        result = unchecked(result + data[i]); // unchecked expression
    }
    return result;
}
```

unchecked object.GetHashCode() . : System.Object.GetHashCode ? .

checked OverflowException throw.

```
int SafeSum(int x, int y) {
    checked { // checked block
        return x + y;
    }
}
```

checked unchecked .

. Enum.ToObject() , Convert.ToInt32() / .

: ( ) **/ checked [+ | -]** . . *checked unchecked* .

goto .

goto A :

:

```
void InfiniteHello()
{
    sayHello:
    Console.WriteLine("Hello!");
    goto sayHello;
}
```

.NET Fiddle

:

```
enum Permissions { Read, Write };

switch (GetRequestedPermission())
{
    case Permissions.Read:
        GrantReadAccess();
        break;

    case Permissions.Write:
        GrantWriteAccess();
        goto case Permissions.Read; //People with write access also get read
}
```

.NET Fiddle

C # fall-through case switch .

```
var exCount = 0;
retry:
try
{
```

```
    //Do work
}
catch (IOException)
{
    exCount++;
    if (exCount < 3)
    {
        Thread.Sleep(100);
        goto retry;
    }
    throw;
}
```

.NET Fiddle

goto .

C # goto :

- switch fall-through case.
- . LINQ .
- . C # .
- (:). / .

enum Enum . Enum ValueType .

```
public enum DaysOfWeek
{
    Monday,
    Tuesday,
}
```

( ) .

```
public enum NotableYear
{
   EndOfWwI = 1918;
   EnfOfWwII = 1945,
}
```

0 . . enum (YourEnumType) 0 (YourEnumType) 0 . YourEnumType enume . 0 enum .

enum int byte , sbyte , short , ushort , int , uint , long ulong . byte .

```
enum Days : byte
{
    Sunday = 0,
    Monday,
    Tuesday,
    Wednesday,
    Thursday,
```

```
    Friday,
    Saturday
};
```

/ .

```
int value = (int)NotableYear.EndOfWwI;
```

enum .

```
void PrintNotes(NotableYear year)
{
    if (!Enum.IsDefined(typeof(NotableYear), year))
        throw InvalidEnumArgumentException("year", (int)year, typeof(NotableYear));

    // ...
}
```

base . .

```
public class Child : SomeBaseClass {
    public Child() : base("some string for the base class")
    {
    }
}

public class SomeBaseClass {
    public SomeBaseClass()
    {
        // new Child() will not call this constructor, as it does not have a parameter
    }
    public SomeBaseClass(string message)
    {
        // new Child() will use this base constructor because of the specified parameter in
Child's constructor
        Console.WriteLine(message);
    }
}
```

```
public override void SomeVirtualMethod() {
    // Do something, then call base implementation
    base.SomeVirtualMethod();
}
```

base . . , .

```
public class Parent
{
    public virtual int VirtualMethod()
    {
        return 1;
    }
}

public class Child : Parent
```

```
{
    public override int VirtualMethod() {
        return 11;
    }

    public int NormalMethod()
    {
        return base.VirtualMethod();
    }

    public void CallMethods()
    {
        Assert.AreEqual(11, VirtualMethod());

        Assert.AreEqual(1, NormalMethod());
        Assert.AreEqual(1, base.VirtualMethod());
    }
}

public class GrandChild : Child
{
    public override int VirtualMethod()
    {
        return 21;
    }

    public void CallAgain()
    {
        Assert.AreEqual(21, VirtualMethod());
        Assert.AreEqual(11, base.VirtualMethod());

        // Notice that the call to NormalMethod below still returns the value
        // from the extreme base class even though the method has been overridden
        // in the child class.
        Assert.AreEqual(1, NormalMethod());
    }
}
```

foreach IEnumerable im .

```
var lines = new string[] {
    "Hello world!",
    "How are you doing today?",
    "Goodbye"
};

foreach (string line in lines)
{
    Console.WriteLine(line);
}
```

.

"!"
" ?"
""

.NET Fiddle

break foreach continue .

```
var numbers = new int[] {1, 2, 3, 4, 5, 6};

foreach (var number in numbers)
{
    // Skip if 2
    if (number == 2)
        continue;

    // Stop iteration if 5
    if (number == 5)
        break;

    Console.Write(number + ", ");
}

// Prints: 1, 3, 4,
```

.NET Fiddle

List .

IEnumerable foreach bool MoveNext() object GetEnumerator() bool MoveNext() object Current { get; } .

params . , 0, .

```
static int AddAll(params int[] numbers)
{
    int total = 0;
    foreach (int number in numbers)
    {
        total += number;
    }

    return total;
}
```

int int .

```
AddAll(5, 10, 15, 20);                // 50
AddAll(new int[] { 5, 10, 15, 20 });  // 50
```

params .

---

params . C # params . :

```
static double Add(params double[] numbers)
{
    Console.WriteLine("Add with array of doubles");
    double total = 0.0;
    foreach (double number in numbers)
```

```
    {
        total += number;
    }

    return total;
}

static int Add(int a, int b)
{
    Console.WriteLine("Add with 2 ints");
    return a + b;
}
```

`params` 2 .

```
Add(2, 3);      //prints "Add with 2 ints"
Add(2, 3.0);    //prints "Add with array of doubles" (doubles are not ints)
Add(2, 3, 4);   //prints "Add with array of doubles" (no 3 argument overload)
```

(for, foreach, do, while) `break` . iterator `yield` .

```
for (var i = 0; i < 10; i++)
{
    if (i == 5)
    {
        break;
    }
    Console.WriteLine("This will appear only 5 times, as the break will stop the loop.");
}
```

.NET Fiddle

```
foreach (var stuff in stuffCollection)
{
    if (stuff.SomeStringProp == null)
        break;
    // If stuff.SomeStringProp for any "stuff" is null, the loop is aborted.
    Console.WriteLine(stuff.SomeStringProp);
}
```

break case default switch-case .

```
switch(a)
{
    case 5:
        Console.WriteLine("a was 5!");
        break;

    default:
        Console.WriteLine("a was something else!");
        break;
}
```

switch case 'break' . '' . 'goto' 'case' .

`0, 1, 2, ..., 9` `.yield break` ().

```
public static IEnumerable<int> GetNumbers()
{
    int i = 0;
    while (true) {
        if (i < 10) {
            yield return i++;
        } else {
            yield break;
        }
    }
    Console.WriteLine("This line will not be executed");
}
```

.NET Fiddle

C # . , .

```
foreach (var outerItem in outerList)
{
    foreach (var innerItem in innerList)
    {
        if (innerItem.ShoudBreakForWhateverReason)
            // This will only break out of the inner loop, the outer will continue:
            break;
    }
}
```

.

- **goto** .
- (`shouldBreak`).
- `return` .

```
bool shouldBreak = false;
while(comeCondition)
{
    while(otherCondition)
    {
        if (conditionToBreak)
        {
            // Either tranfer control flow to the label below...
            goto endAllLooping;

            // OR use a flag, which can be checked in the outer loop:
            shouldBreak = true;
        }
    }

    if(shouldBreakNow)
    {
        break; // Break out of outer loop if flag was set to true
    }
}

endAllLooping: // label from where control flow will continue
```

abstract .

. .

.

```
abstract class Animal
{
    string Name { get; set; }
    public abstract void MakeSound();
}

public class Cat : Animal
{
    public override void MakeSound()
    {
        Console.WriteLine("Meov meov");
    }
}

public class Dog : Animal
{
    public override void MakeSound()
    {
        Console.WriteLine("Bark bark");
    }
}

Animal cat = new Cat();       // Allowed due to Cat deriving from Animal
cat.MakeSound();              // will print out "Meov meov"

Animal dog = new Dog();       // Allowed due to Dog deriving from Animal
dog.MakeSound();              // will print out "Bark bark"

Animal animal = new Animal(); // Not allowed due to being an abstract class
```

abstract , . .

```
abstract class Animal
{
   public abstract string Name { get; set; }
}

public class Cat : Animal
{
    public override string Name { get; set; }
}

public class Dog : Animal
{
    public override string Name { get; set; }
}
```

,,

`float` .NET `System.Single` . IEEE 754 . C # `mscorlib.dll` .

: -3.4 × 10 ^{38} ~ 3.4 × 10 ^{38}

: 6-9

:

```
float f = 0.1259;
var f1 = 0.7895f; // f is literal suffix to represent float values
```

`float` . .

`double` .NET `System.Double` . 64 . C # `mscorlib.dll` .

: ± 5.0 × 10 $^{-324}$ ~ ± 1.7 × 10 ^{308}

: 15-16

:

```
double distance = 200.34; // a double value
double salary = 245; // an integer implicitly type-casted to double value
var marks = 123.764D; // D is literal suffix to represent double values
```

`decimal` .NET `System.Decimal` . 128 . . C # `mscorlib.dll` .

: -7.9 × 10 ^{28} ~ 7.9 × 10 ^{28}

: 28-29

:

```
decimal payable = 152.25m; // a decimal value
var marks = 754.24m; // m is literal suffix to represent decimal values
```

uint

uint . 32 . **uint** Common Type System `System.UInt32` . `mscorlib.dll` `mscorlib.dll` C # . 4 .

0 4,294,967,295 .

```
uint i = 425697; // Valid expression, explicitly stated to compiler
var i1 = 789247U; // Valid expression, suffix allows compiler to determine datatype
uint x = 3.0; // Error, there is no implicit conversion
```

: Microsoft **uint** CLS **int** .

this () . () ( ) .

```
public MyClass {
    int a;

    void set_a(int a)
    {
        //this.a refers to the variable defined outside of the method,
        //while a refers to the passed parameter.
        this.a = a;
    }
}
```

.

```
public MyClass(int arg) : this(arg, null)
{
}
```

:

```
public string this[int idx1, string idx2]
{
    get { /* ... */ }
    set { /* ... */ }
}
```

:

```
public static int Count<TItem>(this IEnumerable<TItem> source)
{
    // ...
}
```

this this.MemberOfType this.MemberOfType MemberOfType . base .

, this . , IEnumerable<> Count .

```
this.Count()  // works like StaticClassForExtensionMethod.Count(this)
```

this .

...

: for (initializer; condition; iterator)

- for .
- initializer .
- condition .

- iterator .

to for .

```
string str = "Hello";
for (int i = 0; i < str.Length; i++)
{
    Console.WriteLine(str[i]);
}
```

:

H

.NET Fiddle

for . , .

```
for( ; ; )
{
    // Your code here
}
```

initializer . condition bool . iterator .

```
string hello = "hello";
for (int i = 0, j = 1, k = 9; i < 3 && k > 0; i++, hello += i) {
    Console.WriteLine(hello);
}
```

:

hello1
hello12

.NET Fiddle

while goto , return , break throw .

while :

( ){ ;}

:

```
int i = 0;
while (i++ < 5)
{
```

```
    Console.WriteLine("While is on loop number {0}.", i);
}
```

:

" 1."
"while 2."
" 3."
" 4."
" 5."

.NET Fiddle□□□ □□□ □□

while **Entry Controlled** , . while .

```
bool a = false;

while (a == true)
{
    Console.WriteLine("This will never be printed.");
}
```

`while` false . .

.

```
while (true)
{
//...
}
```

C # .

```
while (true)
{
// ...
}
```

```
for(;;)
{
// ...
}
```

```
{
:label
// ...
goto label;
}
```

while (bool) ( ) . (`a == x '). ,

```
while (AgriculturalService.MoreCornToPick(myFarm.GetAddress()))
```

```
{
    myFarm.PickCorn();
}
```

MSDN : return . . void return .

```
public int Sum(int valueA, int valueB)
{
    return valueA + valueB;
}


public void Terminate(bool terminateEarly)
{
    if (terminateEarly) return; // method returns to caller if true was passed in
    else Console.WriteLine("Not early"); // prints only if terminateEarly was false
}
```

...

in .

a) foreach LINQ .

```
foreach (var member in sequence)
{
    // ...
}
```

b) .

```
public interface IComparer<in T>
{
    // ...
}
```

c) LINQ .

```
var query = from x in source select new { x.Name, x.ID, };
```

~

using statement using directive using statement using using .

1. **using** :

   using IDisposable . using .

2. **using**

   using using msdn . using .

`sealed` modifier .

```
class A { }
sealed class B : A { }
class C : B { } //error : Cannot derive from the sealed class
```

`virtual` ( ) `sealed` () .

```
public class A
{
    public sealed override string ToString() // Virtual method inherited from class Object
    {
        return "Do not override me!";
    }
}

public class B: A
{
    public override string ToString() // Compile time error
    {
        return "An attempt to override";
    }
}
```

.

```
int byteSize = sizeof(byte) // 1
int sbyteSize = sizeof(sbyte) // 1
int shortSize = sizeof(short) // 2
int ushortSize = sizeof(ushort) // 2
int intSize = sizeof(int) // 4
int uintSize = sizeof(uint) // 4
int longSize = sizeof(long) // 8
int ulongSize = sizeof(ulong) // 8
int charSize = sizeof(char) // 2(Unicode)
int floatSize = sizeof(float) // 4
int doubleSize = sizeof(double) // 8
int decimalSize = sizeof(decimal) // 16
int boolSize = sizeof(bool) // 1
```

`static` . (: `DateTime.Now` .

`static` , , , , , .

.

```
class A
{
    static public int count = 0;

    public A()
    {
        count++;
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        A a = new A();
        A b = new A();
        A c = new A();

        Console.WriteLine(A.count); // 3
    }
}
```

count A .

. .

```
class A
{
    static public DateTime InitializationTime;

    // Static constructor
    static A()
    {
        InitializationTime = DateTime.Now;
        // Guaranteed to only run once
        Console.WriteLine(InitializationTime.ToString());
    }
}
```

static class static . static static constructor .static class .

- .
- Object .
- .
-
- .

. .

```
static class ConversionHelper {
    private static double oneGallonPerLitreRate = 0.264172;

    public static double litreToGallonConversion(int litres) {
        return litres * oneGallonPerLitreRate;
    }
}
```

:

```
public static class Functions
{
  public static int Double(int value)
  {
    return value + value;
  }
```

```
}
```

, . . .

C 6 `static using` . .

, `using static using static` :

```
using System;

public class ConsoleApplication
{
    public static void Main()
    {
         Console.WriteLine("Hello World!"); //Writeline is method belonging to static class
Console
    }

}
```

`using static`

```
using static System.Console;

public class ConsoleApplication
{
    public static void Main()
    {
         WriteLine("Hello World!"); //Writeline is method belonging to static class Console
    }

}
```

.

- AppDomain .
- .

int

`int` 32 `System.Int32` . `mscorlib.dll` . `mscorlib.dll` C # .

: -2,147,483,648 ~ 2,147,483,647

```
int int1 = -10007;
var int2 = 2132012521;
```

long 64 . `mscorlib.dll System.Int64` , C # .

long .

```
long long1 = 9223372036854775806;  // explicit declaration, long keyword used
var long2 = -9223372036854775806L; // implicit declaration, 'L' suffix used
```

-9,223,372,036,854,775,808 9,223,372,036,854,775,807 (: int) .

ulong

64 . mscorlib.dll System.UInt64 . mscorlib.dll C # .

: 0 ~ 18,446,744,073,709,551,615

```
ulong veryLargeInt = 18446744073609451315;
var anotherVeryLargeInt = 154467440636094513 15UL;
```

dynamic . dynamic .

```
using System;
using System.Dynamic;

dynamic info = new ExpandoObject();
info.Id = 123;
info.Another = 456;

Console.WriteLine(info.Another);
// 456

Console.WriteLine(info.DoesntExist);
// Throws RuntimeBinderException
```

Newtonsoft Json.NET dynamic JSON .

```
try
{
    string json = @"{ x : 10, y : ""ho""}";
    dynamic deserializedJson = JsonConvert.DeserializeObject(json);
    int x = deserializedJson.x;
    string y = deserializedJson.y;
    // int z = deserializedJson.z; // throws RuntimeBinderException
}
catch (RuntimeBinderException e)
{
    // This exception is thrown when a property
    // that wasn't assigned to a dynamic variable is used
}
```

. . string SayHello .

```
static class StringExtensions
{
    public static string SayHello(this string s) => $"Hello {s}!";
}
```

() .

```
var person = "Person";
Console.WriteLine(person.SayHello());

dynamic manager = "Manager";
Console.WriteLine(manager.SayHello()); // RuntimeBinderException
```

`RuntimeBinderException` . static .

```
var helloManager = StringExtensions.SayHello(manager);
Console.WriteLine(helloManager);
```

, ,

`virtual` , , . (C # .)

```
public class BaseClass
{
    public virtual void Foo()
    {
        Console.WriteLine("Foo from BaseClass");
    }
}
```

`override` . ( )

```
public class DerivedClass: BaseClass
{
    public override void Foo()
    {
        Console.WriteLine("Foo from DerivedClass");
    }
}
```

. .

, `BaseClass` `DerivedClass` .

```
BaseClass obj1 = new BaseClass();
obj1.Foo(); //Outputs "Foo from BaseClass"

obj1 = new DerivedClass();
obj1.Foo(); //Outputs "Foo from DerivedClass"
```

.

```
public class SecondDerivedClass: DerivedClass {}

var obj1 = new SecondDerivedClass();
obj1.Foo(); //Outputs "Foo from DerivedClass"
```

`virtual` .

```
public class BaseClass
{
    public void Foo()
    {
        Console.WriteLine("Foo from BaseClass");
    }
}

public class DerivedClass: BaseClass
{
    public void Foo()
    {
        Console.WriteLine("Foo from DerivedClass");
    }
}

BaseClass obj1 = new BaseClass();
obj1.Foo(); //Outputs "Foo from BaseClass"

obj1 = new DerivedClass();
obj1.Foo(); //Outputs "Foo from BaseClass" too!
```

.

- `BaseClass` ( ) `BaseClass` .
- `DerivedClass` `DerivedClass` .

( ) **CS0108** .

`new` ( ). `new` .

```
public class BaseClass
{
    public void Foo()
    {
        Console.WriteLine("Foo from BaseClass");
    }
}

public class DerivedClass: BaseClass
{
    public new void Foo()
    {
        Console.WriteLine("Foo from DerivedClass");
    }
}

BaseClass obj1 = new BaseClass();
obj1.Foo(); //Outputs "Foo from BaseClass"

obj1 = new DerivedClass();
obj1.Foo(); //Outputs "Foo from BaseClass" too!
```

C ++ `override` .

```
public class A
{
    public virtual void Foo()
    {
    }
}

public class B : A
{
    public void Foo() // Generates CS0108
    {
    }
}
```

`B.Foo()` `A.Foo()` **CS0108** . `override` `new` . .

.

```
public class A
{
    public void Foo()
    {
    }
}

public class B : A
{
    public override void Foo() // Error: Nothing to override
    {
    }
}
```

( ).

```
    public class A
    {
        public void Foo()
        {
            Console.WriteLine("A");
        }
    }

    public class B : A
    {
        public new virtual void Foo()
        {
            Console.WriteLine("B");
```

```
        }
    }
```

B ( ) `Foo()` A `A.Foo()` .

```
A a = new A();
a.Foo(); // Prints "A";
a = new B();
a.Foo(); // Prints "A";
B b = new B();
b.Foo(); // Prints "B";
```

.

C # . `virtual` . . :

```
public class A
{
    private virtual void Foo() // Error: virtual methods cannot be private
    {
    }
}
```

,.

`await` Visual Studio 2012 C # 5.0 . TPL (Task Parallel Library) . `async await` . `await` `await` . `await` `async` .

`void` `async` . .

:

```
public async Task DoSomethingAsync()
{
    Console.WriteLine("Starting a useless process...");
    Stopwatch stopwatch = Stopwatch.StartNew();
    int delay = await UselessProcessAsync(1000);
    stopwatch.Stop();
    Console.WriteLine("A useless process took {0} milliseconds to execute.",
stopwatch.ElapsedMilliseconds);
}

public async Task<int> UselessProcessAsync(int x)
{
    await Task.Delay(x);
    return x;
}
```

:

" ..."

** ... 1 ... **

" 1000 ."

`Task` `Task<T>` `async` `await` .

:

```
public async Task PrintAndDelayAsync(string message, int delay)
{
    Debug.WriteLine(message);
    await Task.Delay(x);
}
```

.

```
public Task PrintAndDelayAsync(string message, int delay)
{
    Debug.WriteLine(message);
    return Task.Delay(x);
}
```

5.0

C # 5.0 `await` `catch` `finally` .

6.0

C # 6.0 `await` `catch` `finally` .

char . 2 . `mscorlib.dll` `System.Char` `System.Char` C # .

.

1. `char c = 'c';`
2. `char c = '\u0063'; //Unicode`
3. `char c = '\x0063'; //Hex`
4. `char c = (char)99;//Integral`

char `ushort`, `int`, `uint`, `long`, `ulong`, `float`, `double`, `decimal` char .

```
ushort u = c;
```

99 .

char . .

```
ushort u = 99;
 char c = (char)u;
```

`lock` .:

```
private static object _lockObj = new object();
static void Main(string[] args)
{
    Task.Run(() => TaskWork());
    Task.Run(() => TaskWork());
    Task.Run(() => TaskWork());

    Console.ReadKey();
}

private static void TaskWork()
{
    lock(_lockObj)
    {
        Console.WriteLine("Entered");

        Task.Delay(3000);
        Console.WriteLine("Done Delaying");

        // Access shared resources safely

        Console.WriteLine("Leaving");
    }
}

Output:

Entered
Done Delaying
Leaving
Entered
Done Delaying
Leaving
Entered
Done Delaying
Leaving
```

:

. (: `_objLock` 2 lock .

`_objLock` `null` (`static`)

lock `Monitor.Enter(_lockObj);` `Monitor.Enter(_lockObj); Monitor.Exit(_lockObj);` . . C #
-

(null) . null null .

`null` null .

, .

```
object a = null;
string b = null;
int? c = null;
List<int> d  = null;
```

null (null) . .

```
int a = null;
float b = null;
decimal c = null;
```

:

- (`new List<int>()`)
- (`""`)
- 0 (`0` , `0f` , `0m`)
- null (`'\0'`)

null / . System.String.IsNullOrEmpty (String) .

```
private void GreetUser(string userName)
{
    if (String.IsNullOrEmpty(userName))
    {
        //The method that called us either sent in an empty string, or they sent us a null
reference. Either way, we need to report the problem.
        throw new InvalidOperationException("userName may not be null or empty.");
    }
    else
    {
        //userName is acceptable.
        Console.WriteLine("Hello, " + userName + "!");
    }
}
```

internal . .

:

```
public class BaseClass
{
    // Only accessible within the same assembly
    internal static int x = 0;
}
```

.

.

.

.

.

.

. .

`where` LINQ : C # .

```
public class Cup<T>
{
    // ...
}
```

T . T .

.

-
-
-
-

`struct` S (`int` , `boolean` ") .

```
public class Cup<T> where T : struct
{
    // ...
}
```

.

```
public class Cup<T> where T : class
{
    // ...
}
```

I

, . `where T : struct or string` ( ) . "... , SByte, Byte, Int16, UInt16, Int32, UInt32, Int64, UInt64, Single, Double, Decimal, DateTime, Char String" `IConvertible` . " IConvertible .

```
public class Cup<T> where T : IConvertible
{
    // ...
}
```

. ( ) .

```
public class Cup<T> where T : new
{
    // ...
}
```

.

```
public class Cup<T> where T : Beverage
{
    // ...
```

```
}


public class Cup<T> where T : IBeer
{
    // ...
}
```

.

```
public class Cup<T, U> where U : T
{
    // ...
}
```

.

```
public class Cup<T> where T : class, new()
{
    // ...
}
```

, , , .

`where` LINQ . SQL `WHERE` .

```
int[] nums = { 5, 2, 1, 3, 9, 8, 6, 7, 2, 0 };

var query =
    from num in nums
    where num < 5
    select num;

    foreach (var n in query)
    {
        Console.Write(n + " ");
    }
    // prints 2 1 3 2 0
```

`extern` . Interop DllImport . `static` .

:

```
using System.Runtime.InteropServices;
public class MyClass
{
    [DllImport("User32.dll")]
    private static extern int SetForegroundWindow(IntPtr point);

    public void ActivateProcessWindow(Process p)
    {
        SetForegroundWindow(p.MainWindowHandle);
    }
}
```

User32.dll SetForegroundWindow .

. .

.

```
/r:GridV1=grid.dll
/r:GridV2=grid20.dll
```

GridV1 GridV2 . extern . :

```
extern alias GridV1;
extern alias GridV2;
```

`true false` . bool System.Boolean .

bool false.

```
bool b; // default value is false
b = true; // true
b = ((5 + 2) == 6); // false
```

bool null bool .

bool ? null

```
bool? a // default value is null
```

`when` **C # 6** .

`when` catch . .

`when catch` , `when true catch` . `catch` `when` .

```
private void CatchException(Action action)
{
    try
    {
        action.Invoke();
    }

    // exception filter
    catch (Exception ex) when (ex.Message.Contains("when"))
    {
        Console.WriteLine("Caught an exception with when");
    }

    catch (Exception ex)
    {
        Console.WriteLine("Caught an exception without when");
    }
}
```

```
private void Method1() { throw new Exception("message for exception with when"); }
private void Method2() { throw new Exception("message for general exception"); }


CatchException(Method1);
CatchException(Method2);
```

unchecked / .

:

```
const int ConstantMax = int.MaxValue;
unchecked
{
    int1 = 2147483647 + 10;
}
int1 = unchecked(ConstantMax + 10);
```

unchecked .

?

/ (:) .

"void" System.Void .

1. .

```
public void DoSomething()
{
    // Do some work, don't return any value to the caller.
}
```

void return . .

```
public void DoSomething()
{
    // Do some work...

    if (condition)
        return;

    // Do some more work if the condition evaluated to false.
}
```

2. .

, . type* identifier . . int* myInt . void* identifier .

void Microsoft .

if, if ... else, if ... else if

if . if Boolean .

braces {} .

```
int a = 4;
if(a % 2 == 0)
{
     Console.WriteLine("a contains an even number");
}
// output: "a contains an even number"
```

---

if false else .

```
int a = 5;
if(a % 2 == 0)
{
     Console.WriteLine("a contains an even number");
}
else
{
     Console.WriteLine("a contains an odd number");
}
// output: "a contains an odd number"
```

---

if ... else if .

```
int a = 9;
if(a % 2 == 0)
{
     Console.WriteLine("a contains an even number");
}
else if(a % 3 == 0)
{
     Console.WriteLine("a contains an odd number that is a multiple of 3");
}
else
{
     Console.WriteLine("a contains an odd number");
}
// output: "a contains an odd number that is a multiple of 3"
```

, construct.So , ..

C # . .

```
if (someBooleanMethodWithSideEffects() && someOtherBooleanMethodWithSideEffects()) {
  //...
}
```

someOtherBooleanMethodWithSideEffects .

"" . :

```
if (someCollection != null && someCollection.Count > 0) {
   // ..
}
```

.

```
if (someCollection.Count > 0 && someCollection != null) {
```

someCollection null , NullReferenceException .

do false . do-while goto , return , break throw .

do .

do { ; } while ( );

:

```
int i = 0;

do
{
    Console.WriteLine("Do is on loop number {0}.", i);
} while (i++ < 5);
```

:

" 1 ."
"Do 2 ."
" 3 ."
" 4 ."
" 5 ."

while do-while **Exit Controlled** . , do-while .

```
bool a = false;

do
{
    Console.WriteLine("This will be printed once, even if a is false.");
} while (a == true);
```

( ) public static operator .

, 2 .

.

.

```
public struct Vector32
```

```
{

    public Vector32(int x, int y)
    {
        X = x;
        Y = y;
    }

    public int X { get; }
    public int Y { get; }

    public static bool operator ==(Vector32 left, Vector32 right)
        => left.X == right.X && left.Y == right.Y;

    public static bool operator !=(Vector32 left, Vector32 right)
        => !(left == right);

    public static Vector32 operator +(Vector32 left, Vector32 right)
        => new Vector32(left.X + right.X, left.Y + right.Y);

    public static Vector32 operator +(Vector32 left, int right)
        => new Vector32(left.X + right, left.Y + right);

    public static Vector32 operator +(int left, Vector32 right)
        => right + left;

    public static Vector32 operator -(Vector32 left, Vector32 right)
        => new Vector32(left.X - right.X, left.Y - right.Y);

    public static Vector32 operator -(Vector32 left, int right)
        => new Vector32(left.X - right, left.Y - right);

    public static Vector32 operator -(int left, Vector32 right)
        => right - left;

    public static implicit operator Vector64(Vector32 vector)
        => new Vector64(vector.X, vector.Y);

    public override string ToString() => $"{{{X}, {Y}}}";

}

public struct Vector64
{

    public Vector64(long x, long y)
    {
        X = x;
        Y = y;
    }

    public long X { get; }
    public long Y { get; }

    public override string ToString() => $"{{{X}, {Y}}}";

}
```

```
var vector1 = new Vector32(15, 39);
var vector2 = new Vector32(87, 64);
```

```
Console.WriteLine(vector1 == vector2); // false
Console.WriteLine(vector1 != vector2); // true
Console.WriteLine(vector1 + vector2);  // {102, 103}
Console.WriteLine(vector1 - vector2);  // {-72, -25}
```

`struct` (:).

.

```
using static System.Console;

namespace ConsoleApplication1
{
    struct Point
    {
        public int X;
        public int Y;

        public override string ToString()
        {
            return $"X = {X}, Y = {Y}";
        }

        public void Display(string name)
        {
            WriteLine(name + ": " + ToString());
        }
    }

    class Program
    {
        static void Main()
        {
            var point1 = new Point {X = 10, Y = 20};
            // it's not a reference but value type
            var point2 = point1;
            point2.X = 777;
            point2.Y = 888;
            point1.Display(nameof(point1)); // point1: X = 10, Y = 20
            point2.Display(nameof(point2)); // point2: X = 777, Y = 888

            ReadKey();
        }
    }
}
```

,,,,,,, . .

MS :

.

.

- (int, double) .
-

16 .

- .
- .

switch . switch . case . default ( ) default . C # - . case case . case . month 12 case 12 1 2 case 2 . case case break . .

```
int month = DateTime.Now.Month; // this is expected to be 1-12 for Jan-Dec

switch (month)
{
    case 12:
    case 1:
    case 2:
        Console.WriteLine("Winter");
        break;
    case 3:
    case 4:
    case 5:
        Console.WriteLine("Spring");
        break;
    case 6:
    case 7:
    case 8:
        Console.WriteLine("Summer");
        break;
    case 9:
    case 10:
    case 11:
        Console.WriteLine("Autumn");
        break;
    default:
        Console.WriteLine("Incorrect month index");
        break;
}
```

case (: 1 , "str" , Enum.A) variable case const Enum ( ).

interface , . .

interface .

```
interface IProduct
{
    decimal Price { get; }
}

class Product : IProduct
{
    const decimal vat = 0.2M;

    public Product(decimal price)
    {
        _price = price;
    }

    private decimal _price;
```

```
    public decimal Price { get { return _price * (1 + vat); } }
}
```

unsafe .

C # . , , C .

. C # . CLR .

(:) (: interop).

```
// compile with /unsafe
class UnsafeTest
{
   unsafe static void SquarePtrParam(int* p)
   {
      *p *= *p; // the '*' dereferences the pointer.
      //Since we passed in "the address of i", this becomes "i *= i"
   }

   unsafe static void Main()
   {
      int i = 5;
      // Unsafe method: uses address-of operator (&):
      SquarePtrParam(&i); // "&i" means "the address of i". The behavior is similar to "ref i"
      Console.WriteLine(i); // Output: 25
   }
}
```

. (CS0212). " " .

```
void Main()
{
    int[] intArray = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

    UnsafeSquareArray(intArray);
    foreach(int i in intArray)
        Console.WriteLine(i);
}

unsafe static void UnsafeSquareArray(int[] pArr)
{
    int len = pArr.Length;

    //in C or C++, we could say
    // int* a = &(pArr[0])
    // however, C# requires you to "fix" the variable first
    fixed(int* fixedPointer = &(pArr[0]))
    {
        //Declare a new int pointer because "fixedPointer" cannot be written to.
        // "p" points to the same address space, but we can modify it
        int* p = fixedPointer;

        for (int i = 0; i < len; i++)
        {
            *p *= *p; //square the value, just like we did in SquarePtrParam, above
            p++;      //move the pointer to the next memory space.
```

```
                        // NOTE that the pointer will move 4 bytes since "p" is an
                        // int pointer and an int takes 4 bytes

            //the above 2 lines could be written as one, like this:
            // "*p *= *p++;"
        }
    }
}
```

:

```
1
4
9
16
25
36
49
64
81
100
```

`unsafe` stackalloc C _alloca . `stackalloc` .

```
unsafe void Main()
{
    const int len=10;
    int* seedArray = stackalloc int[len];

    //We can no longer use the initializer "{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}" as before.
    // We have at least 2 options to populate the array. The end result of either
    // option will be the same (doing both will also be the same here).

    //FIRST OPTION:
    int* p = seedArray; // we don't want to lose where the array starts, so we
                        // create a shadow copy of the pointer
    for(int i=1; i<=len; i++)
        *p++ = i;
    //end of first option

    //SECOND OPTION:
    for(int i=0; i<len; i++)
        seedArray[i] = i+1;
    //end of second option

    UnsafeSquareArray(seedArray, len);
    for(int i=0; i< len; i++)
        Console.WriteLine(seedArray[i]);
}

//Now that we are dealing directly in pointers, we don't need to mess around with
// "fixed", which dramatically simplifies the code
unsafe static void UnsafeSquareArray(int* p, int len)
{
    for (int i = 0; i < len; i++)
        *p *= *p++;
}
```

( )

implicit . , double Fraction Fraction int .

```
class Fraction(int numerator, int denominator)
{
    public int Numerator { get; } = numerator;
    public int Denominator { get; } = denominator;
    // ...
    public static implicit operator double(Fraction f)
    {
        return f.Numerator / (double) f.Denominator;
    }
    public static implicit operator Fraction(int i)
    {
        return new Fraction(i, 1);
    }
}
```

true false .

1.

```
var myTrueBool = true;
var myFalseBool = false;
```

2.

```
public static bool operator true(MyClass x)
{
    return x.value >= 0;
}

public static bool operator false(MyClass x)
{
    return x.value < 0;
}
```

false Nullable C # 2.0 .

true false .

string ( ) .NET System.String .

:

```
string a = "Hello";
var b = "world";
var f = new string(new []{ 'h', 'i', '!' }); // hi!
```

UTF-16 . , 2 .

16 . ushort System.UInt16 2 .

0 ~ 65535 .

```
ushort a = 50; // 50
ushort b = 65536; // Error, cannot be converted
ushort c = unchecked((ushort)65536); // Overflows (wraps around to 0)
```

8 . `sbyte` `System.SByte` 1 . `byte` .

`-127` `127` ( ).

```
sbyte a = 127; // 127
sbyte b = -127; // -127
sbyte c = 200; // Error, cannot be converted
sbyte d = unchecked((sbyte)129); // -127 (overflows)
```

var

. .

```
var i = 10; // implicitly typed, the compiler must determine what type of variable this is
int i = 10; // explicitly typed, the type of variable is explicitly stated to the compiler

// Note that these both represent the same type of variable (int) with the same value (10).
```

. **var** .

```
var i;
i = 10;

// This code will not run as it is not initialized upon declaration.
```

var . . .

```
var a = new { number = 1, text = "hi" };
```

LINQ

```
public class Dog
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public class DogWithBreed
{
    public Dog Dog { get; set; }
    public string BreedName  { get; set; }
}

public void GetDogsWithBreedNames()
{
    var db = new DogDataContext(ConnectString);
    var result = from d in db.Dogs
                 join b in db.Breeds on d.BreedId equals b.BreedId
                 select new
```

```
                    {
                        DogName = d.Name,
                        BreedName = b.BreedName
                    };

    DoStuff(result);
}
```

foreach var .

```
public bool hasItemInList(List<String> list, string stringToSearch)
{
    foreach(var item in list)
    {
        if( ( (string)item ).equals(stringToSearch) )
            return true;
    }

    return false;
}
```

. .

, , .

```
class DelegateExample
{
    public void Run()
    {
        //using class method
        InvokeDelegate( WriteToConsole );

        //using anonymous method
        DelegateInvoker di = delegate ( string input )
        {
            Console.WriteLine( string.Format( "di: {0} ", input ) );
            return true;
        };
        InvokeDelegate( di );

        //using lambda expression
        InvokeDelegate( input => false );
    }

    public delegate bool DelegateInvoker( string input );

    public void InvokeDelegate(DelegateInvoker func)
    {
        var ret = func( "hello world" );
        Console.WriteLine( string.Format( " > delegate returned {0}", ret ) );
    }

    public bool WriteToConsole( string input )
    {
        Console.WriteLine( string.Format( "WriteToConsole: '{0}'", input ) );
        return true;
    }
}
```

. " .

.

event .

```
public class Server
{
    // defines the event
    public event EventHandler DataChangeEvent;

    void RaiseEvent()
    {
        var ev = DataChangeEvent;
        if(ev != null)
        {
            ev(this, EventArgs.Empty);
        }
    }
}

public class Client
{
    public void Client(Server server)
    {
        // client subscribes to the server's DataChangeEvent
        server.DataChangeEvent += server_DataChanged;
    }

    private void server_DataChanged(object sender, EventArgs args)
    {
        // notified when the server raises the DataChangeEvent
    }
}
```

MSDN

partial class, struct interface . .

File1.cs

```
namespace A
{
    public partial class Test
    {
        public string Var1 {get;set;}
    }
}
```

File2.cs

```
namespace A
{
    public partial class Test
    {
        public string Var2 {get;set;}
```

```
    }
}
```

: . .

`partial` . .

. . . .

- .
- void .
- . .

- MSDN

File1.cs

```
namespace A
{
    public partial class Test
    {
        public string Var1 {get;set;}
        public partial Method1(string str);
    }
}
```

File2.cs

```
namespace A
{
    public partial class Test
    {
        public string Var2 {get;set;}
        public partial Method1(string str)
        {
            Console.WriteLine(str);
        }
    }
}
```

: .

: https://riptutorial.com/ko/csharp/topic/26/

154:

- `myTimer.Interval` - "Tick" ( ).
- `myTimer.Enabled` - / .
- `myTimer.Start()` - .
- `myTimer.Stop()` - .

Visual Studio .

Examples

`System.Threading.Timer` - . .

: DataWrite . 5 "multithread executed ..." Enter 1 .

```
using System;
using System.Threading;
class Program
{
  static void Main()
  {
    // First interval = 5000ms; subsequent intervals = 1000ms
    Timer timer = new Timer (DataWrite, "multithread executed...", 5000, 1000);
    Console.ReadLine();
    timer.Dispose(); // This both stops the timer and cleans up.
  }

  static void DataWrite (object data)
  {
    // This runs on a pooled thread
    Console.WriteLine (data); // Writes "multithread executed..."
  }
}
```

: .

`Change` - .

`Timeout.Infinite` - . .

`System.Timers` - .NET Framework . `System.Threading.Timer` .

:

- `IComponent` - Visual Studio
- `Change` `Interval`
- `delegate` `Elapsed` `event`
- `Enabled` (`default value = false`)
- `Enabled` ( ) `Enabled` `Start` `Stop`
- `AutoReset` - (`default value = true`).

- WPF Windows Forms `Invoke` `BeginInvoke` `SynchronizingObject`

:

```
using System;
using System.Timers; // Timers namespace rather than Threading
class SystemTimer
{
  static void Main()
  {
    Timer timer = new Timer(); // Doesn't require any args
    timer.Interval = 500;
    timer.Elapsed += timer_Elapsed; // Uses an event instead of a delegate
    timer.Start(); // Start the timer
    Console.ReadLine();
    timer.Stop(); // Stop the timer
    Console.ReadLine();
    timer.Start(); // Restart the timer
    Console.ReadLine();
    timer.Dispose(); // Permanently stop the timer
 }

 static void timer_Elapsed(object sender, EventArgs e)
 {
   Console.WriteLine ("Tick");
 }
}
```

`Multithreaded timers` - . , `Elapsed` .

`Elapsed` - `Elapsed` . . OS 10-20ms.

`interop` - Windows . 1ms `winmm.dll` `winmm.dll` .

`timeBeginPeriod` - OS `timeBeginPeriod` .

`timeSetEvent` - `timeBeginPeriod` .

`timeKillEvent` - this .

`timeEndPeriod` - OS .

`dllimport winmm.dll timesetevent` .

(Y). .

: WinForms . WPF `DispatcherTimer` .

```
    using System.Windows.Forms; //Timers use the Windows.Forms namespace

    public partial class Form1 : Form
    {

        Timer myTimer = new Timer(); //create an instance of Timer named myTimer
```

```
        public Form1()
        {
            InitializeComponent();
        }

    }
```

"Tick" Timer

"Tick" .

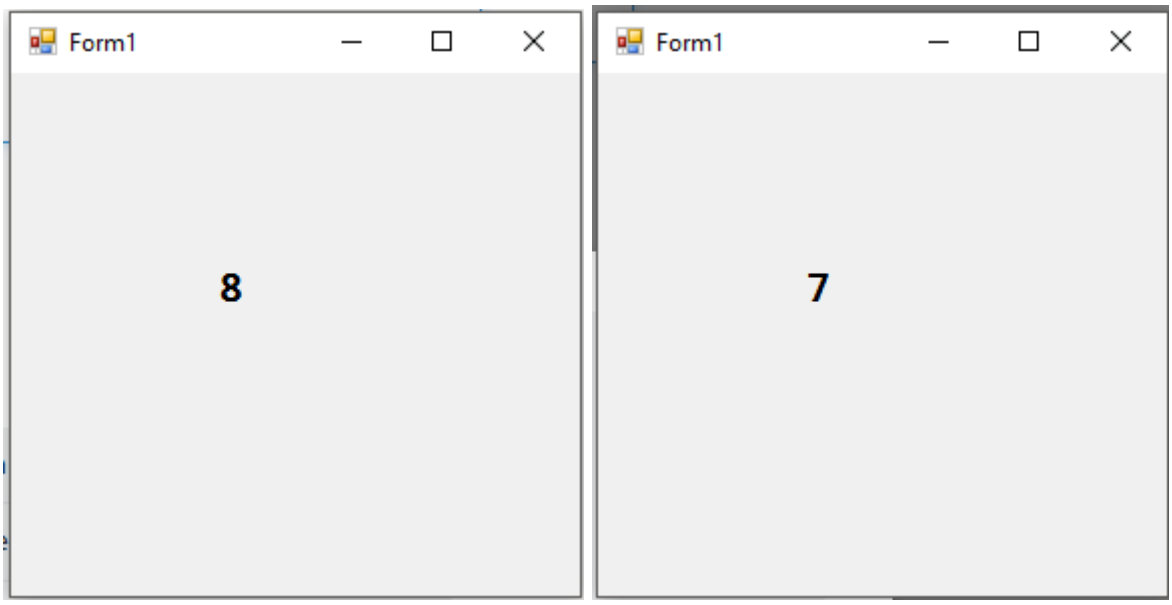
```
public partial class Form1 : Form
{

    Timer myTimer = new Timer();


    public Form1()
    {
        InitializeComponent();

        myTimer.Tick += myTimer_Tick; //assign the event handler named "myTimer_Tick"
    }

    private void myTimer_Tick(object sender, EventArgs e)
    {
        // Perform your actions here.
    }
}
```

: .

```
    public partial class Form1 : Form
    {

    Timer myTimer = new Timer();
    int timeLeft = 10;

        public Form1()
        {
            InitializeComponent();

            //set properties for the Timer
            myTimer.Interval = 1000;
            myTimer.Enabled = true;

            //Set the event handler for the timer, named "myTimer_Tick"
            myTimer.Tick += myTimer_Tick;

            //Start the timer as soon as the form is loaded
            myTimer.Start();

            //Show the time set in the "timeLeft" variable
            lblCountDown.Text = timeLeft.ToString();

        }
```

```
        private void myTimer_Tick(object sender, EventArgs e)
        {
            //perform these actions at the interval set in the properties.
            lblCountDown.Text = timeLeft.ToString();
            timeLeft -= 1;

            if (timeLeft < 0)
            {
                myTimer.Stop();
            }
        }
    }
```

...



...

: https://riptutorial.com/ko/csharp/topic/3829/

155:

Examples

`Tuple<T1>` - `Tuple<T1,T2,T3,T4,T5,T6,T7,T8>` . 1 - 8 . .

```
// tuple with 4 elements
var tuple = new Tuple<string, int, bool, MyClass>("foo", 123, true, new MyClass());
```

`Tuple.Create` . C # .

```
// tuple with 4 elements
var tuple = Tuple.Create("foo", 123, true, new MyClass());
```

7.0

C # 7.0, ValueTuple .

```
var tuple = ("foo", 123, true, new MyClass());
```

.

```
(int number, bool flag, MyClass instance) tuple = (123, true, new MyClass());
```

`Item1` - `Item8` . (, `Tuple<T1,T2> Item3`).

```
var tuple = new Tuple<string, int, bool, MyClass>("foo", 123, true, new MyClass());
var item1 = tuple.Item1; // "foo"
var item2 = tuple.Item2; // 123
var item3 = tuple.Item3; // true
var item4 = tuple.Item4; // new My Class()
```

.

, `Tuple` .

```
List<Tuple<int, string>> list = new List<Tuple<int, string>>();
list.Add(new Tuple<int, string>(2, "foo"));
list.Add(new Tuple<int, string>(1, "bar"));
list.Add(new Tuple<int, string>(3, "qux"));

list.Sort((a, b) => a.Item2.CompareTo(b.Item2)); //sort based on the string element

foreach (var element in list) {
    Console.WriteLine(element);
}

// Output:
// (1, bar)
// (2, foo)
```

```
// (3, qux)
```

:

```
list.Sort((a, b) => b.Item2.CompareTo(a.Item2));
```

. `AddMultiply` (sum, product) .

```
void Write()
{
    var result = AddMultiply(25, 28);
    Console.WriteLine(result.Item1);
    Console.WriteLine(result.Item2);
}

Tuple<int, int> AddMultiply(int a, int b)
{
    return new Tuple<int, int>(a + b, a * b);
}
```

:

53
700

C # 7.0 . `ValueTuple` .

: https://riptutorial.com/ko/csharp/topic/838/

156: I / O

.

- `new System.IO.StreamWriter(string path)`
- `new System.IO.StreamWriter(string path, bool append)`
- `System.IO.StreamWriter.WriteLine(string text)`
- `System.IO.StreamWriter.WriteAsync(string text)`
- `System.IO.Stream.Close()`
- `System.IO.File.ReadAllText(string path)`
- `System.IO.File.ReadAllLines(string path)`
- `System.IO.File.ReadLines(string path)`
- `System.IO.File.WriteAllText(string path, string text)`
- `System.IO.File.WriteAllLines(string path, IEnumerable<string> contents)`
- `System.IO.File.Copy(string source, string dest)`
- `System.IO.File.Create(string path)`
- `System.IO.File.Delete(string path)`
- `System.IO.File.Move(string source, string dest)`
- `System.IO.Directory.GetFiles(string path)`

	.
	true (), false .
	.
	.
	.
	.

- `Stream` . `using` `using myStream.Close()` `using` .
- .
- `@"C:\MyFolder\MyFile.txt"` `@"C:\MyFolder\MyFile.txt"`

Examples

System.IO.File

System.IO.File.ReadAllText .

```
string text = System.IO.File.ReadAllText(@"C:\MyFolder\MyTextFile.txt");
```

System.IO.File.ReadAllLines .

```
string[] lines = System.IO.File.ReadAllLines(@"C:\MyFolder\MyTextFile.txt");
```

System.IO.StreamWriter

System.IO.StreamWriter .

TextWriter .

`WriteLine` .

StreamWriter `using using` .

```
string[] lines = { "My first string", "My second string", "and even a third string" };
using (System.IO.StreamWriter sw = new System.IO.StreamWriter(@"C:\MyFolder\OutputText.txt"))
{
    foreach (string line in lines)
    {
        sw.WriteLine(line);
    }
}
```

StreamWriter `bool` `Append` .

```
bool appendExistingFile = true;
using (System.IO.StreamWriter sw = new System.IO.StreamWriter(@"C:\MyFolder\OutputText.txt",
appendExistingFile ))
{
    sw.WriteLine("This line will be appended to the existing file");
}
```

System.IO.File

System.IO.File.WriteAllText .

```
string text = "String that will be stored in the file";
System.IO.File.WriteAllText(@"C:\MyFolder\OutputFile.txt", text);
```

`IEnumerable<String>` **System.IO.File.WriteAllLines** ( ). .

```
string[] lines = { "My first string", "My second string", "and even a third string" };
System.IO.File.WriteAllLines(@"C:\MyFolder\OutputFile.txt", lines);
```

IEnumerable

`System.IO.File.ReadLines` `IEnumerable<string>` . `System.IO.File.ReadAllLines` .

```
IEnumerable<string> AllLines = File.ReadLines("file_name.txt", Encoding.Default);
```

File.ReadLines . .

`ToArray` , `ToList` `ReadLines` . `foreach` LINQ `IEnumerable` .

File Create File . FileStream . .

ex1 :

```
var fileStream1 = File.Create("samplePath");
/// you can write to the fileStream1
fileStream1.Close();
```

ex2 :

```
using(var fileStream1 = File.Create("samplePath"))
{
    /// you can write to the fileStream1
}
```

ex3 :

```
File.Create("samplePath").Close();
```

FileStream

. .

```
var fileStream2 = new FileStream("samplePath", FileMode.OpenOrCreate, FileAccess.ReadWrite,
FileShare.None);
```

FileMode , FileAccess FileShare . :

FileMode : Answers " ? ? .

FileAccess : Answers " ? ?" .

FileShare : Answers " ?" .

File .

```
File.Copy(@"sourcePath\abc.txt", @"destinationPath\abc.txt");
File.Copy(@"sourcePath\abc.txt", @"destinationPath\xyz.txt");
```

: . . .

.

```
File.Move(@"sourcePath\abc.txt", @"destinationPath\xyz.txt");
```

Remark1 : ( ). .

2 : .

```
string path = @"c:\path\to\file.txt";
```

```
File.Delete(path);
```

`Delete` throw . try-catch `Delete` . lock .

```
 var FileSearchRes = Directory.GetFiles(@Path, "*.*", SearchOption.AllDirectories);
```

`FileInfo` .

```
 var FileSearchRes = Directory.GetFiles(@Path, "*.pdf", SearchOption.AllDirectories);
```

`FileInfo` .

StreamWriter

```
// filename is a string with the full path
// true is to append
using (System.IO.StreamWriter file = new System.IO.StreamWriter(filename, true))
{
   // Can write either a string or char array
   await file.WriteAsync(text);
}
```

I / O : https://riptutorial.com/ko/csharp/topic/4266/---i---o

157:

Examples

C

C # . . . , 2 . . .

```
object a = new object();
object b = a;
System.Object.ReferenceEquals(a, b);  //returns true
```

(==) true false . == true . == .

```
// Numeric equality: True
Console.WriteLine((2 + 2) == 4);

// Reference equality: different objects,
// same boxed value: False.
object s = 1;
object t = 1;
Console.WriteLine(s == t);

// Define some strings:
string a = "hello";
string b = String.Copy(a);
string c = "hello";

// Compare string values of a constant and an instance: True
Console.WriteLine(a == b);

// Compare string references;
// a is a constant but b is an instance: False.
Console.WriteLine((object)a == (object)b);

// Compare string references, both constants
// have the same value, so string interning
// points to same reference: True.
Console.WriteLine((object)a == (object)c);
```

: https://riptutorial.com/ko/csharp/topic/1491/-

158:

unsafe

. unsafe .

System.IntPtr void* . void* .

C C++ . .

C C++ C# . T .

- T .
- T .

Examples

C# C .

```
unsafe
{
    var buffer = new int[1024];
    fixed (int* p = &buffer[0])
    {
        for (var i = 0; i < buffer.Length; i++)
        {
            *(p + i) = i;
        }
    }
}
```

C# unsafe .

fixed C# . .

. .

, int (System.Int32) 4. int 0 int 4 . :

```
var ptr = (int*)IntPtr.Zero;
Console.WriteLine(new IntPtr(ptr)); // prints 0
ptr++;
Console.WriteLine(new IntPtr(ptr)); // prints 4
ptr++;
Console.WriteLine(new IntPtr(ptr)); // prints 8
```

long (System.Int64) 8. long 0 long 8 . :

```
var ptr = (long*)IntPtr.Zero;
Console.WriteLine(new IntPtr(ptr)); // prints 0
ptr++;
Console.WriteLine(new IntPtr(ptr)); // prints 8
ptr++;
Console.WriteLine(new IntPtr(ptr)); // prints 16
```

`void` `void` catch-all . Size-Agnostic `void` .

```
var ptr = (void*)IntPtr.Zero;
Console.WriteLine(new IntPtr(ptr));
ptr++; // compile-time error
Console.WriteLine(new IntPtr(ptr));
ptr++; // compile-time error
Console.WriteLine(new IntPtr(ptr));
```

.

C C ++ (*) . C # .

C, C ++ C # `int` .

```
int* a;
```

C C ++ `int` `int` . C # `int` .

```
int* a, b;
```

C C ++ `int` . C # .

```
int *a, *b;
```

*

C # C C ++ `void*` .

```
void* ptr;
```

`void*` .

```
int* p1 = (int*)IntPtr.Zero;
void* ptr = p1;
```

.

```
int* p1 = (int*)IntPtr.Zero;
void* ptr = p1;
int* p2 = (int*)ptr;
```

->

C # C C ++ . -> .

.

```
struct Vector2
{
    public int X;
    public int Y;
}
```

-> .

```
Vector2 v;
v.X = 5;
v.Y = 10;

Vector2* ptr = &v;
int x = ptr->X;
int y = ptr->Y;
string s = ptr->ToString();

Console.WriteLine(x); // prints 5
Console.WriteLine(y); // prints 10
Console.WriteLine(s); // prints Vector2
```

( ) . .

```
void P<T>(T obj)
    where T : struct
{
    T* ptr = &obj; // compile-time error
}
```

: https://riptutorial.com/ko/csharp/topic/5524/

159:

Examples

C # `unsafe` . .

, . .

.

```
type *var-name;
```

.

```
int    *ip;    /* pointer to an integer */
double *dp;    /* pointer to a double */
float  *fp;    /* pointer to a float */
char   *ch     /* pointer to a character */
```

C # .

```
using System;
namespace UnsafeCodeApplication
{
   class Program
   {
      static unsafe void Main(string[] args)
      {
         int var = 20;
         int* p = &var;
         Console.WriteLine("Data is: {0} ",  var);
         Console.WriteLine("Address is: {0}",  (int)p);
         Console.ReadKey();
      }
   }
}
```

.

```
Data is: 20
Address is: 99215364
```

.

```
// safe code
unsafe
{
    // you can use pointers here
}
// safe code
```

ToString () . .

```
using System;
namespace UnsafeCodeApplication
{
   class Program
   {
      public static void Main()
      {
         unsafe
         {
            int var = 20;
            int* p = &var;
            Console.WriteLine("Data is: {0} " , var);
            Console.WriteLine("Data is: {0} " , p->ToString());
            Console.WriteLine("Address is: {0} " , (int)p);
         }

         Console.ReadKey();
      }
   }
}
```

.

```
Data is: 20
Data is: 20
Address is: 77128984
```

. .

```
using System;
namespace UnsafeCodeApplication
{
   class TestPointer
   {
      public unsafe void swap(int* p, int *q)
      {
         int temp = *p;
         *p = *q;
         *q = temp;
      }

      public unsafe static void Main()
      {
         TestPointer p = new TestPointer();
         int var1 = 10;
         int var2 = 20;
         int* x = &var1;
         int* y = &var2;

         Console.WriteLine("Before Swap: var1:{0}, var2: {1}", var1, var2);
         p.swap(x, y);

         Console.WriteLine("After Swap: var1:{0}, var2: {1}", var1, var2);
         Console.ReadKey();
      }
   }
```

```
}
```

.

```
Before Swap: var1: 10, var2: 20
After Swap: var1: 20, var2: 10
```

C # . `int *p int[] p` . p .

C C ++ fixed .

.

```
using System;
namespace UnsafeCodeApplication
{
   class TestPointer
   {
      public unsafe static void Main()
      {
         int[]  list = {10, 100, 200};
         fixed(int *ptr = list)

         /* let us have array address in pointer */
         for ( int i = 0; i < 3; i++)
         {
            Console.WriteLine("Address of list[{0}]={1}",i,(int)(ptr + i));
            Console.WriteLine("Value of list[{0}]={1}", i, *(ptr + i));
         }

         Console.ReadKey();
      }
   }
}
```

.

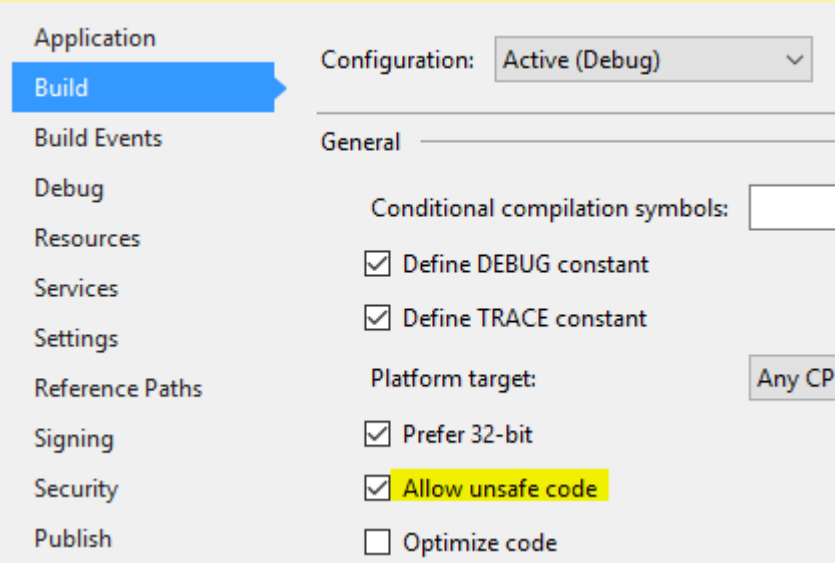
```
Address of list[0] = 31627168
Value of list[0] = 10
Address of list[1] = 31627172
Value of list[1] = 100
Address of list[2] = 31627176
Value of list[2] = 200
```

`/unsafe` .

prog1.cs .

```
csc /unsafe prog1.cs
```

Visual Studio IDE .



:

- .
- .
- " " .

: https://riptutorial.com/ko/csharp/topic/5514/----

160:

. , . Lambda , (:) . .

- <TDelegate> = lambdaExpression;

TDelegate	
lambdaExpression	(: `num => num < 5`)

" " . `decimal CalculateTotalTaxDue(SalesOrder order)` . .NET `decimal taxDue = CalculateTotalTaxDue(order);` . `decimal taxDue = CalculateTotalTaxDue(order);` . (SQL, XML,) ? ! . .

. ( ) .

. . `CalculateTotalTaxDue` .

1.
2. .
3. .
4. .
5. .

.

. .NET ?

MSIL . (MSIL .NET .) MSIL . .

▪

. . `MethodCallExpression` 1) `MethodInfo` , 2) `Expression` , 3) , `Expression` . " " .

. C # . . . `Delegate` (`Action` `Func`) . `LambdaExpression` (`LambdaExpression` `Expression<Action<T>>` `Expression<Func<T>>`) `LambdaExpression` . . *expression tree API* `LambdaExpression` .

. Expressions API . API API `CalculateTotalSalesTax` .

: . (,) `=>` . C # `Delegate` `Expression` . *`LambdaExpression`* (, PascalCase) Expression API .

LINQ

LINQ . LINQ . , LINQ to Entity Framework SQL .

LINQ .

1. : `products.Where(x => x.Cost > 5)`
2. " Cost 5 " .
3. SQL . `SELECT * FROM products WHERE Cost > 5`
4. ORM POCO .

- . (`ExpressionVisitor` `ExpressionVisitor`) .

Examples

API

```
using System.Linq.Expressions;

// Manually build the expression tree for
// the lambda expression num => num < 5.
ParameterExpression numParam = Expression.Parameter(typeof(int), "num");
ConstantExpression five = Expression.Constant(5, typeof(int));
BinaryExpression numLessThanFive = Expression.LessThan(numParam, five);
Expression<Func<int, bool>> lambda1 =
    Expression.Lambda<Func<int, bool>>(
        numLessThanFive,
        new ParameterExpression[] { numParam });
```

```
// Define an expression tree, taking an integer, returning a bool.
Expression<Func<int, bool>> expr = num => num < 5;

// Call the Compile method on the expression tree to return a delegate that can be called.
Func<int, bool> result = expr.Compile();

// Invoke the delegate and write the result to the console.
Console.WriteLine(result(4)); // Prints true

// Prints True.

// You can also combine the compile step with the call/invoke step as below:
Console.WriteLine(expr.Compile()(4));
```

```
using System.Linq.Expressions;

// Create an expression tree.
Expression<Func<int, bool>> exprTree = num => num < 5;

// Decompose the expression tree.
ParameterExpression param = (ParameterExpression)exprTree.Parameters[0];
```

```
BinaryExpression operation = (BinaryExpression)exprTree.Body;
ParameterExpression left = (ParameterExpression)operation.Left;
ConstantExpression right = (ConstantExpression)operation.Right;

Console.WriteLine("Decomposed expression: {0} => {1} {2} {3}",
                  param.Name, left.Name, operation.NodeType, right.Value);

// Decomposed expression: num => num LessThan 5
```

.

```
Expression<Func<int, bool>> lambda = num => num == 42;
```

' ' Expression .

.

```
ParameterExpression parameter = Expression.Parameter(typeof(int), "num"); // num argument
ConstantExpression constant = Expression.Constant(42, typeof(int)); // 42 constant
BinaryExpression equality = Expression.Equals(parameter, constant); // equality of two
expressions (num == 42)
Expression<Func<int, bool>> lambda = Expression.Lambda<Func<int, bool>>(equality, parameter);
```

API

API `CalculateSalesTax` `CalculateSalesTax` . .

1. .
2. .
3. 0 .

```
//For reference, we're using the API to build this lambda expression
    orderLine => orderLine.IsTaxable ? orderLine.Total * orderLine.Order.TaxRate : 0;

//The orderLine parameter we pass in to the method.  We specify it's type (OrderLine) and the
name of the parameter.
    ParameterExpression orderLine = Expression.Parameter(typeof(OrderLine), "orderLine");

//Check if the parameter is taxable;  First we need to access the is taxable property, then
check if it's true
    PropertyInfo isTaxableAccessor = typeof(OrderLine).GetProperty("IsTaxable");
    MemberExpression getIsTaxable = Expression.MakeMemberAccess(orderLine, isTaxableAccessor);
    UnaryExpression isLineTaxable = Expression.IsTrue(getIsTaxable);

//Before creating the if, we need to create the braches
    //If the line is taxable, we'll return the total times the tax rate; get the total and tax
rate, then multiply
    //Get the total
    PropertyInfo totalAccessor = typeof(OrderLine).GetProperty("Total");
    MemberExpression getTotal = Expression.MakeMemberAccess(orderLine, totalAccessor);

    //Get the order
    PropertyInfo orderAccessor = typeof(OrderLine).GetProperty("Order");
    MemberExpression getOrder = Expression.MakeMemberAccess(orderLine, orderAccessor);
```

```
    //Get the tax rate - notice that we pass the getOrder expression directly to the member
access
    PropertyInfo taxRateAccessor = typeof(Order).GetProperty("TaxRate");
    MemberExpression getTaxRate = Expression.MakeMemberAccess(getOrder, taxRateAccessor);

    //Multiply the two - notice we pass the two operand expressions directly to multiply
    BinaryExpression multiplyTotalByRate = Expression.Multiply(getTotal, getTaxRate);

//If the line is not taxable, we'll return a constant value - 0.0 (decimal)
    ConstantExpression zero = Expression.Constant(0M);

//Create the actual if check and branches
    ConditionalExpression ifTaxableTernary = Expression.Condition(isLineTaxable,
multiplyTotalByRate, zero);

//Wrap the whole thing up in a "method" - a LambdaExpression
    Expression<Func<OrderLine, decimal>> method = Expression.Lambda<Func<OrderLine,
decimal>>(ifTaxableTernary, orderLine);
```

, .

, LINQ . .

(DLR) .NET Framework Microsoft (MSIL) .

1. ,
2. System.Linq.Expressions .

.

C # num => num <5 .

```
Expression<Func<int, bool>> lambda = num => num < 5;
```

API

. .

.

1. ParameterExpression
2. MethodCallExpression

API num => num <5 .

```
ParameterExpression numParam = Expression.Parameter(typeof(int), "num");
ConstantExpression five = Expression.Constant(5, typeof(int));
BinaryExpression numLessThanFive = Expression.LessThan(numParam, five);
Expression<Func<int, bool>> lambda1 = Expression.Lambda<Func<int, bool>>(numLessThanFive,new
ParameterExpression[] { numParam });
```

ExpressionVisitor .

```
class PrintingVisitor : ExpressionVisitor {
    protected override Expression VisitConstant(ConstantExpression node) {
        Console.WriteLine("Constant: {0}", node);
        return base.VisitConstant(node);
    }
    protected override Expression VisitParameter(ParameterExpression node) {
        Console.WriteLine("Parameter: {0}", node);
        return base.VisitParameter(node);
    }
    protected override Expression VisitBinary(BinaryExpression node) {
        Console.WriteLine("Binary with operator {0}", node.NodeType);
        return base.VisitBinary(node);
    }
}
```

`Visit` :

```
Expression<Func<int,bool>> isBig = a => a > 1000000;
var visitor = new PrintingVisitor();
visitor.Visit(isBig);
```

: https://riptutorial.com/ko/csharp/topic/75/-

161:

Examples

RPG

. . Flyweight DP .

. RPG . :

- # . .
- $.
- @ . .
- % .

:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

@##############@@@@@######@#$@@@

@##############@@@@@######@###@@@

@########%######@############@@@

@#############################@

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

. .

.

```
public interface IField
{
    string Name { get; }
    char Mark { get; }
    bool CanWalk { get; }
    FieldType Type { get; }
}
```

. ( ):

```
public enum FieldType
{
    GRASS,
    ROCK,
    START,
    CHEST
}
public class Grass : IField
{
    public string Name { get { return "Grass"; } }
```

```
    public char Mark { get { return '#'; } }
    public bool CanWalk { get { return true; } }
    public FieldType Type { get { return FieldType.GRASS; } }
}
public class StartingPoint : IField
{
    public string Name { get { return "Starting Point"; } }
    public char Mark { get { return '$'; } }
    public bool CanWalk { get { return true; } }
    public FieldType Type { get { return FieldType.START; } }
}
public class Rock : IField
{
    public string Name { get { return "Rock"; } }
    public char Mark { get { return '@'; } }
    public bool CanWalk { get { return false; } }
    public FieldType Type { get { return FieldType.ROCK; } }
}
public class TreasureChest : IField
{
    public string Name { get { return "Treasure Chest"; } }
    public char Mark { get { return '%'; } }
    public bool CanWalk { get { return true; } } // you can approach it
    public FieldType Type { get { return FieldType.CHEST; } }
}
```

. . DP . .

```
public class FieldRepository
{
    private List<IField> lstFields = new List<IField>();

    private IField AddField(FieldType type)
    {
        IField f;
        switch(type)
        {
            case FieldType.GRASS: f = new Grass(); break;
            case FieldType.ROCK: f = new Rock(); break;
            case FieldType.START: f = new StartingPoint(); break;
            case FieldType.CHEST:
            default: f = new TreasureChest(); break;
        }
        lstFields.Add(f); //add it to repository
        Console.WriteLine("Created new instance of {0}", f.Name);
        return f;
    }
    public IField GetField(FieldType type)
    {
        IField f = lstFields.Find(x => x.Type == type);
        if (f != null) return f;
        else return AddField(type);
    }
}
```

! .

```
public class Program
{
```

```
    public static void Main(string[] args)
    {
        FieldRepository f = new FieldRepository();
        IField grass = f.GetField(FieldType.GRASS);
        grass = f.GetField(FieldType.ROCK);
        grass = f.GetField(FieldType.GRASS);
    }
}
```

.

Grass

Rock

? `GetField` , .

: https://riptutorial.com/ko/csharp/topic/4619/---

162:

MD5 SHA1 . .

Examples

MD5

.

`MD5` 128 (16 , 32 16) .

`System.Security.Cryptography.MD5 ComputeHash` 16 .

:

```
using System;
using System.Security.Cryptography;
using System.Text;

internal class Program
{
    private static void Main()
    {
        var source = "Hello World!";

        // Creates an instance of the default implementation of the MD5 hash algorithm.
        using (var md5Hash = MD5.Create())
        {
            // Byte array representation of source string
            var sourceBytes = Encoding.UTF8.GetBytes(source);

            // Generate hash value(Byte Array) for input data
            var hashBytes = md5Hash.ComputeHash(sourceBytes);

            // Convert hash byte array to string
            var hash = BitConverter.ToString(hashBytes).Replace("-", string.Empty);

            // Output the MD5 hash
            Console.WriteLine("The MD5 hash of " + source + " is: " + hash);
        }
    }
}
```

: Hello World MD5 ! is : ED076287532E86365E841E92BFC50D8C

:

MD5 . brute-force .

SHA1

```
using System;
using System.Security.Cryptography;
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            string source = "Hello World!";
            using (SHA1 sha1Hash = SHA1.Create())
            {
                //From String to byte array
                byte[] sourceBytes = Encoding.UTF8.GetBytes(source);
                byte[] hashBytes = sha1Hash.ComputeHash(sourceBytes);
                string hash = BitConverter.ToString(hashBytes).Replace("-",String.Empty);

                Console.WriteLine("The SHA1 hash of " + source + " is: " + hash);
            }
        }
    }
 }
```

:

Hello Word SHA1 ! is : 2EF7BDE608CE5404E97D5F042F95F89F1C232871

SHA256

```
using System;
using System.Security.Cryptography;
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            string source = "Hello World!";
            using (SHA256 sha256Hash = SHA256.Create())
            {
                //From String to byte array
                byte[] sourceBytes = Encoding.UTF8.GetBytes(source);
                byte[] hashBytes = sha256Hash.ComputeHash(sourceBytes);
                string hash = BitConverter.ToString(hashBytes).Replace("-", String.Empty);

                Console.WriteLine("The SHA256 hash of " + source + " is: " + hash);
            }
        }
    }
}
```

:

Hello World SHA256 !
7F83B1657FF1FC53B92DC18148A1D65DFC2D4B1FA3D677284ADDD200126D9069.

SHA384

```
using System;
using System.Security.Cryptography;
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            string source = "Hello World!";
            using (SHA384 sha384Hash = SHA384.Create())
            {
                //From String to byte array
                byte[] sourceBytes = Encoding.UTF8.GetBytes(source);
                byte[] hashBytes = sha384Hash.ComputeHash(sourceBytes);
                string hash = BitConverter.ToString(hashBytes).Replace("-", String.Empty);

                Console.WriteLine("The SHA384 hash of " + source + " is: " + hash);
            }
        }
    }
}
```

:

Hello World SHA384 !
BFD76C0EBBD006FEE583410547C1887B0292BE76D582D96C242D2A792723E3FD6FD061F9D5CFD

SHA512

```
using System;
using System.Security.Cryptography;
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            string source = "Hello World!";
            using (SHA512 sha512Hash = SHA512.Create())
            {
                //From String to byte array
                byte[] sourceBytes = Encoding.UTF8.GetBytes(source);
                byte[] hashBytes = sha512Hash.ComputeHash(sourceBytes);
                string hash = BitConverter.ToString(hashBytes).Replace("-", String.Empty);

                Console.WriteLine("The SHA512 hash of " + source + " is: " + hash);
            }
        }
```

```
    }
}
```

: Hello World SHA512 ! is :
861844D6704E8573FEC34D967E20BCFEF3D424CF48BE04E6DC08F2BD58C729743371015EAD891C

PBKDF2

PBKDF2 (" 2") . rfc-2898 .

.NET `Rfc2898DeriveBytes` HMACSHA1 .

```
    using System.Security.Cryptography;

    ...

    public const int SALT_SIZE = 24; // size in bytes
    public const int HASH_SIZE = 24; // size in bytes
    public const int ITERATIONS = 100000; // number of pbkdf2 iterations

    public static byte[] CreateHash(string input)
    {
        // Generate a salt
        RNGCryptoServiceProvider provider = new RNGCryptoServiceProvider();
        byte[] salt = new byte[SALT_SIZE];
        provider.GetBytes(salt);

        // Generate the hash
        Rfc2898DeriveBytes pbkdf2 = new Rfc2898DeriveBytes(input, salt, ITERATIONS);
        return pbkdf2.GetBytes(HASH_SIZE);
    }
```

PBKDF2 .

:

. . PBKDF2 MD5 .

:

. . 1 ( ) .

Pbkdf2

```
using System;
using System.Linq;
using System.Security.Cryptography;

namespace YourCryptoNamespace
{
    /// <summary>
    /// Salted password hashing with PBKDF2-SHA1.
  /// Compatibility: .NET 3.0 and later.
  /// </summary>
```

```
  /// <remarks>See http://crackstation.net/hashing-security.htm for much more on password
hashing.</remarks>
  public static class PasswordHashProvider
  {
    /// <summary>
    /// The salt byte size, 64 length ensures safety but could be increased / decreased
    /// </summary>
    private const int SaltByteSize = 64;
    /// <summary>
    /// The hash byte size,
    /// </summary>
    private const int HashByteSize = 64;
    /// <summary>
    ///  High iteration count is less likely to be cracked
    /// </summary>
    private const int Pbkdf2Iterations = 10000;

    /// <summary>
    /// Creates a salted PBKDF2 hash of the password.
    /// </summary>
    /// <remarks>
    /// The salt and the hash have to be persisted side by side for the password. They could
be persisted as bytes or as a string using the convenience methods in the next class to
convert from byte[] to string and later back again when executing password validation.
    /// </remarks>
    /// <param name="password">The password to hash.</param>
    /// <returns>The hash of the password.</returns>
    public static PasswordHashContainer CreateHash(string password)
    {
      // Generate a random salt
      using (var csprng = new RNGCryptoServiceProvider())
      {
        // create a unique salt for every password hash to prevent rainbow and dictionary
based attacks
        var salt = new byte[SaltByteSize];
        csprng.GetBytes(salt);

        // Hash the password and encode the parameters
        var hash = Pbkdf2(password, salt, Pbkdf2Iterations, HashByteSize);

        return new PasswordHashContainer(hash, salt);
      }
    }
    /// <summary>
    /// Recreates a password hash based on the incoming password string and the stored salt
    /// </summary>
    /// <param name="password">The password to check.</param>
    /// <param name="salt">The salt existing.</param>
    /// <returns>the generated hash based on the password and salt</returns>
    public static byte[] CreateHash(string password, byte[] salt)
    {
      // Extract the parameters from the hash
      return Pbkdf2(password, salt, Pbkdf2Iterations, HashByteSize);
    }

    /// <summary>
    /// Validates a password given a hash of the correct one.
    /// </summary>
    /// <param name="password">The password to check.</param>
    /// <param name="salt">The existing stored salt.</param>
    /// <param name="correctHash">The hash of the existing password.</param>
```

```
    /// <returns><c>true</c> if the password is correct. <c>false</c> otherwise. </returns>
    public static bool ValidatePassword(string password, byte[] salt, byte[] correctHash)
    {
      // Extract the parameters from the hash
      byte[] testHash = Pbkdf2(password, salt, Pbkdf2Iterations, HashByteSize);
      return CompareHashes(correctHash, testHash);
    }
    /// <summary>
    /// Compares two byte arrays (hashes)
    /// </summary>
    /// <param name="array1">The array1.</param>
    /// <param name="array2">The array2.</param>
    /// <returns><c>true</c> if they are the same, otherwise <c>false</c></returns>
    public static bool CompareHashes(byte[] array1, byte[] array2)
    {
      if (array1.Length != array2.Length) return false;
      return !array1.Where((t, i) => t != array2[i]).Any();
    }

    /// <summary>
    /// Computes the PBKDF2-SHA1 hash of a password.
    /// </summary>
    /// <param name="password">The password to hash.</param>
    /// <param name="salt">The salt.</param>
    /// <param name="iterations">The PBKDF2 iteration count.</param>
    /// <param name="outputBytes">The length of the hash to generate, in bytes.</param>
    /// <returns>A hash of the password.</returns>
    private static byte[] Pbkdf2(string password, byte[] salt, int iterations, int
outputBytes)
    {
      using (var pbkdf2 = new Rfc2898DeriveBytes(password, salt))
      {
        pbkdf2.IterationCount = iterations;
        return pbkdf2.GetBytes(outputBytes);
      }
    }
  }

  /// <summary>
  /// Container for password hash and salt and iterations.
  /// </summary>
  public sealed class PasswordHashContainer
  {
    /// <summary>
    /// Gets the hashed password.
    /// </summary>
    public byte[] HashedPassword { get; private set; }
    /// <summary>
    /// Gets the salt.
    /// </summary>
    public byte[] Salt { get; private set; }

    /// <summary>
    /// Initializes a new instance of the <see cref="PasswordHashContainer" /> class.
    /// </summary>
    /// <param name="hashedPassword">The hashed password.</param>
    /// <param name="salt">The salt.</param>
    public PasswordHashContainer(byte[] hashedPassword, byte[] salt)
    {
      this.HashedPassword = hashedPassword;
      this.Salt = salt;
```

```
    }
  }

  /// <summary>
  /// Convenience methods for converting between hex strings and byte array.
  /// </summary>
  public static class ByteConverter
  {
    /// <summary>
    /// Converts the hex representation string to an array of bytes
    /// </summary>
    /// <param name="hexedString">The hexed string.</param>
    /// <returns></returns>
    public static byte[] GetHexBytes(string hexedString)
    {
      var bytes = new byte[hexedString.Length / 2];
      for (var i = 0; i < bytes.Length; i++)
      {
        var strPos = i * 2;
        var chars = hexedString.Substring(strPos, 2);
        bytes[i] = Convert.ToByte(chars, 16);
      }
      return bytes;
    }
    /// <summary>
    /// Gets a hex string representation of the byte array passed in.
    /// </summary>
    /// <param name="bytes">The bytes.</param>
    public static string GetHexString(byte[] bytes)
    {
      return BitConverter.ToString(bytes).Replace("-", "").ToUpper();
    }
  }
}

/*
 * Password Hashing With PBKDF2 (http://crackstation.net/hashing-security.htm).
 * Copyright (c) 2013, Taylor Hornby
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 * this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 * this list of conditions and the following disclaimer in the documentation
 * and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
```

```
*/
```

Crackstation - Salted Password Hashing - . () .

: https://riptutorial.com/ko/csharp/topic/2774/-

163:

Examples

(`public` , `private` , ...), (`abstract`), . . .

```
AccessModifier OptionalModifier ReturnType MethodName(InputParameters)
{
    //Method body
}
```

`AccessModifier` `public` , `protected` , `pirvate` `internal` .

`OptionalModifier` `static` `abstract` `virtual` `override` `new` `sealed` .

`ReturnType` return `void` `int` complex .

Method . (`int a`) (`int a, int b`) .

. (: `int a = 0`). .

.

```
private int Sum(int a, int b)
{
    return a + b;
}
```

:

```
// Single argument
System.Console.WriteLine("Hello World");

// Multiple arguments
string name = "User";
System.Console.WriteLine("Hello, {0}!", name);
```

:

```
string input = System.Console.ReadLine();
```

:

```
int x = 42;
// The instance method called here is Int32.ToString()
string xAsString = x.ToString();
```

```
// Assuming a method 'T[] CreateArray<T>(int size)'
DateTime[] dates = CreateArray<DateTime>(8);
```

(`i, s o`).

```
static void DoSomething(int i, string s, object o) {
    Console.WriteLine(String.Format("i={0}, s={1}, o={2}", i, s, o));
}
```

. .

. .

```
DoSomething(x, "hello", new object());
```

nothing (`void`) .

```
// If you don't want to return a value, use void as return type.
static void ReturnsNothing() {
    Console.WriteLine("Returns nothing");
}

// If you want to return a value, you need to specify its type.
static string ReturnsHelloWorld() {
    return "Hello World";
}
```

. `return` . `return` . `finally` .

(`void`) `return` . `return` .

`return` :

```
return;
return 0;
return x * 2;
return Console.ReadLine();
```

. yield .

.

```
static void SaySomething(string what = "ehh") {
    Console.WriteLine(what);
}

static void Main() {
    // prints "hello"
    SaySomething("hello");
    // prints "ehh"
    SaySomething(); // The compiler compiles this as if we had typed SaySomething("ehh")
}
```

.

.

```
static void SaySomething(string say, string what = "ehh") {
        //Correct
        Console.WriteLine(say + what);
    }

static void SaySomethingElse(string what = "ehh", string say) {
        //Incorrect
        Console.WriteLine(say + what);
    }
```

: . .

: . .

-
-
- **

Area . .

```
public string Area(int value1)
{
    return String.Format("Area of Square is {0}", value1 * value1);
}
```

. (: 5) "Area of Square is 25" .

```
public  double Area(double value1, double value2)
{
    return value1 * value2;
}
```

double double . .

```
public double Area(double value1)
{
    return 3.14 * Math.Pow(value1,2);
}
```

. (radius) .

- Area Area .

```
string squareArea = Area(2);
double rectangleArea = Area(32.0, 17.5);
double circleArea = Area(5.0); // all of these are valid and will compile.
```

** . Area (:

```
public string Area(double width, double height) { ... }
public double Area(double width, double height) { ... }
// This will NOT compile.
```

.

```
public interface IAreaCalculatorString {

    public string Area(double width, double height);

}

public class AreaCalculator : IAreaCalculatorString {

    public string IAreaCalculatorString.Area(double width, double height) { ... }
    // Note that the method call now explicitly says it will be used when called through
    // the IAreaCalculatorString interface, allowing us to resolve the ambiguity.
    public double Area(double width, double height) { ... }
```

. .

```
delegate int IntOp(int lhs, int rhs);
```

```
class Program
{
    static void Main(string[] args)
    {
        // C# 2.0 definition
        IntOp add = delegate(int lhs, int rhs)
        {
            return lhs + rhs;
        };

        // C# 3.0 definition
        IntOp mul = (lhs, rhs) =>
        {
            return lhs * rhs;
        };

        // C# 3.0 definition - shorthand
        IntOp sub = (lhs, rhs) => lhs - rhs;

        // Calling each method
        Console.WriteLine("2 + 3 = " + add(2, 3));
        Console.WriteLine("2 * 3 = " + mul(2, 3));
        Console.WriteLine("2 - 3 = " + sub(2, 3));
    }
}
```

```
// static: is callable on a class even when no instance of the class has been created
public static void MyMethod()

// virtual: can be called or overridden in an inherited class
public virtual  void MyMethod()

// internal: access is limited within the current assembly
internal  void MyMethod()

//private: access is limited only within the same class
private  void MyMethod()
```

```
//public: access right from every class / assembly
public void MyMethod()

//protected: access is limited to the containing class or types derived from it
protected void MyMethod()

//protected internal: access is limited to the current assembly or types derived from the
containing class.
protected internal void MyMethod()
```

: https://riptutorial.com/ko/csharp/topic/60/-

164:

- public static ReturnType MyExtensionMethod (TargetType)
- public static ReturnType MyExtensionMethod (TargetType , TArg1 arg1, ...)

this ""

.

. this .

.

?

.

. , .

Intellisense . Foo , Foo .

" " . IEnumerable System.Linq .

. (: bool IsApproxEqualTo(this double value, double other) System " . .

.

: ?

. string . .

.

1. .
2. .
3. "this" .NET .
4. VS intellisense . . VS intellisense.
5. using .
6. .
7. , .
8. .

Examples

-

C # 3.0 . . . , 3 . . **Method Chaining** .

. , .

. `this` (`this` - C # `this`).

`string` . `Shorten()` `String` `Shorten()` . static `StringExtensions` . `Shorten()` `string` . `Shorten()` `string` `this` . `this string text` . `string` `text` .

```
static class StringExtensions
{
    public static string Shorten(this string text, int length)
    {
        return text.Substring(0, length);
    }
}

class Program
{
    static void Main()
    {
        // This calls method String.ToUpper()
        var myString = "Hello World!".ToUpper();

        // This calls the extension method StringExtensions.Shorten()
        var newString = myString.Shorten(5);

        // It is worth noting that the above call is purely syntactic sugar
        // and the assignment below is functionally equivalent
        var newString2 = StringExtensions.Shorten(myString, 5);
    }
}
```

.NET Fiddle

(`this`) .

, :

```
"some string".Shorten(5);
```

.

```
text: "some string"
length: 5
```

. .NET Framework . .

. LINQ Extensions :

```
using System.Linq; // Allows use of extension methods from the System.Linq namespace

class Program
```

```
{
    static void Main()
    {
        var ints = new int[] {1, 2, 3, 4};

        // Call Where() extension method from the System.Linq namespace
        var even = ints.Where(x => x % 2 == 0);
    }
}
```

.NET Fiddle

C # 6.0 `using static using static . ,using static System.Linq.Enumerable; .` .

. :

```
class Test
{
   public void Hello()
   {
       Console.WriteLine("From Test");
   }
}

static class TestExtensions
{
    public static void Hello(this Test test)
    {
        Console.WriteLine("From extension method");
    }
}

class Program
{
    static void Main()
    {
        Test t = new Test();
        t.Hello(); // Prints "From Test"
    }
}
```

.NET Fiddle

. , using , .

.

`originalTypeInstance.ExtensionMethod()` . .

, :

```
//Calling as though method belongs to string--it seamlessly extends string
```

```
String s = "Hello World";
s.Shorten(5);

//Calling as a traditional static method with two parameters
StringExtensions.Shorten(s, 5);
```

. .

```
static class StringExtensions
{
    public static string Shorten(this string text, int length)
    {
        return text.Substring(0, length);
    }
}
```

:

```
var newString = StringExtensions.Shorten("Hello World", 5);
```

.

- . . .
- . .
- .
- Reflection .
- Visual Studio .

---

using static .:

```
using static OurNamespace.StringExtensions; // refers to class in previous example

// OK: extension method syntax still works.
"Hello World".Shorten(5);
// OK: static method syntax still works.
OurNamespace.StringExtensions.Shorten("Hello World", 5);
// Compile time error: extension methods can't be called as static without specifying class.
Shorten("Hello World", 5);
```

Shorten this .

Null

., null null , throw NullReferenceException . .

.

```
public static class StringExtensions
{
```

```
    public static string EmptyIfNull(this string text)
    {
        return text ?? String.Empty;
    }

    public static string NullIfEmpty(this string text)
    {
        return String.Empty == text ? null : text;
    }
}
```

```
string nullString = null;
string emptyString = nullString.EmptyIfNull();// will return ""
string anotherNullString = emptyString.NullIfEmpty(); // will return null
```

.NET Fiddle

public (internal) .

```
public class SomeClass
{
    public void DoStuff()
    {

    }

    protected void DoMagic()
    {

    }
}

public static class SomeClassExtensions
{
    public static void DoStuffWrapper(this SomeClass someInstance)
    {
        someInstance.DoStuff(); // ok
    }

    public static void DoMagicWrapper(this SomeClass someInstance)
    {
        someInstance.DoMagic(); // compilation error
    }
}
```

. . public (internal , internal), .

generics . :

```
static class Extensions
{
    public static bool HasMoreThanThreeElements<T>(this IEnumerable<T> enumerable)
    {
        return enumerable.Take(4).Count() > 3;
    }
}
```

:

```
IEnumerable<int> numbers = new List<int> {1,2,3,4,5,6};
var hasMoreThanThreeElements = numbers.HasMoreThanThreeElements();
```

:

```
public static TU GenericExt<T, TU>(this T obj)
{
     TU ret = default(TU);
     // do some stuff with obj
     return ret;
}
```

.

```
IEnumerable<int> numbers = new List<int> {1,2,3,4,5,6};
var result = numbers.GenericExt<IEnumerable<int>,String>();
```

.

```
class MyType<T1, T2>
{
}

static class Extensions
{
    public static void Example<T>(this MyType<int, T> test)
    {
    }
}
```

.

```
MyType<int, string> t = new MyType<int, string>();
t.Example();
```

where .

```
public static bool IsDefault<T>(this T obj) where T : struct, IEquatable<T>
{
     return EqualityComparer<T>.Default.Equals(obj, default(T));
}
```

:

```
int number = 5;
var IsDefault = number.IsDefault();
```

() () .

```
public class Base
{
    public virtual string GetName()
    {
        return "Base";
    }
}

public class Derived : Base
{
    public override string GetName()
    {
        return "Derived";
    }
}

public static class Extensions
{
    public static string GetNameByExtension(this Base item)
    {
        return "Base";
    }

    public static string GetNameByExtension(this Derived item)
    {
        return "Derived";
    }
}

public static class Program
{
    public static void Main()
    {
        Derived derived = new Derived();
        Base @base = derived;

        // Use the instance method "GetName"
        Console.WriteLine(derived.GetName()); // Prints "Derived"
        Console.WriteLine(@base.GetName()); // Prints "Derived"

        // Use the static extension method "GetNameByExtension"
        Console.WriteLine(derived.GetNameByExtension()); // Prints "Derived"
        Console.WriteLine(@base.GetNameByExtension()); // Prints "Base"
    }
}
```

.NET Fiddle

dynamic .

```
public class Person
{
    public string Name { get; set; }
}

public static class ExtenionPerson
{
    public static string GetPersonName(this Person person)
    {
        return person.Name;
```

```
    }
}

dynamic person = new Person { Name = "Jon" };
var name = person.GetPersonName(); // RuntimeBinderException is thrown
```

.

```
static class Program
{
    static void Main()
    {
        dynamic dynamicObject = new ExpandoObject();

        string awesomeString = "Awesome";

        // Prints True
        Console.WriteLine(awesomeString.IsThisAwesome());

        dynamicObject.StringValue = awesomeString;

        // Prints True
        Console.WriteLine(StringExtensions.IsThisAwesome(dynamicObject.StringValue));

        // No compile time error or warning, but on runtime throws RuntimeBinderException
        Console.WriteLine(dynamicObject.StringValue.IsThisAwesome());
    }
}

static class StringExtensions
{
    public static bool IsThisAwesome(this string value)
    {
        return value.Equals("Awesome");
    }
}
```

[ ] . `using` .

, DLR `using` . . , .

. , `HttpContext.Items` at cetera ...

```
public static class CacheExtensions
{
    public static void SetUserInfo(this Cache cache, UserInfo data) =>
        cache["UserInfo"] = data;

    public static UserInfo GetUserInfo(this Cache cache) =>
        cache["UserInfo"] as UserInfo;
}
```

. .

`this` "" . / ""API .

```
void Main()
{
    int result = 5.Increment().Decrement().Increment();
    // result is now 6
}

public static class IntExtensions
{
    public static int Increment(this int number) {
        return ++number;
    }

    public static int Decrement(this int number) {
        return --number;
    }
}
```

```
void Main()
{
    int[] ints = new[] { 1, 2, 3, 4, 5, 6};
    int[] a = ints.WhereEven();
    //a is { 2, 4, 6 };
    int[] b = ints.WhereEven().WhereGreaterThan(2);
    //b is { 4, 6 };
}

public static class IntArrayExtensions
{
    public static int[] WhereEven(this int[] array)
    {
        //Enumerable.* extension methods use a fluent approach
        return array.Where(i => (i%2) == 0).ToArray();
    }

    public static int[] WhereGreaterThan(this int[] array, int value)
    {
        return array.Where(i => i > value).ToArray();
    }
}
```

.

```
public interface IInterface
{
   string Do()
}

public static class ExtensionMethods{
    public static string DoWith(this IInterface obj){
      //does something with IInterface instance
    }
}

public class Classy : IInterface
{
   // this is a wrapper method; you could also call DoWith() on a Classy instance directly,
   // provided you import the namespace containing the extension method
   public Do(){
       return this.DoWith();
```

```
    }
}
```

:

```
var classy = new Classy();
classy.Do(); // will call the extension
classy.DoWith(); // Classy implements IInterface so it can also be called this way
```

IList : 2

IList .

`isOrdered` false .

(`T`) `Equals` `GetHashCode` .

:

```
List<string> list1 = new List<string> {"a1", "a2", null, "a3"};
List<string> list2 = new List<string> {"a1", "a2", "a3", null};

list1.Compare(list2);//this gives false
list1.Compare(list2, false);//this gives true. they are equal when the order is disregarded
```

:

```
public static bool Compare<T>(this IList<T> list1, IList<T> list2, bool isOrdered = true)
{
    if (list1 == null && list2 == null)
        return true;
    if (list1 == null || list2 == null || list1.Count != list2.Count)
        return false;

    if (isOrdered)
    {
        for (int i = 0; i < list2.Count; i++)
        {
            var l1 = list1[i];
            var l2 = list2[i];
            if (
                 (l1 == null && l2 != null) ||
                 (l1 != null && l2 == null) ||
                 (!l1.Equals(l2)))
            {
                    return false;
            }
        }
        return true;
    }
    else
    {
        List<T> list2Copy = new List<T>(list2);
        //Can be done with Dictionary without O(n^2)
        for (int i = 0; i < list1.Count; i++)
```

```
        {
            if (!list2Copy.Remove(list1[i]))
                return false;
        }
        return true;
    }
}
```

.

.

```
public enum YesNo
{
    Yes,
    No,
}

public static class EnumExtentions
{
    public static bool ToBool(this YesNo yn)
    {
        return yn == YesNo.Yes;
    }
    public static YesNo ToYesNo(this bool yn)
    {
        return yn ? YesNo.Yes : YesNo.No;
    }
}
```

enum . bool.

```
bool yesNoBool = YesNo.Yes.ToBool(); // yesNoBool == true
YesNo yesNoEnum = false.ToYesNo();   // yesNoEnum == YesNo.No
```

.

```
public enum Element
{
    Hydrogen,
    Helium,
    Lithium,
    Beryllium,
    Boron,
    Carbon,
    Nitrogen,
    Oxygen
    //Etc
}

public static class ElementExtensions
{
    public static double AtomicMass(this Element element)
    {
        switch(element)
        {
            case Element.Hydrogen:  return 1.00794;
```

```
            case Element.Helium:    return 4.002602;
            case Element.Lithium:   return 6.941;
            case Element.Beryllium: return 9.012182;
            case Element.Boron:     return 10.811;
            case Element.Carbon:    return 12.0107;
            case Element.Nitrogen:  return 14.0067;
            case Element.Oxygen:    return 15.9994;
            //Etc
        }
        return double.Nan;
    }
}

var massWater = 2*Element.Hydrogen.AtomicMass() + Element.Oxygen.AtomicMass();
```

DRY .

. . . C # mixin .

`IEnumerable<T>` `System.Linq.Enumerable` . `IEnumerable<T>` Generic Non-generic `GetEnumerator()` . `System.Linq.Enumerable` `IEnumerable<T>` .

.

```
public interface ITimeFormatter
{
   string Format(TimeSpan span);
}

public static class TimeFormatter
{
    // Provide an overload to *all* implementers of ITimeFormatter.
    public static string Format(
        this ITimeFormatter formatter,
        int millisecondsSpan)
        => formatter.Format(TimeSpan.FromMilliseconds(millisecondsSpan));
}

// Implementations only need to provide one method. Very easy to
// write additional implementations.
public class SecondsTimeFormatter : ITimeFormatter
{
   public string Format(TimeSpan span)
   {
       return $"{(int)span.TotalSeconds}s";
   }
}

class Program
{
    static void Main(string[] args)
    {
        var formatter = new SecondsTimeFormatter();
        // Callers get two method overloads!
        Console.WriteLine($"4500ms is rougly {formatter.Format(4500)}");
        var span = TimeSpan.FromSeconds(5);
        Console.WriteLine($"{span} is formatted as {formatter.Format(span)}");
    }
```

```
}
```

if / then "" . null . ,

```
public static class CakeExtensions
{
    public static Cake EnsureTrueCake(this Cake cake)
    {
        //If the cake is a lie, substitute a cake from grandma, whose cakes aren't as tasty
but are known never to be lies. If the cake isn't a lie, don't do anything and return it.
        return CakeVerificationService.IsCakeLie(cake) ? GrandmasKitchen.Get1950sCake() :
cake;
    }
}
```

```
Cake myCake = Bakery.GetNextCake().EnsureTrueCake();
myMouth.Eat(myCake);//Eat the cake, confident that it is not a lie.
```

Extension

. Try Catch . ...

```
using System;
using System.Diagnostics;

namespace Samples
{
    /// <summary>
    /// Wraps a try catch statement as a static helper which uses
    /// Extension methods for the exception
    /// </summary>
    public static class Bullet
    {
        /// <summary>
        /// Wrapper for Try Catch Statement
        /// </summary>
        /// <param name="code">Call back for code</param>
        /// <param name="error">Already handled and logged exception</param>
        public static void Proof(Action code, Action<Exception> error)
        {
            try
            {
                code();
            }
            catch (Exception iox)
            {
                //extension method used here
                iox.Log("BP2200-ERR-Unexpected Error");
                //callback, exception already handled and logged
                error(iox);
            }
        }
        /// <summary>
        /// Example of a logging method helper, this is the extension method
        /// </summary>
        /// <param name="error">The Exception to log</param>
        /// <param name="messageID">A unique error ID header</param>
```

```
        public static void Log(this Exception error, string messageID)
        {
            Trace.WriteLine(messageID);
            Trace.WriteLine(error.Message);
            Trace.WriteLine(error.StackTrace);
            Trace.WriteLine("");
        }
    }
    /// <summary>
    /// Shows how to use both the wrapper and extension methods.
    /// </summary>
    public class UseBulletProofing
    {
        public UseBulletProofing()
        {
            var ok = false;
            var result = DoSomething();
            if (!result.Contains("ERR"))
            {
                ok = true;
                DoSomethingElse();
            }
        }

        /// <summary>
        /// How to use Bullet Proofing in your code.
        /// </summary>
        /// <returns>A string</returns>
        public string DoSomething()
        {
            string result = string.Empty;
            //Note that the Bullet.Proof method forces this construct.
            Bullet.Proof(() =>
            {
                //this is the code callback
                result = "DST5900-INF-No Exceptions in this code";
            }, error =>
            {
                //error is the already logged and handled exception
                //determine the base result
                result = "DTS6200-ERR-An exception happened look at console log";
                if (error.Message.Contains("SomeMarker"))
                {
                    //filter the result for Something within the exception message
                    result = "DST6500-ERR-Some marker was found in the exception";
                }
            });
            return result;
        }

        /// <summary>
        /// Next step in workflow
        /// </summary>
        public void DoSomethingElse()
        {
            //Only called if no exception was thrown before
        }
    }
}
```

. .

```
public interface IVehicle
{
    int MilesDriven { get; set; }
}

public static class Extensions
{
    public static int FeetDriven(this IVehicle vehicle)
    {
        return vehicle.MilesDriven * 5028;
    }
}
```

FeetDriven IVehicle .   IVehicle ,   FeetDriven IVehicle   .

.

```
 public class UserDTO
 {
        public AddressDTO Address { get; set; }
 }

 public class AddressDTO
 {
        public string Name { get; set; }
 }
```

```
public class UserViewModel
{
    public AddressViewModel Address { get; set; }
}

public class AddressViewModel
{
    public string Name { get; set; }
}
```

.

```
public static class ViewModelMapper
{
      public static UserViewModel ToViewModel(this UserDTO user)
      {
            return user == null ?
                null :
                new UserViewModel()
                {
                    Address = user.Address.ToViewModel()
                    // Job = user.Job.ToViewModel(),
                    // Contact = user.Contact.ToViewModel() .. and so on
                };
      }

      public static AddressViewModel ToViewModel(this AddressDTO userAddr)
      {
            return userAddr == null ?
                null :
```

```
                new AddressViewModel()
                {
                    Name = userAddr.Name
                };
      }
}
```

.

```
    UserDTO userDTOObj = new UserDTO() {
            Address = new AddressDTO() {
                Name = "Address of the user"
            }
        };

    UserViewModel user = userDTOObj.ToViewModel(); // My DTO mapped to Viewmodel
```

(ToViewModel)

(: DictList)

List<T> Dictionary .

.

```
public static class DictListExtensions
{
    public static void Add<TKey, TValue, TCollection>(this Dictionary<TKey, TCollection> dict,
TKey key, TValue value)
            where TCollection : ICollection<TValue>, new()
    {
        TCollection list;
        if (!dict.TryGetValue(key, out list))
        {
            list = new TCollection();
            dict.Add(key, list);
        }

        list.Add(value);
    }

    public static bool Remove<TKey, TValue, TCollection>(this Dictionary<TKey, TCollection>
dict, TKey key, TValue value)
        where TCollection : ICollection<TValue>
    {
        TCollection list;
        if (!dict.TryGetValue(key, out list))
        {
            return false;
        }

        var ret = list.Remove(value);
        if (list.Count == 0)
        {
            dict.Remove(key);
        }
        return ret;
```

```
    }
}
```

.

```
var dictList = new Dictionary<string, List<int>>();

dictList.Add("example", 5);
dictList.Add("example", 10);
dictList.Add("example", 15);

Console.WriteLine(String.Join(", ", dictList["example"])); // 5, 10, 15

dictList.Remove("example", 5);
dictList.Remove("example", 10);

Console.WriteLine(String.Join(", ", dictList["example"])); // 15

dictList.Remove("example", 15);

Console.WriteLine(dictList.ContainsKey("example")); // False
```

: https://riptutorial.com/ko/csharp/topic/20/-

165:

Examples

. .

(byte []) . (: WriteByte() .

, . , . .

/ (,) . FileStream .

```
string filePath = @"c:\Users\exampleuser\Documents\userinputlog.txt";
using (FileStream fs = new FileStream(filePath, FileMode.Open, FileAccess.Read,
FileShare.ReadWrite))
{
    // do stuff here...

    fs.Close();
}
```

, MemoryStream .

```
// Read all bytes in from a file on the disk.
byte[] file = File.ReadAllBytes("C:\\file.txt");

// Create a memory stream from those bytes.
using (MemoryStream memory = new MemoryStream(file))
{
   // do stuff here...
}
```

System.Net.Sockets.NetworkStream .

System.IO.Stream . . .NET Framework StreamReader , StreamWriter , BinaryReader BinaryWriter .

StreamReader StreamWriter . . . bool leaveOpen true .

StreamWriter :

```
FileStream fs = new FileStream("sample.txt", FileMode.Create);
StreamWriter sw = new StreamWriter(fs);
string NextLine = "This is the appended line.";
sw.Write(NextLine);
sw.Close();
//fs.Close(); There is no need to close fs. Closing sw will also close the stream it contains.
```

StreamReader :

```
using (var ms = new MemoryStream())
```

```
{
    StreamWriter sw = new StreamWriter(ms);
    sw.Write(123);
    //sw.Close();     This will close ms and when we try to use ms later it will cause an
exception
    sw.Flush();     //You can send the remaining data to stream. Closing will do this
automatically
    // We need to set the position to 0 in order to read
    // from the beginning.
    ms.Position = 0;
    StreamReader sr = new StreamReader(ms);
    var myStr = sr.ReadToEnd();
    sr.Close();
    ms.Close();
}
```

Classes `Stream` , `StreamReader` , `StreamWriter` `IDisposable` `IDisposable` `Dispose()` .

: https://riptutorial.com/ko/csharp/topic/3114/

S. No		Contributors
1	C #	4444, A. Raza, A_Arnold , aalaap, Aaron Hudon, abishekshivan, Ade Stringer, Aleksandur Murfitt, Almir Vuk, Alok Singh, Andrii Abramov, AndroidMechanic, Aravind Suresh, Artemix, Ben Aaronson, Bernard Vander Beken, Bjørn-Roger Kringsjå, Blachshma, Blorgbeard, bpoiss, Br0k3nL1m1ts, Callum Watkins, Carlos Muñoz, Chad Levy, Chris Nantau, Christopher Ronning, Community, Confiqure, crunchy, David G., David Pine, DavidG, DAXaholic , Delphi.Boy, Durgpal Singh, DWright, Ehsan Sajjad, Elie Saad, Emre Bolat, enrico.bacis, fabriciorissetto, FadedAce, Florian Greinacher, Florian Koch , Frankenstine Joe, Gennady Trubach, GingerHead, Gordon Bell, gracacs, G-Wiz, H. Pauwelyn, Happypig375, Henrik H, HodofHod, Hywel Rees, iliketocode, Iordanis , Jamie Rees, Jawa, jnovo, John Slegers, Kayathiri, ken2k , Kevin Montrose, Kritner , Krzyserious, leumas1960, M Monis Ahmed Khan, Mahmoud Elgindy, Malick, Marcus

		Höglund, Mateen Ulhaq, Matt, Matt, Matt, Matt, Michael B, Michael Brandon Morris, Miljen Mikic, Millan Sanchez, Nate Barbettini, Nick, Nick Cox, Nipun Tripathi, NotMyself, Ojen, PashaPash, pijemcolu, Prateek, Raj Rao, Rajput , Rakitić, Rion Williams, RokumDev, RomCoo, Ryan Hilbert, sebingel, SeeuD1, solidcell, Steven Ackley, sumit sharma, Tofix, Tom Bowers, Travis J, Tushar patel, User 00000, user3185569, Ven, Victor Tomaili, viggity, void, Wen Qin, Ziad Akiki , Zze
2	.NET (Roslyn)	4444, Lukáš Lánský
3	.NET	Andrei Rînea, da_sann, Eamon Charles, J3soon, Luke Ryan, Squidward, Suren Srapyan
4	.NET	Andrew Piliser, cbale, codekaizen, Danny Varod, Isac, Jaroslav Kadlec, MSE, Nisarg Shah, Rahul Nikate, Stephen Leppik, Uwe Keim, ZenLulz
5	.zip	4444, DLeh, Naveen Gogineni, Nisarg Shah
6	ASP.NET ID	HappyCoding, Skullomania
7	AssemblyInfo.cs	Adi Lester, Ameya Deshpande, AndreyAkinshin, Boggin, Dodzi Dzakuma, dove,

		Joel Martinez, pinkfloydx33, Ralf Bönning, Theodoros Chatzigiannakis, Wasabi Fan
8	BackgroundWorker	Bovaz, Draken, ephtee, Jacobr365, Will
9	BigInteger	4444, Ed Marty, James Hughes, Rob, The_Outsider
10	C # 3.0	0xFF, bob0the0mighty, FrenkyB, H. Pauwelyn, ken2k, Maniero, Rob
11	C # 4.0	Benjamin Hodgson, Botond Balázs, H. Pauwelyn, Proxima, Sibeesh Venu, Squidward, Theodoros Chatzigiannakis
12	C # 5.0	Abob, alex.b, H. Pauwelyn
13	C # 6.0	A_Arnold, Aaron Anodide, Aaron Hudon, Adil Mammadov, Adriano Repetti, AER, AGB, Akshay Anand, Alan McBee, Alex Logan, Amitay Stern, anaximander, andre_ss6, Andrea, AndroidMechanic, Ares, Arthur Rizzo, Ashwin Ramaswami, avishayp, Balagurunathan Marimuthu, Bardia, Ben Aaronson, Blubberguy22, Bobson, bpoiss, Bradley Uffner, Bret Copeland, C4u, Callum Watkins, Chad Levy, Charlie H, ChrFin, Community,

		Conrad.Dean, Cyprien Autexier, Dan, Daniel Minnaar, Daniel Stradowski, DarkV1, dasblinkenlight, David, David G., David Pine, Deepak gupta, DLeh, dotctor, Durgpal Singh, Ehsan Sajjad, el2iot2, Emre Bolat, enrico.bacis, Erik Schierboom, fabriciorissetto, faso, Franck Dernoncourt, FrankerZ, Gabor Kecskemeti, Gary, Gates Wong, Geoff, GingerHead, Gordon Bell, Guillaume Pascal, H. Pauwelyn, hankide, Henrik H, iliketocode, Iordanis , Irfan, Ivan Yurchenko, J. Steen, Jacob Linney, Jamie Rees, Jason Sturges, Jeppe Stig Nielsen, Jim, JNYRanger, Joe, Joel Etherton, John Slegers, Johnbot, Jojodmo, Jonas S, Juan, Kapep, ken2k, Kit, Konamiman, Krikor Ailanjian, Lafexlos, LaoR , Lasse Vågsæther Karlsen, M.kazem Akhgary, Mafii, Magisch, Makyen, MANISH KUMAR CHOUDHARY, Marc, MarcinJuraszek, Mark Shevchenko, Matas Vaitkevicius, Mateen Ulhaq, Matt, Matt, Matt, Matt Thomas, Maximillian Laumeister, mbrdev, Mellow, Michael Mairegger, Michael Richardson, Michał Perłakowski, mike z, Minhas Kamal, Mitch

		Talmadge, Mohammad Mirmostafa, Mr.Mindor, mshsayem, MuiBienCarlota, Nate Barbettini, Nicholas Sizer , nik, nollidge, Nuri Tasdemir, Oliver Mellet, Orlando William, Osama AbuSitta, Panda, Parth Patel, Patrick, Pavel Voronin, PSN, qJake, QoP, Racil Hilan, Radouane ROUFID, Rahul Nikate, Raidri, Rajeev, Rakitić, ravindra, rdans, Reeven, Richa Garg, Richard, Rion Williams, Rob, Robban, Robert Columbia, Ryan Hilbert, ryanyuyu, Sam, Sam Axe, Samuel, Sender, Shekhar, Shoe, Slayther, solidcell, Squidward, Squirrel, stackptr, stark, Stilgar, Sunny R Gupta, Suren Srapyan, Sworgkh, syb0rg, takrl, Tamir Vered, Theodoros Chatzigiannakis, Timothy Shields, Tom Droste, Travis J, Trent, Trikaldarshi, Troyen, Tushar patel, tzachs, Uri Agassi, Uriil, uTeisT, vcsjones, Ven, viggity, Vishal Madhvani, Vlad, Wai Ha Lee, Xiaoy312, Yury Kerbitskov, Zano, Ze Rubeus, Zimm1
14	C # 7.0	Adil Mammadov, afuna, Amitay Stern, Amr Badawy, Andreas Pähler , Andrew Diamond, Avi Turner, Benjamin Hodgson, Blorgbeard,

bluray, Botond Balázs, Bovaz, Cerbrus, Clueless, Conrad.Dean, Dale Chen, David Pine, Degusto, Didgeridoo, Diligent Key Presser, ECC-Dan, Emre Bolat, fallaciousreasoning, ferday, Florian Greinacher, ganchito55, Ginkgo, H. Pauwelyn, Henrik H, Icy Defiance, Igor Ševo, iliketocode, Jatin Sanghvi, Jean-Bernard Pellerin, Jesse Williams, Jon Schoning, Kimmax, Kobi, Kris Vandermotten, Kritner, leppie, Llwyd, Maakep, maf-soft, Marc Gravell, MarcinJuraszek, Mariano Desanze, Matt Rowland, Matt Thomas, MemphiZ, mnoronha, MotKohn, Name, Nate Barbettini, Nico, Niek, nietras, NikolayKondratyev, Nuri Tasdemir, PashaPash, Pavel Mayorov, PeteGO, petrzjunior, Philippe, Pratik, Priyank Gadhiya, Pyritie, qJake, Raidri, Rakitić, RamenChef, Ray Vega, RBT, René Vogt, Rob, samuelesque , Squidward, Stavm, Stefano, Stefano d'Antonio, Stilgar, Tim Pohlmann, Uriil, user1304444, user2321864, user3185569, uTeisT, Uwe Keim, Vlad, Vlad, Wai Ha Lee, Wasabi Fan , WerWet, wezten, Wojciech Czerniak, Zze

15	C # Structs Union (C)	DLeh, Milton Hernandez, Squidward, usr
16	C #	Aaron Hudon, Andrew Diamond, Denuath, Jeremy Kato, Jon Schneider, Jorge, Juha Palomäki, Leon Husmann, Michael Mairegger, Michael Richardson, Nikita, rene, Rob, Sebi, TarkaDaal, wertzui, Will Ray
17	C #	mehrandvd, Squidward, Stephen Leppik
18	C #	Abbas Galiyakotwala
19	C # SQLite	Carmine, NikolayKondratyev, th1rdey3, Tim Yusupov
20	C #	Yashar Aliabasi
21	C #	A. Can Aydemir, Adi Lester, Alexander Mandt , DLeh, J3soon, Rob
22	CLSCompliantAttribute	mybirthname, Rob
23	DateTime	AbdulRahman Ansari, C4u, Christian Gollhardt, Felipe Oriani, Guilherme de Jesus Santos, James Hughes, Matas Vaitkevicius, midnightsyntax, Mostafiz , Oluwafemi, Pavel Yermalovich, Sondre, theinarasu, Thulani Chivandikwa
24	FileSystemWatcher	Sondre
25	Func	Theodoros Chatzigiannakis, Valentin

26	Google	4444, Supraj v
27	HTTP	Gordon Bell, Jon Schneider, Mark Shevchenko
28	ICloneable	ja72, Rob
29	IComparable	alex
30	IDisposable	Aaron Hudon, Adam, BatteryBackupUnit, binki, Bogdan Gavril, Bryan Crosby, ChrisWue, Dmitry Bychenko, Ehsan Sajjad, H. Pauwelyn, Jarrod Dixon, Josh Peterson, Matas Vaitkevicius, Maxime, Nicholas Sizer, OliPro007, Pavel Mayorov, pinkfloydx33, pyrocumulus, RamenChef, Rob, Thennarasan, Will Ray
31	IEnumerable	4444, Avia, Benjamin Hodgson, Luke Ryan, Olivier De Meulder, The_Outsider
32	ILGenerator	Aleks Andreev, thehennyy
33	INotifyPropertyChanged	mbrdev, Stephen Leppik, Vlad
34	IQueryable	lucavgobbi, Michiel van Oosterhout, RamenChef, Rob
35	json.net	Aleks Andreev, Snipzwolf
36	LINQ to XML	Denis Elkhov, Stephen Leppik, Uali
37	Linq	brijber, Christian Gollhardt, FortyTwo,

		Kevin Green, Raphael Pantaleão, Simon Halsey, Tanveer Badar
38	LINQ	Adam Clifford, Ade Stringer, Adi Lester, Adil Mammadov, Akshay Anand, Aleksey L., Alexey Koptyaev, AMW, anaximander, Andrew Piliser, Ankit Vijay, Aphelion, bbonch, Benjamin Hodgson, bmadtiger, BOBS, BrunoLM, Budi, bumbeishvili, callisto, cbale, Chad McGrath, Chris, Chris H., coyote, Daniel Argüelles, Daniel Corzo, darcyq, David, David G., David Pine, DavidG, die maus, Diligent Key Presser, Dmitry Bychenko, Dmitry Egorov, dotctor, Ehsan Sajjad, Erick, Erik Schierboom, EvenPrime, fabriciorissetto, faso, Finickyflame, Florin M, forsvarir, fubo, gbellmann, Gene, Gert Arnold, Gilad Green, H. Pauwelyn, Hari Prasad, hellyale, HimBromBeere, hWright, iliketocode, Ioannis Karadimas, Jagadisha B S, James Ellis-Jones, jao, jiaweizhang, Jodrell, Jon Bates, Jon G, Jon Schneider, Jonas S, karaken12, KevinM, Koopakiller, leppie, LINQ , Lohitha Palagiri, ltiveron, Mafii, Martin Zikmund, Matas Vaitkevicius, Mateen

		Ulhaq, Matt, Maxime, mburleigh, Meloviz, Mikko Viitala, Mohammad Dehghan, mok, Nate Barbettini, Neel, Neha Jain, Néstor Sánchez A., Nico, Noctis , Pavel Mayorov, Pavel Yermalovich, Paweł Hemperek, Pedro, Phuc Nguyen, pinkfloydx33, przno, qJake, Racil Hilan , rdans, Rémi, Rion Williams, rjdevereux, RobPethi, Ryan Abbott, S. Rangeley, S.Akbari, S.L. Barth, Salvador Rubio Martinez, Sanjay Radadiya, Satish Yadav, sebingel, Sergio Domínguez, SilentCoder, Sivanantham Padikkasu, slawekwin, Sondre, Squidward, Stephen Leppik, Steve Trout, Tamir Vered, techspider, teo van kot, th1rdey3, Theodoros Chatzigiannakis, Tim Iles , Tim S. Van Haren, Tobbe, Tom, Travis J, tungns304, Tushar patel, user1304444, user3185569, Valentin, varocarbas, VictorB, Vitaliy Fedorchenko, vivek nuna, void, wali, wertzui, WMios, Xiaoy312, Yaakov Ellis, Zev Spitz
39	Microsoft.Exchange.WebServices	Bassie
40	Null	Benjamin Hodgson, Braydie, DmitryG, Gordon Bell, Jasmin Solanki, Jon Schneider,

		Konstantin Vdovkin, Maximilian Ast, Mikko Viitala, Nicholas Sizer, Patrick Hofman, Pavel Mayorov, pinkfloydx33, Vitaliy Fedorchenko
41	Null-Coalescing	aashishkoirala, Ankit Rana, Aristos, Bradley Uffner, David Arno, David G., David Pine, demonplus, Denis Elkhov, Diligent Key Presser, Eamon Charles, Ehsan Sajjad, eouw0o83hf, Fernando Matsumoto, H. Pauwelyn , Jodrell, Jon Schneider, Jonesopolis, Martin Zikmund, Mike C, Nate Barbettini, Nic Foster, petelids, Prateek, Rahul Nikate, Rion Williams, Rob, smead, tonirush, Wasabi Fan, Will Ray
42	NullReferenceException	4444, Agramer, Ashutosh, krimog, Kyle Trauberman, Mathias Müller, Philip C, RamenChef, S.L. Barth, Shelby115, Squidward, vicky, Zikato
43	O (n)	AFT
44	ObservableCollection	demonplus, GeralexGR, Jonathan Anctil, MuiBienCarlota
45	String.Format	Aaron Hudon, Akshay Anand, Alexander Mandt , Andrius, Aseem Gautam, Benjol, BrunoLM, Dmitry Egorov , Don Vince, Dweeberly, ebattulga, ejhn5, gdoron, H. Pauwelyn, Hossein

		Narimani Rad, Jasmin Solanki, Marek Musielak, Mark Shevchenko, Matas Vaitkevicius, Mendhak, MGB, nikchi, Philip C, Rahul Nikate, Raidri, RamenChef, Richard, Richard, Rion Williams, ryanyuyu, teo van kot, Vincent, void, Wyck
46	StringBuilder	ATechieThought, brijber, Jeremy Kato, Jon Schneider, Robert Columbia, The_Outsider
47	System.DirectoryServices.Protocols.LdapConnection	Andrew Stollak
48	System.Management.Automation	Mikko Viitala
49	T4	lloyd, Pavel Mayorov
50	Windows Communication Foundation	NtFreX
51	Windows Form MessageBox	Mansel Davies, Vaibhav_Welcomes_You
52	XDocument System.Xml.Linq	Crowcoder, Jon Schneider
53	XML	Alexander Mandt, James , jHilscher, Jon Schneider, Nathan Tuggy, teo van kot, tsjnsn
54	XmlDocument System.Xml	Alexander Petrov, Rokey Ge, Rubens Farias, Timon Post, Willy David Jr
55	.	Wyck
56		Abdul Rehman Sayed, Adam, Amir Pourmand, Blubberguy22, Chronocide, Craig Brett, docesam, GWigWam,

		matiaslauriti, meJustAndrew, Michael Mairegger, Michele Ceo, Moe Farag, Nate Barbettini, RamenChef, Rob, scher, Snympi, Tagc, Theodoros Chatzigiannakis
57	GetHashCode	Alexey, BanksySan, hatcyl, ja72, Jeppe Stig Nielsen, meJustAndrew, Rob, scher, Timitry, viggity
58		Andrei, Kroltan, LeopardSkinPillBoxHat, Marco, Nick DeVore, Stephen Leppik
59		Almir Vuk, andre_ss6, Andrew Diamond, Barathon, Ben Aaronson , Ben Fogel, Benjol, David L, deloreyk, Ehsan Sajjad, harriyott, ja72, Jon Ericson, Karthik, Konamiman, MarcE, Matas Vaitkevicius, Pete Uh, Rion Williams, Robert Columbia, Steven , Suren Srapyan, VirusParadox, Yehuda Shapira
60		Akshay Anand, Nuri Tasdemir, tonirush
61		Dunno, Petr Hudeček, Stephen Leppik, TorbenJ
62		Timon Post
63		abto, Alexey Groshev, Benjamin Hodgson, Botz3000, David, Elad Lachmi, ganchito55, Jon

		Schneider, NikolayKondratyev
64		Bales, Facebamm
65		Andrei Epure, Boggin, Botond Balázs, richard
66		Racil Hilan, Rahul Nikate , Stephen Leppik
67		Alexander Mandt, David, F_V, Haseeb Asif, matteeyah, Patrick Hofman, Wai Ha Lee
68		Alpha, dazerdude, DLeh, Draken, George Duckett, Jon Schneider, Kobi, Max, Nathan, Nicholas Sizer, Rob, Stephen Leppik, tehDorf, Timothy Shields, topolm, Wasabi Fan
69		Adi Lester, Nicholaus Lawson, Salih Karagoz, shawty, Squirrel, Xander Luciano
70		Ade Stringer, ganchito55 , H. Pauwelyn, Karthik, Maximilian Ast, void
71		Aaron Hudon, Adam, Ben Aaronson, Benjamin Hodgson, Bradley Uffner , CalmBit, Cihan Yakar, CodeWarrior, EyasSH, Huseyin Durmus, Jasmin Solanki, Jeppe Stig Nielsen, Jon G, Jonas S, Matt, NikolayKondratyev, niksofteng, Rajput, Richa Garg, Sam Farajpour Ghamari, Shog9, Stu, Thulani Chivandikwa, trashr0x

72		Maxime, Mikko Viitala, The_Outsider, Will Ray
73		ATechieThought, ravindra, Rion Williams, The_Outsider, user2321864
74		Jan Bońkowski
75		Daryl, David, H. Pauwelyn, Kilazur, Mark Shevchenko, Nate Barbettini, Rob
76		Andrei Rînea, Benjamin Hodgson, Benjol, David L, David Pine, Federico Allocati, Feelbad Soussi Wolfgun DZ, Fernando Matsumoto, H. Pauwelyn , haim770, Matas Vaitkevicius, Matt Sherman, Michael Mairegger, Michael Richardson, NotEnoughData, Oly, RubberDuck, S.L. Barth, Sunny R Gupta, Tagc, Thriggle
77		Artificial Stupidity, Stephen Leppik, Tommy
78		Alisson, Andrei Rînea, B Hawkins, Benjamin Hodgson, Botond Balázs , connor, Dialecticus, DJCubed, Freelex, Jon Schneider, Oluwafemi, Racil Hilan, Squidward, Testing123, Tolga Evcimen
79	(Rx)	stefankmitph
80		jaycer, NotEnoughData, Racil Hilan

81		Adam Houldsworth, Ahmar, Akshay Anand, Alex Wiese, andre_ss6, Aphelion, Benjol, Boris Callens, Bradley Grainger, Bradley Uffner, bubbleking, Chris Marisic , ChrisWue, Cristian T, cubrr, Dan Ling, Danny Chen, dav_i, David Stockinger, dazerdude, Denis Elkhov, Dmitry Bychenko, Erik Schierboom, Florian Greinacher, gdoron, H. Pauwelyn, Herbstein, Jon Schneider, Jon Skeet, Jonesopolis, JT., Ken Keenan, Kev, Kobi, Kyle Trauberman, Lasse Vågsæther Karlsen, LegionMammal978, Lorentz Vedeler, Martin, Martin Zikmund, Maxime , Nuri Tasdemir, Peter K, Philip C, pid, René Vogt, Rion Williams, Ryan Abbott, Scott Koland, Sean, Sparrow, styfle, Sunny R Gupta, Sworgkh, Thaoden, The_Cthulhu_Kid, Tom Droste, Tot Zam, Zaheer Ul Hassan
82		Cihan Yakar, Danny Chen, mehrandvd, Pan, Pavel Mayorov, Stephen Leppik
83		RamenChef, Sibeesh Venu, Testing123, The_Outsider, Tim Yusupov
84		Arjan Einbu, ATechieThought, avs099 , bluray, Brendan L,

		Dave Zych, DLeh, Ehsan Sajjad, fabriciorissetto, Guilherme de Jesus Santos, H. Pauwelyn, Jon Skeet, Nate Barbettini, RamenChef, Rion Williams, Squidward, Stephen Leppik, Tushar patel, Wasabi Fan
85		Abdul Rehman Sayed, Callum Watkins, ChaoticTwist, Doruk, Dweeberly, Jon Schneider, Oluwafemi, Rob, RubberDuck, Testing123, The_Outsider
86		Benjol, Botond Balázs, cubrr, Ed Gibbs, Jeppe Stig Nielsen, LegionMammal978, Michael Richardson, Peter Gordon, Petr Hudeček, Squidward, tonirush
87		Blachshma, Jon Schneider, sferencik, The_Outsider
88	FormatException	Rakitić, un-lucky
89		Bovaz, Stephen Leppik, yumaikas
90		Botond Balázs, Lijo, Nate Barbettini, Tagc
91		Alexander Mandt, Aman Sharma, artemisart, Aseem Gautam, Axarydax, Benjamin Hodgson, Botond Balázs, Carson McManus, Cigano Morrison Mendez, Cihan Yakar, da_sann,

		DVJex, Ehsan Sajjad, H. Pauwelyn, Haim Bendanan, HimBromBeere, James Ellis-Jones, James Hughes, Jamie Rees, Jan Peldřimovský, Johny Skovdal, JSF, Kobi, Konamiman, Kristijan, Lovy, Matas Vaitkevicius , Mourndark, Nuri Tasdemir, pinkfloydx33, Rekshino, René Vogt, Sachin Chavan, Shuffler, Sjoerd222888, Sklivvz, Tamir Vered, Thriggle, Travis J, uygar.raf, Vadim Ovchinnikov, wablab, Wai Ha Lee
92		A_Arnold, Aaron Hudon, Alexey Groshev, Anas Tasadduq, Andrii Abramov, Baddie, Benjamin Hodgson, bluray, coyote, D.J., das_keyboard, Fernando Matsumoto, granmirupa, Jaydip Jadhav, Jeppe Stig Nielsen, Jon Schneider, Ogoun, RamenChef, Robert Columbia, Shyju, The_Outsider, Thomas Weller, tonirush, Tormod Haugene, Wasabi Fan, Wen Qin, Xiobiq, Yotam Salmon
93	LINQ (PLINQ)	Adi Lester
94		Ben Jenkinson, Jonas S, Rahul Nikate, Stephen Leppik, Taras, The_Outsider
95		Boggin, Jon Schneider, Oluwafemi, Tim

		Ebenezer
96	/ , ,	Dieter Meemken, Kyrylo M, nik, Pavel Mayorov, sebingel, Underscore, Xander Luciano, Yehor Hromadskyi
97		codeape, Mark Shevchenko, RamenChef
98		Aaron Hudon, AGB, aholmes, Ant P, Benjol, BrunoLM, Conrad.Dean, Craig Brett, Donald Webb, EJoshuaS, EvilTak, gdyrrahitis, George Duckett, Grimm The Opiner, Guanxi, guntbert, H. Pauwelyn, jdpilgrim, ken2k, Kevin Montrose, marshal craft, Michael Richardson, Moerwald, Nate Barbettini, nickguletskii, Pavel Mayorov, Pavel Voronin, pinkfloydx33, Rob, Serg Rogovtsev, Stefano d'Antonio, Stephen Leppik, SynerCoder, trashr0x, Tseng, user2321864, Vincent
99		Timon Post
100		Mohsin khan
101		Balen Danny, Benjamin Hodgson, Bovaz, Craig Brett, Dean Van Greunen, Gajendra, Jan Bońkowski, Kimmax, Marc Wittmann, Martin, Pavel Durov, René Vogt, RomCoo, Squidward

102		Adam Sills, Adi Lester, Adriano Repetti, Andrei Rînea, Andrew Diamond, Arjan Einbu, Avia, BackDoorNoBaby, BanksySan, Ben Fogel, Benjamin Hodgson, Benjol, Bogdan Gavril, Bovaz, Carlos Muñoz, Dan Hulme, Daryl, DLeh, Dmitry Bychenko, drusellers, Ehsan Sajjad, Fernando Matsumoto, guntbert, hatchet, Ian, Jeremy Kato, Jon Skeet, Julien Roncaglia, kamilk, Konamiman, ltiveron, Michael Richardson, Neel, Oly, Pavel Mayorov, Pavel Sapehin, Pavel Voronin, Peter Hommel, pinkfloydx33, Robert Columbia, RomCoo, Roy Dictus, Sam, Saravanan Sachi, Seph, Sklivvz, The_Cthulhu_Kid, Tim Medora, usr, Verena Haunschmid, void, Wouter, ZenLulz
103		Botond Balázs, Rahul Nikate, Sam Johnson, ZenLulz
104		Botond Balázs, Callum Watkins, Jeremy Kato, John, JohnLBevan, niksofteng, Stephen Leppik, Zohar Peled
105		Blorgbeard, hatchet, jaycer, Michael Sorens, Parth Patel, Stephen Leppik
106		Aaron Hudon, Andrew Diamond, Ben Aaronson

		, ChrisPatrick, Damon Smithies, David G., David Pine, Dmitry Bychenko, dotctor, Ehsan Sajjad, erfanrazi, Gajendra, George Duckett, H. Pauwelyn, HimBromBeere, Jeremy Kato, João Lourenço, Joe Amenta, Julien Roncaglia, just.ru, Karthik AMR, Mark Shevchenko, Michael Richardson, MuiBienCarlota, Myster, Nate Barbettini, Noctis, Nuri Tasdemir, Olivier De Meulder, OP313, ravindra, Ricardo Amores, Rion Williams, rocky, Sompom, Tot Zam, un-lucky, Vlad, void, Wasabi Fan, Xiaoy312, ZenLulz
107		Aaron Hudon, Alexander Petrov, Austin T French, captainjamie, Eldar Dordzhiev, H. Pauwelyn, ionmike, Jacob Linney, JohnLBevan, leondepdelaw, Mamta D, Matthijs Wessels, Mellow , RamenChef, Zoba
108		S.L. Barth
109		Adam, demonplus, dotctor, Gavin Greenwalt , Jeppe Stig Nielsen, Sondre
110	: Json with C #	Neo Vijay, Rob, VitorCioletti
111		Aaron Hudon, Adam, Adi Lester, Andrei Rînea, cbale, Disk Crasher,

		Ehsan Sajjad, Krzysztof Branicki, lothlarias, Mark Shevchenko, Pavel Mayorov, Sklivvz, snickro, Squidward, Squirrel, Stephen Leppik , Victor Tomaili, Xandrmoro
112		Bearington, Botond Balázs, elibyy, Jonas S, Osama AbuSitta, Sherantha, TarkaDaal, The_Outsider, Tim Ebenezer, void
113	(System.Security.Cryptography)	glaubergft, MikeS159, Ogglas, Pete
114		Botond Balázs, H. Pauwelyn, hatcyl, John, Justin Rohr, Kobi, Robert Woods, Thaoden, ZenLulz
115		Lokesh_Ram
116		Adam, Alexey Mitev, Durgpal Singh, Tolga Evcimen
117		Adam Houldsworth, Adi Lester, Adil Mammadov, Akshay Anand, Alan McBee, Avi Turner, Ben Fogel, Blorgbeard, Blubberguy22, Chris Jester-Young, David Basarab, DLeh, Dmitry Bychenko, dotctor, Ehsan Sajjad, fabriciorissetto, Fernando Matsumoto, H. Pauwelyn, Henrik H, Jake Farley, Jasmin Solanki, Jephron, Jeppe Stig Nielsen, Jesse Williams, Joe,

		JohnLBevan, Jon Schneider, Jonas S, Kevin Montrose, Kimmax , lokusking, Matas Vaitkevicius, meJustAndrew, Mikko Viitala, mmushtaq, Mohamed Belal, Nate Barbettini, Nico, Oly, pascalhein, Pavel Voronin, petelids, Philip C, Racil Hilan, RhysO, Robert Columbia, Rodolfo Fadino Junior, Sachin Joseph, Sam, slawekwin, slinzerthegod , Squidward, Testing123, TyCobb, Wasabi Fan, Xiaoy312, Zaheer Ul Hassan
118		Aaron Hudon, Abdul Rehman Sayed, Adrian Iftode, aholmes, alex, Blachshma, Chris Oldwood, Diligent Key Presser, dlatikay, Dmitry Bychenko, dove, Ghost4Man, H. Pauwelyn, ja72, Jon Schneider, Kit, konkked, Kyle Trauberman, Martin Zikmund, Matthew Whited, Maxime, mbrdev , Michael Mairegger, MuiBienCarlota, NikolayKondratyev, Osama AbuSitta, PSGuy , recursive, Richa Garg, Richard, Rob, sdgfsdh, Sergii Lischuk, Squirrel, Stefano d'Antonio, Tanner Swett, TarkaDaal , Theodoros Chatzigiannakis, vesi, Wasabi Fan, Yanai

119		0x49D1, Abdul Rehman Sayed, Adam Lear, Adil Mammadov, Andrew Diamond, Aseem Gautam, Athafoud, Botond Balázs, Collin Stevens, Danny Chen, Dmitry Bychenko, dove, Eldar Dordzhiev, fabriciorissetto, faso, flq, George Duckett, Gilad Naaman, Gudradain, Jack, James Hughes, Jamie Rees, John Meyer , Jonesopolis, MadddinTribleD, Marimba, Matas Vaitkevicius, Matt, matteeyah, Mendhak, Michael Bisbjerg, Nate Barbettini, Nathaniel Ford, nik0lias, niksofteng , Oly, Pavel Pája Halbich , Pavel Voronin, PMF, Racil Hilan, raidensan, Rasa , Robert Columbia, RomCoo, Sam Hanley, Scott Koland, Squidward , Steve Dunn, Thulani Chivandikwa, vesi
120		Chad, Danny Chen, heltonbiker, Kane, MotKohn, Philip C, pinkfloydx33, Racil Hilan, Rob, Robert Columbia, Sender, Sondre, Stephen Leppik, Wasabi Fan
121		Community, connor, Ehsan Sajjad, Lijo
122		Bad, Botond Balázs, Jonathan Zúñiga, MrDKOz, Ranjit Singh, Squidward

123		Buh Buh, iaminvinicble, Kyle Trauberman, Wiktor Dębski
124		Ben Aaronson, Callum Watkins, PMF, ZenLulz
125		Aaron Hudon, Adi Lester , Benjol, CheGuevarasBeret, dcastro, matteeyah, meJustAndrew, mhoward, nik, niksofteng , NotEnoughData, OliPro007, paulius_l, PSGuy, Reza Aghaei, Roy Dictus, Squidward, Steven, vbnet3d
126		David, Maxim, RamenChef, Stephen Leppik
127		Fernando Matsumoto, goric, Stephen Leppik
128		A_Arnold, Ehsan Sajjad, jHilscher
129		Avia, Botond Balázs, CyberFox, harriyott, hellyale, Jeremy Kato, MarcE, MSE, PMF, Preston, Sigh, Sometowngeek, Stagg, Steven, user2441511
130		4444, PedroSouki
131	C # (csc.exe)	delete me
132		Austin T French, Blachshma, bluish, CharithJ, Chief Wiggum, cyberj0g, Daryl, deloreyk , jaycer, Jaydip Jadhav, Jon G, Jon Schneider, juergen d, Konamiman, Maniero, Paul Weiland,

		Racil Hilan, RoelF, Stefan Steiger, Steven, The_Outsider, tiedied61, un-lucky, WizardOfMenlo
133		Benjamin Hodgson, Brandon, Collin Stevens, i3arnon, Mokhtar Ashour , Murtuza Vohra
134	(TPL)	Droritos, Stephen Leppik
135		Aaron Hudon, Alexey Groshev, Andrei Rînea, Benjamin Hodgson, Botond Balázs, Christopher Currens, Cihan Yakar, David Ben Knoble, Denis Elkhov, Diligent Key Presser, George Duckett, George Polevoy, Jargon, Jasmin Solanki, Jivan, Mark Shevchenko, Matas Vaitkevicius, Mikko Viitala, Nuri Tasdemir, Oluwafemi, Pavel Mayorov, Richard, Rob, Scott Hannen, Squidward, Vahid Farahmandian
136		Alexey Groshev, Botond Balázs, connor, ephtee, Florian Koch, Kroltan, Michael Brandon Morris, Mulder, Pan, qJake, Robert Columbia, Roy Dictus, SlaterCodes, Yves Schelpe
137		Andrei, Gilad Naaman, Matas Vaitkevicius, qJake, RamenChef, theB, volvis
138		C4u

139		MCronin, The_Outsider, Xiaoy312
140		AGB, andre_ss6, Ben Aaronson, Benjamin Hodgson, Benjol, Bobson, Carsten, darth_phoenixx, dymanoid, Eamon Charles, Ehsan Sajjad, Gajendra, GregC, H. Pauwelyn, ja72, Jim, Kroltan, Matas Vaitkevicius, mehmetgil, meJustAndrew, Mord Zuber, Mujassir Nasir, Oly, Pavel Voronin, Richa Garg, Sam, Sebi, Sjoerd222888, Theodoros Chatzigiannakis, user3185569, VictorB, void, Wallace Zhang
141		Alexander Mandt, Ameya Deshpande, EJoshuaS, H. Pauwelyn, Hayden, Kroltan, RamenChef, Sklivvz
142		Benjamin Hodgson, MSE , RamenChef, StriplingWarrior
143		Fernando Matsumoto, Jesse Williams, JohnLBevan, Kit, Michael Freidgeim, Nuri Tasdemir, RamenChef, Tot Zam
144		Jasmin Solanki, Luke Ryan, TylerH
145		DWright, Jan Bońkowski, Mark Shevchenko, Parth Patel, PedroSouki, Pierre Theate, Sondre,

		Tushar patel
146		Alan McBee, Amitay Stern, Andrew Diamond, Aphelion, Arjan Einbu, avb, Bryan Crosby, Charlie H, David G., devuxer, DLeh, Ehsan Sajjad, Freelex, goric, Jared Hooper, Jeremy Kato, Jonas S, Kevin Montrose, Kilazur, Mateen Ulhaq, Ricardo Amores, Rion Williams, Sam Johnson, Sophie Jackson-Lee, Squirrel, th1rdey3
147		Aliaksei Futryn, th1rdey3
148		Aphelion, ASh, Bart Jolling, Chronocide, CodeCaster, CyberFox, DLeh, Jacob Linney, Jeromy Irvine, Jonas S, Matas Vaitkevicius, Rob, robert demartino, rudygt, Squidward, Tamir Vered, TarkaDaal, Thulani Chivandikwa, WMios
149		MegaTron
150		Roy Dictus
151		4444, A_Arnold, Aaron Hudon, Ade Stringer, Adi Lester, Aditya Korti, Adriano Repetti, AJ., Akshay Anand, Alex Filatov, Alexander Pacha , Amir Pourmand, Andrei Rînea, Andrew Diamond, Angela, Anna, Avia, Bart , Ben, Ben Fogel, Benjamin Hodgson, Bjørn-Roger Kringsjå, Botz3000, Brandon,

		brijber, BrunoLM, BunkerMentality, BurnsBA, bwegs, Callum Watkins, Chris, Chris Akridge, Chris H., Chris Skardon, ChrisPatrick, Chuu, Cihan Yakar, cl3m , Craig Brett, Daniel, Daniel J.G., Danny Chen , Darren Davies, Daryl, dasblinkenlight, David, David G., David L, David Pine, DAXaholic, deadManN, DeanoMachino, digitlworld, Dmitry Bychenko, dotctor, DPenner1, Drew Kennedy, DrewJordan, Ehsan Sajjad, EJoshuaS , Elad Lachmi, Eric Lippert, EvenPrime, F_V, Felix, fernacolo, Fernando Matsumoto, forsvarir, Francis Lord, Gavin Greenwalt, gdoron , George Duckett, Gilad Naaman, goric, greatwolf , H. Pauwelyn, Happypig375, Icemanind , Jack, Jacob Linney, Jake, James Hughes, Jcoffman, Jeppe Stig Nielsen, jHilscher, João Lourenço, John Slegers, JohnD, Jon Schneider, Jon Skeet, JoshuaBehrens, Kilazur, Kimmax, Kirk Woll, Kit, Kjartan, kjhf, Konamiman , Kyle Trauberman, kyurthich, levininja, lokusking, Mafii, Mamta D, Mango Wong, MarcE, MarcinJuraszek, Marco Scabbiolo, Martin, Martin Klinke, Martin Zikmund,

		Matas Vaitkevicius, Mateen Ulhaq, Matěj Pokorný, Mat's Mug, Matthew Whited, Max, Maximilian Ast, Medeni Baykal, Michael Mairegger, Michael Richardson, Michel Keijzers, Mihail Shishkov , mike z, Mr.Mindor, Myster, Nicholas Sizer, Nicholaus Lawson, Nick Cox, Nico, nik, niksofteng, NotEnoughData, numaroth, Nuri Tasdemir , pascalhein, Pavel Mayorov, Pavel Pája Halbich, Pavel Yermalovich, Pavieł Kraskoŭski, Paweł Mach, petelids, Peter Gordon, Peter L., PMF, Rakitić, RamenChef, ranieuwe, Razan, RBT, Renan Gemignani, Ringil, Rion Williams, Rob, Robert Columbia, ro binmckenzie, RobSiklos, Romain Vincent, RomCoo, ryanyuyu, Sain Pradeep, Sam, Sándor Mátyás Márton, Sanjay Radadiya, Scott, sebingel, Skipper, Sobieck, sohnryang, somebody, Sondre, Squidward, Stephen Leppik, Sujay Sarma, Suyash Kumar Singh, svick, TarkaDaal, th1rdey3, Thaoden, Theodoros Chatzigiannakis, Thorsten Dittmar, Tim Ebenezer, titol, tonirush,

		topolm, Tot Zam, user3185569, Valentin, vcsjones, void, Wasabi Fan, Wavum, Woodchipper, Xandrmoro, Zaheer Ul Hassan, Zalomon, Zohar Peled
152		Adam, Akshay Anand, Benjamin Kozuch, ephtee, RamenChef, Thennarasan
153		Bovaz, Chawin, EFrank, H. Pauwelyn, Mark Benovsky, Muhammad Albarmawi, Nathan Tuggy, Nikita, Nuri Tasdemir, petrzjunior, PMF, RaYell, slawekwin, Squidward, tire0011
154	I / O	BanksySan, Blachshma, dbmuller, DJCubed, Feelbad Soussi Wolfgun DZ, intox, Mikko Viitala, Sender, Squidward, Tolga Evcimen, Wasabi Fan
155		Vadim Martynov
156		Jeppe Stig Nielsen, Theodoros Chatzigiannakis
157		Aaron Hudon, Botond Balázs, undefined
158		Benjamin Hodgson, dasblinkenlight, Dileep, George Duckett, just.another.programmer, Matas Vaitkevicius, matteeyah, meJustAndrew, Nathan Tuggy,

		NikolayKondratyev, Rob, Ruben Steins, Stephen Leppik, Рахул Маквана
159		Jan Bońkowski
160		Adi Lester, Callum Watkins, EvenPrime, ganchito55, Igor, jHilscher, RamenChef, ZenLulz
161		Botz3000, F_V, fubo, H. Pauwelyn, Icy Defiance, Jasmin Solanki, Jeremy Kato, Jon Schneider, ken2k, Marco, meJustAndrew, MSL, S.Dav, Sjoerd222888, TarkaDaal, un-lucky
162		Aaron Hudon, AbdulRahman Ansari, Adi Lester, Adil Mammadov, AGB, AldoRomo88, anaximander, Aphelion, Ashwin Ramaswami, ATechieThought, Ben Aaronson, Benjol, binki, Bjørn-Roger Kringsjå, Blachshma, Blorgbeard, Brett Veenstra, brijber, Callum Watkins, Chad McGrath, Charlie H, Chris Akridge, Chronocide, CorrectorBot, cubrr, Dan-Cook, Daniel Stradowski, David G., David Pine, Deepak gupta, diiN_____, DLeh, Dmitry Bychenko, DoNot, DWright, Đan, Ehsan Sajjad, ekolis, el2iot2, Elton, enrico.bacis, Erik

Schierboom, ethorn10, extremeboredom, Ezra, fahadash, Federico Allocati, Fernando Matsumoto, FrankerZ, gdziadkiewicz, Gilad Naaman, GregC, Gudradain, H. Pauwelyn, HimBromBeere, Hsu Wei Cheng, Icy Defiance, Jamie Rees, Jeppe Stig Nielsen, John Peters, John Slegers, Jon Erickson, Jonas S, Jonesopolis, Kev, Kevin Avignon, Kevin DiTraglia , Kobi, Konamiman, krillgar, Kurtis Beavers, Kyle Trauberman, Lafexlos, LMK, lothlarias, Lukáš Lánský, Magisch, Marc, MarcE, Marek Musielak, Martin Zikmund, Matas Vaitkevicius, Matt, Matt Dillard, Maximilian Ast, mbrdev, MDTech.us_MAN, meJustAndrew, Michael Benford, Michael Freidgeim, Michael Richardson, Michał Perłakowski, Nate Barbettini, Nick Larsen, Nico, Nisarg Shah, Nuri Tasdemir, Parth Patel, pinkfloydx33, PMF, Prashanth Benny, QoP, Raidri, Reddy, Reeven, Ricardo Amores, Richard , Rion Williams, Rob, Robert Columbia, Ryan Hilbert, ryanyuyu, S. Tarık Çetin, Sam Axe, Shoe , Sibeesh Venu, solidcell , Sondre, Squidward, Steven, styfle, SysVoid,

		Tanner Swett, Timothy Rascher, TKharaishvili, T-moty, Tobbe, Tushar patel, unarist, user3185569, user40521, Ven, Victor Tomaili, viggity
163		Danny Bogers, jlawcordova, Jon Schneider, Nuri Tasdemir, Pushpendra