LEARNING dependency-injection

#dependency-injection

Table of Contents

About

Chapter 1: Getting started with dependency-injection

Remarks

In object-oriented programming, objects often depend on other objects in order to do things.

Dependency Injection (DI) is giving an object the things that it depends on so that it doesn't have to worry about getting them itself. That is, the dependencies are *injected* into the object. This is most often done with *constructor injection* or *property injection*.

Dependency injection is a form of **Inversion of Control** (IoC). IoC is a broader term that describes a pattern of software design.

In traditional procedural programming, the flow of control follows logically in steps. The control is in the hands of the object or function performing operations. Step-by-step the program performs a series of operations that it controls explicitly.

Instead of the object or function detailing every step, the flow of control can be *inverted* by making the operations be performed by more generic and abstract objects - usually a framework that is broader in scope.

Examples

What is a basic example of dependency injection?

Here is a class (Dog) creating its own dependency (Food):

```
class Dog {
    public Dog() {
        var food = new Food();

        this.eat(food);
    }
}
```

Here is the same class being injected with its dependency using constructor injection:

```
class Dog {
    public Dog(Food food) {
        this.eat(food);
    }
}
```

Read Getting started with dependency-injection online: https://riptutorial.com/dependency-injection/topic/4361/getting-started-with-dependency-injection

Chapter 2: .NET - Pure DI examples

Remarks

An example of how to use dependency injection in .net without using a container. Based on examples by Mark Seemann http://blog.ploeh.dk/

Examples

Web Api

```
public interface ISingleton : IDisposable { }
public class TransientDependency { }

public class Singleton : ISingleton
{
    public void Dispose() { }
}

public class CompositionRoot : IDisposable, IHttpControllerActivator
{
    private readonly ISingleton _singleton;

    // pass in any true singletons i.e. cross application instance singletons
    public CompositionRoot()
    {
        // intitialise any application instance singletons
        _singleton = new Singleton();
    }

    public void Dispose()
    {
        _singleton.Dispose();
    }

    public IHttpController Create(HttpRequestMessage request, HttpControllerDescriptor
controllerDescriptor, Type controllerType)
    {
        // Per-Request-scoped services are declared and initialized here
        if (controllerType == typeof(SomeApiController))
        {
            // Transient services are created and directly injected here
            return new SomeApiController(_singleton, new TransientDependency());
        }

        var argumentException = new ArgumentException(@"Unexpected controller type! " +
controllerType.Name,
            nameof(controllerType));
        Log.Error(argumentException, "don't know how to instantiate API controller:
{controllerType}", controllerType.Name);
        throw argumentException;
    }
}
```

```
public static class DependencyInjection
{
    public static void WireUp()
    {
        var compositionRoot = new CompositionRoot();
        System.Web.Http.GlobalConfiguration.Configuration.Services.Replace(typeof
(IHttpControllerActivator), compositionRoot);
    }
}
```

MVC

```
public interface ISingleton : IDisposable { }
public class TransientDependency { }

public class Singleton : ISingleton
{
    public void Dispose() { }
}

public class CompositionRoot : IDisposable, IControllerFactory
{
    private readonly ISingleton _singleton;

    // pass in any true singletons i.e. cross application instance singletons
    public CompositionRoot()
    {
        // intitialise any application instance singletons
        _singleton = new Singleton();
    }

    public void Dispose()
    {
        _singleton.Dispose();
    }

    public IController CreateController(RequestContext requestContext, string controllerName)
    {
        if (controllerName.ToLower() == "home")
        {
            return new HomeController(_singleton, new TransientDependency());
        }

        var argumentException = new ArgumentException(@"Unexpected controller! " +
controllerName);
        Log.Error("don't know how to instantiate MVC controller: {controllerType}. redirecting
to help", controllerName);
        throw argumentException; // Alternatively would return some default Page Not Found
placeholder controller;
    }

    public SessionStateBehavior GetControllerSessionBehavior(RequestContext requestContext,
string controllerName)
    {
        return SessionStateBehavior.Default;
    }

    public void ReleaseController(IController controller)
    {
        // anything to clean up?
    }
}

public static class DependencyInjection
{
    public static void WireUp()
    {
        var compositionRoot = new CompositionRoot();
        System.Web.Mvc.ControllerBuilder.Current.SetControllerFactory(compositionRoot);
    }
}
```

ASP.NET Core

in Startup.cs

```
// This method gets called by the runtime. Use this method to add services to the container.
public void ConfigureServices(IServiceCollection services)
{
    // Add framework services.
    services.AddMvc();

    var controllerActivator = new CompositionRoot();
    services.AddSingleton<IControllerActivator>(controllerActivator);
}
```

CompositionRoot.cs

```
public class CompositionRoot : IControllerActivator, IDisposable
    {
        // Singletons
        private readonly ISingleton _singleton;

        public CompositionRoot()
        {
            // Create singletons
            _singleton = new Singleton();
        }

        public object Create(ControllerContext c) =>
this.Create(c.ActionDescriptor.ControllerTypeInfo.AsType());
        public void Release(ControllerContext c, object controller) => (controller as
IDisposable)?.Dispose();

        public Controller Create(Type type)
        {
            // scoped
            var scoped = new Scoped();
            // transient get new()-ed up below
            if (type == typeof(HomeController))
                return new HomeController(_singleton, scoped, new Transient());

            throw new InvalidOperationException($"Unknown controller {type}.");
        }

        public void Dispose()
        {
            // dispose stuff
        }
    }
```

Read .NET - Pure DI examples online: https://riptutorial.com/dependency-injection/topic/4556/-net---pure-di-examples

Chapter 3: Constructor Injection

Examples

Constructor Injection

Classes have instance variable (dependencies), on which they call methods.

Example taken from http://www.jamesshore.com/Blog/Dependency-Injection-Demystified.html for reference

```
public class Example {
  private DatabaseThingie myDatabase;

  public Example() {
    myDatabase = new DatabaseThingie();
  }

  public void DoStuff() {
    ...
    myDatabase.GetData();
    ...
  }
}
```

This class has one dependency called `DatabaseThingie`. Here in this example class **Example** is responsible for creating its own dependencies, thereby violating **Single Responsibility Principle**. Class has some primary responsibility + it is creating its won dependencies. If dependency creation mechanism changes, Let say now the dependency take argument or Instead of DatabaseThingie I want some other type. The class will change.

Injecting dependency from External Source

```
public class Example {
  private DatabaseThingie myDatabase;

  public Example(DatabaseThingie useThisDatabaseInstead) {
    myDatabase = useThisDatabaseInstead;
  }

  public void DoStuff() {
    ...
    myDatabase.GetData();
    ...
  }
}
```

Here we are Injecting dependency from external source using **Constructor Injection** (Passing dependency in constructor)

We don't care what type of dependency is injected, We just use it. The class won't change due to this.

The main benefit of Dependency Injection is when writing **tests**.

```
public class ExampleTest {

  void testDoStuff() {
    MockDatabase mockDatabase = new MockDatabase();

    // MockDatabase is a subclass of DatabaseThingie, so we can
    // "inject" it here:
    Example example = new Example(mockDatabase);

    example.DoStuff();
    mockDatabase.AssertGetDataWasCalled();
  }
}
```

Read Constructor Injection online: https://riptutorial.com/dependency-injection/topic/5474/constructor-injection

Chapter 4: Method injection

Examples

A Simple Example of Method Injection in c#

```
using System;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            var foo = new Dependency();
            var bar = new ClassWithDependency();

            bar.DoSomething(foo); //Inject dependency as method parameter

            Console.ReadLine();
        }
    }

    public interface IDependency
    {
        void DoSomething();
    }

    public class Dependency: IDependency
    {
        public void DoSomething()
        {
            Console.WriteLine("Hello");
        }
    }

    public class ClassWithDependency
    {
        public void DoSomething(IDependency dependency)
        {
            dependency.DoSomething();
        }
    }
}
```

Read Method injection online: https://riptutorial.com/dependency-injection/topic/6638/method-injection

Chapter 5: Property Injection

Examples

A Very Simple Example of Property Injection with C# with a Lazy-loaded Local Default

```
using System;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            var foo = new ClassWithDependency();

            foo.DoSomething();

            var bar = new InjectedDependency();

            foo.Dependency = bar; //Injecting the dependency via a property

            foo.DoSomething();

            Console.ReadLine();
        }

    }

    public interface IDependency
    {
        void DoSomething();
    }

    public class DefaultDependency: IDependency
    {
        public void DoSomething()
        {
            Console.WriteLine("Default behaviour");
        }
    }

    public class InjectedDependency: IDependency
    {
        public void DoSomething()
        {
            Console.WriteLine("Different behaviour");
        }
    }

    public class ClassWithDependency
    {
        private IDependency _dependency;

        public IDependency Dependency
        {
            get
            {
                if (_dependency == null) Dependency = new DefaultDependency();
                return _dependency;
            }
            set { _dependency = value; }
        }

        public void DoSomething()
        {
            Dependency.DoSomething();
        }
    }
}
```

A Simple Example of Property Injection, setting the default via constructor injection

```
using System;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            var dep = new DefaultDependency();
            var foo = new ClassWithDependency(dep);

            foo.DoSomething();

            var bar = new InjectedDependency();

            foo.Dependency = bar; //Injecting the dependency via a property

            foo.DoSomething();

            Console.ReadLine();
        }

    }

    public interface IDependency
    {
        void DoSomething();
    }

    public class DefaultDependency: IDependency
    {
        public void DoSomething()
        {
            Console.WriteLine("Default behaviour");
        }
    }

    public class InjectedDependency: IDependency
    {
        public void DoSomething()
        {
            Console.WriteLine("Different behaviour");
        }
    }

    public class ClassWithDependency
    {
        public ClassWithDependency(IDependency dependency)
        {
            Dependency = dependency;
        }

        public IDependency Dependency { get; set; }

        public void DoSomething()
        {
            Dependency.DoSomething();
        }
    }
}
```

Read Property Injection online: https://riptutorial.com/dependency-injection/topic/6288/property-injection

Credits

S. No	Chapters	Contributors
1	Getting started with dependency-injection	Community, Paul D'Ambra, Rowan Freeman
2	.NET - Pure DI examples	tomliversidge
3	Constructor Injection	Alex P, Irfan
4	Method injection	mark_h
5	Property Injection	mark_h