Kostenloses eBook

LERNEN

Visual Basic .NET Language

Free unaffiliated eBook created from
Stack Overflow contributors.

#vb.net

Inhaltsverzeichnis

Über

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

It is an unofficial and free Visual Basic .NET 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 Visual Basic .NET 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

Kapitel 1: Erste Schritte mit Visual Basic .NET Language

Bemerkungen

Visual Basic .NET ist der offizielle Nachfolger der ursprünglichen Visual Basic-Programmiersprache von Microsoft. Visual Basic [.NET] scheint Ähnlichkeiten mit Python zu haben, da es keine Semikolons und Klammern gibt, teilt jedoch mit C ++ die grundlegende Struktur von Funktionen. Geschweifte Klammern sind in VB .NET nicht vorhanden, sondern werden durch Phrasen wie `End If` , `Next` und `End Sub` .

Versionen

VB.NET Version	Visual Studio-Version	.NET Framework-Version	Veröffentlichungsdatum
7,0	2002	1,0	2002-02-13
7.1	2003	1.1	2003-04-24
8,0	2005	2,0 / 3,0	2005-10-18
9,0	2008	3,5	2007-11-19
10,0	2010	4,0	2010-04-12
11,0	2012	4,5	2012-08-15
12,0	2013	4.5.1 / 4.5.2	2013-10-17
14,0	2015	4.6.0 ~ 4.6.2	2015-07-20
15,0	2017	4.7	2017-03-07

Examples

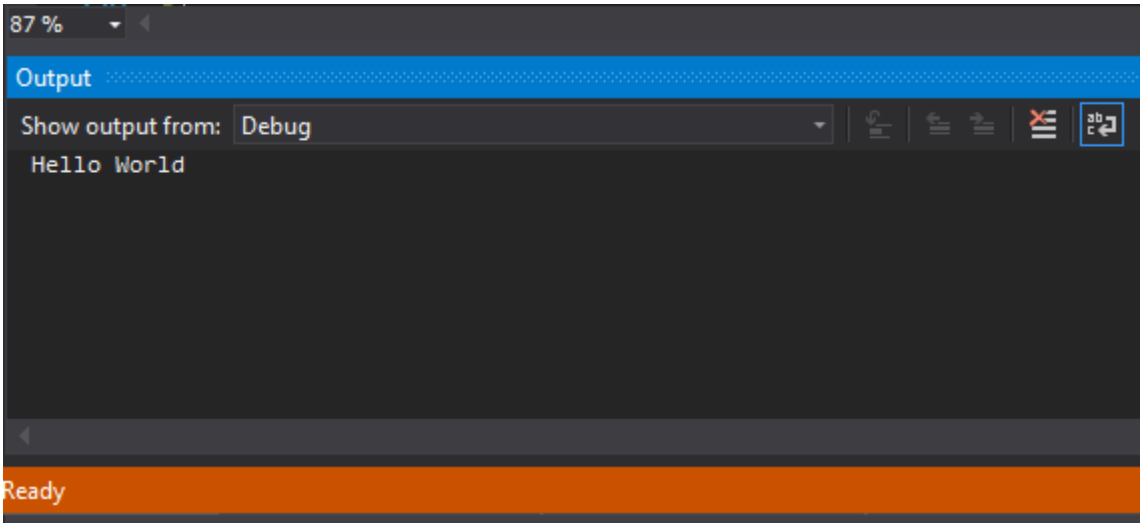
Hallo Welt

Installieren Sie zunächst eine Version von Microsoft Visual Studio , einschließlich der kostenlosen Community Edition. Erstellen Sie dann ein Visual Basic Console Application-Projekt des Typs *Console Application* , und der folgende Code druckt die Zeichenfolge `'Hello World'` an die Konsole:

```
Module Module1
```

```
    Sub Main()
        Console.WriteLine("Hello World")
    End Sub

End Module
```

Speichern Sie dann und drücken Sie `F5` auf der Tastatur (oder *rufen Sie* das *Debug*- Menü auf und klicken *Sie auf Ohne Debug* oder *Ausführen*), um das Programm zu kompilieren und auszuführen. `'Hello World'` sollte im Konsolenfenster angezeigt werden.



Hallo Welt in einem Textfeld beim Klicken eines Buttons

Ziehen Sie 1 Textfeld und 1 Schaltfläche

Doppelklicken Sie auf button1, und Sie werden zum `Button1_Click event`

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click

    End Sub
End Class
```

`textbox1` Sie den Namen des Objekts ein, für das Sie ein `textbox1` . In unserem Fall handelt es sich um das `textbox1` . `.Text` ist die Eigenschaft, die wir verwenden möchten, wenn wir einen Text darauf `.Text` möchten.

`Property Textbox.Text, gets or sets the current text in the TextBox` . Jetzt haben wir `Textbox1.Text`

Wir müssen den Wert dieser `Textbox1.Text` damit wir das `=` -Zeichen verwenden. Der Wert, den wir

in `Textbox1.Text` ist `Hello World` . Insgesamt ist dies der Gesamtcode für die `Textbox1.Text` eines Wertes von `Hello World` in `Textbox1.Text`

```
TextBox1.Text = "Hello World"
```

Hinzufügen dieses Codes zum `clicked event` von `button1`

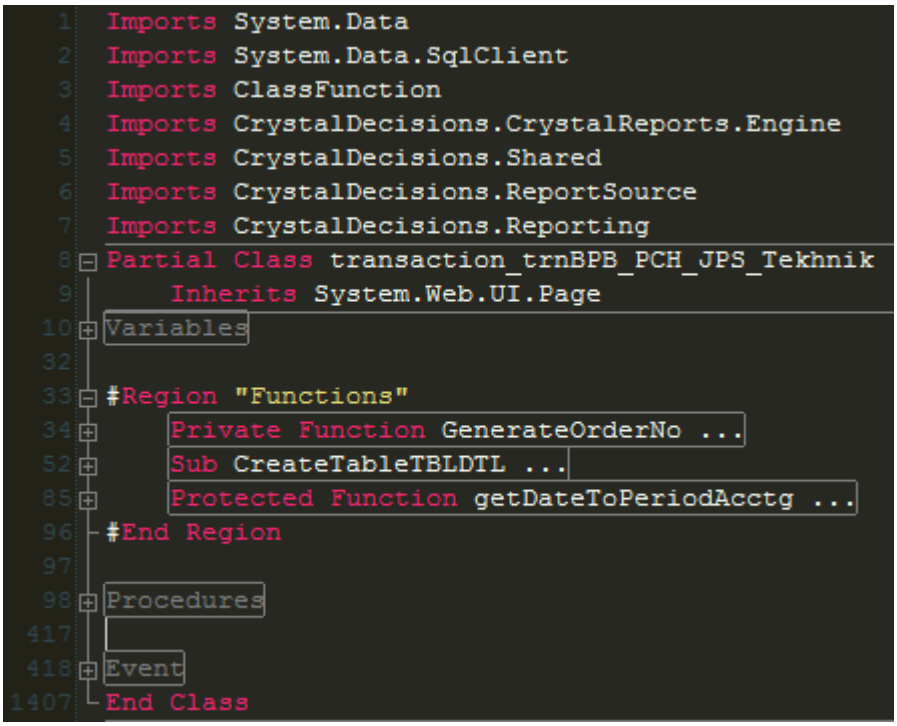
```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        TextBox1.Text = "Hello World"
    End Sub
End Class
```

Region

Aus Gründen der Lesbarkeit, die für Anfänger beim Lesen von VB-Code sowie für Vollzeitentwickler zur Pflege des Codes nützlich sein wird, können Sie mit "Region" einen Bereich derselben Gruppe von Ereignissen, Funktionen oder Variablen festlegen:

```
#Region "Events"
    Protected Sub txtPrice_TextChanged(...) Handles txtPrice.TextChanged
        'Do the ops here...
    End Sub

    Protected Sub txtTotal_TextChanged(...) Handles txtTotal.TextChanged
        'Do the ops here...
    End Sub

    'Some other events....

#End Region
```

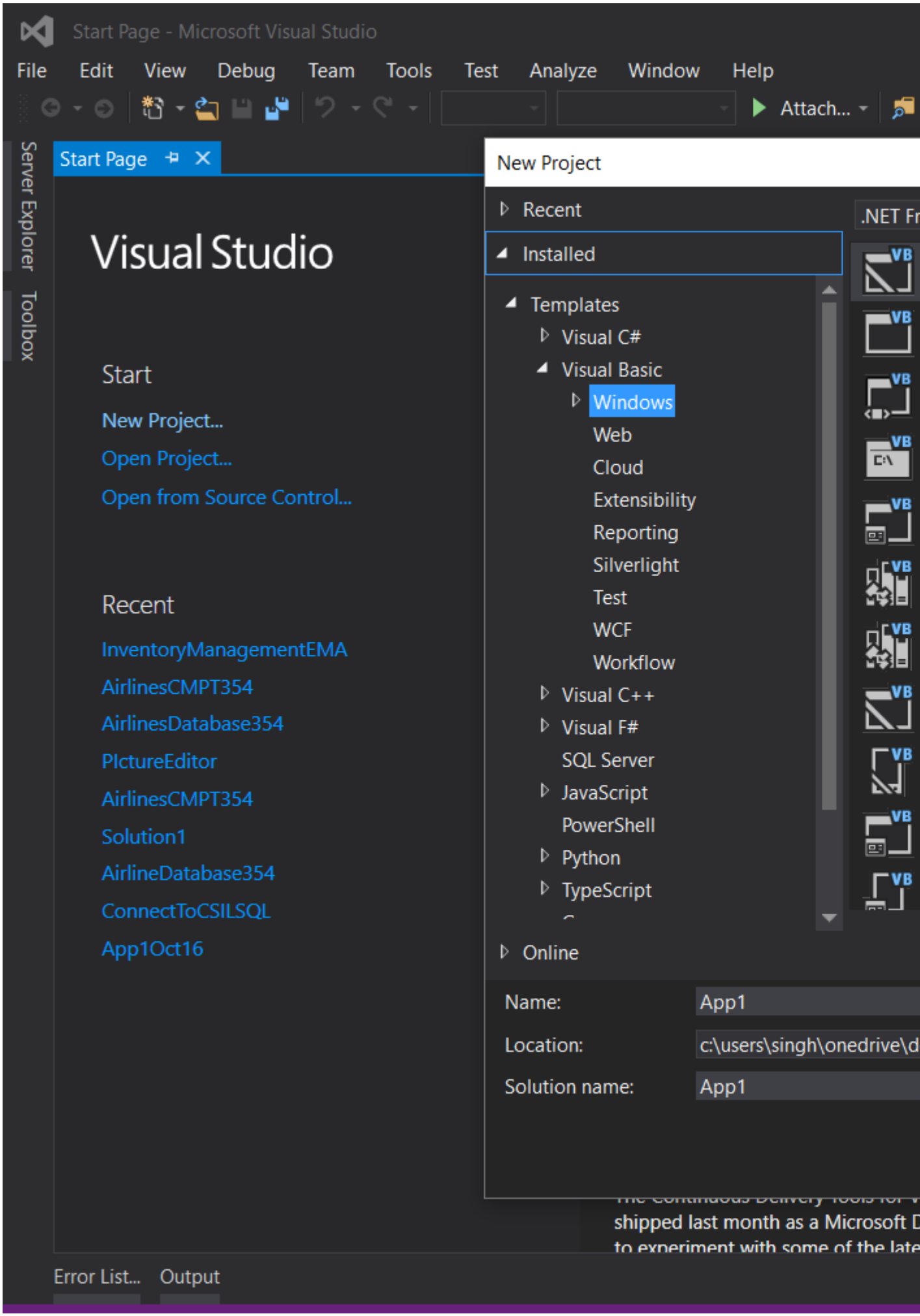
Dieser Bereichsblock könnte reduziert werden, um visuelle Hilfe zu erhalten, wenn die Codezeile 1000+ überschreitet. Es spart auch Ihre Scroll-Bemühungen.



Getestet am VS 2005, 2008 2010, 2015 und 2017.

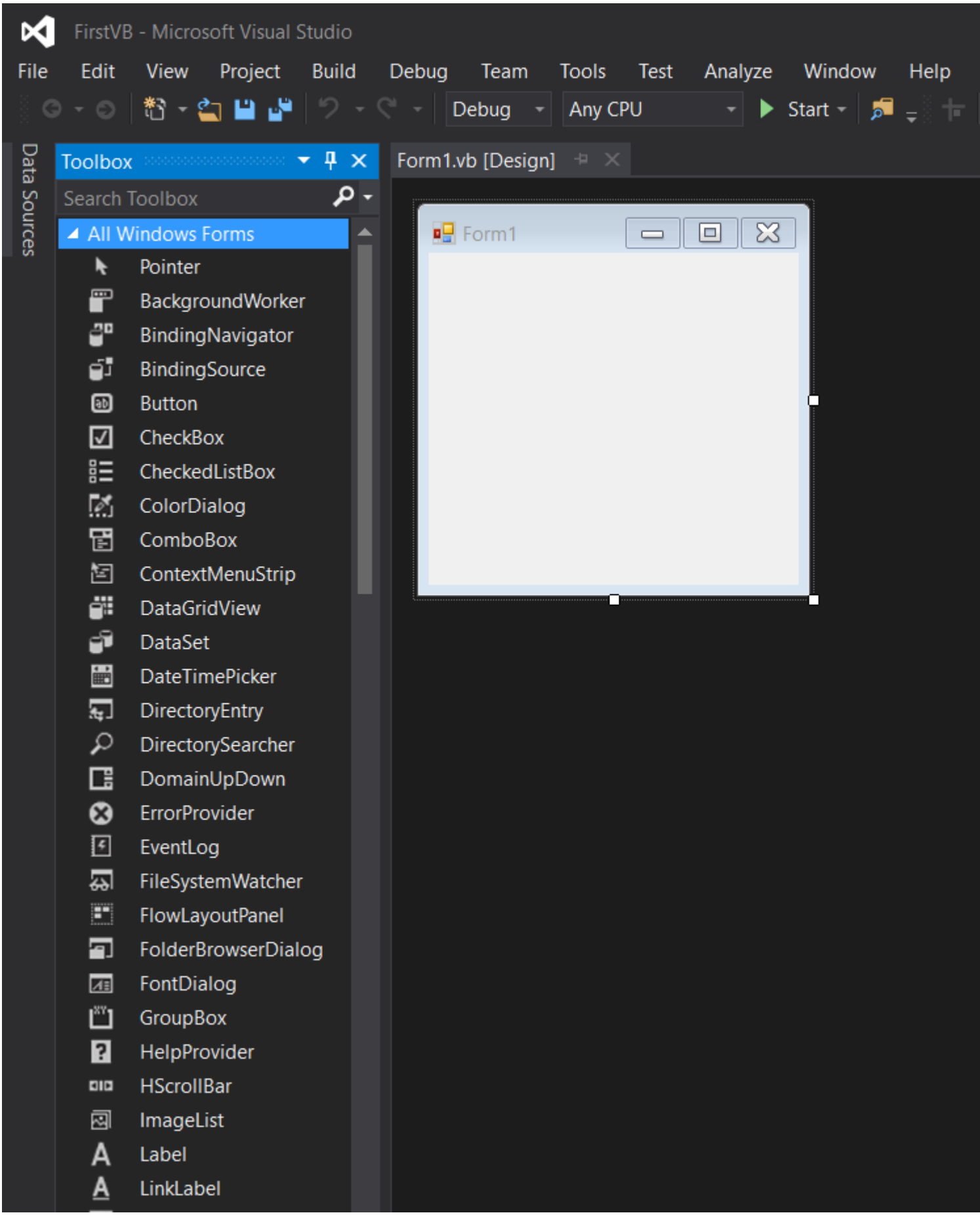
Erstellen eines einfachen Rechners, um sich mit der Schnittstelle und dem Code vertraut zu machen.

1. Starten Sie nach der Installation von Visual Studio von https://www.visualstudio.com/downloads/ ein neues Projekt.

2. 
Start Page - Microsoft Visual Studio
File Edit View Debug Team Tools Test Analyze Window Help
Attach...
Server Explorer
Toolbox
Start Page
Visual Studio
Start
New Project...
Open Project...
Open from Source Control...
Recent
InventoryManagementEMA
AirlinesCMPT354
AirlinesDatabase354
PIctureEditor
AirlinesCMPT354
Solution1
AirlineDatabase354
ConnectToCSILSQL
App1Oct16
New Project
Recent
Installed
Templates
Visual C#
Visual Basic
Windows
Web
Cloud
Extensibility
Reporting
Silverlight
Test
WCF
Workflow
Visual C++
Visual F#
SQL Server
JavaScript
PowerShell
Python
TypeScript
Online
Name: App1
Location: c:\users\singh\onedrive\d
Solution name: App1
shipped last month as a Microsoft D
Error List... Output

3. Wählen Sie "Windows Forms-Anwendung" auf der Registerkarte "Visual Basic" aus. Sie können es hier umbenennen, wenn Sie müssen.

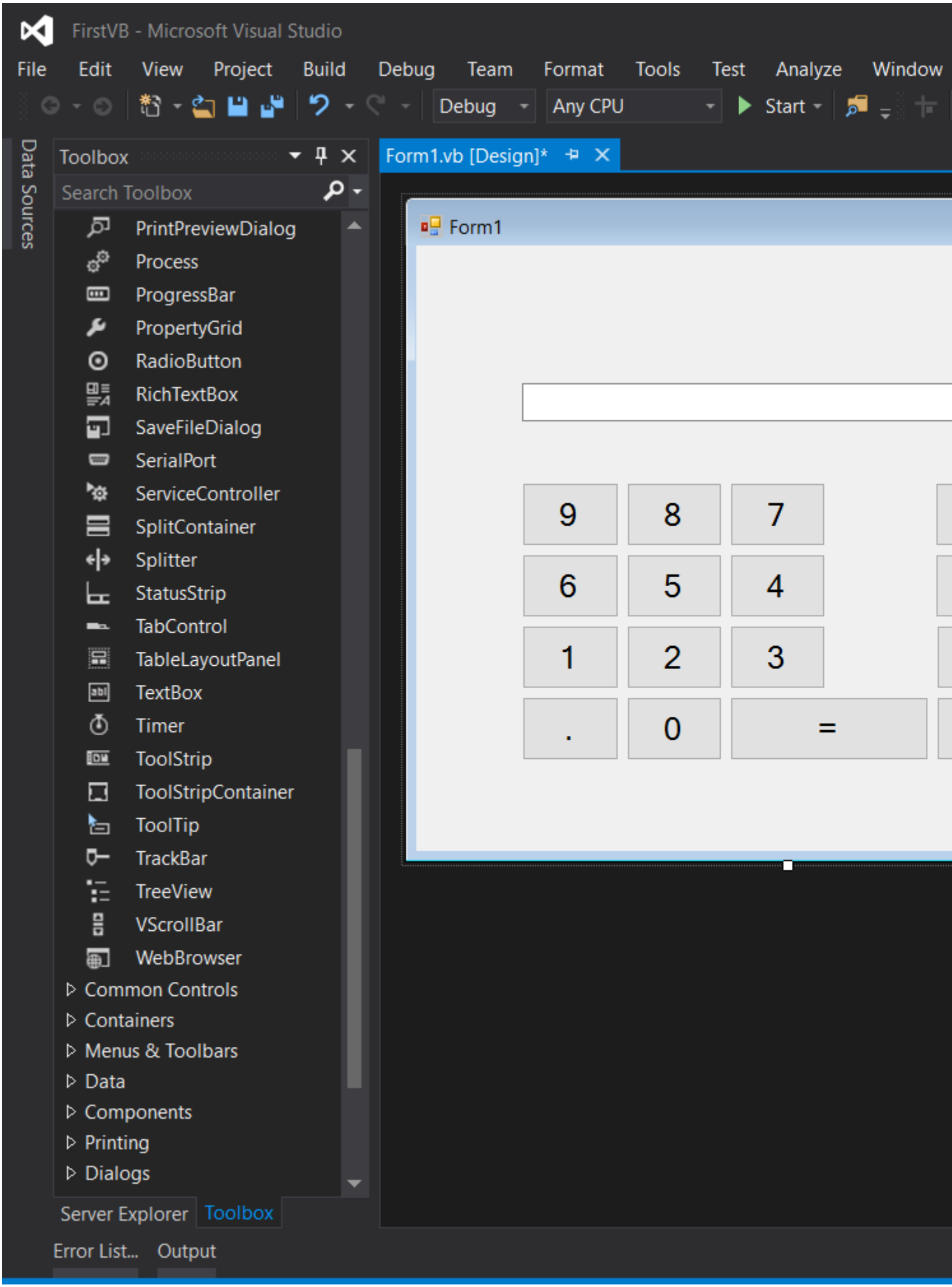
4. Sobald Sie auf "OK" klicken, wird dieses Fenster angezeigt:



5. Klicken Sie links auf die Registerkarte "Toolbox". In der Symbolleiste ist die Option zum automatischen Ausblenden standardmäßig aktiviert. Um diese Option zu deaktivieren,

klicken Sie auf das kleine Symbol zwischen dem Pfeil nach unten und dem Symbol "x" in der oberen rechten Ecke des Toolbox-Fensters.

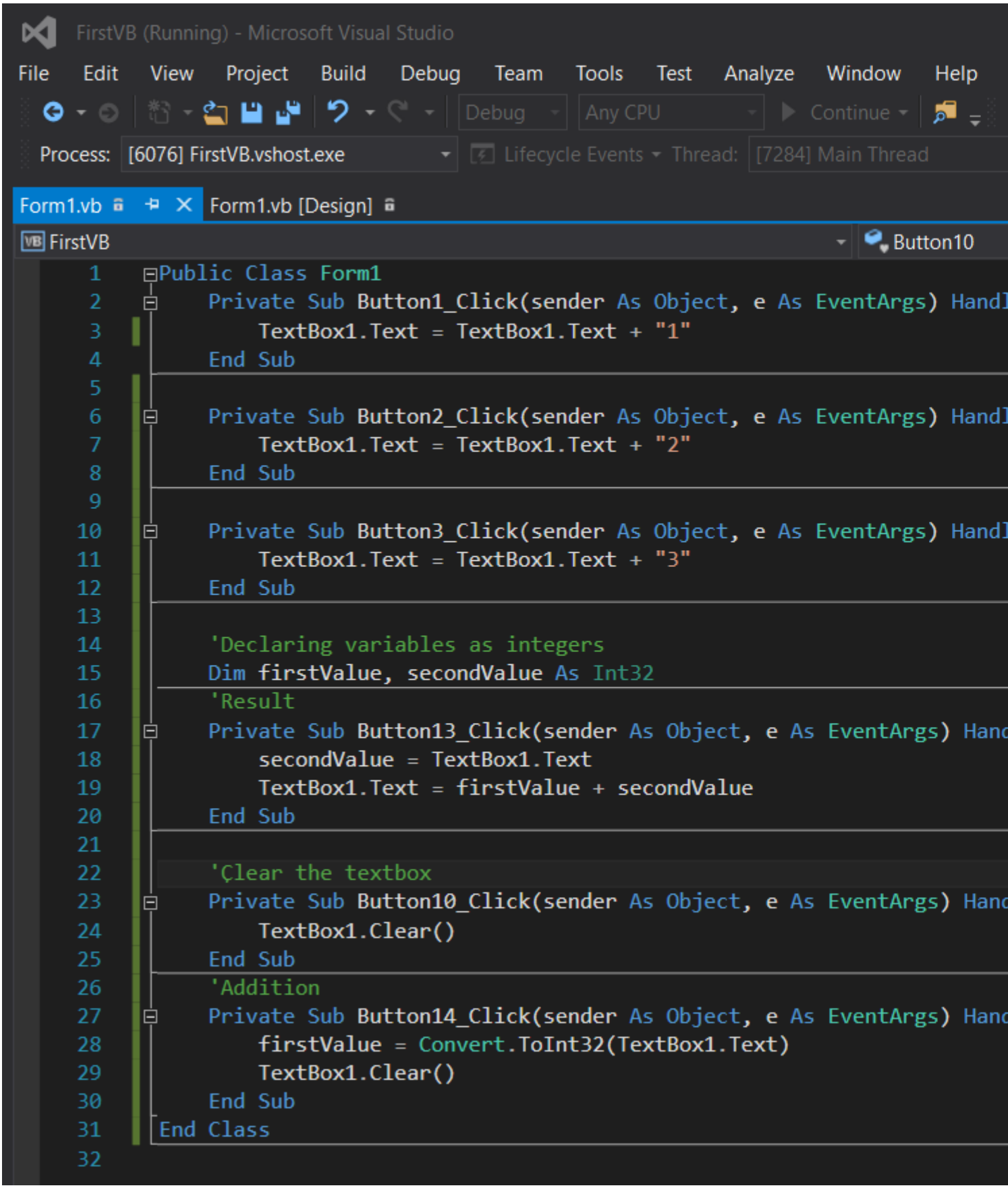
6. Machen Sie sich mit den im Lieferumfang enthaltenen Tools vertraut. Ich habe eine Taschenrechner-Schnittstelle mit Schaltflächen und einer Textbox erstellt.



7. Klicken Sie auf die Registerkarte *Eigenschaften*

(sie befindet sich auf der rechten Seite des Editors). Sie können die *Texteigenschaft* einer Schaltfläche und das Textfeld ändern, um sie umzubenennen. *Mit der* Eigenschaft font können Sie die Schriftart der Steuerelemente ändern.

8. Um die spezifische Aktion für ein Ereignis zu schreiben (z. B. Klicken auf eine Schaltfläche), doppelklicken Sie auf das Steuerelement. Das Codefenster wird geöffnet.



9. VB.Net ist eine leistungsstarke Sprache, die für eine schnelle Entwicklung konzipiert wurde. Hohe Einkapselung und Abstraktion kostet es. Sie müssen kein *Semikolon* hinzufügen, um das Ende einer Anweisung anzuzeigen, es gibt keine Klammern, und in den meisten Fällen korrigiert es automatisch die Groß- und Kleinschreibung der Buchstaben.

10. Der im Bild enthaltene Code sollte einfach zu verstehen sein. *Dim* ist das Schlüsselwort, mit dem eine Variable initialisiert wird, und *new* weist Speicher zu. Alles, was Sie in das Textfeld eingeben, hat standardmäßig den Typ *string* . Casting ist erforderlich, um den Wert als einen anderen Typ zu verwenden.

Viel Spaß bei Ihrer ersten Kreation in VB.Net!

Erste Schritte mit Visual Basic .NET Language online lesen: https://riptutorial.com/de/vb-net/topic/352/erste-schritte-mit-visual-basic--net-language

Kapitel 2: Anweisung verwenden

Syntax

- Verwenden von *a* = New *DisposableClass* [, *b* = ...]
  ...
  Ende mit

- Verwenden von *a* = *GetDisposable (...)* [, *b* = ...]
  ...
  Ende mit

Examples

Siehe Beispiele unter Einwegobjekte

Grundkonzept von IDisposable

Anweisung verwenden online lesen: https://riptutorial.com/de/vb-net/topic/7965/anweisung-verwenden

Kapitel 3: Array

Bemerkungen

```
Dim myArray(2) As Integer

someFunc(myArray)
```

Ein Array ist eine nach Index geordnete Sammlung von Objekten. Der Objekttyp wird durch den in der Array-Deklaration angegebenen Typ definiert.

Arrays in Visual Basic .NET sind am häufigsten (und standardmäßig) null (0), was bedeutet, dass der erste Index 0 ist. Ein Array mit 10 Elementen hat einen Indexbereich von 0 bis 9. Beim Zugriff auf Array-Elemente ist der maximal zugreifbare Index um eins geringer als die Gesamtzahl der Elemente. Aus diesem Grund sollten Schleifen, die inkrementell auf Array-Indizes zugreifen, immer eine Bereichsüberprüfung durchführen, bei der der Wert kleiner als die Arraylänge ist.

Examples

Array-Definition

```
Dim array(9) As Integer ' Defines an array variable with 10 Integer elements (0-9).

Dim array = New Integer(10) {} ' Defines an array variable with 11 Integer elements (0-10)
                               'using New.

Dim array As Integer() = {1, 2, 3, 4} ' Defines an Integer array variable and populate it
                                      'using an array literal. Populates the array with
                                      '4 elements.

ReDim Preserve array(10) ' Redefines the size of an existing array variable preserving any
                         'existing values in the array. The array will now have 11 Integer
                         'elements (0-10).

ReDim array(10) ' Redefines the size of an existing array variable discarding any
                'existing values in the array. The array will now have 11 Integer
                'elements (0-10).
```

Nullbasiert

Alle Arrays in VB.NET sind nullbasiert. Mit anderen Worten, der Index des ersten Elements (der unteren Grenze) in einem VB.NET-Array ist immer `0` . Ältere Versionen von VB, z. B. VB6 und VBA, waren standardmäßig einseitig, boten jedoch die Möglichkeit, die Standardgrenzen zu überschreiben. In diesen früheren Versionen von VB konnten die Unter- und Obergrenzen explizit angegeben werden (z. B. `Dim array(5 To 10)` . In VB.NET wurde diese Flexibilität und die Untergrenze für die Kompatibilität mit anderen .NET-Sprachen aufgehoben Von `0` wird jetzt immer erzwungen, die `To` Syntax kann jedoch weiterhin in VB.NET verwendet werden, wodurch der

Bereich möglicherweise deutlicher wird. Die folgenden Beispiele entsprechen beispielsweise den oben aufgelisteten:

```
Dim array(0 To 9) As Integer

Dim array = New Integer(0 To 10) {}

ReDim Preserve array(0 To 10)

ReDim array(0 To 10)
```

Verschachtelte Array-Deklarationen

```
Dim myArray = {{1, 2}, {3, 4}}
```

Deklarieren Sie ein Single-Dimension-Array und legen Sie Array-Elementwerte fest

```
Dim array = New Integer() {1, 2, 3, 4}
```

oder

```
Dim array As Int32() = {1, 2, 3, 4}
```

Array-Initialisierung

```
Dim array() As Integer = {2, 0, 1, 6}                  ''Initialize an array of four
Integers.
Dim strings() As String = {"this", "is", "an", "array"} ''Initialize an array of four Strings.
Dim floats() As Single = {56.2, 55.633, 1.2, 5.7743, 22.345}
              ''Initialize an array of five Singles, which are the same as floats in C#.
Dim miscellaneous() as Object = { New Object(), "Hello", New List(of String) }
              ''Initialize an array of three references to any reference type objects
              ''and point them to objects of three different types.
```

Multidimensionale Array-Initialisierung

```
Dim array2D(,) As Integer = {{1, 2, 3}, {4, 5, 6}}
' array2D(0, 0) is 1 ; array2D(0, 1) is 2 ; array2D(1, 0) is 4

Dim array3D(,,) As Integer = {{{1, 2, 3}, {4, 5, 6}}, {{7, 8, 9}, {10, 11, 12}}}
' array3D(0, 0, 0) is 1 ; array3D(0, 0, 1) is 2
' array3D(0, 1, 0) is 4 ; array3D(1, 0, 0) is 7
```

Zackige Array-Initialisierung

Beachten Sie die Klammer, um zwischen einem gezackten und einem mehrdimensionalen Array zu unterscheiden

```
Dim jaggedArray()() As Integer = { ({1, 2, 3}), ({4, 5, 6}), ({7}) }
' jaggedArray(0) is {1, 2, 3} and so jaggedArray(0)(0) is 1
' jaggedArray(1) is {4, 5, 6} and so jaggedArray(1)(0) is 4
' jaggedArray(2) is {7} and so jaggedArray(2)(0) is 7
```

Null-Array-Variablen

Da Arrays Referenztypen sind, kann eine Arrayvariable null sein. Um eine Null-Array-Variable zu deklarieren, müssen Sie sie ohne Größe deklarieren:

```
Dim array() As Integer
```

Oder

```
Dim array As Integer()
```

Um zu überprüfen, ob ein Array null ist, testen Sie, ob es `Is Nothing` :

```
Dim array() As Integer
If array Is Nothing Then
    array = {1, 2, 3}
End If
```

Um eine vorhandene Array-Variable auf null zu setzen, setzen Sie sie einfach auf `Nothing` :

```
Dim array() As Integer = {1, 2, 3}
array = Nothing
Console.WriteLine(array(0))  ' Throws a NullReferenceException
```

Oder verwenden Sie `Erase` , was dasselbe bewirkt:

```
Dim array() As Integer = {1, 2, 3}
Erase array
Console.WriteLine(array(0))  ' Throws a NullReferenceException
```

Verweis auf dasselbe Array aus zwei Variablen

Da Arrays Referenztypen sind, können mehrere Variablen auf dasselbe Arrayobjekt verweisen.

```
Dim array1() As Integer = {1, 2, 3}
Dim array2() As Integer = array1
array1(0) = 4
Console.WriteLine(String.Join(", ", array2))  ' Writes "4, 2, 3"
```

Nicht-Null-Untergrenzen

Mit der `Option Strict On` sind in .NET Framework zwar Single-Dimension-Arrays mit niedrigeren Grenzen als Null möglich, sie sind jedoch keine "Vektoren" und daher nicht mit VB.NET-Arrays kompatibel. Dies bedeutet, dass sie nur als `Array` und daher keine normalen Array- `(index)`

Referenzen verwenden können.

```
Dim a As Array = Array.CreateInstance(GetType(Integer), {4}, {-1})
For y = LBound(a) To UBound(a)
    a.SetValue(y * y, y)
Next
For y = LBound(a) To UBound(a)
    Console.WriteLine($"{y}: {a.GetValue(y)}")
Next
```

Neben der Verwendung von `Option Strict Off` können Sie die `(index) IList` zurückholen, indem Sie das Array als `IList` behandeln. `IList` handelt sich jedoch nicht um ein Array. `LBound` können Sie `LBound` und `UBound` für diesen Variablennamen (und Sie 'verwenden). noch immer vermeiden Boxen nicht:)

```
Dim nsz As IList = a
For y = LBound(a) To UBound(a)
    nsz(y) = 2 - CInt(nsz(y))
Next
For y = LBound(a) To UBound(a)
    Console.WriteLine($"{y}: {nsz(y)}")
Next
```

Mehrdimensionale nicht-null-Arrays mit niedrigerer Begrenzung *sind* mit mehrdimensionalen VB.NET-Arrays kompatibel:

```
Dim nza(,) As Integer = DirectCast(Array.CreateInstance(GetType(Integer),
                                    {4, 3}, {1, -1}), Integer(,))
For y = LBound(nza) To UBound(nza)
    For w = LBound(nza, 2) To UBound(nza, 2)
        nza(y, w) = -y * w + nza(UBound(nza) - y + LBound(nza),
                                 UBound(nza, 2) - w + LBound(nza, 2))
    Next
Next
For y = LBound(nza) To UBound(nza)
    Dim ly = y
    Console.WriteLine(String.Join(" ",
        Enumerable.Repeat(ly & ":", 1).Concat(
            Enumerable.Range(LBound(nza, 2), UBound(nza, 2) - LBound(nza, 2) + 1) _
            .Select(Function(w) CStr(nza(ly, w))))))
Next
```

MSDN-Referenz: Array.CreateInstance

Array online lesen: https://riptutorial.com/de/vb-net/topic/805/array

Kapitel 4: Aufgabenbasiertes asynchrones Muster

Examples

Grundlegende Verwendung von Async / Await

Sie können einen langsamen Prozess parallel starten und anschließend die Ergebnisse sammeln:

```
Public Sub Main()
    Dim results = Task.WhenAll(SlowCalculation, AnotherSlowCalculation).Result

    For Each result In results
        Console.WriteLine(result)
    Next
End Sub

Async Function SlowCalculation() As Task(Of Integer)
     Await Task.Delay(2000)

     Return 40
End Function

Async Function AnotherSlowCalculation() As Task(Of Integer)
    Await Task.Delay(2000)

    Return 60
End Function
```

Nach zwei Sekunden sind beide Ergebnisse verfügbar.

TAP mit LINQ verwenden

Sie können ein `IEnumerable` von `Task` erstellen, indem Sie `AddressOf AsyncMethod` an die **LINQ** `Select` Methode übergeben und dann alle Ergebnisse mit `Task.WhenAll` starten und warten

Wenn Ihre Methode über Parameter verfügt, die mit dem vorherigen **LINQ-** Kettenaufruf übereinstimmen, werden diese automatisch zugeordnet.

```
Public Sub Main()
    Dim tasks = Enumerable.Range(0, 100).Select(AddressOf TurnSlowlyIntegerIntoString)

    Dim resultingStrings = Task.WhenAll(tasks).Result

    For Each value In resultingStrings
        Console.WriteLine(value)
    Next
End Sub

Async Function TurnSlowlyIntegerIntoString(input As Integer) As Task(Of String)
    Await Task.Delay(2000)
```

```
    Return input.ToString()
End Function
```

Um verschiedene Argumente `AddressOf Method` können Sie `AddressOf Method` durch ein Lambda ersetzen:

```
Function(linqData As Integer) MyNonMatchingMethod(linqData, "Other parameter")
```

Aufgabenbasiertes asynchrones Muster online lesen: https://riptutorial.com/de/vb-net/topic/2936/aufgabenbasiertes-asynchrones-muster

Kapitel 5: AxWindowsMediaPlayer in VB.Net verwenden

Einführung

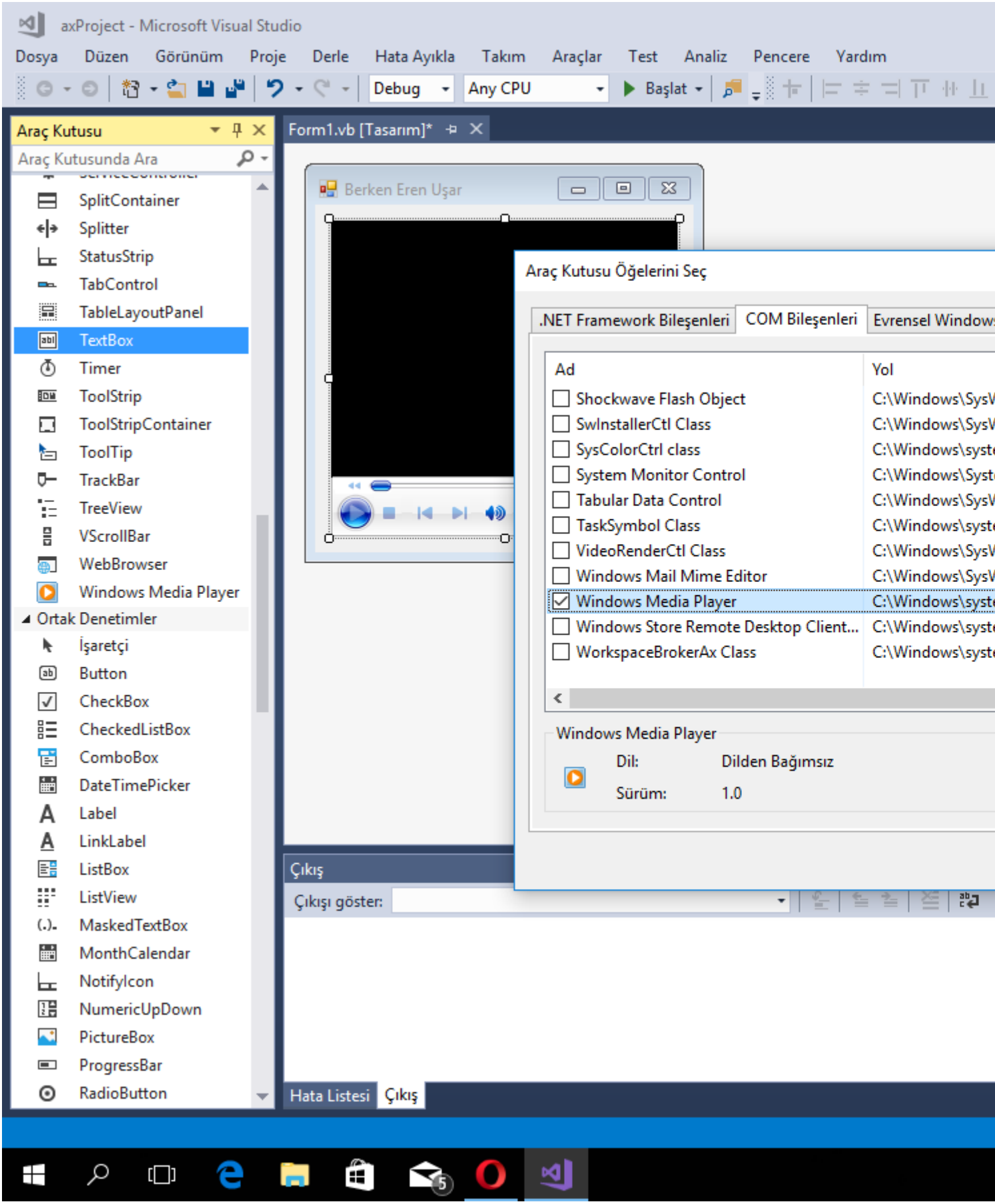
axWindowsMediaPlayer steuert die Wiedergabe von Multimedia-Dateien wie Videos und Musik.

Examples

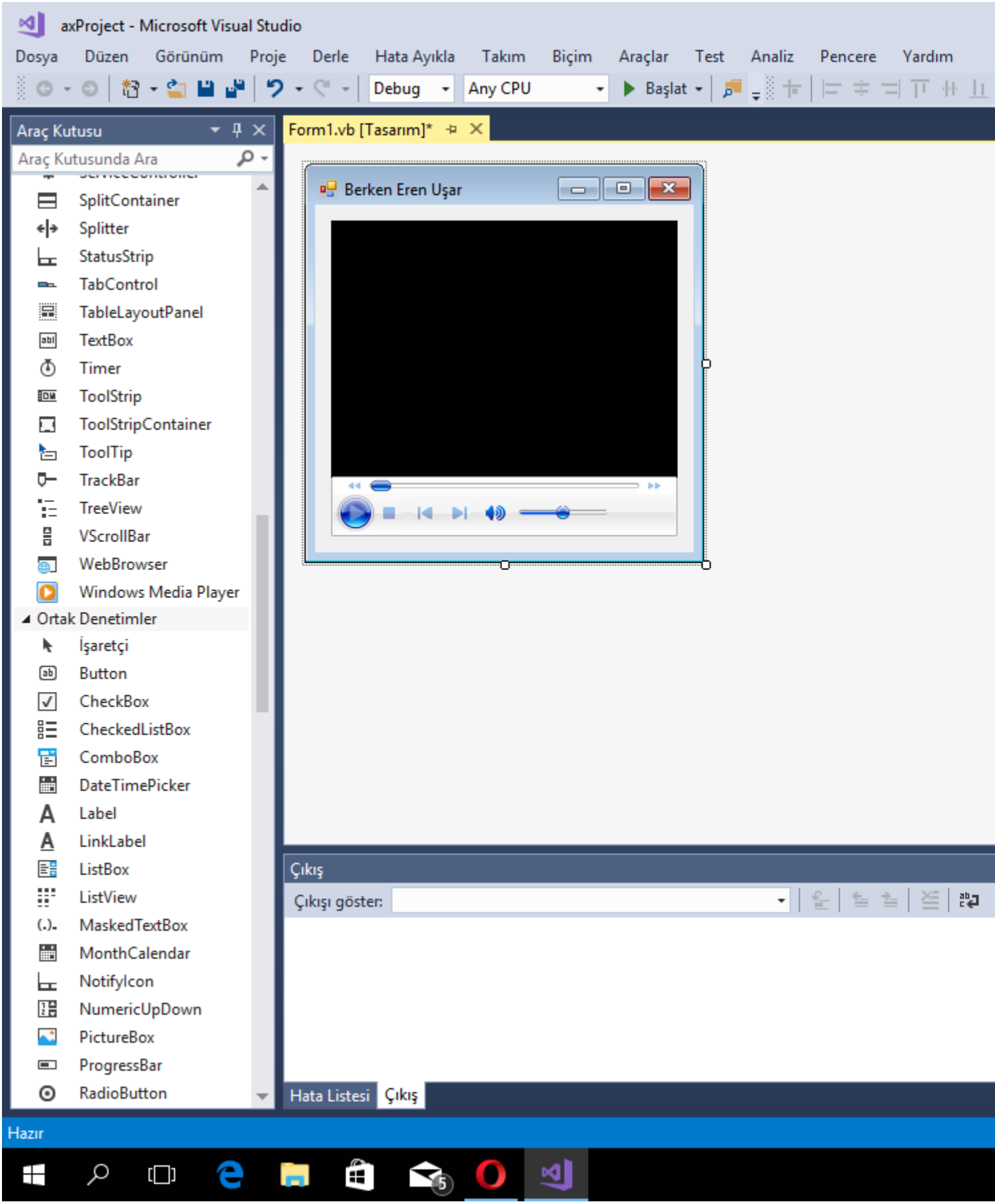
AxWindowsMediaPlayer hinzufügen

- Klicken Sie mit der rechten Maustaste auf die Toolbox, und klicken Sie dann auf "Artikel auswählen".
- Wählen Sie die Registerkarte COM-Komponenten aus, und überprüfen Sie Windows Media Player.
- axWindowsMediaPlayer wird zur Toolbox hinzugefügt.

Aktivieren Sie dieses Kontrollkästchen, um den axWindowsMediaPlayer zu verwenden



Dann kannst du axWindowsMediaPlayer verwenden :)



Abspielen einer Multimedia-Datei

```
AxWindowsMediaPlayer1.URL = "C:\My Files\Movies\Avatar.mp4"
```

```
AxWindowsMediaPlayer1.Ctlcontrols.play()
```

Dieser Code spielt Avatar im axWindowsMediaPlayer.

AxWindowsMediaPlayer in VB.Net verwenden online lesen: https://riptutorial.com/de/vb-net/topic/10096/axwindowsmediaplayer-in-vb-net-verwenden

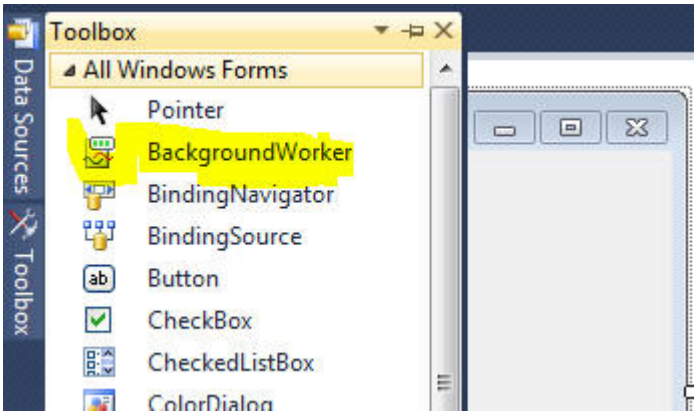
Kapitel 6: BackgroundWorker

Examples

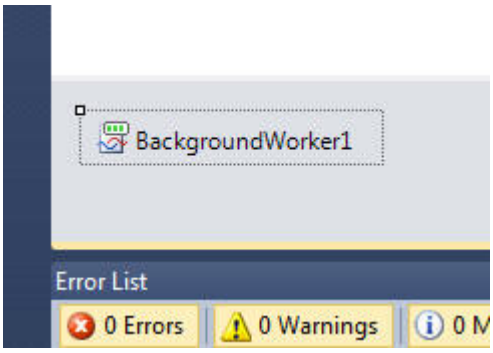
BackgroundWorker verwenden

Ausführen einer Aufgabe mit dem Hintergrundarbeiter

Doppelklicken Sie in der Toolbox auf das `BackgroundWorker` Steuerelement



So erscheint der BackgroundWorker, nachdem er hinzugefügt wurde.



Doppelklicken Sie auf das hinzugefügte Steuerelement, um das `BackgroundWorker1_DoWork` Ereignis zu erhalten, und fügen Sie den Code hinzu, der ausgeführt werden soll, wenn der BackgroundWorker aufgerufen wird. Etwas wie das:

```
Private Sub BackgroundWorker1_DoWork(ByVal sender As System.Object, ByVal e As
System.ComponentModel.DoWorkEventArgs) Handles BackgroundWorker1.DoWork

    'Do the time consuming background task here

End Sub
```

Das Aufrufen des BackgroundWorker zum Ausführen der Aufgabe kann bei jedem Ereignis wie `Button_Click` , `Textbox_TextChanged` usw. wie folgt durchgeführt werden:

```
BackgroundWorker1.RunWorkerAsync()
```

`RunWorkerCompleted` Ereignis " `RunWorkerCompleted` , um das Ereignis "Vorgang abgeschlossen" des BackgroundWorker wie folgt zu erfassen:

```
Private Sub BackgroundWorker1_RunWorkerCompleted(ByVal sender As Object, ByVal e As
System.ComponentModel.RunWorkerCompletedEventArgs) Handles
BackgroundWorker1.RunWorkerCompleted
    MsgBox("Done")
End Sub
```

Daraufhin wird ein Meldungsfeld mit der Meldung `Done` angezeigt, wenn der Mitarbeiter die ihm zugewiesene Aufgabe abgeschlossen hat.

Zugriff auf GUI-Komponenten in BackgroundWorker

Sie können vom BackgroudWorker aus nicht auf GUI-Komponenten zugreifen. Zum Beispiel, wenn Sie versuchen, so etwas zu tun

```
Private Sub BackgroundWorker1_DoWork(sender As Object, e As DoWorkEventArgs)
    TextBox1.Text = "Done"
End Sub
```

Sie erhalten einen Laufzeitfehler, der besagt, dass "Cross-Thread-Vorgang nicht gültig ist: Auf das Steuerelement" TextBox1 "wurde von einem anderen Thread als dem Thread zugegriffen, für den es erstellt wurde.

Dies liegt daran, dass der BackgroundWorker Ihren Code in einem anderen Thread parallel zum Hauptthread ausführt und die GUI-Komponenten nicht threadsicher sind. Sie müssen Ihren Code so einstellen, dass er im Hauptthread mithilfe der `Invoke` Methode ausgeführt wird, und ihm einen Delegaten `Invoke` :

```
Private Sub BackgroundWorker1_DoWork(sender As Object, e As DoWorkEventArgs)
    Me.Invoke(New MethodInvoker(Sub() Me.TextBox1.Text = "Done"))
End Sub
```

Oder Sie können die ReportProgress-Methode des BackgroundWorker verwenden:

```
Private Sub BackgroundWorker1_DoWork(sender As Object, e As DoWorkEventArgs)
    Me.BackgroundWorker1.ReportProgress(0, "Done")
End Sub

Private Sub BackgroundWorker1_ProgressChanged(sender As Object, e As ProgressChangedEventArgs)
    Me.TextBox1.Text = DirectCast(e.UserState, String)
End Sub
```

BackgroundWorker online lesen: https://riptutorial.com/de/vb-net/topic/6242/backgroundworker

Kapitel 7: BackgroundWorker verwenden

Examples

Grundlegende Implementierung der Hintergrundarbeiterklasse

Sie müssen System.ComponentModel für die Verwendung des Hintergrundarbeiters importieren

```
Imports System.ComponentModel
```

Deklarieren Sie dann eine private Variable

```
Private bgWorker As New BackgroundWorker
```

Sie müssen zwei Methoden für DoWork- und RunWorkerCompleted-Ereignisse des Hintergrundarbeiters erstellen und diese zuweisen.

```
Private Sub MyWorker_DoWork(ByVal sender As System.Object, ByVal e As
System.ComponentModel.DoWorkEventArgs)
  'Add your codes here for the worker to execute

End Sub
```

Das folgende Sub wird ausgeführt, wenn der Arbeiter den Job beendet

```
Private Sub MyWorker_RunWorkerCompleted(ByVal sender As Object, ByVal e As
System.ComponentModel.RunWorkerCompletedEventArgs)
    'Add your codes for the worker to execute after finishing the work.

End Sub
```

Fügen Sie dann in Ihrem Code die folgenden Zeilen hinzu, um den Hintergrundarbeiter zu starten

```
    bgWorker = New BackgroundWorker
    AddHandler bgWorker.DoWork, AddressOf MyWorker_DoWork
    AddHandler bgWorker.RunWorkerCompleted, AddressOf MyWorker_RunWorkerCompleted
    bgWorker.RunWorkerAsync()
```

Wenn Sie die Funktion RunWorkerAsync () aufrufen, wird MyWorker_DoWork ausgeführt.

BackgroundWorker verwenden online lesen: https://riptutorial.com/de/vb-net/topic/6401/backgroundworker-verwenden

Kapitel 8: Bedingungen

Examples

WENN ... Dann ... Else

```
Dim count As Integer = 0
Dim message As String

If count = 0 Then
    message = "There are no items."
ElseIf count = 1 Then
    message = "There is 1 item."
Else
    message = "There are " & count & " items."
End If
```

Wenn Betreiber

9,0

```
If(condition > value, "True", "False")
```

Wir können den **If-** Operator anstelle von **If ... Then ... Else..End If-** Anweisung blockieren.

Betrachten Sie das folgende Beispiel:

```
If 10 > 9 Then
     MsgBox("True")
Else
     MsgBox("False")
End If
```

ist das gleiche wie

```
MsgBox(If(10 > 9, "True", "False"))
```

`If()` verwendet eine *Kurzschlussbewertung* , dh es werden nur die von ihm verwendeten Argumente ausgewertet. Wenn die Bedingung falsch ist (oder ein `Nullable` , der `Nothing`), wird die erste Alternative überhaupt nicht bewertet und keine ihrer Nebenwirkungen wird beobachtet. Dies ist im Grunde dasselbe wie der ternäre Operator von C # in Form von `condition?a:b` .

Dies ist besonders nützlich, um Ausnahmen zu vermeiden:

```
Dim z As Integer = If(x = 0, 0, y/x)
```

Wir alle wissen, dass das Teilen durch Null eine Ausnahme auslöst, aber `If()` hier verhindert, dass ein Kurzschluss nur zu dem Ausdruck führt, den die Bedingung bereits gewährleistet hat.

Ein anderes Beispiel:

```
Dim varDate as DateTime = If(varString <> "N/A", Convert.ToDateTime(varString), Now.Date)
```

Wenn `varString <> "N/A"` Wert `False` `varDate` , wird der Wert von `Now.Date` als `Now.Date` ohne den ersten Ausdruck auszuwerten.

9,0

Ältere Versionen von VB verfügen nicht über den Operator `If()` und müssen mit der integrierten Funktion `IIf()` . Da es sich um eine Funktion und nicht um einen Operator handelt, wird *kein* Kurzschluss verursacht. Alle Ausdrücke werden mit allen möglichen Nebeneffekten ausgewertet, einschließlich Leistungsstrafen, Statusänderungen und Ausnahmen. (Die beiden obigen Beispiele, die Ausnahmen vermeiden, würden bei einer Konvertierung in `IIf` .) Wenn eine dieser Nebenwirkungen ein Problem darstellt, können keine Inline-Bedingungen verwendet werden. `If..Then` stattdessen auf `If..Then` Blöcke wie üblich.

Bedingungen online lesen: https://riptutorial.com/de/vb-net/topic/7484/bedingungen

Kapitel 9: ByVal- und ByRef-Schlüsselwörter

Examples

ByVal-Schlüsselwort

ByVal-Schlüsselwort vor dem Methodenparameter (oder kein Schlüsselwort, da standardmäßig ByVal angenommen wird) besagt, dass der Parameter so gesendet wird, dass die Methode die dem Parameter zugrunde liegende Variable **nicht** ändern kann (einen neuen Wert zuweisen).
Es verhindert nicht, dass der Inhalt (oder Zustand) des Arguments geändert wird, wenn es sich um eine Klasse handelt.

```
Class SomeClass
    Public Property Member As Integer
End Class

Module Program
    Sub Main()
        Dim someInstance As New SomeClass With {.Member = 42}

        Foo (someInstance)
        ' here someInstance is not Nothing (still the same object)
        ' but someInstance.Member is -42 (internal state can still be changed)

        Dim number As Integer = 42
        Foo(number)
        ' here number is still 42
    End Sub

    Sub Foo(ByVal arg As SomeClass)
        arg.Member = -arg.Member ' change argument content
        arg = Nothing ' change (re-assign) argument
    End Sub

    Sub Foo(arg As Integer) ' No ByVal or ByRef keyword, ByVal is assumed
        arg = -arg
    End Sub
End Module
```

ByRef-Schlüsselwort

Das ByRef-Schlüsselwort "method" gibt an, dass der Parameter so gesendet wird, dass die Methode die dem Parameter zugrunde liegende Variable ändern kann (einen neuen Wert zuweisen).

```
Class SomeClass
    Public Property Member As Integer
End Class

Module Program
    Sub Main()
        Dim someInstance As New SomeClass With {.Member = 42}
```

```
        Foo (someInstance)
        ' here someInstance is not Nothing
        ' but someInstance.Member is -42

        Bar(someInstance)
        ' here someInstance is Nothing
    End Sub

    Sub Foo(ByVal arg As SomeClass)
        arg.Member = -arg.Member ' change argument content
        arg = Nothing ' change (re-assign) argument
    End Sub

    Sub Bar(ByRef param As Integer)
        arg.Member = -arg.Member ' change argument content
        arg = Nothing ' change (re-assign) argument
    End Sub
End Module
```

ByVal- und ByRef-Schlüsselwörter online lesen: https://riptutorial.com/de/vb-net/topic/4653/byval--und-byref-schlusselworter

Kapitel 10: Datei- / Ordner-Komprimierung

Examples

Erstellen eines ZIP-Archivs aus einem Verzeichnis

```
System.IO.Compression.ZipFile.CreateFromDirectory("myfolder", "archive.zip")
```

Erstellen Sie die Datei archive.zip, die Dateien enthält, die sich im `myfolder` . In diesem Beispiel beziehen sich die Pfade auf das Arbeitsverzeichnis des Programms. Sie können absolute Pfade angeben.

Zip-Archiv in ein Verzeichnis extrahieren

```
System.IO.Compression.ZipFile.ExtractToDirectory("archive.zip", "myfolder")
```

Extrahiert die Datei archive.zip in das Verzeichnis myfolder. In diesem Beispiel beziehen sich die Pfade auf das Arbeitsverzeichnis des Programms. Sie können absolute Pfade angeben.

Zip-Archiv dynamisch erstellen

```
' Create filestream to file
Using fileStream = New IO.FileStream("archive.zip", IO.FileMode.Create)
    ' open zip archive from stream
    Using archive = New System.IO.Compression.ZipArchive(fileStream,
IO.Compression.ZipArchiveMode.Create)
        ' create file_in_archive.txt in archive
        Dim zipfile = archive.CreateEntry("file_in_archive.txt")

        ' write Hello world to file_in_archive.txt in archive
        Using sw As New IO.StreamWriter(zipfile.Open())
            sw.WriteLine("Hello world")
        End Using

    End Using
End Using
```

Dateikomprimierung zu Ihrem Projekt hinzufügen

1. Gehen Sie im Projektmappen- *Explorer* zu Ihrem Projekt, klicken Sie mit der rechten Maustaste auf *Verweise und* dann auf *Verweis hinzufügen.*
2. Suchen Sie nach Kompression und wählen Sie *System.IO.Compression.FileSystem aus,* und drücken Sie OK.
3. Fügen Sie `Imports System.IO.Compression` am Anfang Ihrer `Imports System.IO.Compression` (vor einer Klasse oder einem Modul mit den anderen `Imports` Anweisungen).

```
Option Explicit On
```

```
Option Strict On

Imports System.IO.Compression

Public Class Foo

    ...

End Class
```

Bitte beachten Sie, dass diese Klasse (ZipArchive) erst ab .NET Version 4.5 verfügbar ist

Datei- / Ordner-Komprimierung online lesen: https://riptutorial.com/de/vb-net/topic/4638/datei----ordner-komprimierung

Kapitel 11: Dateibehandlung

Syntax

- `System.IO.File.ReadAllLines(path As String)`
- `System.IO.File.ReadAllText(path As String)`
- `System.IO.File.WriteAllText(path As String, contents As String)`
- `System.IO.File.WriteAllLines(path As String, contents() As String)`

Examples

Daten in eine Datei schreiben

So schreiben Sie den Inhalt einer Zeichenfolge in eine Datei:

```
Dim toWrite As String = "This will be written to the file."
System.IO.File.WriteAllText("filename.txt", toWrite)
```

`WriteAllText` öffnet die angegebene Datei, schreibt die Daten und schließt die Datei. Wenn die Zieldatei vorhanden ist, wird sie überschrieben. Wenn die Zieldatei nicht vorhanden ist, wird sie erstellt.

So schreiben Sie den Inhalt eines Arrays in eine Datei:

```
Dim toWrite As String() = {"This", "Is", "A", "Test"}
System.IO.File.WriteAllLines("filename.txt", toWrite)
```

`WriteAllLines` öffnet die angegebene Datei, schreibt jeden Wert des Arrays in eine neue Zeile und schließt die Datei. Wenn die Zieldatei vorhanden ist, wird sie überschrieben. Wenn die Zieldatei nicht vorhanden ist, wird sie erstellt.

Lesen Sie den gesamten Inhalt einer Datei

So lesen Sie den Inhalt einer Datei in eine Stringvariable:

```
Dim fileContents As String = System.IO.File.ReadAllText("filename.txt")
```

`ReadAllText` öffnet die angegebene Datei, liest die Daten bis zum Ende und schließt die Datei.

Um eine Datei zu lesen, trennen Sie sie für jede Zeile in ein Array-Element:

```
Dim fileLines As String() = System.IO.File.ReadAllLines("filename.txt")
```

`ReadAllLines` öffnet die angegebene Datei, liest jede Zeile der Datei in einen neuen Index in einem Array bis zum Ende der Datei und schließt dann die Datei.

Zeilen einzeln mit StreamWriter in eine Textdatei schreiben

```
Using sw As New System.IO.StreamWriter("path\to\file.txt")
    sw.WriteLine("Hello world")
End Using
```

Die Verwendung eines `Using` Blocks wird empfohlen, wenn ein Objekt verwendet wird, das `IDisposable`

Dateibehandlung online lesen: https://riptutorial.com/de/vb-net/topic/2413/dateibehandlung

Kapitel 12: Datenzugriff

Examples

Feld aus Datenbank lesen

```
Public Function GetUserFirstName(UserName As String) As String
    Dim Firstname As String = ""

    'Specify the SQL that you want to use including a Parameter
    Dim SQL As String = "select firstname from users where username=@UserName"

    'Provide a Data Source
    Dim DBDSN As String = "Data Source=server.address;Initial Catalog=DatabaseName;Persist
Security Info=True;User ID=UserName;Password=UserPassword"

    Dim dbConn As New SqlConnection(DBDSN)

    Dim dbCommand As New SqlCommand(SQL, dbConn)

    'Provide one or more Parameters
    dbCommand.Parameters.AddWithValue("@UserName", UserName)

    'An optional Timeout
    dbCommand.CommandTimeout = 600

    Dim reader As SqlDataReader
    Dim previousConnectionState As ConnectionState = dbConn.State
    Try
        If dbConn.State = ConnectionState.Closed Then
            dbConn.Open()
        End If
        reader = dbCommand.ExecuteReader
        Using reader
            With reader
                If .HasRows Then
                    'Read the 1st Record
                    reader.Read()
                    'Read required field/s
                    Firstname = .Item("FirstName").ToString
                End If

            End With

        End Using

    Catch
        'Handle the error here
    Finally
        If previousConnectionState = ConnectionState.Closed Then
            dbConn.Close()
        End If
        dbConn.Dispose()
        dbCommand.Dispose()

    End Try
    'Pass the data back from the function
```

```
    Return Firstname

End Function
```

Die Verwendung der obigen Funktion ist einfach:

```
   Dim UserFirstName as string=GetUserFirstName(UserName)
```

Einfache Funktion zum Lesen aus der Datenbank und Rückgabe als DataTable

Diese einfache Funktion führt den angegebenen Select SQL-Befehl aus und gibt das Ergebnis als Datensatz zurück.

```
Public Function ReadFromDatabase(ByVal DBConnectionString As String, ByVal SQL As String) As
DataTable
    Dim dtReturn As New DataTable
    Try
        'Open the connection using the connection string
        Using conn As New SqlClient.SqlConnection(DBConnectionString)
            conn.Open()

            Using cmd As New SqlClient.SqlCommand()
                cmd.Connection = conn
                cmd.CommandText = SQL
                Dim da As New SqlClient.SqlDataAdapter(cmd)
                da.Fill(dtReturn)
            End Using
        End Using
    Catch ex As Exception
        'Handle the exception
    End Try


    'Return the result data set
    Return dtReturn
End Function
```

Jetzt können Sie die obige Funktion mit den folgenden Codes ausführen

```
Private Sub MainFunction()
    Dim dtCustomers As New DataTable
    Dim dtEmployees As New DataTable
    Dim dtSuppliers As New DataTable


    dtCustomers = ReadFromDatabase("Server=MYDEVPC\SQLEXPRESS;Database=MyDatabase;User
Id=sa;Password=pwd22;", "Select * from [Customers]")
    dtEmployees = ReadFromDatabase("Server=MYDEVPC\SQLEXPRESS;Database=MyDatabase;User
Id=sa;Password=pwd22;", "Select * from [Employees]")
    dtSuppliers = ReadFromDatabase("Server=MYDEVPC\SQLEXPRESS;Database=MyDatabase;User
Id=sa;Password=pwd22;", "Select * from [Suppliers]")

End Sub
```

Im obigen Beispiel wird davon ausgegangen, dass Ihre SQL Express-Instanz "SQLEXPRESS"

derzeit auf "MYDEVPC" installiert ist und Ihre Datenbank "MyDatabase" Tabellen "Customers", "Suppliers" und "Employees" enthält. Das Kennwort "sa" lautet "pwd22". Bitte ändern Sie diese Werte entsprechend Ihrem Setup, um die gewünschten Ergebnisse zu erhalten.

Holen Sie sich Skalardaten

Diese einfache Funktion kann verwendet werden, um einen Wert aus genau einem Feld eines Datensatzabfrageergebnisses zu erhalten

```
Public Function getDataScalar(ssql As String)
    openConnection()

    Try
        Dim q As New MySqlCommand

        q.Connection = db
        q.CommandText = ssql
        getDataScalar = q.ExecuteScalar

    Catch ex As Exception
        'Exception
    End Try
End Function
```

Wie man es benutzt:

```
Dim userid as String = getDataScalar("select username from user where userid=99")
```

Die Variable 'Benutzername' würde als Ergebnis dieser Abfrage mit dem Wert des Felds Benutzername gefüllt werden.

Datenzugriff online lesen: https://riptutorial.com/de/vb-net/topic/3113/datenzugriff

Kapitel 13: Datum

Examples

Konvertieren (Parsen) einer Zeichenfolge in ein Datum

Wenn Sie das Format der zu konvertierenden Zeichenfolge kennen (Parsing), sollten Sie `DateTime.ParseExact`

```
Dim dateString As String = "12.07.2003"
Dim dateFormat As String = "dd.MM.yyyy"
Dim dateValue As Date

dateValue = DateTime.ParseExact(dateString, dateFormat,
Globalization.CultureInfo.InvariantCulture)
```

Wenn Sie nicht sicher sind, in `DateTime.TryParseExact` Format die Zeichenfolge steht, können Sie `DateTime.TryParseExact` und das Ergebnis testen, um zu sehen, ob es analysiert wird oder nicht

```
Dim dateString As String = "23-09-2013"
Dim dateFormat As String = "dd-MM-yyyy"
Dim dateValue As Date

If DateTime.TryParseExact(dateString, dateFormat, Globalization.CultureInfo.InvariantCulture,
DateTimeStyles.None, dateValue) Then
    'the parse worked and the dateValue variable now holds the datetime that was parsed as it
is passing in ByRef
Else
    'the parse failed
End If
```

Konvertieren eines Datums in einen String

Verwenden `.ToString` einfach die `.ToString` Überladung eines `DateTime` Objekts, um das `DateTime` Format zu erhalten:

```
Dim dateValue As DateTime = New DateTime(2001, 03, 06)
Dim dateString As String = dateValue.ToString("yyyy-MM-dd") '2001-03-06
```

Datum online lesen: https://riptutorial.com/de/vb-net/topic/3727/datum

Kapitel 14: Debuggen Ihrer Anwendung

Einführung

Wann immer Sie ein Problem in Ihrem Code haben, ist es immer eine gute Idee zu wissen, was in Ihrem Computer vor sich geht. Die Klasse System.Diagnostics.Debug in .NET Framework wird Ihnen bei dieser Aufgabe sehr helfen.

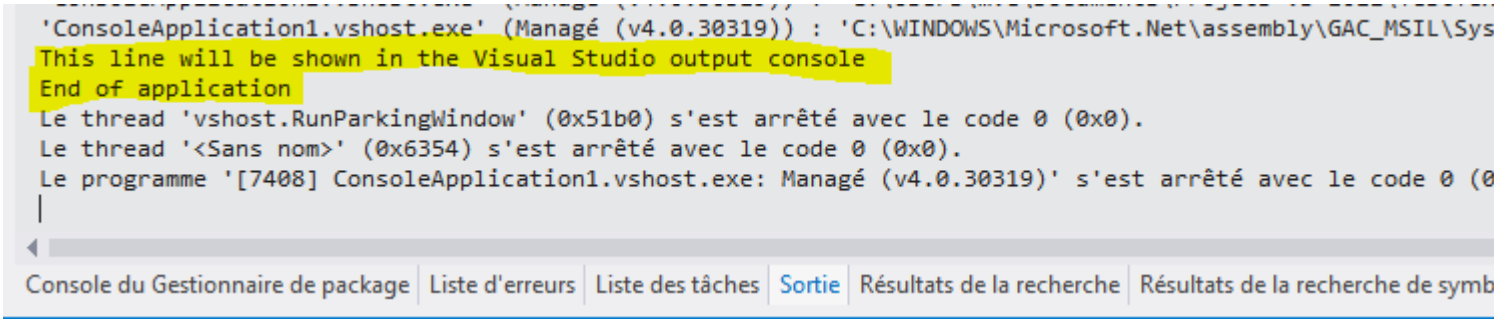
Der erste Vorteil der Debug-Klasse ist, dass sie nur dann Code erzeugt, wenn Sie Ihre Anwendung im Debug-Modus erstellen. Wenn Sie Ihre Anwendung im Freigabemodus erstellen, wird aus den Debug-Aufrufen kein Code generiert.

Examples

Debuggen Sie in der Konsole

```
Module Module1
    Sub Main()
        Debug.WriteLine("This line will be shown in the Visual Studio output console")

        Console.WriteLine("Press a key to exit")
        Console.ReadKey()

        Debug.WriteLine("End of application")
    End Sub
End Module
```

wird herstellen:

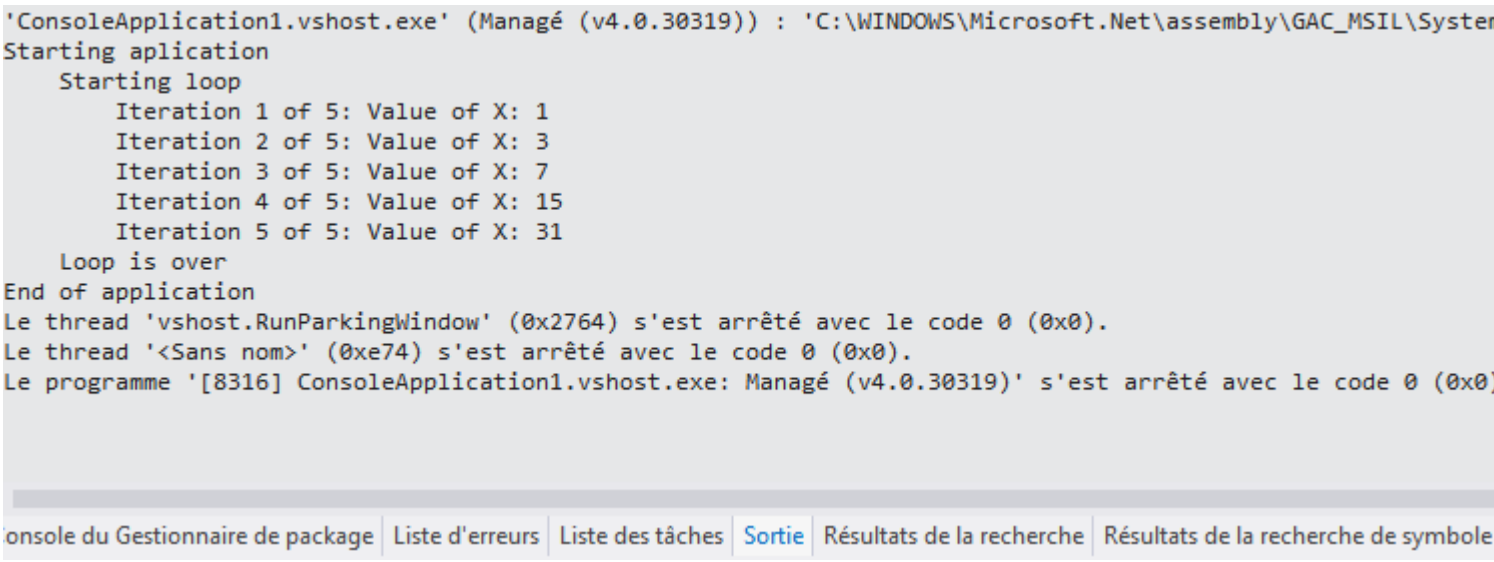


Einrücken Ihrer Debug-Ausgabe

```
Module Module1

    Sub Main()
        Debug.WriteLine("Starting aplication")

        Debug.Indent()
        LoopAndDoStuff(5)
        Debug.Unindent()
```

```
        Console.WriteLine("Press a key to exit")
        Console.ReadKey()

        Debug.WriteLine("End of application")
    End Sub

    Sub LoopAndDoStuff(Iterations As Integer)
        Dim x As Integer = 0
        Debug.WriteLine("Starting loop")
        Debug.Indent()
        For i As Integer = 0 To Iterations - 1
            Debug.Write("Iteration " & (i + 1).ToString() & " of " & Iterations.ToString() &
": Value of X: ")
            x += (x + 1)
            Debug.WriteLine(x.ToString())
        Next
        Debug.Unindent()
        Debug.WriteLine("Loop is over")
    End Sub
End Module
```

wird herstellen:



Debuggen Sie in einer Textdatei

Zu Beginn Ihrer Anwendung müssen Sie der Listeners-Liste der Debug-Klasse einen TextWriterTraceListener hinzufügen.

```
Module Module1

    Sub Main()
        Debug.Listeners.Add(New TextWriterTraceListener("Debug of " & DateTime.Now.ToString()
& ".txt"))

        Debug.WriteLine("Starting aplication")

        Console.WriteLine("Press a key to exit")
        Console.ReadKey()

        Debug.WriteLine("End of application")
    End Sub
```

```
End Module
```

Der gesamte Debug-Code wird in der Visual Studio-Konsole UND in der von Ihnen ausgewählten Textdatei ausgegeben.

Wenn die Datei immer gleich ist:

```
Debug.Listeners.Add(New TextWriterTraceListener("Debug.txt"))
```

Die Ausgabe wird jedes Mal an die Datei angehängt UND eine neue Datei beginnt mit einer GUID. Dann wird Ihr Dateiname generiert.

Debuggen Ihrer Anwendung online lesen: https://riptutorial.com/de/vb-net/topic/8631/debuggen-ihrer-anwendung

Kapitel 15: Einfädeln

Examples

Thread-sichere Aufrufe mit Control.Invoke () durchführen

Mit der `Control.Invoke()` Methode können Sie die Ausführung einer Methode oder Funktion von einem Hintergrund-Thread in den Thread verschieben, in dem das Steuerelement erstellt wurde. `Control.Invoke()` ist normalerweise der UI-Thread (User Interface). Auf diese Weise wird Ihr Code in die Warteschlange gestellt, um stattdessen im Thread des Steuerelements ausgeführt zu werden, wodurch die Möglichkeit der Parallelität beseitigt wird.

Die `Control.InvokeRequired` Eigenschaft sollte auch überprüft werden, um festzustellen, ob Sie aufrufen müssen oder ob der Code bereits im selben Thread wie das Steuerelement ausgeführt wird.

Die `Invoke()` Methode nimmt einen Delegaten als ersten Parameter. Ein Delegat enthält die Referenz, die Parameterliste und den Rückgabetyp für eine andere Methode.

In Visual Basic 2010 (10.0) oder höher können *Lambda-Ausdrücke* verwendet werden, um spontan eine Delegat-Methode zu erstellen:

```
If LogTextBox.InvokeRequired = True Then
    LogTextBox.Invoke(Sub() LogTextBox.AppendText("Check passed"))
Else
    LogTextBox.AppendText("Check passed")
End If
```

In Visual Basic 2008 (9.0) oder niedriger müssen Sie den Delegierten jedoch selbst deklarieren:

```
Delegate Sub AddLogText(ByVal Text As String)

If LogTextBox.InvokeRequired = True Then
    LogTextBox.Invoke(New AddLogText(AddressOf UpdateLog), "Check passed")
Else
    UpdateLog("Check passed")
End If

Sub UpdateLog(ByVal Text As String)
    LogTextBox.AppendText(Text)
End Sub
```

Thread-sichere Anrufe mit Async / Await durchführen

Wenn wir versuchen, ein Objekt im UI-Thread von einem anderen Thread aus zu ändern, wird eine Cross-Thread-Operationsausnahme angezeigt:

```
Private Sub Button_Click(sender As Object, e As EventArgs) Handles MyButton.Click
    ' Cross thread-operation exception as the assignment is executed on a different thread
```

```
    ' from the UI one:
    Task.Run(Sub() MyButton.Text = Thread.CurrentThread.ManagedThreadId)
End Sub
```

Vor **VB 14.0** und **.NET 4.5 hat** die Lösung die Zuweisung auf und das Objekt auf dem UI-Thread aufgerufen:

```
Private Sub Button_Click(sender As Object, e As EventArgs) Handles MyButton.Click
    ' This will run the conde on the UI thread:
    MyButton.Invoke(Sub() MyButton.Text = Thread.CurrentThread.ManagedThreadId)
End Sub
```

Mit **VB 14.0** können wir eine `Task` in einem anderen Thread ausführen und dann den Kontext wiederherstellen, sobald die Ausführung abgeschlossen ist, und dann die Zuweisung mit Async / Await durchführen:

```
Private Async Sub Button_Click(sender As Object, e As EventArgs) Handles MyButton.Click
    ' This will run the code on a different thread then the context is restored
    ' so the assignment happens on the UI thread:
    MyButton.Text = Await Task.Run(Function() Thread.CurrentThread.ManagedThreadId)
End Sub
```

Einfädeln online lesen: https://riptutorial.com/de/vb-net/topic/1913/einfadeln

Kapitel 16: Einführung in die Syntax

Examples

Bemerkungen

Die erste interessante Sache zu wissen ist, wie man Kommentare schreibt.

In VB .NET schreiben Sie einen Kommentar, indem Sie ein Apostroph schreiben oder `REM` schreiben. Das bedeutet, dass der Rest der Zeile vom Compiler nicht berücksichtigt wird.

```
'This entire line is a comment
Dim x As Integer = 0 'This comment is here to say we give 0 value to x

REM There are no such things as multiline comments
'So we have to start everyline with the apostrophe or REM
```

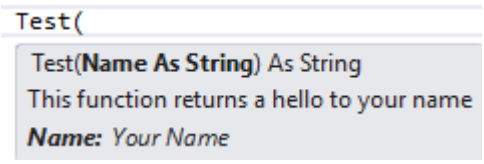
Intellisense-Helfer

Eine interessante Sache ist die Möglichkeit, eigene Kommentare in Visual Studio Intellisense hinzuzufügen. So können Sie selbst geschriebene Funktionen und Klassen selbsterklärend gestalten. Dazu müssen Sie das Kommentarsymbol dreimal in die Zeile oberhalb Ihrer Funktion eingeben.

Sobald dies erledigt ist, fügt Visual Studio automatisch eine XML-Dokumentation hinzu:

```
''' <summary>
''' This function returns a hello to your name
''' </summary>
''' <param name="Name">Your Name</param>
''' <returns></returns>
''' <remarks></remarks>
Public Function Test(Name As String) As String
    Return "Hello " & Name
End Function
```

Wenn Sie Ihre Testfunktion irgendwo in Ihren Code eingeben, wird diese kleine Hilfe angezeigt:



Eine Variable deklarieren

In VB.NET muss jede Variable vor ihrer Verwendung deklariert werden (wenn Option Explicit auf **On gesetzt ist**). Es gibt zwei Möglichkeiten, Variablen zu deklarieren:

- In einer `Function` oder einem `Sub` :

```
Dim w 'Declares a variable named w of type Object (invalid if Option Strict is On)
Dim x As String 'Declares a variable named x of type String
Dim y As Long = 45 'Declares a variable named y of type Long and assigns it the value 45
Dim z = 45 'Declares a variable named z whose type is inferred
           'from the type of the assigned value (Integer here) (if Option Infer is On)
           'otherwise the type is Object (invalid if Option Strict is On)
           'and assigns that value (45) to it
```

In dieser Antwort finden Sie alle Details zu `Option Explicit` , `Strict` und `Infer` .

- Innerhalb einer `Class` oder eines `Module` :

Auf diese Variablen (in diesem Kontext auch als Felder bezeichnet) kann für jede Instanz der `Class` sie deklariert sind, zugegriffen werden. Sie können je nach Modifikator (`Public` , `Private` , `Protected` , `Protected Friend` oder `Friend`) auch außerhalb der deklarierten `Class` verfügbar sein.

```
Private x 'Declares a private field named x of type Object (invalid if Option Strict is On)
Public y As String 'Declares a public field named y of type String
Friend z As Integer = 45 'Declares a friend field named z of type Integer and assigns it the
value 45
```

Diese Felder können auch mit `Dim` deklariert werden, die Bedeutung ändert sich jedoch je nach dem einschließenden Typ:

```
Class SomeClass
    Dim z As Integer = 45 ' Same meaning as Private z As Integer = 45
End Class

Structure SomeStructure
    Dim y As String ' Same meaning as Public y As String
End Structure
```

Modifikatoren

Modifikatoren geben an, wie externe Objekte auf die Daten eines Objekts zugreifen können.

- Öffentlichkeit

Dies bedeutet, dass jedes Objekt uneingeschränkt darauf zugreifen kann

- Privatgelände

Bedeutet, dass nur das deklarierende Objekt darauf zugreifen kann

- Geschützt

Bedeutet nur das deklarierende Objekt und jedes Objekt, das es erbt, kann darauf zugreifen und es anzeigen.

- Freund

Bedeutet nur das delcaring-Objekt, jedes Objekt, das davon erbt, und jedes Objekt im selben

Namespace können darauf zugreifen und es anzeigen.

```
Public Class MyClass
    Private x As Integer

    Friend Property Hello As String

    Public Sub New()
    End Sub

    Protected Function Test() As Integer
        Return 0
    End Function
End Class
```

Funktion schreiben

Eine Funktion ist ein Codeblock, der während der Ausführung mehrmals aufgerufen wird. Anstatt den gleichen Code immer wieder zu schreiben, kann man diesen Code in eine Funktion schreiben und diese Funktion jederzeit aufrufen.

Eine Funktion :

- Muss in einer *Klasse* oder einem *Modul* deklariert *werden*
- Gibt einen Wert zurück (angegeben durch den Rückgabetyp)
- Hat einen *Modifikator*
- Kann Parameter für die Verarbeitung übernehmen

```
Private Function AddNumbers(X As Integer, Y As Integer) As Integer
    Return X + Y
End Function
```

Ein Funktionsname könnte als Rückgabeanweisung verwendet werden

```
Function sealBarTypeValidation() as Boolean
    Dim err As Boolean = False

    If rbSealBarType.SelectedValue = "" Then
        err = True
    End If

    Return err
End Function
```

ist genauso wie

```
Function sealBarTypeValidation() as Boolean
    sealBarTypeValidation = False

    If rbSealBarType.SelectedValue = "" Then
        sealBarTypeValidation = True
    End If
```

```
End Function
```

Objektinitialisierer

- Benannte Typen

```
Dim someInstance As New SomeClass(argument) With {
    .Member1 = value1,
    .Member2 = value2
    '...
}
```

Ist äquivalent zu

```
Dim someInstance As New SomeClass(argument)
someInstance.Member1 = value1
someInstance.Member2 = value2
'...
```

- Anonyme Typen *(Option Infer muss aktiviert sein)*

```
 Dim anonymousInstance = New With {
     .Member1 = value1,
     .Member2 = value2
     '...
 }
```

Eine ähnliche `anonymousInstance` zwar nicht denselben Typ wie `someInstance`

Der Mitgliedsname muss im anonymen Typ eindeutig sein und kann einer Variablen oder einem anderen Objektmitgliedernamen entnommen werden

```
 Dim anonymousInstance = New With {
     value1,
     value2,
     foo.value3
     '...
 }
 ' usage : anonymousInstance.value1 or anonymousInstance.value3
```

Jedem Mitglied kann das `Key` Schlüssel vorangestellt werden. Diese Member sind `ReadOnly` Eigenschaften, diejenigen ohne Read / Write-Eigenschaften

```
 Dim anonymousInstance = New With {
     Key value1,
     .Member2 = value2,
     Key .Member3 = value3
     '...
 }
```

Zwei anonyme Instanzen, die mit denselben Mitgliedern (Name, Typ, Vorhandensein von `Key` und Reihenfolge) definiert sind, haben denselben anonymen Typ.

```
Dim anon1 = New With { Key .Value = 10 }
Dim anon2 = New With { Key .Value = 20 }

anon1.GetType Is anon2.GetType ' True
```

Anonyme Typen sind strukturell gleichwertig. Zwei Instanzen desselben anonymen Typs mit mindestens einer `Key` Eigenschaft mit denselben `Key` Werten sind gleich. Sie müssen die `Equals` Methode verwenden, um sie zu testen, mit `=` wird nicht kompiliert und `Is` vergleicht die Objektreferenz.

```
Dim anon1 = New With { Key .Name = "Foo", Key .Age = 10, .Salary = 0 }
Dim anon2 = New With { Key .Name = "Bar", Key .Age = 20, .Salary = 0 }
Dim anon3 = New With { Key .Name = "Foo", Key .Age = 10, .Salary = 10000 }

anon1.Equals(anon2) ' False
anon1.Equals(anon3) ' True although non-Key Salary isn't the same
```

Der Initialisierer Named und Anonymous kann verschachtelt und gemischt werden

```
Dim anonymousInstance = New With {
    value,
    Key .someInstance = New SomeClass(argument) With {
        .Member1 = value1,
        .Member2 = value2
        '...
    }
    '...
}
```

Collection Initializer

- Arrays

```
Dim names = {"Foo", "Bar"} ' Inferred as String()
Dim numbers = {1, 5, 42} ' Inferred as Integer()
```

- Container (`List(Of T)` , `Dictionary(Of TKey, TValue)` usw.)

```
Dim names As New List(Of String) From {
    "Foo",
    "Bar"
    '...
}

Dim indexedDays As New Dictionary(Of Integer, String) From {
    {0, "Sun"},
    {1, "Mon"}
    '...
}
```

Ist äquivalent zu

```
Dim indexedDays As New Dictionary(Of Integer, String)
indexedDays.Add(0, "Sun")
indexedDays.Add(1, "Mon")
'...
```

Elemente können das Ergebnis eines Konstruktors, eines Methodenaufrufs oder eines Eigenschaftszugriffs sein. Es kann auch mit dem Objektinitialisierer gemischt werden.

```
Dim someList As New List(Of SomeClass) From {
    New SomeClass(argument),
    New SomeClass With { .Member = value },
    otherClass.PropertyReturningSomeClass,
    FunctionReturningSomeClass(arguments)
    '...
}
```

Es ist nicht möglich, die Objektinitialisierersyntax **UND die** Erfassungsinitialisierersyntax gleichzeitig für dasselbe Objekt zu verwenden. Zum Beispiel funktionieren diese **nicht**

```
Dim numbers As New List(Of Integer) With {.Capacity = 10} _
                                   From { 1, 5, 42 }

Dim numbers As New List(Of Integer) From {
    .Capacity = 10,
    1, 5, 42
}

Dim numbers As New List(Of Integer) With {
    .Capacity = 10,
    1, 5, 42
}
```

- Benutzerdefinierter Typ

  Wir können auch die Syntax für die Initialisierung von Auflistungen zulassen, indem wir einen benutzerdefinierten Typ angeben.
  Es muss `IEnumerable` implementieren und über eine durch Überladungsregeln zugängliche und kompatible Methode verfügen. `Add` Methode (Instanz, Shared oder sogar Erweiterungsmethode)

  Gedachtes Beispiel:

```
Class Person
    Implements IEnumerable(Of Person) ' Inherits from IEnumerable

    Private ReadOnly relationships As List(Of Person)

    Public Sub New(name As String)
        relationships = New List(Of Person)
    End Sub

    Public Sub Add(relationName As String)
        relationships.Add(New Person(relationName))
    End Sub
```

```
    Public Iterator Function GetEnumerator() As IEnumerator(Of Person) _
        Implements IEnumerable(Of Person).GetEnumerator

        For Each relation In relationships
            Yield relation
        Next
    End Function

    Private Function IEnumerable_GetEnumerator() As IEnumerator _
        Implements IEnumerable.GetEnumerator

        Return GetEnumerator()
    End Function
End Class

' Usage
Dim somePerson As New Person("name") From {
    "FriendName",
    "CoWorkerName"
    '...
}
```

Wenn Sie der `List(Of Person)` eine Person hinzufügen möchten, indem Sie einfach den Namen in den Initialisierungssatz der Auflistung eingeben (die Klasse List (Person) kann jedoch nicht geändert werden), können wir eine Erweiterungsmethode verwenden

```
' Inside a Module
<Runtime.CompilerServices.Extension>
Sub Add(target As List(Of Person), name As String)
    target.Add(New Person(name))
End Sub

' Usage
Dim people As New List(Of Person) From {
    "Name1", ' no need to create Person object here
    "Name2"
}
```

Einführung in die Syntax online lesen: https://riptutorial.com/de/vb-net/topic/3997/einfuhrung-in-die-syntax

Kapitel 17: Einweggegenstände

Examples

Grundkonzept von IDisposable

Jedes Mal, wenn Sie eine Klasse instanziieren, die `IDisposable` , sollten Sie `.Dispose` [1] für diese Klasse aufrufen, wenn Sie die Verwendung beendet haben. Dadurch kann die Klasse verwaltete oder nicht verwaltete Abhängigkeiten bereinigen, die sie möglicherweise verwendet. Andernfalls kann es zu einem Speicherverlust kommen.

Das Schlüsselwort `Using` sorgt dafür, dass `.Dispose` aufgerufen wird, ohne dass Sie es *explizit* aufrufen müssen.

Zum Beispiel ohne `Using` :

```
Dim sr As New StreamReader("C:\foo.txt")
Dim line = sr.ReadLine
sr.Dispose()
```

Jetzt mit `Using` :

```
Using sr As New StreamReader("C:\foo.txt")
    Dim line = sr.ReadLine
End Using '.Dispose is called here for you
```

Ein großer Vorteil bei der `Using` ist, wenn eine Ausnahme ausgelöst wird, da *sichergestellt* ist, dass `.Dispose` aufgerufen wird.

Folgendes berücksichtigen. Wenn eine Ausnahme ausgelöst wird, müssen Sie daran denken, .Dispose aufzurufen, müssen jedoch möglicherweise auch den Status des Objekts überprüfen, um sicherzustellen, dass Sie keinen Nullreferenzfehler erhalten.

```
    Dim sr As StreamReader = Nothing
    Try
        sr = New StreamReader("C:\foo.txt")
        Dim line = sr.ReadLine
    Catch ex As Exception
        'Handle the Exception
    Finally
        If sr IsNot Nothing Then sr.Dispose()
    End Try
```

Ein using-Block bedeutet, dass Sie sich nicht daran erinnern müssen, und Sie können Ihr Objekt innerhalb des `try` deklarieren:

```
    Try
        Using sr As New StreamReader("C:\foo.txt")
            Dim line = sr.ReadLine
```

```
        End Using
    Catch ex As Exception
        'sr is disposed at this point
    End Try
```

---

[1] Muss ich Dispose () immer für meine DbContext-Objekte aufrufen? Nee

Mehrere Objekte in einem deklarieren Using

Manchmal müssen Sie zwei `Disposable` in einer Reihe erstellen. Es gibt eine einfache Möglichkeit, das Verschachteln `Using` Blöcken zu vermeiden.

Dieser Code

```
Using File As New FileStream("MyFile", FileMode.Append)
    Using Writer As New BinaryWriter(File)
        'You code here
        Writer.Writer("Hello")
    End Using
End Using
```

kann in diese gekürzt werden. Der Hauptvorteil ist, dass Sie eine Einrückungsstufe erhalten:

```
Using File As New FileStream("MyFile", FileMode.Append), Writer As New BinaryWriter(File)
    'You code here
    Writer.Writer("Hello")
End Using
```

Einweggegenstände online lesen: https://riptutorial.com/de/vb-net/topic/3204/einweggegenstande

Kapitel 18: Enum

Examples

Aufzählungsdefinition

Ein Enum ist eine Menge logisch verwandter Konstanten.

```
Enum Size
    Small
    Medium
    Large
End Enum

Public Sub Order(shirtSize As Size)
    Select Case shirtSize
        Case Size.Small
            ' ...
        Case Size.Medium
            ' ...
        Case Size.Large
            ' ...
    End Select
End Sub
```

Member-Initialisierung

Jedes Enumerationsmitglied kann mit einem Wert initialisiert werden. Wenn für ein Element kein Wert angegeben ist, wird der Standardwert auf 0 gesetzt (wenn es das erste Mitglied in der Mitgliederliste ist) oder auf einen Wert, der um 1 größer ist als der Wert des vorhergehenden Mitglieds.

```
Module Module1

    Enum Size
        Small
        Medium = 3
        Large
    End Enum

    Sub Main()
        Console.WriteLine(Size.Small)     ' prints 0
        Console.WriteLine(Size.Medium)    ' prints 3
        Console.WriteLine(Size.Large)     ' prints 4

        ' Waits until user presses any key
        Console.ReadKey()
    End Sub

End Module
```

Das Attribut Flags

Mit dem Attribut `<Flags>` wird die Aufzählung zu einer Gruppe von Flags. Dieses Attribut ermöglicht das Zuweisen mehrerer Werte zu einer Aufzählungsvariablen. Die Mitglieder einer Flag-Enum sollten mit Potenzen von 2 (1, 2, 4, 8 ...) initialisiert werden.

```
Module Module1

    <Flags>
    Enum Material
        Wood = 1
        Plastic = 2
        Metal = 4
        Stone = 8
    End Enum

    Sub Main()
        Dim houseMaterials As Material = Material.Wood Or Material.Stone
        Dim carMaterials as Material = Material.Plastic Or Material.Metal
        Dim knifeMaterials as Material = Material.Metal

        Console.WriteLine(houseMaterials.ToString()) 'Prints "Wood, Stone"
        Console.WriteLine(CType(carMaterials, Integer)) 'Prints 6
    End Sub

End Module
```

HasFlag ()

Mit der `HasFlag()` -Methode kann `HasFlag()` werden, ob ein Flag gesetzt ist.

```
Module Module1

    <Flags>
    Enum Material
        Wood = 1
        Plastic = 2
        Metal = 4
        Stone = 8
    End Enum

    Sub Main()
        Dim houseMaterials As Material = Material.Wood Or Material.Stone

        If houseMaterials.HasFlag(Material.Stone) Then
            Console.WriteLine("the house is made of stone")
        Else
            Console.WriteLine("the house is not made of stone")
        End If
    End Sub

End Module
```

Weitere Informationen zum Flags-Attribut und seiner Verwendung finden Sie in der offiziellen Microsoft-Dokumentation .

String-Analyse

Eine Enum-Instanz kann erstellt werden, indem eine Zeichenfolgendarstellung des Enums analysiert wird.

```
Module Module1

    Enum Size
        Small
        Medium
        Large
    End Enum

    Sub Main()
        Dim shirtSize As Size =  DirectCast([Enum].Parse(GetType(Size), "Medium"), Size)

        ' Prints 'Medium'
        Console.WriteLine(shirtSize.ToString())

        ' Waits until user presses any key
        Console.ReadKey()
    End Sub

End Module
```

Siehe auch: Eine Zeichenfolge in einen Enum-Wert in VB.NET parsen

GetNames ()

Gibt die Namen der Konstanten im angegebenen Enum als String-Array zurück:

```
Module Module1

  Enum Size
    Small
    Medium
    Large
  End Enum

  Sub Main()
    Dim sizes = [Enum].GetNames(GetType(Size))

    For Each size In sizes
      Console.WriteLine(size)
      Next
  End Sub

End Module
```

Ausgabe:

Klein

Mittel

Groß

GetValues ()

"Diese Methode ist hilfreich für das Durchlaufen von Enum-Werten."

```
Enum Animal
    Dog = 1
    Cat = 2
    Frog = 4
End Enum

Dim Animals = [Enum].GetValues(GetType(Animal))

For Each animal in Animals
    Console.WriteLine(animal)
Next
```

Drucke:

1

2

4

ToString ()

Die `ToString` Methode für eine Aufzählung gibt den Namen der Aufzählung zurück. Zum Beispiel:

```
Module Module1
    Enum Size
        Small
        Medium
        Large
    End Enum

    Sub Main()
        Dim shirtSize As Size = Size.Medium
        Dim output As String = shirtSize.ToString()
        Console.WriteLine(output)   ' Writes "Medium"
    End Sub
End Module
```

Wenn jedoch die Zeichenfolgendarstellung des tatsächlichen Ganzzahlwerts der Aufzählung gewünscht wird, können Sie die Aufzählung in eine `Integer` `ToString` und `ToString` :

```
Dim shirtSize As Size = Size.Medium
Dim output As String = CInt(shirtSize).ToString()
Console.WriteLine(output)   ' Writes "1"
```

Bestimmen Sie, ob für ein Enum FlagsAttribute angegeben wurde oder nicht

Das nächste Beispiel kann verwendet werden, um zu bestimmen, ob für eine Enumeration **FlagsAttribute** angegeben wurde. Die angewandte Methodik basiert auf **Reflection** .

Dieses Beispiel ergibt ein `True` Ergebnis:

```
Dim enu As [Enum] = New FileAttributes()
Dim hasFlags As Boolean = enu.GetType().GetCustomAttributes(GetType(FlagsAttribute),
inherit:=False).Any()
Console.WriteLine("{0} Enum has FlagsAttribute?: {1}", enu.GetType().Name, hasFlags)
```

Dieses Beispiel liefert ein `False` Ergebnis:

```
Dim enu As [Enum] = New ConsoleColor()
Dim hasFlags As Boolean = enu.GetType().GetCustomAttributes(GetType(FlagsAttribute),
inherit:=False).Any()
Console.WriteLine("{0} Enum has FlagsAttribute?: {1}", enu.GetType().Name, hasFlags)
```

Wir können eine generische Nutzungserweiterungsmethode wie diese entwerfen:

```
<DebuggerStepThrough>
<Extension>
<EditorBrowsable(EditorBrowsableState.Always)>
Public Function HasFlagsAttribute(ByVal sender As [Enum]) As Boolean
    Return sender.GetType().GetCustomAttributes(GetType(FlagsAttribute), inherit:=False).Any()
End Function
```

Verwendungsbeispiel:

```
Dim result As Boolean = (New FileAttributes).HasFlagsAttribute()
```

For-Each-Flag (Flag-Iteration)

In einigen sehr spezifischen Szenarien würden wir die Notwendigkeit sehen, für jedes Flag der Quellenzählung eine spezifische Aktion auszuführen.

Wir können eine einfache *generische* Erweiterungsmethode schreiben, um diese Aufgabe zu realisieren.

```
<DebuggerStepThrough>
<Extension>
<EditorBrowsable(EditorBrowsableState.Always)>
Public Sub ForEachFlag(Of T)(ByVal sender As [Enum],
                             ByVal action As Action(Of T))

    For Each flag As T In sender.Flags(Of T)
        action.Invoke(flag)
    Next flag

End Sub
```

Verwendungsbeispiel:

```
Dim flags As FileAttributes = (FileAttributes.ReadOnly Or FileAttributes.Hidden)
```

```
flags.ForEachFlag(Of FileAttributes)(
      Sub(ByVal x As FileAttributes)
          Console.WriteLine(x.ToString())
      End Sub)
```

Bestimmen Sie die Anzahl der Flags in einer Flagkombination

Das nächste Beispiel soll die Anzahl der Flags in der angegebenen Flagkombination zählen.

Das Beispiel wird als Erweiterungsmethode bereitgestellt:

```
<DebuggerStepThrough>
<Extension>
<EditorBrowsable(EditorBrowsableState.Always)>
Public Function CountFlags(ByVal sender As [Enum]) As Integer
    Return sender.ToString().Split(","c).Count()
End Function
```

Verwendungsbeispiel:

```
Dim flags As FileAttributes = (FileAttributes.Archive Or FileAttributes.Compressed)
Dim count As Integer = flags.CountFlags()
Console.WriteLine(count)
```

Finden Sie den nächsten Wert in einer Aufzählung

Der nächste Code veranschaulicht, wie der nächste Wert einer **Aufzählung ermittelt wird** .

Zuerst definieren wir dieses **Enum** , das zur Angabe von Suchkriterien (Suchrichtung) dient.

```
Public Enum EnumFindDirection As Integer
    Nearest = 0
    Less = 1
    LessOrEqual = 2
    Greater = 3
    GreaterOrEqual = 4
End Enum
```

Und jetzt implementieren wir den Suchalgorithmus:

```
<DebuggerStepThrough>
Public Shared Function FindNearestEnumValue(Of T)(ByVal value As Long,
                                                  ByVal direction As EnumFindDirection) As T

    Select Case direction

        Case EnumFindDirection.Nearest
            Return (From enumValue As T In [Enum].GetValues(GetType(T)).Cast(Of T)()
                    Order By Math.Abs(value - Convert.ToInt64(enumValue))
                    ).FirstOrDefault

        Case EnumFindDirection.Less
            If value < Convert.ToInt64([Enum].GetValues(GetType(T)).Cast(Of T).First) Then
```

```
                Return [Enum].GetValues(GetType(T)).Cast(Of T).FirstOrDefault

            Else
                Return (From enumValue As T In [Enum].GetValues(GetType(T)).Cast(Of T)()
                        Where Convert.ToInt64(enumValue) < value
                        ).LastOrDefault
            End If

        Case EnumFindDirection.LessOrEqual
            If value < Convert.ToInt64([Enum].GetValues(GetType(T)).Cast(Of T).First) Then
                Return [Enum].GetValues(GetType(T)).Cast(Of T).FirstOrDefault

            Else
                Return (From enumValue As T In [Enum].GetValues(GetType(T)).Cast(Of T)()
                        Where Convert.ToInt64(enumValue) <= value
                        ).LastOrDefault
            End If

        Case EnumFindDirection.Greater
            If value > Convert.ToInt64([Enum].GetValues(GetType(T)).Cast(Of T).Last) Then
                Return [Enum].GetValues(GetType(T)).Cast(Of T).LastOrDefault

            Else
                Return (From enumValue As T In [Enum].GetValues(GetType(T)).Cast(Of T)()
                        Where Convert.ToInt64(enumValue) > value
                        ).FirstOrDefault
            End If

        Case EnumFindDirection.GreaterOrEqual
            If value > Convert.ToInt64([Enum].GetValues(GetType(T)).Cast(Of T).Last) Then
                Return [Enum].GetValues(GetType(T)).Cast(Of T).LastOrDefault

            Else
                Return (From enumValue As T In [Enum].GetValues(GetType(T)).Cast(Of T)()
                        Where Convert.ToInt64(enumValue) >= value
                        ).FirstOrDefault
            End If

    End Select

End Function
```

Verwendungsbeispiel:

```
Public Enum Bitrate As Integer
    Kbps128 = 128
    Kbps192 = 192
    Kbps256 = 256
    Kbps320 = 320
End Enum

Dim nearestValue As Bitrate = FindNearestEnumValue(Of Bitrate)(224,
EnumFindDirection.GreaterOrEqual)
```

Enum online lesen: https://riptutorial.com/de/vb-net/topic/1809/enum

Kapitel 19: Erweiterungsmethoden

Bemerkungen

Erweiterungsmethoden sind Methoden (`Sub` oder `Function`), die einem Typ `Function` hinzufügen (dies kann ein Referenztyp oder ein Werttyp sein). Diese Typen können Ihnen gehören oder nicht.

Sie befinden sich möglicherweise in derselben Baugruppe wie der Typ, den sie ändern möchten. Sie können ein Opt-In für Ihre Erweiterungsmethoden zulassen, indem Sie sie in ihrem eigenen Namespace isolieren. Wenn Sie möchten, können Sie sie immer verfügbar machen, indem Sie sie in demselben Namespace wie der Typ angeben, den sie ändern (vorausgesetzt, dass alle Assemblyreferenzen vorhanden und korrekt sind). Ein gutes Beispiel für den Opt-In-Stil von Erweiterungsmethoden finden Sie im Entity Framework Core 1.0-Projekt auf GitHub.

Erweiterungsmethoden in VB haben einige Anforderungen:

- Erweiterungsmethoden dürfen nur in Modulen deklariert werden.
- Erweiterungsmethoden müssen mit dem `Extension()` Attribut versehen werden.
- Der ExtensionAttribute-Namespace muss in Ihrem Modul verfügbar sein.
  `Imports System.Runtime.CompilerServices`
- Der erste Parameter der Methode muss von einem Typ sein, an den diese Methode angehängt wird.
- Der erste Parameter der Methode repräsentiert die Instanz, für die diese Methode ausgeführt wird. (Entspricht `Me` wenn dies eine echte Instanzmethode wäre).
- Eine Erweiterungsmethode kann als reguläre Methode aufgerufen werden, indem alle Parameter angegeben werden, sofern sie nicht für das instanziierte Objekt aufgerufen werden.

Examples

Erweiterungsmethode erstellen

Erweiterungsmethoden sind nützlich, um das Verhalten von Bibliotheken zu erweitern, die uns nicht gehören.

Sie werden dank des syntaktischen Zuckers des Compilers ähnlich wie die Instanzmethoden verwendet:

```
Sub Main()
    Dim stringBuilder = new StringBuilder()

    'Extension called directly on the object.
    stringBuilder.AppendIf(true, "Condition was true")

    'Extension called as a regular method. This defeats the purpose
    'of an extension method but should be noted that it is possible.
    AppendIf(stringBuilder, true, "Condition was true")
```

```
End Sub

<Extension>
Public Function AppendIf(stringBuilder As StringBuilder, condition As Boolean, text As String)
As StringBuilder
    If(condition) Then stringBuilder.Append(text)

    Return stringBuilder
End Function
```

Für eine verwendbare Erweiterungsmethode benötigt die Methode das `Extension` und muss in einem `Module` deklariert werden.

Die Sprache mit Erweiterungsmethoden funktioneller machen

Eine gute Verwendung der Erweiterungsmethode besteht darin, die Sprache funktionaler zu gestalten

```
Sub Main()
    Dim strings = { "One", "Two", "Three" }

    strings.Join(Environment.NewLine).Print()
End Sub

<Extension>
Public Function Join(strings As IEnumerable(Of String), separator As String) As String
    Return String.Join(separator, strings)
End Function

<Extension>
Public Sub Print(text As String)
    Console.WriteLine(text)
End Sub
```

Padding Numerics

```
Public Module Usage
  Public Sub LikeThis()
    Dim iCount As Integer
    Dim sCount As String

    iCount = 245
    sCount = iCount.PadLeft(4, "0")

    Console.WriteLine(sCount)
    Console.ReadKey()
  End Sub
End Module



Public Module Padding
  <Extension>
  Public Function PadLeft(Value As Integer, Length As Integer) As String
    Return Value.PadLeft(Length, Space(Length))
  End Function
```

```
  <Extension>
  Public Function PadRight(Value As Integer, Length As Integer) As String
    Return Value.PadRight(Length, Space(Length))
  End Function



  <Extension>
  Public Function PadLeft(Value As Integer, Length As Integer, Character As Char) As String
    Return CStr(Value).PadLeft(Length, Character)
  End Function



  <Extension>
  Public Function PadRight(Value As Integer, Length As Integer, Character As Char) As String
    Return CStr(Value).PadRight(Length, Character)
  End Function
End Module
```

Montageversion von einem starken Namen abrufen

Beispiel für den Aufruf einer Erweiterungsmethode als Erweiterung und als reguläre Methode.

```
public Class MyClass
  Sub Main()

        'Extension called directly on the object.
        Dim Version = Assembly.GetExecutingAssembly.GetVersionFromAssembly()

        'Called as a regular method.
        Dim Ver = GetVersionFromAssembly(SomeOtherAssembly)

    End Sub
End Class
```

Die Erweiterungsmethode in einem Modul. Machen Sie das Modul öffentlich, wenn Erweiterungen zu einer DLL kompiliert werden und in einer anderen Assembly referenziert werden.

```
Public Module Extensions
    ''' <summary>
    ''' Returns the version number from the specified assembly using the assembly's strong
name.
    ''' </summary>
    ''' <param name="Assy">[Assembly] Assembly to get the version info from.</param>
    ''' <returns>[String]</returns>
    <Extension>
    Friend Function GetVersionFromAssembly(ByVal Assy As Assembly) As String
        Return Split(Split(Assy.FullName,",")(1),"=")(1)
    End Function
End Module
```

Erweiterungsmethoden online lesen: https://riptutorial.com/de/vb-

net/topic/1592/erweiterungsmethoden

Kapitel 20: Fehlerbehandlung

Examples

Versuchen Sie ... Fang ... Endlich Statement

Struktur:

```
Try
    'Your program will try to run the code in this block.
    'If any exceptions are thrown, the code in the Catch Block will be executed,
    'without executing the lines after the one which caused the exception.
Catch ex As System.IO.IOException
    'If an exception occurs when processing the Try block, each Catch statement
    'is examined in textual order to determine which handles the exception.
    'For example, this Catch block handles an IOException.
Catch ex As Exception
    'This catch block handles all Exception types.
    'Details of the exception, in this case, are in the "ex" variable.
    'You can show the error in a MessageBox with the below line.
    MessageBox.Show(ex.Message)
Finally
    'A finally block is always executed, regardless of if an Exception occurred.
End Try
```

Beispielcode:

```
Try
    Dim obj = Nothing
    Dim prop = obj.Name 'This line will throw a NullReferenceException

    Console.WriteLine("Test.") ' This line will NOT be executed
Catch ex As System.IO.IOException
    ' Code that reacts to IOException.
Catch ex As NullReferenceException
    ' Code that reacts to a NullReferenceException
    Console.WriteLine("NullReferenceException: " & ex.Message)
    Console.WriteLine("Stack Trace: " & ex.StackTrace)
Catch ex As Exception
    ' Code that reacts to any other exception.
Finally
    ' This will always be run, regardless of if an exception is thrown.
    Console.WriteLine("Completed")
End Try
```

Benutzerdefinierte Ausnahme erstellen und auslösen

Sie können eine benutzerdefinierte Ausnahme erstellen und diese während der Ausführung Ihrer Funktion auslösen. In der Regel sollten Sie nur dann eine Ausnahme auslösen, wenn Ihre Funktion ihre definierte Funktionalität nicht erreichen kann.

```
Private Function OpenDatabase(Byval Server as String, Byval User as String, Byval Pwd as
```

```
String)
    if Server.trim="" then
        Throw new Exception("Server Name cannot be blank")
    elseif User.trim ="" then
        Throw new Exception("User name cannot be blank")
    elseif Pwd.trim="" then
        Throw new Exception("Password cannot be blank")
    endif

    'Here add codes for connecting to the server
End function
```

Versuchen Sie Catch in Database Operation

Sie können Try..Catch zum Rollback der Datenbankoperation verwenden, indem Sie die Rollback-Anweisung im Catch-Segment platzieren.

```
    Try
        'Do the database operation...
        xCmd.CommandText = "INSERT into ...."
        xCmd.ExecuteNonQuery()

        objTrans.Commit()
        conn.Close()
    Catch ex As Exception
        'Rollback action when something goes off
        objTrans.Rollback()
        conn.Close()
    End Try
```

Die nicht einfangbare Ausnahme

Obwohl `Catch ex As Exception` behauptet, dass es alle Ausnahmen verarbeiten kann, gibt es eine Ausnahme (kein Wortspiel vorgesehen).

```
Imports System
Static Sub StackOverflow() ' Again no pun intended
    StackOverflow()
End Sub
Static Sub Main()
    Try
        StackOverflow()
    Catch ex As Exception
        Console.WriteLine("Exception caught!")
    Finally
        Console.WriteLine("Finally block")
    End Try
End Sub
```

Hoppla ... Es gibt eine nicht erfasste `System.StackOverflowException` während die Konsole nichts gedruckt hat! Laut MSDN

> Ab .NET Framework 2.0 können Sie ein StackOverflowException-Objekt mit einem try / catch-Block nicht abfangen. Der entsprechende Prozess wird standardmäßig beendet.

> Daher sollten Sie Ihren Code schreiben, um einen Stapelüberlauf zu erkennen und zu verhindern.

Daher ist `System.StackOverflowException` nicht `System.StackOverflowException` . Hüte dich davor!

Kritische Ausnahmen

Im Allgemeinen sind die meisten Ausnahmen nicht so kritisch, aber es gibt einige wirklich schwerwiegende Ausnahmen, mit denen Sie möglicherweise nicht umgehen können, z. B. die berühmte `System.StackOverflowException` . Es gibt jedoch andere, die möglicherweise durch `Catch ex As Exception` ausgeblendet werden, z. B. `System.OutOfMemoryException` , `System.BadImageFormatException` und `System.InvalidProgramException` . Es ist eine gute Programmierpraxis, diese wegzulassen, wenn Sie nicht richtig damit umgehen können. Um diese Ausnahmen herauszufiltern, benötigen wir eine Hilfsmethode:

```
Public Shared Function IsCritical(ex As Exception) As Boolean
    Return TypeOf ex Is OutOfMemoryException OrElse
           TypeOf ex Is AppDomainUnloadedException OrElse
           TypeOf ex Is AccessViolationException OrElse
           TypeOf ex Is BadImageFormatException OrElse
           TypeOf ex Is CannotUnloadAppDomainException OrElse
           TypeOf ex Is ExecutionEngineException OrElse ' Obsolete one, but better to include
           TypeOf ex Is InvalidProgramException OrElse
           TypeOf ex Is System.Threading.ThreadAbortException
End Function
```

Verwendungszweck:

```
Try
    SomeMethod()
Catch ex As Exception When Not IsCritical(ex)
    Console.WriteLine("Exception caught: " & ex.Message)
End Try
```

Fehlerbehandlung online lesen: https://riptutorial.com/de/vb-net/topic/4232/fehlerbehandlung

Kapitel 21: FTP-Server

Syntax

- My.Computer.Network.DownloadFile (serverFile As String, localFile As String)
- My.Computer.Network.DownloadFile (serverFile As String, localFile As String, Benutzer As String, Passwort As String)
- My.Computer.Network.UploadFile (localFile As String, serverFile As String)
- My.Computer.Network.UploadFile (localFile As String, serverFile As String, Benutzer As String, Kennwort As String)

Examples

Laden Sie die Datei vom FTP-Server herunter

```
My.Computer.Network.DownloadFile("ftp://server.my/myfile.txt", "donwloaded_file.txt")
```

Dieser Befehl lädt die Datei `myfile.txt` vom Server `server.my` und speichert sie als `donwloaded_file.txt` im Arbeitsverzeichnis. Sie können den absoluten Pfad für die heruntergeladene Datei angeben.

Laden Sie die Datei vom FTP-Server herunter, wenn Sie sich anmelden möchten

```
My.Computer.Network.DownloadFile("ftp://srv.my/myfile.txt", "donwload.txt", "Peter", "1234")
```

Dieser Befehl lädt die Datei `myfile.txt` vom Server mit dem Namen `srv.my` und speichert sie als `donwload.txt` im Arbeitsverzeichnis. Sie können den absoluten Pfad für die heruntergeladene Datei angeben. Die Datei wird vom Benutzer Peter mit dem Kennwort 1234 heruntergeladen.

Laden Sie die Datei auf den FTP-Server hoch

```
My.Computer.Network.UploadFile("example.txt", "ftp://server.my/server_example.txt")
```

Dieser Befehl lädt die Datei `example.txt` aus dem Arbeitsverzeichnis (Sie können den absoluten Pfad angeben, wenn Sie möchten) auf den Server `server.my` . Die auf dem Server gespeicherte Datei wird als `server_example.txt` .

Laden Sie die Datei auf den FTP-Server hoch, wenn Sie sich anmelden müssen

```
My.Computer.Network.UploadFile("doc.txt", "ftp://server.my/on_server.txt", "Peter", "1234")
```

Dieser Befehl lädt `doc.txt` Datei `doc.txt` aus dem Arbeitsverzeichnis (Sie können den absoluten Pfad angeben, wenn Sie möchten) auf den Server namens `server.my` . Die auf dem Server gespeicherte Datei wird als `server_example.txt` . Fill wird auf dem Server von Benutzer Peter und Passwort 1234 gesendet.

FTP-Server online lesen: https://riptutorial.com/de/vb-net/topic/4078/ftp-server

Kapitel 22: Funktionen

Einführung

Die Funktion ist wie ein Sub. Die Funktion gibt jedoch einen Wert zurück. Eine Funktion kann einzelne oder mehrere Parameter akzeptieren.

Examples

Funktion definieren

Es ist sehr einfach, die Funktionen zu definieren.

```
Function GetAreaOfARectangle(ByVal Edge1 As Integer, ByVal Edge2 As Integer) As Integer
    Return Edge1 * Edge2
End Function
```

```
Dim Area As Integer = GetAreaOfARectangle(5, 8)
Console.Writeline(Area) 'Output: 40
```

Eine Funktion definieren # 2

```
Function Age(ByVal YourAge As Integer) As String
    Select Case YourAge
        Case Is < 18
            Return("You are younger than 18! You are teen!")
        Case 18 to 64
            Return("You are older than 18 but younger than 65! You are adult!")
        Case Is >= 65
            Return("You are older than 65! You are old!")
    End Select
End Function
```

```
Console.WriteLine(Age(48)) 'Output: You are older than 18 but younger than 65! You are adult!
```

Funktionen online lesen: https://riptutorial.com/de/vb-net/topic/10088/funktionen

Kapitel 23: GDI +

Examples

Grafikobjekt erstellen

Es gibt drei Möglichkeiten, ein Grafikobjekt zu erstellen

1. Aus dem **Paint Event**

Jedes Mal, wenn das Steuerelement neu gezeichnet (skaliert, aktualisiert ...) wird, wird dieses Ereignis aufgerufen. Verwenden Sie diesen Weg, wenn das Steuerelement konsistent auf das Steuerelement zugreifen soll

```
'this will work on any object's paint event, not just the form
Private Sub Form1_Paint(sender as Object, e as PaintEventArgs) Handles Me.Paint
  Dim gra as Graphics
  gra = e.Graphics
End Sub
```

2. **Grafik erstellen**

Dies wird meistens verwendet, wenn Sie eine einmalige Grafik im Steuerelement erstellen möchten oder das Steuerelement nicht selbst neu zeichnen soll

```
Dim btn as New Button
Dim g As Graphics = btn.CreateGraphics
```

3. Aus einer **vorhandenen Grafik**

Verwenden Sie diese Methode, wenn Sie eine vorhandene Grafik zeichnen und ändern möchten

```
'The existing image can be from a filename, stream or Drawing.Graphic
Dim image = New Bitmap("C:\TempBit.bmp")
Dim gr As Graphics = Graphics.FromImage(image)
```

Formen zeichnen

Zum Zeichnen einer Form müssen Sie ein Stiftobjekt definieren. Der `Pen` akzeptiert zwei Parameter:

1. Stiftfarbe oder Pinsel
2. Stiftbreite

Mit dem Stiftobjekt wird eine **Kontur** des Objekts erstellt, das Sie zeichnen möchten

Nach dem Definieren des Stiftes können Sie bestimmte Stifteigenschaften festlegen

```
   Dim pens As New Pen(Color.Purple)
   pens.DashStyle = DashStyle.Dash 'pen will draw with a dashed line
   pens.EndCap = LineCap.ArrowAnchor 'the line will end in an arrow
   pens.StartCap = LineCap.Round 'The line draw will start rounded
   '*Notice* - the Start and End Caps will not show if you draw a closed shape
```

Verwenden Sie dann das von Ihnen erstellte Grafikobjekt, um die Form zu zeichnen

```
  Private Sub GraphicForm_Paint(sender As Object, e As PaintEventArgs) Handles MyBase.Paint
    Dim pen As New Pen(Color.Blue, 15) 'Use a blue pen with a width of 15
    Dim point1 As New Point(5, 15) 'starting point of the line
    Dim point2 As New Point(30, 100) 'ending point of the line
    e.Graphics.DrawLine(pen, point1, point2)

    e.Graphics.DrawRectangle(pen, 60, 90, 200, 300) 'draw an outline of the rectangle
```

Standardmäßig entspricht die Breite des Stifts 1

```
    Dim pen2 as New Pen(Color.Orange) 'Use an orange pen with width of 1
    Dim origRect As New Rectangle(90, 30, 50, 60) 'Define bounds of arc
    e.Graphics.DrawArc(pen2, origRect, 20, 180) 'Draw arc in the rectangle bounds

End Sub
```

Formen füllen

Graphics.FillShapes zeichnet eine Form und füllt sie mit der angegebenen Farbe aus. Füllformen können verwenden

1. `Brush` - Füllt die Form mit einer Volltonfarbe

   ```
   Dim rect As New Rectangle(50, 50, 50, 50)
   e.Graphics.FillRectangle(Brushes.Green, rect) 'draws a rectangle that is filled with
   green

   e.Graphics.FillPie(Brushes.Silver, rect, 0, 180) 'draws a half circle that is filled with
   silver
   ```

2. `HatchBrush` Werkzeug - um die Form mit einem Muster zu füllen

```
Dim hBrush As New HatchBrush(HatchStyle.ZigZag, Color.SkyBlue, Color.Gray)
'creates a HatchBrush Tool with a background color of blue, foreground color of gray,
'and will fill with a zigzag pattern
Dim rectan As New Rectangle(100, 100, 100, 100)
e.Graphics.FillRectangle(hBrush, rectan)
```

3. `LinearGradientBrush` - `LinearGradientBrush` die Form mit einem Farbverlauf

   ```
   Dim lBrush As New LinearGradientBrush(point1, point2, Color.MediumVioletRed,
   Color.PaleGreen)
    Dim rect As New Rectangle(50, 50, 200, 200)
    e.Graphics.FillRectangle(lBrush, rect)
   ```

4. `TextureBrush` - um die Form mit einem Bild zu füllen

Sie können ein Bild aus Ressourcen, einer bereits definierten Bitmap oder aus einem Dateinamen auswählen

```
   Dim textBrush As New TextureBrush(New Bitmap("C:\ColorPic.jpg"))
    Dim rect As New Rectangle(400, 400, 100, 100)
    e.Graphics.FillPie(textBrush, rect, 0, 360)
```

Das `Hatch Brush Tool` und `LinearGradientBrush` importieren die folgende Anweisung: **Importiert System.Drawing.Drawing2D**

Text

Verwenden `DrawString` zum Zeichnen von Text in das Formular die `DrawString` Methode

Wenn Sie eine Zeichenfolge zeichnen, können Sie einen der vier oben aufgelisteten Pinsel verwenden

```
Dim lBrush As New LinearGradientBrush(point1, point2, Color.MediumVioletRed, Color.PaleGreen)
e.Graphics.DrawString("HELLO", New Font("Impact", 60, FontStyle.Bold), lBrush, New Point(40,
400))
'this will draw the word "Hello" at the given point, with a linearGradient Brush
```

Da Sie die Breite oder Höhe des Textes nicht definieren können, überprüfen Sie die Textgröße mithilfe von Text `Measure Text`

```
Dim lBrush As New LinearGradientBrush(point1, point2, Color.MediumVioletRed, Color.PaleGreen)
Dim TextSize =  e.Graphics.MeasureString("HELLO", New Font("Impact", 60, FontStyle.Bold),
lBrush)
'Use the TextSize to determine where to place the string, or if the font needs to be smaller
```

> Beispiel: Sie müssen das Wort "Test" auf das Formular zeichnen. Die Breite des Formulars beträgt 120. Verwenden Sie diese Schleife, um die Schriftgröße zu verringern, bis sie in die Formularbreite passt

```
Dim FontSize as Integer = 80
Dim TextSize = e.graphics.measeString("Test", New Font("Impact",FontSize, FontStyle.Bold), new
Brush(colors.Blue, 10)
Do while TextSize.Width >120
FontSize = FontSize -1
TextSize = e.graphics.measeString("Test", New Font("Impact",FontSize, FontStyle.Bold), new
Brush(colors.Blue, 10)
Loop
```

GDI + online lesen: https://riptutorial.com/de/vb-net/topic/5096/gdi-plus

Kapitel 24: Generics

Examples

Erstellen Sie eine generische Klasse

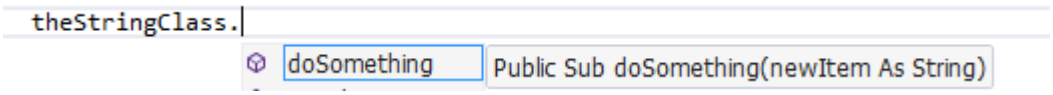
Es wird ein generischer Typ erstellt, um sich anzupassen, sodass für verschiedene Datentypen auf dieselbe Funktionalität zugegriffen werden kann.

```
Public Class SomeClass(Of T)
    Public Sub doSomething(newItem As T)
        Dim tempItem As T
        ' Insert code that processes an item of data type t.
    End Sub
End Class
```

Instanz einer generischen Klasse

Durch das Erstellen einer Instanz derselben Klasse mit einem anderen angegebenen Typ ändert sich das Interface der Klasse abhängig vom angegebenen Typ.

```
Dim theStringClass As New SomeClass(Of String)
Dim theIntegerClass As New SomeClass(Of Integer)
```



Definieren Sie eine 'generische' Klasse

Eine generische Klasse ist eine Klasse, die sich an einen später angegebenen Typ anpasst, so dass dieselbe Funktionalität für verschiedene Typen angeboten werden kann.

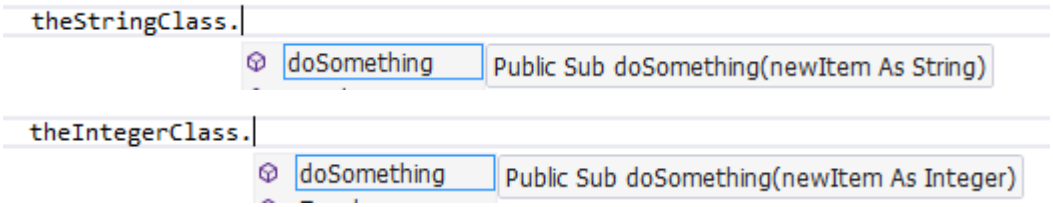
In diesem grundlegenden Beispiel wird eine generische Klasse erstellt. Es gibt ein Sub, das den generischen Typ T verwendet. Während der Programmierung dieser Klasse kennen wir den Typ von T. In diesem Fall hat T alle Eigenschaften von Object.

```
Public Class SomeClass(Of T)
    Public Sub doSomething(newItem As T)
        Dim tempItem As T
        ' Insert code that processes an item of data type t.
    End Sub
End Class
```

Verwenden Sie eine generische Klasse

In diesem Beispiel werden 2 Instanzen der SomeClass-Klasse erstellt. Je nach Typ haben die 2 Instanzen eine andere Schnittstelle:

```
Dim theStringClass As New SomeClass(Of String)
Dim theIntegerClass As New SomeClass(Of Integer)
```



Die bekannteste generische Klasse ist List (of)

Begrenzen Sie die möglichen Typen

Die möglichen Typen, die an eine neue Instanz von SomeClass übergeben werden, müssen SomeBaseClass erben. Dies kann auch eine Schnittstelle sein. Die Merkmale von SomeBaseClass sind innerhalb dieser Klassendefinition verfügbar.

```
Public Class SomeClass(Of T As SomeBaseClass)
    Public Sub DoSomething(newItem As T)
        newItem.DoSomethingElse()
        ' Insert code that processes an item of data type t.
    End Sub
End Class

Public Class SomeBaseClass
    Public Sub DoSomethingElse()
    End Sub
End Class
```

Erstellen Sie eine neue Instanz des angegebenen Typs

Das Erstellen einer neuen Bedeutung eines generischen Typs kann zur Kompilierzeit durchgeführt / überprüft werden.

```
Public Class SomeClass(Of T As {New})
    Public Function GetInstance() As T
        Return New T
    End Function
End Class
```

Oder bei begrenzten Typen:

```
Public Class SomeClass(Of T As {New, SomeBaseClass})
    Public Function GetInstance() As T
        Return New T
    End Function
End Class

Public Class SomeBaseClass
End Class
```

Die baseClass (wenn keine Object angegeben ist) muss einen Konstruktor ohne Parameter

haben.

Dies kann auch zur Laufzeit durch Reflektion erfolgen

Generics online lesen: https://riptutorial.com/de/vb-net/topic/6572/generics

Kapitel 25: Google Maps in einem Windows-Formular

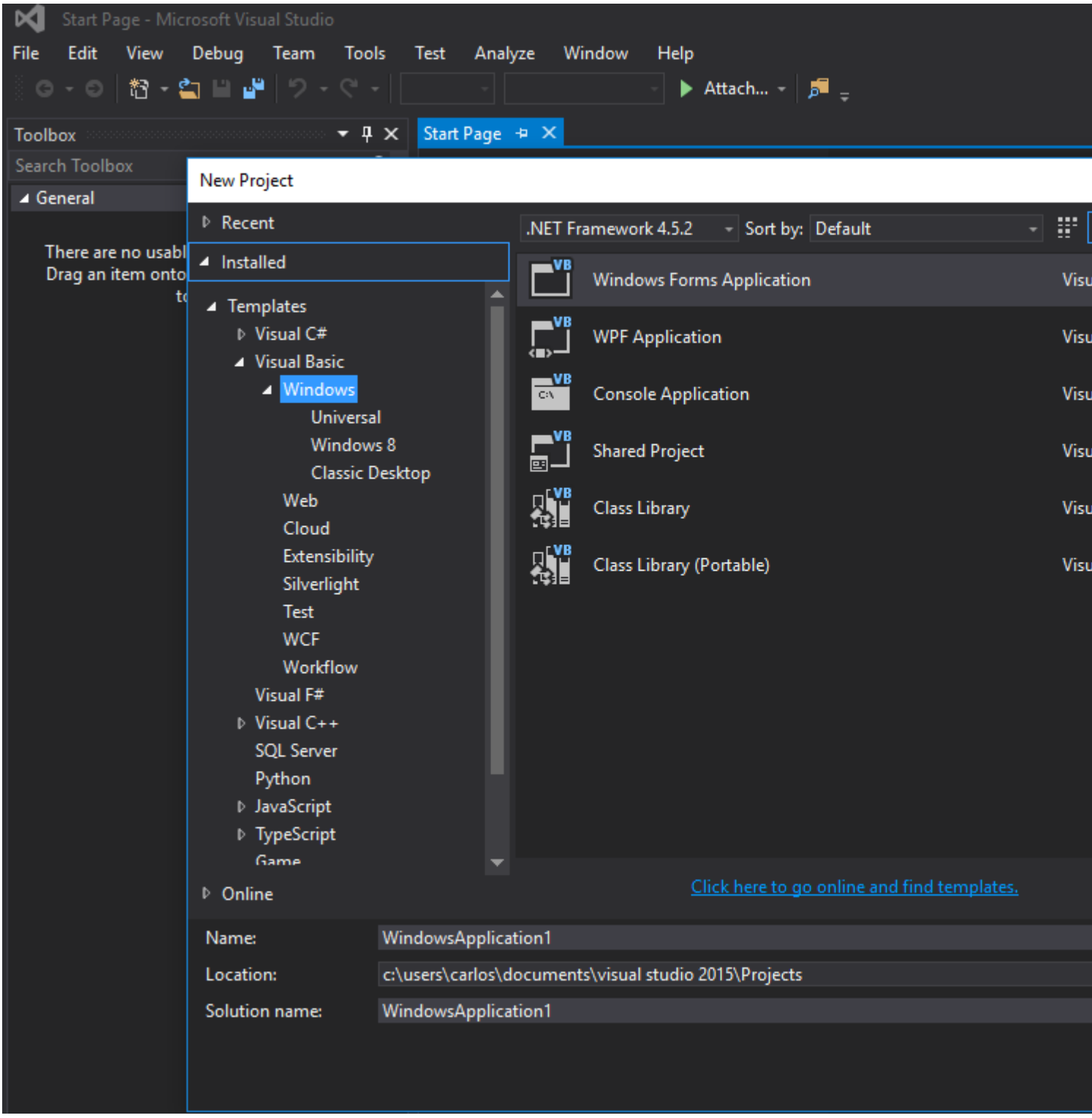
Examples

So verwenden Sie eine Google Map in einem Windows Form

Der erste Teil dieses Beispiels erläutert die Implementierung. Im zweiten werde ich erklären, wie es funktioniert. Dies versucht ein allgemeines Beispiel zu sein. Die Vorlage für die Karte (siehe Schritt 3) und die Beispielfunktionen sind vollständig anpassbar.

################################## **IMPLEMENTIERUNG** ################## ##################

Schritt 1. Erstellen Sie zunächst ein neues Projekt und wählen Sie Windows Form Application aus. Lassen wir den Namen als "Form1".



Schritt 2. Fügen Sie Ihrem Form1 ein Webbrowser-Steuerelement hinzu (das Ihre Karte enthalten wird). Nennen wir es "wbmap"

Schritt 3. Erstellen Sie eine HTML-Datei mit dem Namen "googlemap_template.html" mit Ihrem bevorzugten Texteditor, und fügen Sie den folgenden Code ein:

googlemap_template.html

```
<!DOCTYPE html>
<html>
  <head>
```

```
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge"/>
     <style type="text/css">
      html, body {
        height: 100%;
        margin: 0;
        padding: 0;
      }
      #gmap {
        height: 100%;
      }
     </style>
    <script type="text/javascript"
src="http://maps.google.com/maps/api/js?sensor=false"></script>
    <script type="text/javascript">
        function initialize() {
            //Use window.X instead of var X to make a variable globally available
            window.markers = new Array();
            window.marker_data = [[MARKER_DATA]];
            window.gmap = new google.maps.Map(document.getElementById('gmap'), {
            zoom: 15,
            center: new google.maps.LatLng(marker_data[0][0], marker_data[0][1]),
            mapTypeId: google.maps.MapTypeId.ROADMAP
          });
          var infowindow = new google.maps.InfoWindow();
          var newmarker, i;
          for (i = 0; i < marker_data.length; i++) {
              if (marker_data[0].length == 2) {
                  newmarker = new google.maps.Marker({
                      position: new google.maps.LatLng(marker_data[i][0], marker_data[i][1]),
                      map: gmap
                  });
              } else if (marker_data[0].length == 3) {
                  newmarker = new google.maps.Marker({
                      position: new google.maps.LatLng(marker_data[i][0], marker_data[i][1]),
                      map: gmap,
                      title: (marker_data[i][2])
                  });
              } else {
                  newmarker = new google.maps.Marker({
                      position: new google.maps.LatLng(marker_data[i][0], marker_data[i][1]),
                      map: gmap,
                      title: (marker_data[i][2]),
                      icon: (marker_data[i][3])
                  });
              }
            google.maps.event.addListener(newmarker, 'click', (function (newmarker, i) {
                return function () {
                    if (newmarker.title) {
                        infowindow.setContent(newmarker.title);
                        infowindow.open(gmap, newmarker);
                    }
                    gmap.setCenter(newmarker.getPosition());
                    // Calling functions written in the WF
                    window.external.showVbHelloWorld();
                    window.external.getMarkerDataFromJavascript(newmarker.title,i);
                }
            })(newmarker, i));
            markers[i] = newmarker;
          }
        }
```

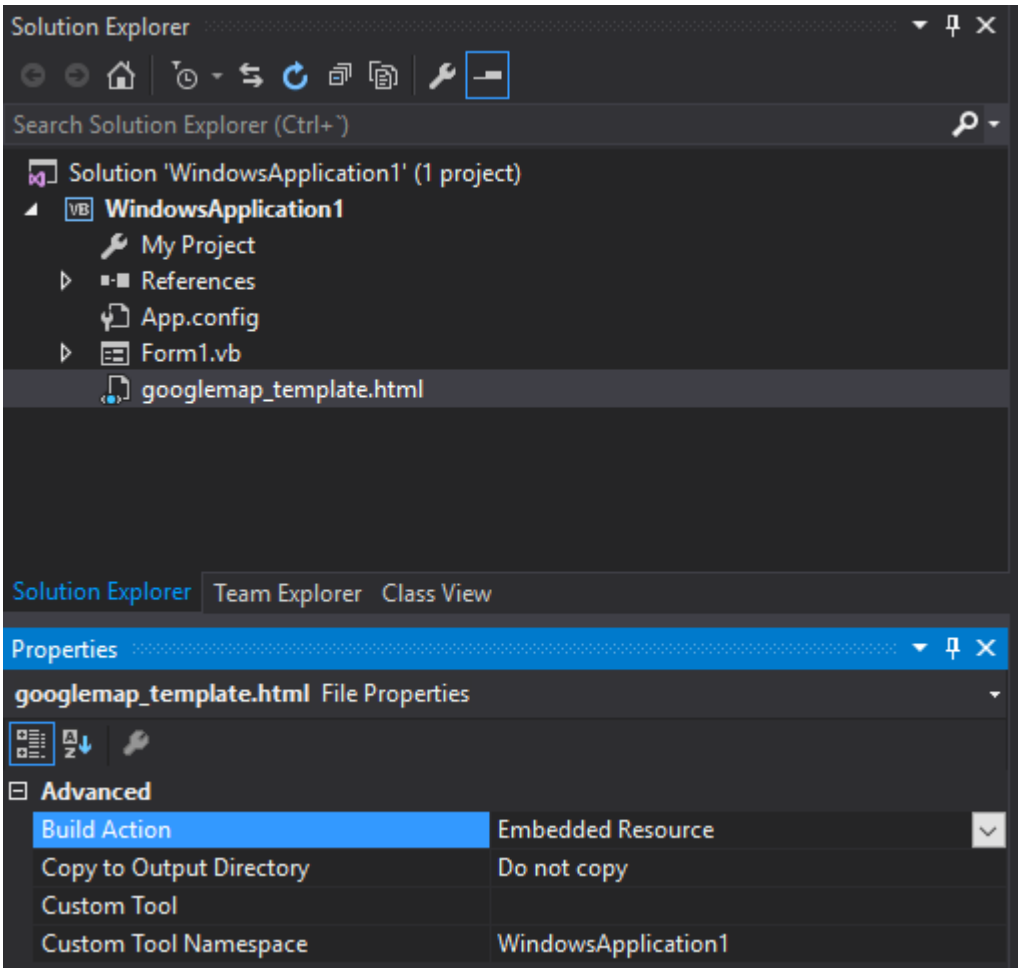
```
        google.maps.event.addDomListener(window, 'load', initialize);
    </script>
    <script type="text/javascript">
        // Function triggered from the WF with no arguments
        function showJavascriptHelloWorld() {
            alert("Hello world in HTML from WF");
        }
     </script>
      <script type="text/javascript">
        // Function triggered from the WF with a String argument
        function focusMarkerFromIdx(idx) {
            google.maps.event.trigger(markers[idx], 'click');
        }
      </script>
  </head>
  <body>
    <div id="gmap"></div>
  </body>
</html>
```

Dies wird als unsere Kartenvorlage dienen. Ich werde später erklären, wie es funktioniert.

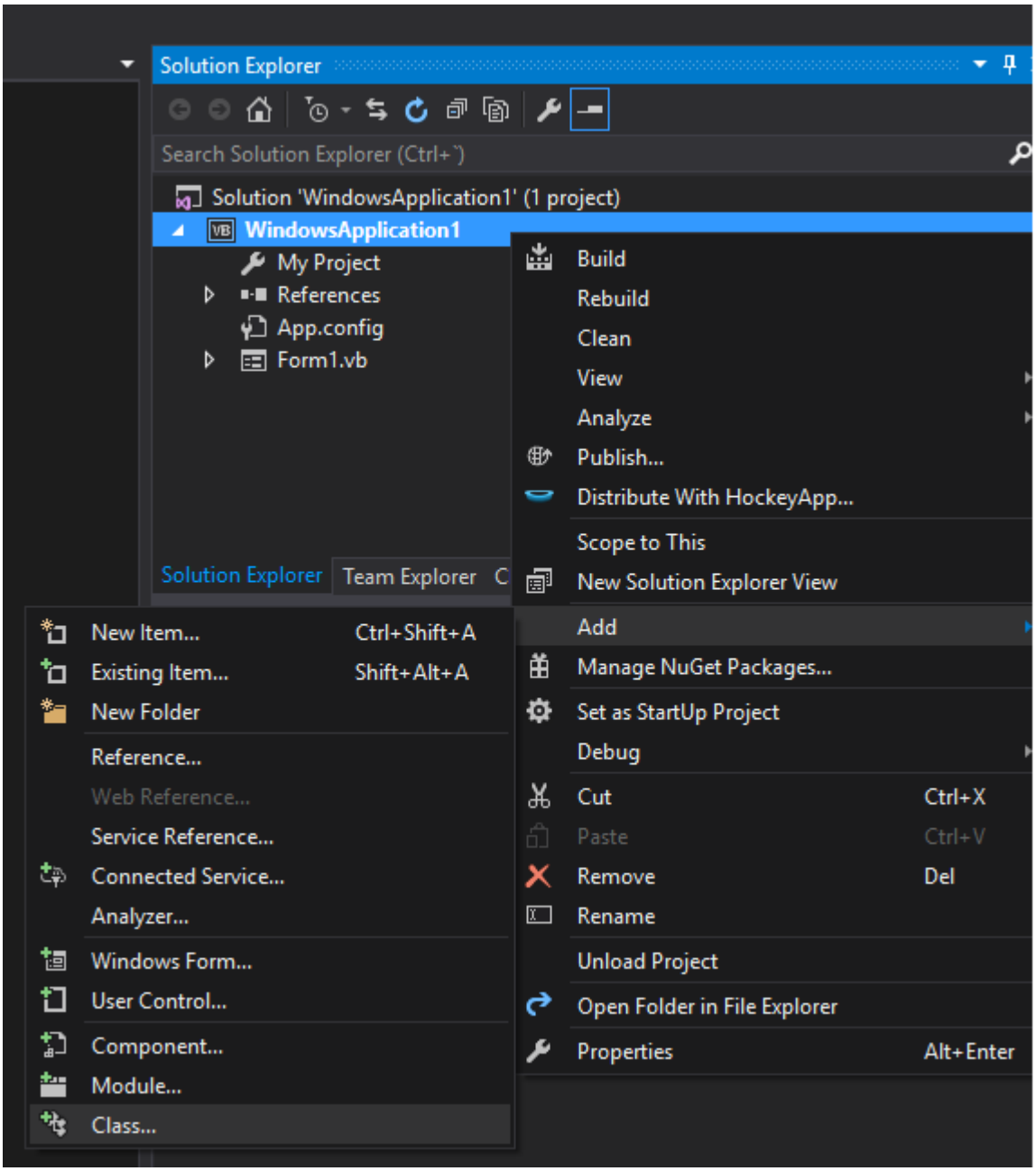
Schritt 4. Fügen Sie die Datei googlemap_template.hmtl Ihrem Projekt hinzu (klicken Sie mit der rechten Maustaste auf Ihr Projekt -> Hinzufügen -> Vorhandenes Element).

Schritt 5. Wenn es im Projektmappen-Explorer angezeigt wird, legen Sie seine Eigenschaften auf Folgendes fest:

- Aktion erstellen -> Eingebettete Ressource
- Custom Tool Namespace -> Geben Sie den Namen des Projekts ein



Schritt 6. Fügen Sie eine neue Klasse hinzu (klicken Sie mit der rechten Maustaste auf Ihr Projekt -> Hinzufügen -> Klasse). In meinem Beispiel werde ich es GoogleMapHelper nennen.



Schritt 7. Fügen Sie den folgenden Code in Ihre Klasse ein:

GoogleMapHelper.vb

```
    Imports System.IO
    Imports System.Reflection
    Imports System.Text

    Public Class GoogleMapHelper

    ' 1- googlemap_template.html must be copied in the main project folder
    ' 2- add the file into the Visual Studio Solution Explorer (add existing file)
    ' 3- set the properties of the file to:
    '                                   Build Action -> Embedded Resource
    '                                   Custom Tool Namespace -> write the name of the project

    Private Const ICON_FOLDER As String = "marker_icons/" 'images must be stored in a folder
inside  Debug/Release folder
    Private Const MAP_TEMPLATE As String = "WindowsApplication1.googlemap_template.html"
```

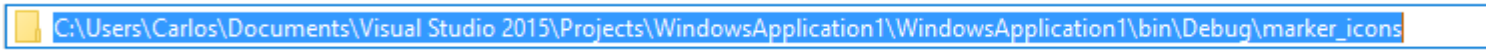
```
    Private Const TEXT_TO_REPLACE_MARKER_DATA As String = "[[MARKER_DATA]]"
    Private Const TMP_NAME As String = "tmp_map.html"


    Private mWebBrowser As WebBrowser

    'MARKER POSITIONS
    Private mPositions As Double(,) 'lat, lon
    ' marker data allows different formats to include lat,long and optionally title and icon:
    ' op1: mMarkerData = New String(N-1, 1) {{lat1, lon1}, {lat2, lon2}, {latN, lonN}}
    ' op2: mMarkerData = New String(N-1, 2) {{lat1, lon1,'title1'}, {lat2, lon2,'title2'},
{latN, lonN, 'titleN'}}
    ' op3: mMarkerData = New String(N-1, 3) {{lat1, lon1,'title1','image1.png'}, {lat2,
lon2,'title2','image2.png'}, {latN, lonN, 'titleN','imageN.png'}}
    Private mMarkerData As String(,) = Nothing


    Public Sub New(ByRef wb As WebBrowser, pos As Double(,))
        mWebBrowser = wb
        mPositions = pos
        mMarkerData = getMarkerDataFromPositions(pos)
    End Sub

    Public Sub New(ByRef wb As WebBrowser, md As String(,))
        mWebBrowser = wb
        mMarkerData = md
    End Sub

    Public Sub loadMap()
        mWebBrowser.Navigate(getMapTemplate())
    End Sub

    Private Function getMapTemplate() As String

        If mMarkerData Is Nothing Or mMarkerData.GetLength(1) > 4 Then
            MessageBox.Show("Marker data has not the proper size. It must have 2, 3 o 4
columns")
            Return Nothing
        End If

        Dim htmlTemplate As New StringBuilder()
        Dim tmpFolder As String = Environment.GetEnvironmentVariable("TEMP")
        Dim dataSize As Integer = mMarkerData.GetLength(1) 'number of columns
        Dim mMarkerDataAsText As String = String.Empty
        Dim myresourcePath As String = My.Resources.ResourceManager.BaseName
        Dim myresourcefullPath As String =
Path.GetFullPath(My.Resources.ResourceManager.BaseName)
        Dim localPath = myresourcefullPath.Replace(myresourcePath, "").Replace("\", "/") &
ICON_FOLDER

        htmlTemplate.AppendLine(getStringFromResources(MAP_TEMPLATE))
        mMarkerDataAsText = "["

        For i As Integer = 0 To mMarkerData.GetLength(0) - 1
            If i <> 0 Then
                mMarkerDataAsText += ","
            End If
            If dataSize = 2 Then 'lat,lon
                mMarkerDataAsText += "[" & mMarkerData(i, 0) & "," + mMarkerData(i, 1) & "]"
            ElseIf dataSize = 3 Then 'lat,lon and title
                mMarkerDataAsText += "[" & mMarkerData(i, 0) & "," + mMarkerData(i, 1) & ",'"
```

```
& mMarkerData(i, 2) & "']"
            ElseIf dataSize = 4 Then 'lat,lon,title and image
                mMarkerDataAsText += "[" & mMarkerData(i, 0) & "," + mMarkerData(i, 1) & ",'"
& mMarkerData(i, 2) & "','" & localPath & mMarkerData(i, 3) & "']" 'Ojo a las comillas simples
en las columnas 3 y 4
            End If
        Next

        mMarkerDataAsText += "]"
        htmlTemplate.Replace(TEXT_TO_REPLACE_MARKER_DATA, mMarkerDataAsText)

        Dim tmpHtmlMapFile As String = (tmpFolder & Convert.ToString("\")) + TMP_NAME
        Dim existsMapFile As Boolean = False
        Try
            existsMapFile = createTxtFile(tmpHtmlMapFile, htmlTemplate)
        Catch ex As Exception
            MessageBox.Show("Error writing temporal file", "Writing Error",
MessageBoxButtons.OK, MessageBoxIcon.[Error])
        End Try

        If existsMapFile Then
            Return tmpHtmlMapFile
        Else
            Return Nothing
        End If
    End Function

    Private Function getMarkerDataFromPositions(pos As Double(,)) As String(,)
        Dim md As String(,) = New String(pos.GetLength(0) - 1, 1) {}
        For i As Integer = 0 To pos.GetLength(0) - 1
            md(i, 0) = pos(i, 0).ToString("g", New System.Globalization.CultureInfo("en-US"))
            md(i, 1) = pos(i, 1).ToString("g", New System.Globalization.CultureInfo("en-US"))
        Next
        Return md
    End Function

    Private Function getStringFromResources(resourceName As String) As String
        Dim assem As Assembly = Me.[GetType]().Assembly

        Using stream As Stream = assem.GetManifestResourceStream(resourceName)
            Try
                Using reader As New StreamReader(stream)
                    Return reader.ReadToEnd()
                End Using
            Catch e As Exception
                Throw New Exception((Convert.ToString("Error de acceso al Recurso '") &
resourceName) + "'" & vbCr & vbLf + e.ToString())
            End Try
        End Using
    End Function

    Private Function createTxtFile(mFile As String, content As StringBuilder) As Boolean
        Dim mPath As String = Path.GetDirectoryName(mFile)
        If Not Directory.Exists(mPath) Then
            Directory.CreateDirectory(mPath)
        End If
        If File.Exists(mFile) Then
            File.Delete(mFile)
        End If
        Dim sw As StreamWriter = File.CreateText(mFile)
        sw.Write(content.ToString())
```

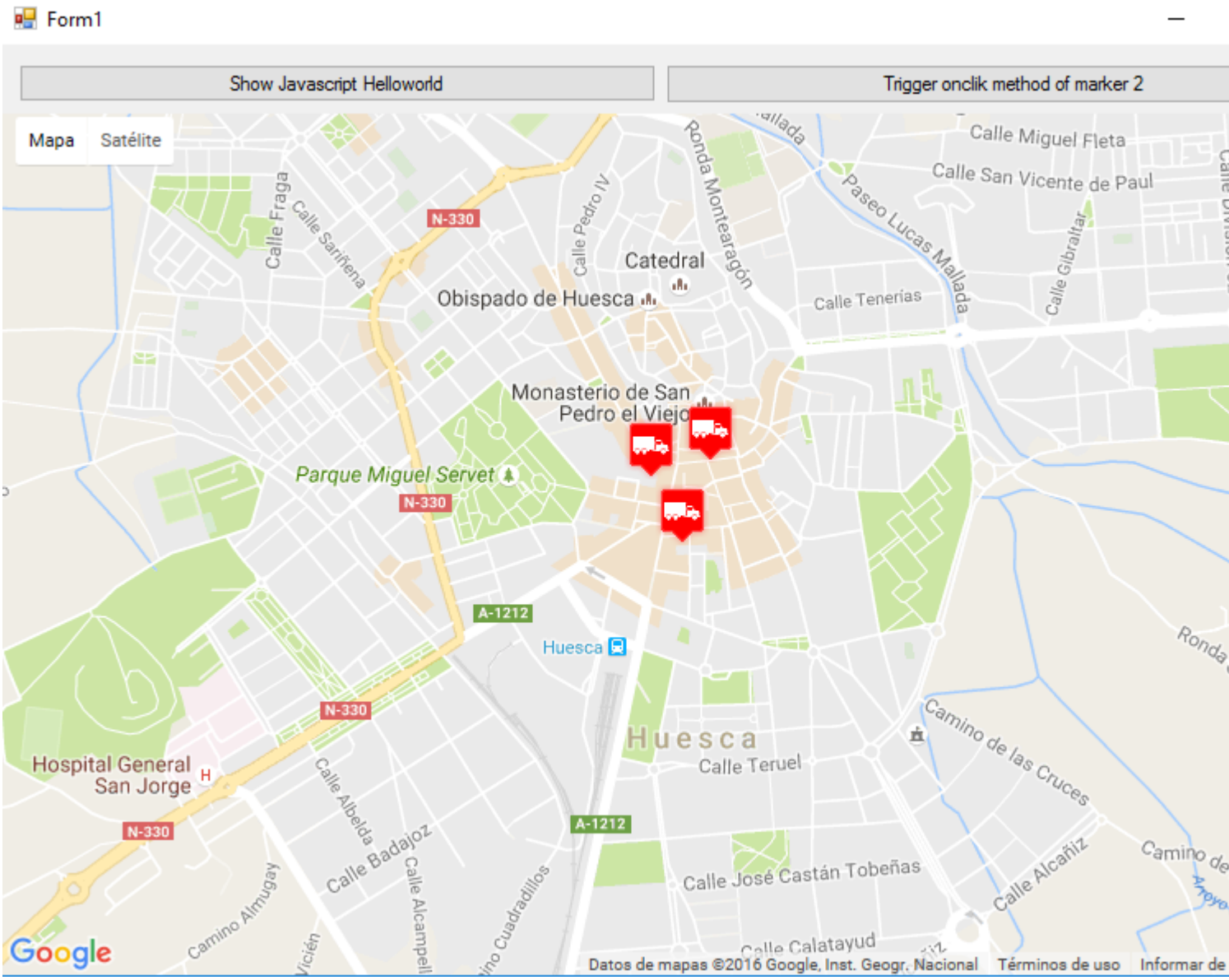
```
        sw.Close()
        Return True
    End Function
    End Class
```

Hinweis: Die MAP_TEMPLATE-Konstante muss den Namen Ihres Projekts enthalten

Schritt 8. Jetzt können wir unsere GoogleMapHelper-Klasse verwenden, um die Karte in unseren Webbrowser zu laden, indem Sie einfach die Methode loadMap () erstellen, eine Instanz erstellen und aufrufen. Wie Sie Ihre markerData erstellen, liegt bei Ihnen. In diesem Beispiel schreibe ich sie zur Klarstellung von Hand. Es gibt 3 Optionen zum Definieren der Markierungsdaten (siehe Kommentare zur GoogleMapHelper-Klasse). Wenn Sie die dritte Option (einschließlich Titel und Symbole) verwenden, müssen Sie einen Ordner mit dem Namen "marker_icons" (oder was auch immer Sie in der GoogleMapHelper-Konstante ICON_FOLDER definieren) in Ihrem Debug / Release-Ordner erstellen und Ihre PNG-Dateien dort ablegen. In meinem Fall:



Ich habe zwei Schaltflächen in Form1 erstellt, um zu veranschaulichen, wie die Karte und der WF miteinander interagieren. So sieht es aus:



Und hier ist der Code:

Form1.vb

```
Imports System.IO
Imports System.Reflection
Imports System.Security.Permissions
Imports System.Text
<PermissionSet(SecurityAction.Demand, Name:="FullTrust")>
<System.Runtime.InteropServices.ComVisible(True)>
Public Class Form1

Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load

    Me.wbmap.ObjectForScripting = Me

    Dim onlyPositions As Double(,) = New Double(2, 1) {{42.13557, -0.40806}, {42.13684, -
0.40884}, {42.13716, -0.40729}}
    Dim positonAndTitles As String(,) = New String(2, 2) {{"42.13557", "-0.40806", "marker0"},
{"42.13684", "-0.40884", "marker1"}, {"42.13716", "-0.40729", "marker2"}}
    Dim positonTitlesAndIcons As String(,) = New String(2, 3) {{"42.13557", "-0.40806",
"marker0", "truck_red.png"}, {"42.13684", "-0.40884", "marker1", "truck_red.png"},
```

```
{"42.13716", "-0.40729", "marker2", "truck_red.png"}}

    'Dim gmh As GoogleMapHelper = New GoogleMapHelper(wbmap, onlyPositions)
    'Dim gmh As GoogleMapHelper = New GoogleMapHelper(wbmap, positonAndTitles)
    Dim gmh As GoogleMapHelper = New GoogleMapHelper(wbmap, positonTitlesAndIcons)
    gmh.loadMap()
End Sub

'############################## CALLING JAVASCRIPT METHODS ##############################
'This methods call methods written in googlemap_template.html
Private Sub callMapJavascript(sender As Object, e As EventArgs) Handles Button1.Click
    wbmap.Document.InvokeScript("showJavascriptHelloWorld")
End Sub

Private Sub callMapJavascriptWithArguments(sender As Object, e As EventArgs) Handles
Button2.Click
    wbmap.Document.InvokeScript("focusMarkerFromIdx", New String() {2})
End Sub
'########################################################################################

'############################## METHODS CALLED FROM JAVASCRIPT ##########################
'This methods are called by the javascript defined in googlemap_template.html when some events
are triggered
Public Sub getMarkerDataFromJavascript(title As String, idx As String)
    MsgBox("Title: " & title & " idx: " & idx)
End Sub

Public Sub showVbHelloWorld()
    MsgBox("Hello world in WF from HTML")
End Sub
End Class
```

WICHTIG: Vergessen Sie nicht, diese Zeilen vor der Definition der Klasse Form1 hinzuzufügen:

```
<PermissionSet(SecurityAction.Demand, Name:="FullTrust")>
<System.Runtime.InteropServices.ComVisible(True)>
```

Sie müssen dem .NET-Framework mitteilen, dass wir fulltrust wollen und die Klasse für COM sichtbar machen, damit Form1 für JavaScript sichtbar ist.

Vergessen Sie das auch nicht in Ihrer Form1-Ladefunktion:

```
Me.wbmap.ObjectForScripting = Me
```

Es macht Ihre Form1-Klasse für JavaScript auf der Seite googlemap_template.hmtl verfügbar.

Jetzt können Sie ausführen und es sollte funktionieren

################################## **WIE ES FUNKTIONIERT**##############
##################

Unsere GoogleMapHelper-Klasse liest im Wesentlichen unsere googlemap_template.html, erstellt eine temporäre Kopie, ersetzt den Code, der mit den Markern ([[MARKER_DATA]]) zusammenhängt und führt die Seite in der Webbrowser-Steuerung unseres Formulars aus. Diese HTML-Datei durchläuft alle Markierungen und weist jedem einen Klick-Listener zu. Diese

Klickfunktion ist offensichtlich vollständig anpassbar. Im Beispiel wird ein Infofenster geöffnet, wenn die Markierung einen Titel hat, die Karte in dieser Markierung zentriert und zwei externe Funktionen aufgerufen, die in unserer Klasse Form1 definiert sind.

Andererseits können wir andere Javascript-Funktionen (mit oder ohne Argumente) in dieser HTML-Datei definieren, die von unserem Windows Form (mit wbmap.Document.InvokeScript) aufgerufen werden.

Google Maps in einem Windows-Formular online lesen: https://riptutorial.com/de/vb-net/topic/5903/google-maps-in-einem-windows-formular

Kapitel 26: Klassen

Einführung

Eine Klasse fasst verschiedene Funktionen, Methoden, Variablen und Eigenschaften zusammen, die als ihre Mitglieder bezeichnet werden. Eine Klasse kapselt die Member ein, auf die eine Instanz der Klasse, Objekt genannt, zugreifen kann. Klassen sind für den Programmierer äußerst nützlich, da sie die Aufgabe bequem und schnell machen, mit Eigenschaften wie Modularität, Wiederverwendbarkeit, Wartbarkeit und Lesbarkeit des Codes.

Klassen sind die Bausteine objektorientierter Programmiersprachen.

Examples

Klassen erstellen

Klassen bieten die Möglichkeit, eigene Typen innerhalb des .NET-Frameworks zu erstellen. In eine Klassendefinition können Sie Folgendes einschließen:

- Felder
- Eigenschaften
- Methoden
- Konstrukteure
- Veranstaltungen

Um eine Klasse zu deklarieren, verwenden Sie folgende Syntax:

```
Public Class Vehicle
End Class
```

Andere .NET-Typen können innerhalb der Klasse gekapselt und entsprechend angezeigt werden, wie unten gezeigt:

```
Public Class Vehicle
     Private Property _numberOfWheels As Integer
     Private Property _engineSize As Integer

     Public Sub New(engineSize As Integer, wheels As Integer)
         _numberOfWheels = wheels
         _engineSize = engineSize
     End Sub

     Public Function DisplayWheelCount() As Integer
         Return _numberOfWheels
     End Function
End Class
```

Abstrakte Klassen

Wenn Klassen eine gemeinsame Funktionalität haben, können Sie diese in einer Basis- oder abstrakten Klasse gruppieren. Abstrakte Klassen können eine teilweise oder keine Implementierung enthalten und ermöglichen es dem abgeleiteten Typ, die Basisimplementierung zu überschreiben.

Abstrakte Klassen in VisualBasic.NET müssen als `MustInherit` deklariert werden und können nicht instanziiert werden.

```
Public MustInherit Class Vehicle
     Private Property _numberOfWheels As Integer
     Private Property _engineSize As Integer

     Public Sub New(engineSize As Integer, wheels As Integer)
         _numberOfWheels = wheels
         _engineSize = engineSize
     End Sub

     Public Function DisplayWheelCount() As Integer
         Return _numberOfWheels
     End Function
End Class
```

Ein Untertyp kann dann diese abstrakte Klasse wie folgt `inherit` :

```
Public Class Car
    Inherits Vehicle
End Class
```

Das Auto erbt alle deklarierten Typen innerhalb des Fahrzeugs, kann jedoch nur basierend auf dem zugrunde liegenden Zugriffsmodifizierer darauf zugreifen.

```
Dim car As New Car()
car.DisplayWheelCount()
```

Im obigen Beispiel wird eine neue Car-Instanz erstellt. Dann wird die `DisplayWheelCount()` Methode aufgerufen, die die `Vehicles` Implementierung der Basisklasse `Vehicles` .

Klassen online lesen: https://riptutorial.com/de/vb-net/topic/3522/klassen

Kapitel 27: Komprimierte Textdatei im Handumdrehen lesen

Examples

Lesen der .gz-Textdatei Zeile für Zeile

Diese Klasse öffnet eine GZ-Datei (übliches Format komprimierter Protokolldateien) und gibt bei jedem Aufruf von `.NextLine()` eine Zeile zurück.

Für die temporäre Dekomprimierung wird kein Speicherplatz benötigt. Dies ist sehr nützlich für große Dateien.

```
Imports System.IO

Class logread_gz

  Private ptr As FileStream
  Private UnGZPtr As Compression.GZipStream
  Private line_ptr As StreamReader
  Private spath As String

  Sub New(full_filename As String)
    spath = full_filename
  End Sub

  Sub Open()
     Me.ptr = File.OpenRead(spath)
     Me.UnGZPtr = New Compression.GZipStream(ptr, Compression.CompressionMode.Decompress)
     Me.line_ptr = New StreamReader(UnGZPtr)
  End Sub()

  Function NextLine() As String
    'will return Nothing if EOF
    Return Me.line_ptr.ReadLine()
  End Function

  Sub Close()
    Me.line_ptr.Close()
    Me.line_ptr.Dispose()
    Me.UnGZPtr.Close()
    Me.UnGZPtr.Dispose()
    Me.ptr.Close()
    Me.ptr.Dispose()
  End Sub

End Class
```

Hinweis: Aus Gründen der Lesbarkeit ist keine Ausfallsicherheit gegeben.

Komprimierte Textdatei im Handumdrehen lesen online lesen: https://riptutorial.com/de/vb-net/topic/6960/komprimierte-textdatei-im-handumdrehen-lesen

Kapitel 28: Konsole

Examples

Console.ReadLine ()

```
Dim input as String = Console.ReadLine()
```

`Console.ReadLine()` liest die Konsoleneingabe vom Benutzer, bis die nächste neue Zeile erkannt wird (normalerweise durch Drücken der Eingabetaste oder der Eingabetaste). Die Codeausführung wird im aktuellen Thread angehalten, bis eine neue Zeile angegeben wird. Danach wird die nächste Codezeile ausgeführt.

Console.WriteLine ()

```
Dim x As Int32 = 128
Console.WriteLine(x) ' Variable '
Console.WriteLine(3) ' Integer '
Console.WriteLine(3.14159) ' Floating-point number '
Console.WriteLine("Hello, world") ' String '
Console.WriteLine(myObject) ' Outputs the value from calling myObject.ToString()
```

Die `Console.WriteLine()` -Methode druckt die angegebenen Argumente **mit** einer neuen Zeile am Ende. Dadurch werden alle bereitgestellten Objekte gedruckt, einschließlich, aber nicht beschränkt auf, Zeichenfolgen, Ganzzahlen, Variablen und Gleitkommazahlen.

Beim Schreiben von Objekten, die nicht explizit von den verschiedenen `WriteLine` Überladungen `WriteLine` werden (`WriteLine` Sie verwenden die Überladung, die einen Wert des Typs `Object` erwartet), verwendet WriteLine die `.ToString()` -Methode, um einen `String` zum tatsächlichen Schreiben zu generieren Objekte sollten die `.ToString` Methode `.ToString` und etwas aussagekräftiger als die Standardimplementierung erzeugen (die normalerweise nur den vollständig qualifizierten `.ToString` schreibt).

Console.Write ()

```
Dim x As Int32 = 128
Console.Write(x) ' Variable '
Console.Write(3) ' Integer '
Console.Write(3.14159) ' Floating-point number '
Console.Write("Hello, world") ' String '
```

Die `Console.Write()` -Methode ist mit der `Console.WriteLine()` -Methode identisch, mit der Ausnahme, dass das angegebene Argument bzw. die angegebenen Argumente **ohne** eine neue Zeile am Ende gedruckt werden. Diese Methode kann funktional mit `WriteLine` identisch `WriteLine` werden, indem am Ende der angegebenen Argumente eine Newline-Zeichenfolge hinzugefügt wird:

```
Console.Write("this is the value" & Environment.NewLine)
```

Console.Read ()

```
Dim inputCode As Integer = Console.Read()
```

`Console.Read()` erwartet eine Eingabe vom Benutzer und gibt bei Erhalt einen ganzzahligen Wert zurück, der dem Zeichencode des eingegebenen Zeichens entspricht. Wenn der Eingabestrom auf irgendeine Weise beendet ist, bevor die Eingabe erhalten werden kann, wird stattdessen -1 zurückgegeben.

Console.ReadKey ()

```
Dim inputChar As ConsoleKeyInfo = Console.ReadKey()
```

`Console.ReadKey()` erwartet eine Eingabe des Benutzers und gibt bei Erhalt ein Objekt der Klasse `ConsoleKeyInfo` , das Informationen zu dem Zeichen enthält, das der Benutzer als Eingabe angegeben hat. Ausführliche Informationen zu den bereitgestellten Informationen finden Sie in der MSDN-Dokumentation .

Prototyp der Eingabeaufforderung

```
Module MainPrompt
Public Const PromptSymbol As String = "TLA > "
Public Const ApplicationTitle As String = GetType(Project.BaseClass).Assembly.FullName
REM Or you can use a custom string
REM Public Const ApplicationTitle As String = "Short name of the application"

Sub Main()
    Dim Statement As String
    Dim BrokenDownStatement As String()
    Dim Command As String
    Dim Args As String()
    Dim Result As String

    Console.ForegroundColor = ConsoleColor.Cyan
    Console.Title = ApplicationTitle & " command line console"

    Console.WriteLine("Welcome to " & ApplicationTitle & "console frontend")
    Console.WriteLine("This package is version " &
GetType(Project.BaseClass).Assembly.GetName().Version.ToString)
    Console.WriteLine()
    Console.Write(PromptSymbol)

    Do While True
        Statement = Console.ReadLine()
        BrokenDownStatement = Statement.Split(" ")
        ReDim Args(BrokenDownStatement.Length - 1)
        Command = BrokenDownStatement(0)

        For i = 1 To BrokenDownStatement.Length - 1
            Args(i - 1) = BrokenDownStatement(i)
        Next
```

```
        Select Case Command.ToLower
            Case "example"
                Result = DoSomething(Example)
            Case "exit", "quit"
                Exit Do
            Case "ver"
                Result = "This package is version " &
GetType(Project.BaseClass).Assembly.GetName().Version.ToString
            Case Else
                Result = "Command not acknowldged: -" & Command & "-"
        End Select
        Console.WriteLine(" " & Result)
        Console.Write(PromptSymbol)
    Loop

    Console.WriteLine("I am exiting, time is " & DateTime.Now.ToString("u"))
    Console.WriteLine("Goodbye")
    Environment.Exit(0)
End Sub
End Module
```

Dieser Prototyp generiert einen grundlegenden Befehlszeileninterpreter.

Es erhält automatisch den Anwendungsnamen und die Version, um mit dem Benutzer zu kommunizieren. Er erkennt für jede Eingabezeile den Befehl und eine beliebige Liste von Argumenten, die alle durch Leerzeichen getrennt sind.

Als grundlegendes Beispiel versteht dieser Code die Befehle *Ver* , *Quit* und *Exit* .

Der Parameter *Project.BaseClass* ist eine Klasse Ihres Projekts, in der die Assembly-Details festgelegt werden.

Konsole online lesen: https://riptutorial.com/de/vb-net/topic/602/konsole

Kapitel 29: Kurzschlussoperatoren (AndAlso - OrElse)

Syntax

- result = expression1 AndAlso expression2
- result = expression1 oder Else expression2

Parameter

Parameter	Einzelheiten
Ergebnis	Erforderlich. Jeder boolesche Ausdruck. Das Ergebnis ist das boolesche Ergebnis des Vergleichs der beiden Ausdrücke.
expression1	Erforderlich. Jeder boolesche Ausdruck.
expression2	Erforderlich. Jeder boolesche Ausdruck.

Bemerkungen

'AndAlso' und **'OrElse'** sind **ShortCircuiting-** Operatoren. **Dies** bedeutet, dass die Ausführung kürzer ist, da der Compiler nicht alle Ausdrücke in einem booleschen Vergleich auswertet, wenn der erste das gewünschte Ergebnis liefert.

Examples

Und auch Verwendung

```
' Sometimes we don't need to evaluate all the conditions in an if statement's boolean check.

' Let's suppose we have a list of strings:

Dim MyCollection as List(Of String) = New List(of String)()

' We want to evaluate the first value inside our list:

If MyCollection.Count > 0 And MyCollection(0).Equals("Somevalue")
    Console.WriteLine("Yes, I've found Somevalue in the collection!")
End If

' If MyCollection is empty, an exception will be thrown at runtime.
' This because it evaluates both first and second condition of the
' if statement regardless of the outcome of the first condition.

' Now let's apply the AndAlso operator
```

```
If MyCollection.Count > 0 AndAlso MyCollection(0).Equals("Somevalue")
    Console.WriteLine("Yes, I've found Somevalue in the collection!")
End If

' This won't throw any exception because the compiler evaluates just the first condition.
' If the first condition returns False, the second expression isn't evaluated at all.
```

Oder Else Nutzung

```
' The OrElse operator is the homologous of AndAlso. It lets us perform a boolean
' comparison evaluating the second condition only if the first one is False

If testFunction(5) = True OrElse otherFunction(4) = True Then
    ' If testFunction(5) is True, otherFunction(4) is not called.
    ' Insert code to be executed.
End If
```

NullReferenceException vermeiden

7,0

Oder Else

```
Sub Main()
    Dim elements As List(Of Integer) = Nothing

    Dim average As Double = AverageElementsOrElse(elements)
    Console.WriteLine(average) ' Writes 0 to Console

    Try
        'Throws ArgumentNullException
        average = AverageElementsOr(elements)
    Catch ex As ArgumentNullException
        Console.WriteLine(ex.Message)
    End Try
End Sub

Public Function AverageElementsOrElse(ByVal elements As IEnumerable(Of Integer)) As Double
    ' elements.Count is not called if elements is Nothing so it cannot crash
    If (elements Is Nothing OrElse elements.Count = 0) Then
        Return 0
    Else
        Return elements.Average()
    End If
End Function

Public Function AverageElementsOr(ByVal elements As IEnumerable(Of Integer)) As Double
    ' elements.Count is always called so it can crash if elements is Nothing
    If (elements Is Nothing Or elements.Count = 0) Then
        Return 0
    Else
        Return elements.Average()
    End If
End Function
```

7,0

Und auch

```
Sub Main()
    Dim elements As List(Of Integer) = Nothing

    Dim average As Double = AverageElementsAndAlso(elements)
    Console.WriteLine(average) ' Writes 0 to Console

    Try
        'Throws ArgumentNullException
        average = AverageElementsAnd(elements)
    Catch ex As ArgumentNullException
        Console.WriteLine(ex.Message)
    End Try
End Sub

Public Function AverageElementsAndAlso(ByVal elements As IEnumerable(Of Integer)) As Double
    ' elements.Count is not called if elements is Nothing so it cannot crash
    If (Not elements Is Nothing AndAlso elements.Count > 0) Then
        Return elements.Average()
    Else
        Return 0
    End If
End Function

Public Function AverageElementsAnd(ByVal elements As IEnumerable(Of Integer)) As Double
    ' elements.Count is always called so it can crash if elements is Nothing
    If (Not elements Is Nothing And elements.Count > 0) Then
        Return elements.Average()
    Else
        Return 0
    End If
End Function
```

14,0

In Visual Basic 14.0 wurde der bedingungslose Operator " `AndAlso` eingeführt , mit dem die Funktionen sauberer geschrieben werden können, um das Verhalten der `AndAlso` Version des Beispiels nachzuahmen.

Kurzschlussoperatoren (AndAlso - OrElse) online lesen: https://riptutorial.com/de/vb-net/topic/2509/kurzschlussoperatoren--andalso---orelse-

Kapitel 30: LINQ

Einführung

LINQ (Language Integrated Query) ist ein Ausdruck, der Daten aus einer Datenquelle abruft. LINQ vereinfacht diese Situation, indem es ein konsistentes Modell für die Arbeit mit Daten über verschiedene Arten von Datenquellen und -formaten anbietet. In einer LINQ-Abfrage arbeiten Sie immer mit Objekten. Sie verwenden dieselben grundlegenden Codierungsmuster zum Abfragen und Umwandeln von Daten in XML-Dokumenten, SQL-Datenbanken, ADO.NET-Datensätzen, .NET-Sammlungen und allen anderen Formaten, für die ein LINQ-Anbieter verfügbar ist.

Examples

Projektion

```
' sample data
Dim sample = {1, 2, 3, 4, 5}

' using "query syntax"
Dim squares = From number In sample Select number * number

' same thing using "method syntax"
Dim squares = sample.Select (Function (number) number * number)
```

Wir können auch mehrere Ergebnisse gleichzeitig projizieren

```
Dim numbersAndSquares =
    From number In sample Select number, square = number * number

Dim numbersAndSquares =
    sample.Select (Function (number) New With {Key number, Key .square = number * number})
```

Auswahl aus Feld mit einfachen Bedingungen

```
Dim sites() As String = {"Stack Overflow", "Super User", "Ask Ubuntu", "Hardware
Recommendations"}
Dim query = From x In sites Where x.StartsWith("S")
' result = "Stack Overflow", "Super User"
```

Die Abfrage ist ein aufzählbares Objekt, das `Stack Overflow` und `Super User` . `x` in der Abfrage ist eine Iterationsvariable, in der jedes von der `Where` Klausel überprüfte Objekt gespeichert wird.

Zuordnungsfeld durch Auswahlklausel

```
Dim sites() As String = {"Stack Overflow",
                         "Super User",
                         "Ask Ubuntu",
```

```
                          "Hardware Recommendations"}
Dim query = From x In sites Select x.Length
' result = 14, 10, 10, 24
```

Das Abfrageergebnis ist ein aufzählbares Objekt, das die Länge der Zeichenfolgen im Eingabearray enthält. In diesem Beispiel wären dies die Werte 14, 10, 10, 24. x in der Abfrage ist eine Iterationsvariable, in der jedes Objekt aus dem Eingabearray gespeichert wird.

Ausgabe bestellen

```
Dim sites() As String = {"Stack Overflow",
                         "Super User",
                         "Ask Ubuntu",
                         "Hardware Recommendations"}

Dim query = From x In sites
            Order By x.Length

' result = "Super User", "Ask Ubuntu", "Stack Overflow", "Hardware Recommendations"
```

Die OrderBy-Klausel ordnet die Ausgabe nach dem von der Klausel zurückgegebenen Wert an. In diesem Beispiel ist dies die Länge jeder Zeichenfolge. Die Standardausgabereihenfolge ist aufsteigend. Wenn Sie absteigend benötigen, können Sie nach der Klausel das `Descending` Schlüsselwort angeben.

```
Dim query = From x In sites
            Order By x.Length Descending
```

Wörterbuch aus IEnumerable generieren

```
' Just setting up the example
Public Class A
    Public Property ID as integer
    Public Property Name as string
    Public Property OtherValue as Object
End Class

Public Sub Example()
    'Setup the list of items
    Dim originalList As New List(Of A)
    originalList.Add(New A() With {.ID = 1, .Name = "Item 1", .OtherValue = "Item 1 Value"})
    originalList.Add(New A() With {.ID = 2, .Name = "Item 2", .OtherValue = "Item 2 Value"})
    originalList.Add(New A() With {.ID = 3, .Name = "Item 3", .OtherValue = "Item 3 Value"})

    'Convert the list to a dictionary based on the ID
    Dim dict As Dictionary(Of Integer, A) = originalList.ToDictionary(function(c) c.ID,
function(c) c)

    'Access Values From The Dictionary
    console.Write(dict(1).Name) ' Prints "Item 1"
    console.Write(dict(1).OtherValue) ' Prints "Item 1 Value"
End Sub
```

Ermitteln verschiedener Werte (mithilfe der Distinct-Methode)

```
Dim duplicateFruits = New List(Of String) From {"Grape", "Apple", "Grape", "Apple", "Grape"}
'At this point, duplicateFruits.Length = 5

Dim uniqueFruits = duplicateFruits.Distinct();
'Now, uniqueFruits.Count() = 2
'If iterated over at this point, it will contain 1 each of "Grape" and "Apple"
```

LINQ online lesen: https://riptutorial.com/de/vb-net/topic/3111/linq

Kapitel 31: Listen

Syntax

- List.Add (Artikel als Typ)
- List.RemoveRange (Index als ganze Zahl, Anzahl als ganze Zahl)
- List.Remove (Index als Ganzzahl)
- List.AddRange (Sammlung)
- List.Find (Übereinstimmung als Prädikat (von String))
- List.Insert (Index als Ganzzahl, Element als Typ)
- List.Contains (Element als Typ)

Examples

Erstelle eine Liste

Listen können mit jedem beliebigen Datentyp und mit dem Format gefüllt werden

```
Dim aList as New List(Of Type)
```

Zum Beispiel:

Erstellen Sie eine neue, leere Liste von Zeichenfolgen

```
Dim aList As New List(Of String)
```

Erstellen Sie eine neue Liste von Zeichenfolgen und füllen Sie einige Daten auf

VB.NET 2005/2008:

```
Dim aList as New List(Of String)(New String() {"one", "two", "three"})
```

VB.NET 2010:

```
Dim aList as New List(Of String) From {"one", "two", "three"}
```

-

VB.NET 2015:

```
Dim aList as New List(Of String)(New String() {"one", "two", "three"})
```

HINWEIS:

Wenn Sie Folgendes erhalten, wenn der Code ausgeführt wird:

Der Objektverweis wurde nicht auf eine Instanz eines Objekts festgelegt.

Vergewissern Sie sich, dass Sie entweder `New` `Dim aList as New List(Of String)` deklarieren. Wenn Sie ohne `New` deklarieren, stellen Sie sicher, dass Sie die Liste auf eine neue Liste setzen - `Dim aList as List(Of String) = New List(Of String)`

Elemente zu einer Liste hinzufügen

```
Dim aList as New List(Of Integer)
aList.Add(1)
aList.Add(10)
aList.Add(1001)
```

Um mehrere **Elemente** gleichzeitig **hinzuzufügen,** verwenden Sie **AddRange** . Fügt immer am Ende der Liste hinzu

```
Dim blist as New List(of Integer)
blist.AddRange(alist)


Dim aList as New List(of String)
alist.AddRange({"one", "two", "three"})
```

Um Elemente zur Mitte der Liste hinzuzufügen, verwenden Sie **Einfügen**

Durch Einfügen wird das Element im Index platziert und die restlichen Elemente neu nummeriert

```
Dim aList as New List(Of String)
aList.Add("one")
aList.Add("three")
alist(0) = "one"
alist(1) = "three"
alist.Insert(1,"two")
```

Neue Ausgabe:

```
alist(0) = "one"
alist(1) = "two"
alist(2) = "three"
```

Elemente aus einer Liste entfernen

```
Dim aList As New List(Of String)
aList.Add("Hello")
aList.Add("Delete Me!")
aList.Add("World")

'Remove the item from the list at index 1
aList.RemoveAt(1)

'Remove a range of items from a list, starting at index 0, for a count of 1)
'This will remove index 0, and 1!
aList.RemoveRange(0, 1)
```

```
'Clear the entire list
alist.Clear()
```

Elemente aus einer Liste abrufen

```
Dim aList as New List(Of String)
aList.Add("Hello, World")
aList.Add("Test")

Dim output As String = aList(0)
```

`output` :

Hallo Welt

Wenn Sie den Index des Elements nicht kennen oder nur einen Teil der Zeichenfolge kennen, verwenden **Sie die** Methode **Find** oder **FindAll**

```
Dim aList as New List(Of String)
aList.Add("Hello, World")
aList.Add("Test")

Dim output As String = aList.Find(Function(x) x.StartWith("Hello"))
```

`output` :

Hallo Welt

Die **FindAll-** Methode gibt eine neue List (of String) zurück.

```
Dim aList as New List(Of String)
aList.Add("Hello, Test")
aList.Add("Hello, World")
aList.Add("Test")

Dim output As String = aList.FindAll(Function(x) x.Contains("Test"))
```

Ausgabe (0) = "Hallo, Test"

Ausgang (1) = "Test"

Schleife durch Elemente in der Liste

```
Dim aList as New List(Of String)
aList.Add("one")
aList.Add("two")
aList.Add("three")

For Each str As String in aList
    System.Console.WriteLine(str)
Next
```

Erzeugt die folgende Ausgabe:

```
one
two
three
```

Eine weitere Option wäre das Durchlaufen des Index jedes Elements:

```
Dim aList as New List(Of String)
aList.Add("one")
aList.Add("two")
aList.Add("three")

For i = 0 to aList.Count - 1 'We use "- 1" because a list uses 0 based indexing.
    System.Console.WriteLine(aList(i))
Next
```

Prüfen Sie, ob das Element in einer Liste vorhanden ist

```
    Sub Main()
        Dim People = New List(Of String)({"Bob Barker", "Ricky Bobby", "Jeff Bridges"})
        Console.WriteLine(People.Contains("Rick James"))
        Console.WriteLine(People.Contains("Ricky Bobby"))
        Console.WriteLine(People.Contains("Barker"))
        Console.Read
    End Sub
```

Erzeugt die folgende Ausgabe:

```
False
True
False
```

Listen online lesen: https://riptutorial.com/de/vb-net/topic/3636/listen

Kapitel 32: Looping

Examples

Fürs nächste

`For` ... `Next` Loop wird verwendet, um dieselbe Aktion endlos oft zu wiederholen. Die Anweisungen innerhalb der folgenden Schleife werden 11-mal ausgeführt. Beim ersten Mal wird `i` den Wert 0 haben, beim zweiten Mal wird es den Wert 1 haben, beim letzten Mal wird es den Wert 10 haben.

```
For i As Integer = 0 To 10
    'Execute the action
    Console.Writeline(i.ToString)
Next
```

Es kann ein ganzzahliger Ausdruck verwendet werden, um die Schleife zu parametrisieren. Es ist zulässig, aber nicht erforderlich, dass die Steuergröße (in diesem Fall `i`) auch nach dem `Next` . Es ist zulässig, dass die Steuervariable im Voraus und nicht innerhalb der `For` Anweisung deklariert wird.

```
Dim StartIndex As Integer = 3
Dim EndIndex As Integer = 7
Dim i As Integer

For i = StartIndex To EndIndex - 1
    'Execute the action
    Console.Writeline(i.ToString)
Next i
```

Durch die Möglichkeit, die Start- und End-Ganzzahl zu definieren, können Schleifen erstellt werden, die direkt auf andere Objekte verweisen, z.

```
For i = 0 to DataGridView1.Rows.Count - 1
    Console.Writeline(DataGridView1.Rows(i).Cells(0).Value.ToString)
Next
```

Dies würde dann jede Zeile in `DataGridView1` und die Aktion ausführen, indem der Wert von Spalte 1 in die Konsole geschrieben wird. *(Das -1 ist, weil die erste Zeile der gezählten Zeilen 1 wäre, nicht 0).*

Es kann auch definiert werden, wie die Steuervariable erhöht werden muss.

```
For i As Integer = 1 To 10 Step 2
    Console.Writeline(i.ToString)
Next
```

Dies gibt aus:

1 3 5 7 9

Es ist auch möglich, die Steuervariable zu dekrementieren (Countdown).

```
For i As Integer = 10 To 1 Step -1
    Console.Writeline(i.ToString)
Next
```

Dies gibt aus:

10 9 8 7 6 5 4 3 2 1

Sie sollten nicht versuchen, die Steuervariable außerhalb der Schleife zu verwenden (zu lesen oder zu aktualisieren).

For Each ... Nächste Schleife zum Durchlaufen der Sammlung von Elementen

Sie können eine `For Each...Next` Schleife verwenden, um einen beliebigen `IEnumerable` Typ zu `IEnumerable` . Dazu gehören Arrays, Listen und andere Elemente, die möglicherweise vom Typ IEnumerable sind oder ein IEnumerable zurückgeben.

Ein Beispiel für das Durchlaufen einer Rows-Eigenschaft einer DataTable würde folgendermaßen aussehen:

```
For Each row As DataRow In DataTable1.Rows
    'Each time this loops, row will be the next item out of Rows
    'Here we print the first column's value from the row variable.
    Debug.Print(Row.Item(0))
Next
```

Wichtig ist, dass die Auflistung nicht in einer `For Each` Schleife geändert werden darf. `System.InvalidOperationException` wird eine `System.InvalidOperationException` mit der Nachricht `System.InvalidOperationException` :

> Sammlung wurde geändert; Enumerationsoperation wird möglicherweise nicht ausgeführt.

Während der Schleife zum Durchlaufen, während eine Bedingung erfüllt ist

Eine `While` Schleife beginnt mit der Auswertung einer Bedingung. Ist dies der Fall, wird der Hauptteil der Schleife ausgeführt. Nachdem der Body der Schleife ausgeführt wurde, wird die `While` Bedingung erneut ausgewertet, um zu bestimmen, ob der Body erneut ausgeführt werden soll.

```
Dim iteration As Integer = 1
While iteration <= 10
  Console.Writeline(iteration.ToString() & " ")

  iteration += 1
End While
```

Dies gibt aus:

1 2 3 4 5 6 7 8 9 10

Warnung: Eine `While` Schleife kann zu einer *Endlosschleife führen* . Berücksichtigen Sie, was passieren würde, wenn die Codezeile, die die `iteration` erhöht, entfernt wird. In einem solchen Fall wäre die Bedingung niemals wahr und die Schleife würde unbegrenzt fortgesetzt.

Mach ... Schleife

Verwenden Sie `Do...Loop` , um einen Anweisungsblock zu wiederholen, `While` oder `Until` eine Bedingung erfüllt ist. Überprüfen Sie die Bedingung entweder am Anfang oder am Ende der Schleife.

```
Dim x As Integer = 0
Do
    Console.Write(x & " ")
    x += 1
Loop While x < 10
```

oder

```
Dim x As Integer = 0
Do While x < 10
    Console.Write(x & " ")
    x += 1
Loop
```

0 1 2 3 4 5 6 7 8 9

```
Dim x As Integer = 0
Do
    Console.Write(x & " ")
    x += 1
Loop Until x = 10
```

oder

```
Dim x As Integer = 0
Do Until x = 10
    Console.Write(x & " ")
    x += 1
Loop
```

0 1 2 3 4 5 6 7 8 9

`Continue Do` kann verwendet werden, um zur nächsten Iteration der Schleife zu springen:

```
Dim x As Integer = 0
Do While x < 10
    x += 1
    If x Mod 2 = 0 Then
```

```
        Continue Do
    End If
    Console.Write(x & " ")
Loop
```

1 3 5 7 9

Sie können die Schleife mit `Exit Do` beenden. Beachten Sie, dass das Fehlen einer Bedingung in diesem Beispiel eine Endlosschleife verursachen würde:

```
Dim x As Integer = 0
Do
    Console.Write(x & " ")
    x += 1
    If x = 10 Then
        Exit Do
    End If
Loop
```

0 1 2 3 4 5 6 7 8 9

Kurzschluss

Jede Schleife kann jederzeit mit den Anweisungen `Exit` oder `Continue` beendet oder vorzeitig `Continue` werden.

Beenden

Sie können jede Schleife stoppen, indem Sie sie vorzeitig beenden. Dazu können Sie das Schlüsselwort `Exit` zusammen mit dem Namen der Schleife verwenden.

Schleife	Exit Statement
Zum	`Exit For`
Für jeden	`Exit For`
Während tun	`Exit Do`
Während	`Exit While`

Das frühe Beenden einer Schleife ist eine großartige Möglichkeit, die Leistung zu steigern, indem nur die erforderliche Anzahl Schleifen ausgeführt wird, um die Anforderungen der Anwendung zu erfüllen. Unten ist ein Beispiel, wo die Schleife beendet wird, sobald sie die Nummer `2` .

```
Dim Numbers As Integer() = {1,2,3,4,5}
Dim SoughtValue As Integer = 2
Dim SoughtIndex
For Each i In Numbers
    If i = 2 Then
        SoughtIndex = i
```

```
        Exit For
    End If
Next
Debug.Print(SoughtIndex)
```

Auch weiterhin

Neben dem vorzeitigen Beenden können Sie auch entscheiden, dass Sie einfach mit der nächsten Wiederholungsschleife fortfahren müssen. Dies erfolgt einfach mit der `Continue` Anweisung. Genau wie `Exit` wird mit dem Namen der Schleife fortgefahren.

Schleife	Setzen Sie die Anweisung fort
Zum	`Continue For`
Für jeden	`Continue For`
Während tun	`Continue Do`
Während	`Continue While`

Hier ein Beispiel, um zu verhindern, dass gerade Zahlen zur Summe hinzugefügt werden.

```
Dim Numbers As Integer() = {1,2,3,4,5}
Dim SumOdd As Integer = 0
For Each i In Numbers
    If Numbers(i) \ 2 = 0 Then Continue For
    SumOdd += Numbers(i)
Next
```

Verwendungshinweise

Es gibt zwei alternative Techniken, die anstelle von `Exit` oder `Continue` .

Sie können eine neue boolesche Variable deklarieren, sie auf einen Wert initialisieren und bedingt auf den anderen Wert in der Schleife setzen. Sie verwenden dann eine bedingte Anweisung (z. B. `If`), die auf dieser Variablen basiert, um die Ausführung der Anweisungen in der Schleife in nachfolgenden Iterationen zu vermeiden.

```
Dim Found As Boolean = False
Dim FoundIndex As Integer
For i As Integer = 0 To N - 1
    If Not Found AndAlso A(i) = SoughtValue Then
        FoundIndex = i
        Found = True
    End If
Next
```

Einer der Einwände gegen diese Technik ist, dass sie möglicherweise ineffizient ist. Wenn zum Beispiel in dem obigen Beispiel `N` 1000000 ist und das erste Element des Arrays `A` gleich

`SoughtValue` , wird die Schleife weitere 999999 Mal wiederholt, ohne irgendetwas Nützliches zu tun. Diese Technik kann jedoch in einigen Fällen den Vorteil größerer Klarheit haben.

Sie können die `GoTo` Anweisung verwenden, um aus der Schleife zu springen. Beachten Sie, dass Sie `GoTo` nicht verwenden können, um *in* eine Schleife zu springen.

```
    Dim FoundIndex As Integer
    For i As Integer = 0 To N - 1
        If A(i) = SoughtValue Then
            FoundIndex = i
            GoTo Found
        End If
    Next
    Debug.Print("Not found")
Found:
    Debug.Print(FoundIndex)
```

Diese Technik kann manchmal die beste Methode sein, um aus der Schleife zu springen und eine oder mehrere Anweisungen zu vermeiden, die unmittelbar nach dem natürlichen Ende der Schleife ausgeführt werden.

Sie sollten alle Alternativen in Betracht ziehen und die für Ihre Anforderungen am besten geeignete Lösung verwenden, z. B. Effizienz, Schnelligkeit beim Schreiben des Codes und Lesbarkeit (also Wartbarkeit).

Lassen Sie sich mit `GoTo` nicht `GoTo` wenn dies die beste Alternative ist.

Verschachtelte Schleife

```
A nested loop is a loop within a loop, an inner loop within the body of an outer one. How this
works is that the first pass of the outer loop triggers the inner loop, which executes to
completion. Then the second pass of the outer loop triggers the inner loop again. This repeats
until the outer loop finishes. a break within either the inner or outer loop would interrupt
this process.
```

Die Struktur einer for next geschachtelten Schleife lautet:

```
For counter1=startNumber to endNumber (Step increment)

    For counter2=startNumber to endNumber (Step increment)

        One or more VB statements

    Next counter2

Next  counter1
```

Beispiel:

```
    For  firstCounter = 1 to 5

      Print “First Loop of ” + firstCounter
```

```
    For   secondCounter= 1 to 4

      Print “Second Loop of ” + secondCounter

    Next secondCounter

  Next firstCounter
```

Looping online lesen: https://riptutorial.com/de/vb-net/topic/1639/looping

Kapitel 33: Mit Windows Forms arbeiten

Examples

Verwenden der Standardformularinstanz

VB.NET bietet Standardformularinstanzen. Der Entwickler muss die Instanz nicht erstellen, da sie im Hintergrund erstellt wird. *Es ist jedoch **nicht** vorzuziehen* , die Standardinstanz außer den einfachsten Programmen zu verwenden.

```
Public Class Form1

    Public Sub Foo()
        MessageBox.Show("Bar")
    End Sub

End Class

Module Module1

    Public Sub Main()
        ' Default instance
        Form1.Foo()
        ' New instance
        Dim myForm1 As Form1 = New Form1()
        myForm1.Foo()

    End Sub

End Module
```

Siehe auch:

- Müssen Sie explizit eine Instanz eines Formulars in VB.NET erstellen?
- Warum gibt es eine Standardinstanz jedes Formulars in VB.Net, aber nicht in C #?

Daten von einem Formular zum anderen übergeben

Manchmal möchten Sie Informationen, die in einem Formular generiert wurden, an ein anderes Formular zur weiteren Verwendung übergeben. Dies ist nützlich für Formulare, die ein Suchwerkzeug oder eine Einstellungsseite für viele andere Zwecke anzeigen.

Angenommen , Sie möchten eine `DataTable` zwischen einem bereits geöffneten *Formular (MainForm)* und einem neuen Formular *(NewForm) übergeben* :

In der MainForm:

```
  Private Sub Open_New_Form()
       Dim NewInstanceOfForm As New NewForm(DataTable1)
       NewInstanceOfForm.ShowDialog()
    End Sub
```

In der NewForm

```
Public Class NewForm
    Dim NewDataTable as Datatable

    Public Sub New(PassedDataTable As Datatable)
       InitializeComponent()
       NewDataTable= PassedDataTable
    End Sub

End Class
```

Wenn nun die *NewForm* geöffnet wird, wird `DataTable1` von *MainForm übergeben* und als `NewDataTable` in *NewForm* zur Verwendung durch dieses Formular gespeichert.

Dies kann äußerst nützlich sein, wenn Sie versuchen, große Informationsmengen zwischen Formularen zu übergeben, insbesondere wenn Sie alle Informationen in einer einzigen `ArrayList` und die `ArrayList` an das neue Formular übergeben.

Mit Windows Forms arbeiten online lesen: https://riptutorial.com/de/vb-net/topic/4636/mit-windows-forms-arbeiten

Kapitel 34: Multithreading

Examples

Multithreading mit Thread-Klasse

In diesem Beispiel wird die `Thread` Klasse verwendet. Multithread-Anwendungen können jedoch auch mit `BackgroundWorker` . Die Funktionen `AddNumber` , `SubstractNumber` und `DivideNumber` werden von separaten Threads ausgeführt:

Bearbeiten: Jetzt wartet der UI-Thread auf den Abschluss der untergeordneten Threads und zeigt das Ergebnis.

```
Module Module1
    'Declare the Thread and assign a sub to that
    Dim AddThread As New Threading.Thread(AddressOf AddNumber)
    Dim SubstractThread As New Threading.Thread(AddressOf SubstractNumber)
    Dim DivideThread As New Threading.Thread(AddressOf DivideNumber)

    'Declare the variable for holding the result
    Dim addResult As Integer
    Dim SubStractResult As Integer
    Dim DivisionResult As Double

    Dim bFinishAddition As Boolean = False
    Dim bFinishSubstration As Boolean = False
    Dim bFinishDivision As Boolean = False

    Dim bShownAdditionResult As Boolean = False
    Dim bShownDivisionResult As Boolean = False
    Dim bShownSubstractionResult As Boolean = False

    Sub Main()

        'Now start the trheads
        AddThread.Start()
        SubstractThread.Start()
        DivideThread.Start()

        'Wait and display the results in console
        Console.WriteLine("Waiting for threads to finish...")
        Console.WriteLine("")

        While bFinishAddition = False Or bFinishDivision = False Or bFinishSubstration = False
            Threading.Thread.Sleep(50)     'UI thread is sleeping
            If bFinishAddition And Not bShownAdditionResult Then
                Console.WriteLine("Addition Result : " & addResult)
                bShownAdditionResult = True
            End If

            If bFinishSubstration And Not bShownSubstractionResult Then
                Console.WriteLine("Substraction Result : " & SubStractResult)
                bShownSubstractionResult = True
            End If
```

```
            If bFinishDivision  And Not bShownDivisionResult Then
                Console.WriteLine("Division Result : " & DivisionResult)
                bShownDivisionResult = True
            End If

        End While

        Console.WriteLine("")
        Console.WriteLine("Finished all threads.")
        Console.ReadKey()
    End Sub


    Private Sub AddNumber()
        Dim n1 As Integer = 22
        Dim n2 As Integer = 11

        For i As Integer = 0 To 100
            addResult = addResult + (n1 + n2)
            Threading.Thread.Sleep(50)      'sleeping Add thread
        Next
        bFinishAddition = True
    End Sub

    Private Sub SubstractNumber()
        Dim n1 As Integer = 22
        Dim n2 As Integer = 11

        For i As Integer = 0 To 80
            SubStractResult = SubStractResult - (n1 - n2)
            Threading.Thread.Sleep(50)
        Next
        bFinishSubstration = True
    End Sub

    Private Sub DivideNumber()
        Dim n1 As Integer = 22
        Dim n2 As Integer = 11
        For i As Integer = 0 To 60
            DivisionResult = DivisionResult + (n1 / n2)
            Threading.Thread.Sleep(50)
        Next
        bFinishDivision = True
    End Sub

End Module
```

Multithreading online lesen: https://riptutorial.com/de/vb-net/topic/6756/multithreading

Kapitel 35: NullReferenceException

Bemerkungen

NullReferenceException wird ausgelöst, wenn eine Variable leer ist und auf eine ihrer Methoden / Eigenschaften verwiesen wird. Um dies zu vermeiden, stellen Sie sicher, dass alle Variablen korrekt initialisiert sind (`new` Operator) und alle Methoden einen Wert ungleich Null zurückgeben.

Examples

Nicht initialisierte Variable

BAD CODE

```
Dim f As System.Windows.Forms.Form
f.ShowModal()
```

GUTER CODE

```
Dim f As System.Windows.Forms.Form = New System.Windows.Forms.Form
' Dim f As New System.Windows.Forms.Form ' alternative syntax
f.ShowModal()
```

NOCH BESSERER CODE (**Sicheres** Entsorgen des IDisposable-Objekts weitere Informationen)

```
Using f As System.Windows.Forms.Form = New System.Windows.Forms.Form
' Using f As New System.Windows.Forms.Form ' alternative syntax
    f.ShowModal()
End Using
```

Leere Rückkehr

```
Function TestFunction() As TestClass
    Return Nothing
End Function
```

BAD CODE

```
TestFunction().TestMethod()
```

GUTER CODE

```
Dim x = TestFunction()
If x IsNot Nothing Then x.TestMethod()
```

14,0

Null bedingter Operator

```
TestFunction()?.TestMethod()
```

NullReferenceException online lesen: https://riptutorial.com/de/vb-net/topic/4076/nullreferenceexception

Kapitel 36: OOP-Schlüsselwörter

Examples

Klasse definieren

Klassen sind wesentliche Aspekte von OOP. Eine Klasse ist wie der "Bauplan" eines Objekts. Ein Objekt hat die Eigenschaften einer Klasse, aber die Merkmale werden nicht in der Klasse selbst definiert. Da jedes Objekt unterschiedlich sein kann, definieren sie ihre eigenen Merkmale.

```
Public Class Person
End Class

Public Class Customer
End Class
```

Eine Klasse kann auch *Unterklassen* enthalten. Eine Unterklasse erbt die gleichen Eigenschaften und Verhaltensweisen wie ihre übergeordnete Klasse, kann jedoch eigene eindeutige Eigenschaften und Klassen haben.

Vererbungsmodifikatoren (für Klassen)

Erbt

Gibt die Basisklasse (oder übergeordnete Klasse) an

```
Public Class Person
End Class

Public Class Customer
    Inherits Person

End Class

'One line notation
Public Class Student : Inherits Person
End Class
```

Mögliche Objekte:

```
Dim p As New Person
Dim c As New Customer
Dim s As New Student
```

Nicht vererbbar

Verhindert, dass Programmierer die Klasse als Basisklasse verwenden.

```
Public NotInheritable Class Person
End Class
```

Mögliche Objekte:

```
Dim p As New Person
```

MustInherit

Gibt an, dass die Klasse nur als Basisklasse verwendet werden soll. (Abstrakte Klasse)

```
Public MustInherit Class Person
End Class

Public Class Customer
    Inherits Person
End Class
```

Mögliche Objekte:

```
Dim c As New Customer
```

Vererbungsmodifikatoren (zu Eigenschaften und Methoden)

Überschreibbar

Ermöglicht das Überschreiben einer Eigenschaft oder Methode in einer Klasse in einer abgeleiteten Klasse.

```
Public Class Person
    Public Overridable Sub DoSomething()
        Console.WriteLine("Person")
    End Sub
End Class
```

Überschreibungen

Überschreibt eine überschreibbare Eigenschaft oder Methode, die in der Basisklasse definiert ist.

```
Public Class Customer
    Inherits Person

    'Base Class must be Overridable
    Public Overrides Sub DoSomething()
```

```
        Console.WriteLine("Customer")
    End Sub
End Class
```

Nichtüberwindbar

Verhindert, dass eine Eigenschaft oder Methode in einer erbenden Klasse überschrieben wird. Standardverhalten Kann nur bei **Überschreibungsmethoden** deklariert **werden**

```
Public Class Person

    Public Overridable Sub DoSomething()
        Console.WriteLine("Person")
    End Sub

End Class

Public Class Customer
    Inherits Person

    Public NotOverridable Overrides Sub DoSomething()
        Console.WriteLine("Customer")
    End Sub

End Class

Public Class DetailedCustomer
    Inherits Customer

    'DoSomething can't be overridden
End Class
```

Verwendungsbeispiel:

```
Dim p As New Person
p.DoSomething()

Dim c As New Customer
c.DoSomething()

Dim d As New DetailedCustomer
d.DoSomething()
```

Ausgabe:

```
Person
Customer
Customer
```

MustOverride

Erfordert, dass eine abgeleitete Klasse die Eigenschaft oder Methode überschreibt.

MustOverride-Methoden müssen in **MustInherit-Klassen deklariert werden.**

```
Public MustInherit Class Person

    Public MustOverride Sub DoSomething()
    'No method definition here

End Class

Public Class Customer
    Inherits Person

    'DoSomething must be overridden
    Public Overrides Sub DoSomething()
        Console.WriteLine("Customer")
    End Sub

End Class
```

Verwendungsbeispiel:

```
Dim c As New Customer
c.DoSomething()
```

Ausgabe:

```
Customer
```

MyBase

Das MyBase-Schlüsselwort verhält sich wie eine Objektvariable, die auf die Basisklasse der aktuellen Instanz einer Klasse verweist.

```
Public Class Person
    Public Sub DoSomething()
        Console.WriteLine("Person")
    End Sub
End Class

Public Class Customer
    Inherits Person

    Public Sub DoSomethingElse()
        MyBase.DoSomething()
    End Sub

End Class
```

Anwendungsbeispiel:

```
Dim p As New Person
p.DoSomething()
```

```
Console.WriteLine("----")

Dim c As New Customer
c.DoSomething()
c.DoSomethingElse()
```

Ausgabe:

```
Person
----
Person
Person
```

Ich gegen MyClass

Ich verwendet die aktuelle Objektinstanz.

MyClass verwendet die Memberdefinition in der Klasse, in der der Member aufgerufen wird

```
Class Person
    Public Overridable Sub DoSomething()
        Console.WriteLine("Person")
    End Sub

    Public Sub useMe()
        Me.DoSomething()
    End Sub

    Public Sub useMyClass()
        MyClass.DoSomething()
    End Sub
End Class

Class Customer
    Inherits Person

    Public Overrides Sub DoSomething()
        Console.WriteLine("Customer")
    End Sub
End Class
```

Verwendungsbeispiel:

```
Dim c As New Customer
c.useMe()
c.useMyClass()
```

Ausgabe:

```
Customer
Person
```

Überlastung

Überladen ist das Erstellen von mehr als einer Prozedur, einem Instanzkonstruktor oder einer Eigenschaft in einer Klasse mit demselben Namen, aber unterschiedlichen Argumenttypen.

```
Class Person
    Overloads Sub Display(ByVal theChar As Char)
        ' Add code that displays Char data.
    End Sub

    Overloads Sub Display(ByVal theInteger As Integer)
        ' Add code that displays Integer data.
    End Sub

    Overloads Sub Display(ByVal theDouble As Double)
        ' Add code that displays Double data.
    End Sub
End Class
```

Schatten

Es deklariert ein Mitglied, das nicht überschrieben werden kann. Nur Aufrufe an die Instanz sind betroffen. Code innerhalb der Basisklassen wird davon nicht betroffen.

```
Public Class Person
    Public Sub DoSomething()
        Console.WriteLine("Person")
    End Sub


    Public Sub UseMe()
        Me.DoSomething()
    End Sub
End Class
Public Class Customer
    Inherits Person
    Public Shadows Sub DoSomething()
        Console.WriteLine("Customer")
    End Sub

End Class
```

Verwendungsbeispiel:

```
Dim p As New Person
Dim c As New Customer
p.UseMe()
c.UseMe()
Console.WriteLine("----")
p.DoSomething()
c.DoSomething()
```

Ausgabe:

```
Person
Person
----
```

```
Person
Customer
```

Fallstricke :

Beispiel1, Erstellen eines neuen Objekts durch ein generisches Objekt. Welche Funktion wird verwendet?

```
Public Sub CreateAndDoSomething(Of T As {Person, New})()
    Dim obj As New T
    obj.DoSomething()
End Sub
```

Verwendungsbeispiel:

```
Dim p As New Person
p.DoSomething()
Dim s As New Student
s.DoSomething()
Console.WriteLine("----")
CreateAndDoSomething(Of Person)()
CreateAndDoSomething(Of Student)()
```

Ausgabe: Nach Intuition sollte das Ergebnis das gleiche sein. Das stimmt aber nicht.

```
Person
Student
----
Person
Person
```

Beispiel 2

```
Dim p As Person
Dim s As New Student
p = s
p.DoSomething()
s.DoSomething()
```

Ausgabe: Durch Intuition könnten Sie denken, dass p und s gleich sind und sich gleich benehmen. Das stimmt aber nicht.

```
Person
Student
```

In diesen einfachen Beispielen ist es leicht, das merkwürdige Verhalten von Schatten zu lernen. Im wirklichen Leben bringt es jedoch viele Überraschungen. Es ist ratsam, die Verwendung von Schatten zu verhindern. Man sollte so viele Alternativen wie möglich verwenden (Überschreibungen usw.).

Schnittstellen

```
Public Interface IPerson
    Sub DoSomething()
End Interface

Public Class Customer
    Implements IPerson
    Public Sub DoSomething() Implements IPerson.DoSomething
        Console.WriteLine("Customer")
    End Sub

End Class
```

OOP-Schlüsselwörter online lesen: https://riptutorial.com/de/vb-net/topic/4273/oop-schlusselworter

Kapitel 37: Operatoren

Bemerkungen

Operatoren dienen zum Zuweisen oder Vergleichen von Werten. Sie bestehen aus einem einzelnen Symbol oder Schlüsselwort und sind normalerweise zwischen einem linken und einem rechten Wert angeordnet. Zum Beispiel: `right = left` .

Operatoren sind sprachintensiv (wie `=`) und keine Funktionen wie die von System.Math bereitgestellten.

Examples

Vergleich

Vergleichsoperatoren vergleichen zwei Werte und geben als Ergebnis einen Booleschen Wert (`True` oder `False`) zurück.

Gleichberechtigung

- Das Gleichheitszeichen `=` wird sowohl für den Vergleich der Gleichheit als auch für die Zuweisung verwendet.

  ```
  If leftValue = rightValue Then ...
  ```

Ungleichheit

- Die linke eckige Klammer verschachtelt mit der rechten eckigen Klammer `<>` führt einen ungleichen Vergleich durch.

  ```
  If leftValue <> rightValue Then ...
  ```

Größer als

- Die linke spitze Klammer `<` führt einen größeren Vergleich durch.

  ```
  If leftValue < rightValue Then ...
  ```

Größer als oder gleich

- Das Gleichheitszeichen verschachtelt mit der linken spitzen Klammer `=>` führt einen Vergleich mit größer oder gleich aus.

  ```
  If leftValue =< rightValue Then ...
  ```

Weniger als

- Die rechtwinklige Klammer `>` führt einen Vergleich aus.

  ```
  If leftValue > rightValue Then ...
  ```

Weniger als oder gleich

- Das Gleichheitszeichen für die rechtwinklige Klammer `=>` führt einen Vergleich mit größer oder gleich aus.
  `If leftValue => rightValue Then ...`

Mögen

- Der `Like`- Operator prüft die Gleichheit einer Zeichenfolge und eines Suchmusters.
- Der `Like`- Operator basiert auf der Option Compare-Anweisung
- In der folgenden Tabelle sind die verfügbaren Muster aufgeführt. Quelle: https://msdn.microsoft.com/en-us/library/swf8kaxw.aspx (Abschnitt "Bemerkungen")

Zeichen im *Muster*	Übereinstimmungen im *String*
?	Ein beliebiges einzelnes Zeichen
*	Null oder mehr Zeichen
#	Beliebige einstellige Zahl (0 - 9)
[charlist]	Ein beliebiges einzelnes Zeichen in der *Charlist*
[! charlist]	Ein beliebiges einzelnes Zeichen, das nicht in *Charlist steht*

- Weitere Informationen zu MSDN finden Sie in den Anmerkungen.
  `If string Like pattern Then ...`

Zuordnung

Es gibt einen einzigen Zuweisungsoperator in VB.

- Das Gleichheitszeichen `=` wird sowohl für den Vergleich der Gleichheit als auch für die Zuweisung verwendet.
  `Dim value = 5`

Anmerkungen
Achten Sie auf den Vergleich von Zuordnung und Gleichheit.

```
Dim result = leftValue = rightValue
```

In diesem Beispiel sehen Sie, dass das Gleichheitszeichen im Gegensatz zu anderen Sprachen sowohl als Vergleichsoperator als auch als Zuweisungsoperator verwendet wird. In diesem Fall ist das `result` vom Typ `Boolean` und enthält den Wert des Gleichheitsvergleichs zwischen `leftValue` und `rightValue` .

Verwandte: Verwenden von Option Strict On, um Variablen richtig zu deklarieren

Mathematik

Wenn Sie die folgenden Variablen haben

```
Dim leftValue As Integer = 5
Dim rightValue As Integer = 2
Dim value As Integer = 0
```

Zusatz Wird durch das Pluszeichen `+ ausgeführt` .

```
value  = leftValue + rightValue

'Output the following:
'7
```

Subtraktion Wird durch das Minuszeichen `- ausgeführt` .

```
value = leftValue - rightValue

'Output the following:
'3
```

Multiplikation Wird durch das Sternsymbol `* ausgeführt` .

```
value = leftValue * rightValue

'Output the following:
'10
```

Division Wird durch das Schrägstrichsymbol `/ ausgeführt` .

```
value = leftValue / rightValue

'Output the following:
'2.5
```

Integer Division Wird durch das Backslash-Symbol `\ ausgeführt` .

```
value = leftValue \ rightValue

'Output the following:
'2
```

Modul, das vom Schlüsselwort `Mod ausgeführt wird` .

```
value = leftValue Mod rightValue

'Output the following:
'1
```

Erhöhen Sie die Potenz des `^` -Zeichen.

```
value = leftValue ^ rightValue

'Output the following:
'25
```

Verbreiterung und Verengung

Muss bearbeitet werden.

Überladung des Bedieners

Muss bearbeitet werden.

Bitweise

Dies sind die bitweisen Operatoren in VB.NET: Und, oder, Xor, Not

Beispiel für und bitweise Operation

```
Dim a as Integer
a = 3 And 5
```

Der Wert von a ist 1. Das Ergebnis wird nach einem binären Vergleich von 3 und 5 erhalten. 3 in binärer Form ist 011 und 5 in binärer Form ist 101. Der Operator And setzt 1, wenn beide Bits 1. Wenn eines der Bits 0 ist, ist der Wert 0

```
3 And 5 will be  011
                 101
                 ---
                 001
```

Das binäre Ergebnis ist also 001 und wenn das Ergebnis in Dezimalzahl umgewandelt wird, lautet die Antwort 1.

Oder Operatorplätze 1, wenn beide oder ein Bit 1 ist

```
   3 Or 5 will be 011
                  101
                  ---
                  111
```

Xor-Operator setzt 1, wenn nur eines der Bits 1 ist (nicht beide)

```
   3 Xor 5 will be  011
                    101
                    ---
                    110
```

Nicht-Operator setzt die Bits einschließlich Vorzeichen zurück

```
Not 5 will be - 010
```

String-Verkettung

Stringverkettung ist, wenn Sie zwei oder mehr Strings in einer einzigen Stringvariablen kombinieren.

String-Verkettung wird mit dem Symbol **`&` durchgeführt** .

```
Dim one As String = "Hello "
Dim two As String = "there"
Dim result As String = one & two
```

Nicht-String-Werte werden bei Verwendung von **`&`** in String konvertiert.

```
Dim result as String = "2" & 10 ' result  = "210"
```

*Verwenden Sie immer **`&`** (kaufmännisches Und), um eine String-Verkettung durchzuführen.*

TUN SIE NICHT
Obwohl es in den *einfachsten* Fällen möglich ist, das Zeichen `+` für die Verkettung von Zeichenfolgen zu verwenden, sollten Sie dies niemals tun. Wenn eine Seite des Plus-Symbols keine Zeichenfolge ist und Option strikt deaktiviert ist, wird das Verhalten nicht intuitiv. Wenn strenge Option aktiviert ist, wird ein Compiler-Fehler ausgegeben. Erwägen:

```
Dim value = "2" + 10    ' result = 12  (data type Double)
Dim value = "2" + "10"  ' result = "210"  (data type String)
Dim value = "2g" + 10   ' runtime error
```

Das Problem hierbei ist, dass, wenn der Operator `+` einen Operanden sieht, der ein numerischer Typ ist, er davon ausgeht, dass der Programmierer eine arithmetische Operation ausführen wollte und versucht, den anderen Operanden in den entsprechenden numerischen Typ umzuwandeln. Wenn der andere Operand eine Zeichenfolge ist, die eine Nummer enthält (z. B. "10"), wird die Zeichenfolge *in eine Zahl konvertiert* und dann dem anderen Operanden *arithmetisch* hinzugefügt. Wenn der andere Operand nicht in eine Zahl konvertiert werden kann (z. B. "2g"), stürzt die Operation aufgrund eines Datenkonvertierungsfehlers ab. Der Operator `+` führt nur dann eine String-Verkettung durch, wenn *beide* Operanden vom Typ `String` .

Der Operator `&` ist jedoch für die Verkettung von Strings ausgelegt und wandelt Nicht-String-Typen in Strings um.

Operatoren online lesen: https://riptutorial.com/de/vb-net/topic/3257/operatoren

Kapitel 38: Option explizit

Bemerkungen

`Option Explicit On` ist eine empfohlene bewährte Methode für Visual Basic .Net. Es hilft Ihnen als Entwickler, saubereren, stabileren, fehlerfreieren und wartungsfreundlicheren Code zu erstellen. In einigen Fällen kann es auch helfen, Programme mit besserer Leistung zu schreiben!

mit ref an https://support.microsoft.com/de-in/kb/311329#bookmark-3 Die Option strict kann auch anstelle der Option explizit verwendet werden. Die Option strict erbt die Option explizit.

Examples

Was ist es?

Sie zwingt Sie dazu, alle Variablen explizit zu deklarieren.

Was ist der Unterschied zwischen explizit und implizit deklarieren einer Variablen?

Eine Variable explizit deklarieren:

```
Dim anInteger As Integer = 1234
```

Eine Variable implizit deklarieren:

```
'Did not declare aNumber using Dim
aNumber = 1234
```

Fazit

Daher sollten Sie immer `Option Explicit On` da Sie während der Zuweisung eine Variable falsch schreiben können, wodurch das Programm unerwartet reagiert.

Wie schalte ich es ein?

Dokumentebene

Sie ist standardmäßig aktiviert, Sie können jedoch eine zusätzliche Schutzschicht verwenden, indem Sie `Option Explicit On` oben in der Codedatei platzieren. Die Option gilt für das gesamte Dokument.

Projektebene

Sie können es über das Menü in Visual Studio einschalten:

> Projekt> [Projekt] Eigenschaften> Registerkarte Kompilieren> Option Explicit

Wählen `On` im Dropdown-Menü die Option Ein. Die Option gilt für das gesamte Dokument.

Alle neuen Projekte

Sie können es standardmäßig für alle neuen Projekte aktivieren, indem Sie Folgendes auswählen:

> Extras> Optionen> Projekte und Lösungen> VB-Standardeinstellungen> Option Explicit

Wählen `On` im Dropdown-Menü die Option Ein.

Option explizit online lesen: https://riptutorial.com/de/vb-net/topic/4725/option-explizit

Kapitel 39: Option Infer

Examples

Was ist es?

Ermöglicht die Verwendung lokaler Typinferenz bei der Deklaration von Variablen.

Was ist Typ Inferenz?

Sie können lokale Variablen deklarieren, ohne explizit einen Datentyp anzugeben. Der Compiler entnimmt den Datentyp einer Variablen aus dem Typ ihres Initialisierungsausdrucks.

Option Infer am :

```
Dim aString  = "1234" '--> Will be treated as String by the compiler
Dim aNumber  = 4711   '--> Will be treated as Integer by the compiler
```

vs. explizite Typdeklaration:

```
'State a type explicitly
Dim aString as String = "1234"
Dim aNumber as Integer = 4711
```

Option Infer Off:
Das Verhalten des Compilers bei `Option Infer Off` hängt von der `Option Strict` Einstellung ab, die hier bereits dokumentiert ist .

- **Option Infer Off - Option Strict Off**
  Alle Variablen ohne explizite Typdeklarationen werden als `Object` deklariert.

  ```
  Dim aString  = "1234" '--> Will be treated as Object by the compiler
  ```

- **Option Infer Off - Option Strict On**
  Der Compiler lässt Sie keine Variablen ohne expliziten Typ deklarieren.

  ```
  'Dim aString  = "1234" '--> Will not compile due to missing type in declaration
  ```

So aktivieren / deaktivieren Sie es

Dokumentebene

Sie ist standardmäßig aktiviert, aber Sie können sie festlegen, indem Sie `Option Infer On|Off` am Anfang der `Option Infer On|Off` . Die Option gilt für das gesamte Dokument.

Projektebene

Sie können es über das Menü in Visual Studio ein- und ausschalten:

> Projekt> [Projekt] Eigenschaften> Registerkarte Kompilieren> Option

Wählen `On|Off` im Dropdown-Menü `On|Off` . Die Option gilt für das gesamte Dokument.

Alle neuen Projekte

Sie können es standardmäßig für alle neuen Projekte aktivieren, indem Sie Folgendes auswählen:

> Extras> Optionen> Projekte und Lösungen> VB-Standardeinstellungen> Option Infer

Wählen `On|Off` im Dropdown-Menü `On|Off` .

Wann Typ-Inferenz verwenden soll

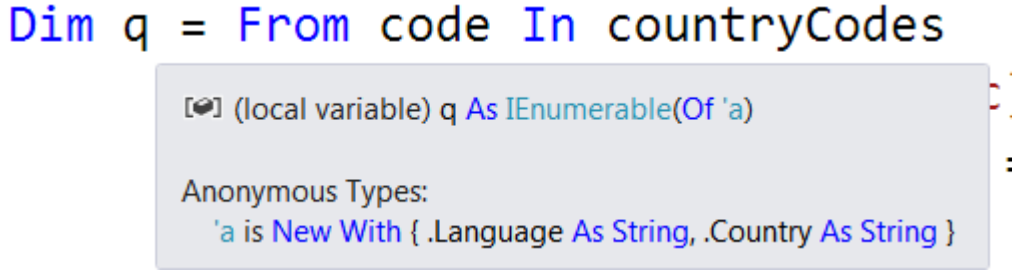
Grundsätzlich können Sie Typinferenz verwenden, wann immer dies möglich ist.
`Option Infer Off` Sie jedoch vorsichtig, wenn Sie die `Option Infer Off` und `Option Strict Off` kombinieren, da dies zu unerwünschtem Laufzeitverhalten führen kann:

```
Dim someVar = 5
someVar.GetType.ToString() '--> System.Int32
someVar = "abc"
someVar.GetType.ToString() '--> System.String
```

Anonymer Typ
Anonyme Typen können **nur** mit `Option Infer On` .
Sie werden häufig beim Umgang mit LINQ verwendet :

```
Dim countryCodes = New List(Of String)
countryCodes.Add("en-US")
countryCodes.Add("en-GB")
countryCodes.Add("de-DE")
countryCodes.Add("de-AT")

Dim q = From code In countryCodes
        Let split = code.Split("-"c)
        Select New With {.Language = split(0), .Country = split(1)}
```

- **Option Infer ein**
  Der Compiler erkennt den anonymen Typ:



- **Option Infer aus**
  Der Compiler gibt entweder einen Fehler aus (mit `Option Strict On`)

oder wird `q` als `object` (mit `Option Strict Off`).
In beiden Fällen wird das Ergebnis erzielt, dass Sie den anonymen Typ nicht verwenden können.

Doppelte / Dezimalzahlen
Numerische Variablen mit Dezimalstellen werden standardmäßig als `Double` :

```
Dim aNumber = 44.11 '--> Will be treated as type `Double` by the compiler
```

Wenn ein anderer Typ wie `Decimal` gewünscht wird, muss der Wert, mit dem die Variable initialisiert wurde, markiert werden:

```
Dim mDecimal = 47.11D '--> Will be treated as type `Decimal` by the compiler
```

Option Infer online lesen: https://riptutorial.com/de/vb-net/topic/5095/option-infer

Kapitel 40: Option Strict

Syntax

- Option Strict {Ein | Aus }

Bemerkungen

`Option Strict On` ist eine empfohlene bewährte Methode für Visual Basic .Net. Es hilft Ihnen als Entwickler, saubereren, stabileren, fehlerfreieren und wartungsfreundlicheren Code zu erstellen. In einigen Fällen kann dies auch dazu beitragen, Programme mit besserer Leistung zu schreiben, um beispielsweise Implizite Konvertierung zu vermeiden.

`On` ist *nicht* die Standardeinstellung für eine Neuinstallation von Visual Studio. Es sollte eines der ersten Dinge sein, bevor Sie mit der Programmierung beginnen, wenn Sie VB.NET verwenden. Der Grund dafür ist nicht die Standardeinstellung, weil die ersten Editionen von Visual Studio von Programmierern erwartet wurden, dass sie Projekte von VB6 migrieren.

Examples

Warum es benutzen?

`Option Strict On` verhindert, dass drei Dinge passieren:

1. Implizite eingrenzende Konvertierungsfehler

Es verhindert, dass Sie einer Variablen mit einer *geringeren Genauigkeit oder einer geringeren Kapazität* (eine engere Konvertierung) ohne explizite Umwandlung zuweisen. Dies würde zu Datenverlust führen.

```
Dim d As Double = 123.4
Dim s As Single = d 'This line does not compile with Option Strict On
```

2. Späte verbindliche Anrufe

Eine späte Bindung ist nicht zulässig. Dies verhindert, dass Tippfehler kompiliert werden, die jedoch zur Laufzeit fehlschlagen

```
Dim obj As New Object
obj.Foo 'This line does not compile with Option Strict On
```

3. Implizite Objekttypfehler

Dies verhindert, dass eine Variable als Objekt abgeleitet wird, obwohl sie eigentlich als Typ deklariert werden sollte

```
Dim something = Nothing. 'This line does not compile with Option Strict On
```

Fazit

Wenn Sie keine späte Bindung durchführen müssen, sollten Sie immer über `Option Strict On` verfügen, da die genannten Fehler zu Fehlern bei der Kompilierung anstatt zu Laufzeitausnahmen führen.

Wenn Sie die späte Bindung zu tun *haben,* können Sie *entweder*

- Packen Sie alle Ihre späten Bindungsaufrufe in eine Klasse / ein Modul und verwenden Sie `Option Strict Off` oben in der Codedatei (dies ist die bevorzugte Methode, da dies die Wahrscheinlichkeit von Tippfehlern in anderen Dateien verringert) *oder*
- Festlegen, dass die späte Bindung keinen Kompilierungsfehler verursacht (`Project Properties > Compile Tab > Warning Configuration`)

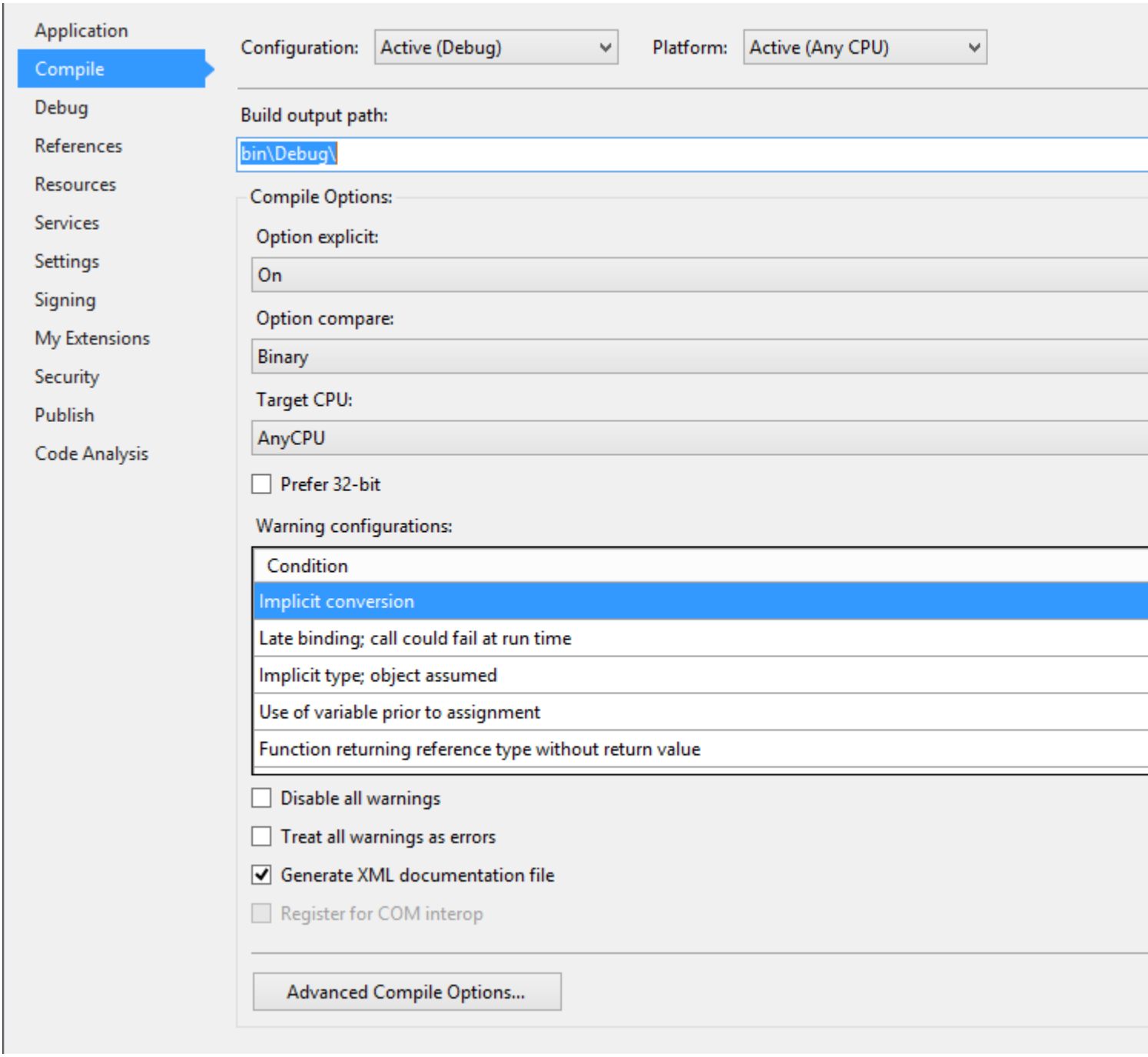
So schalten Sie es ein

- Sie können es auf Modul- / Klassenebene aktivieren, indem Sie die Direktive oben in der Codedatei platzieren.

  ```
  Option Strict On
  ```

- Sie können es auf Projektebene über das Menü in Visual Studio einschalten

  Projekt> [Projekt] Eigenschaften> Registerkarte Kompilieren> Option Strict> Ein



- Sie können es standardmäßig für alle neuen Projekte aktivieren, indem Sie Folgendes auswählen:

  Extras> Optionen> Projekte und Lösungen> VB-Standardeinstellungen> Option Strict Stellen Sie es auf `On` .

Option Strict online lesen: https://riptutorial.com/de/vb-net/topic/4022/option-strict

Kapitel 41: Reflexion

Examples

Rufen Sie Eigenschaften für eine Instanz einer Klasse ab

```
Imports System.Reflection

Public Class PropertyExample

    Public Function GetMyProperties() As PropertyInfo()
        Dim objProperties As PropertyInfo()
        objProperties = Me.GetType.GetProperties(BindingFlags.Public Or BindingFlags.Instance)
        Return objProperties
    End Function

    Public Property ThisWillBeRetrieved As String = "ThisWillBeRetrieved"

    Private Property ThisWillNot As String = "ThisWillNot"

    Public Shared Property NeitherWillThis As String = "NeitherWillThis"

    Public Overrides Function ToString() As String
        Return String.Join(",", GetMyProperties.Select(Function(pi) pi.Name).ToArray)
    End Function
End Class
```

Der Parameter von GetProperties definiert, welche Eigenschaften von der Funktion zurückgegeben werden. Da wir Public und Instance übergeben, gibt die Methode nur Eigenschaften zurück, die sowohl öffentlich als auch nicht freigegeben sind. Weitere Informationen zum Kombinieren von Flag-Enums finden Sie unter Das Attribut Flags .

Holen Sie sich die Mitglieder eines Typs

```
Dim flags = BindingFlags.Static Or BindingFlags.Public Or BindingFlags.Instance
Dim members = GetType(String).GetMembers(flags)
For Each member In members
    Console.WriteLine($"{member.Name}, ({member.MemberType})")
Next
```

Holen Sie sich eine Methode und rufen Sie sie auf

Statische Methode:

```
Dim parseMethod = GetType(Integer).GetMethod("Parse",{GetType(String)})
Dim result = DirectCast(parseMethod.Invoke(Nothing,{"123"}), Integer)
```

Instanzmethode:

```
 Dim instance = "hello".ToUpper
```

```
Dim method = Gettype(String).GetMethod("ToUpper",{})
Dim result = method.Invoke(instance,{})
Console.WriteLine(result) 'HELLO
```

Erstellen Sie eine Instanz eines generischen Typs

```
    Dim openListType = GetType(List(Of ))
    Dim typeParameters = {GetType(String)}
    Dim stringListType = openListType.MakeGenericType(typeParameters)
    Dim instance = DirectCast(Activator.CreateInstance(stringListType), List(Of String))
    instance.Add("Hello")
```

Reflexion online lesen: https://riptutorial.com/de/vb-net/topic/1598/reflexion

Kapitel 42: Rekursion

Examples

N-te Fibonacci-Zahl berechnen

Visual Basic.NET, wie die meisten Sprachen erlaubt Rekursion, ein Verfahren , mit dem eine Funktion *selbst* unter bestimmten Bedingungen fordert.

Hier ist eine grundlegende Funktion in Visual Basic .NET zum Berechnen von Fibonacci- Zahlen.

```
''' <summary>
''' Gets the n'th Fibonacci number
''' </summary>
''' <param name="n">The 1-indexed ordinal number of the Fibonacci sequence that you wish to
receive. Precondition: Must be greater than or equal to 1.</param>
''' <returns>The nth Fibonacci number. Throws an exception if a precondition is
violated.</returns>
Public Shared Function Fibonacci(ByVal n as Integer) as Integer
    If n<1
        Throw New ArgumentOutOfRangeException("n must be greater than or equal to one.")
    End If
    If (n=1) or (n=2)
        ''Base case. The first two Fibonacci numbers (n=1 and n=2) are both 1, by definition.
        Return 1
    End If
    ''Recursive case.
    ''Get the two previous Fibonacci numbers via recursion, add them together, and return the
result.
    Return Fibonacci(n-1) + Fibonacci(n-2)
End Function
```

Diese Funktion überprüft zunächst, ob die Funktion mit dem Parameter `n` gleich `1` oder `2` aufgerufen wurde. Definitionsgemäß sind die ersten beiden Werte in der Fibonacci-Sequenz 1 und 1, daher sind keine weiteren Berechnungen erforderlich, um dies zu bestimmen. Wenn `n` größer als 2 ist, können wir den zugehörigen Wert nicht so leicht nachschlagen. Wir wissen jedoch, dass eine solche Fibonacci-Zahl gleich der Summe der vorherigen beiden Zahlen ist. *Daher* fordern wir diese über *Rekursion* (Aufruf unserer eigenen Fibonacci-Funktion) an. Da aufeinanderfolgende rekursive Aufrufe mit immer kleineren Zahlen über Verringerungen von -1 und -2 aufgerufen, wissen wir , dass irgendwann werden sie Zahlen erreichen , die als 2. Sobald diese Bedingungen kleiner sind (so genannte *Basis Fälle)* erreicht werden, wickelt sich der Stapel und wir Holen Sie sich unser Endergebnis.

Rekursion online lesen: https://riptutorial.com/de/vb-net/topic/7862/rekursion

Kapitel 43: Typumwandlung

Syntax

- CBool (Ausdruck)
- CByte (Ausdruck)
- CChar (Ausdruck)
- CDate (Ausdruck)
- CDbl (Ausdruck)
- CDec (Ausdruck)
- CInt (Ausdruck)
- CLng (Ausdruck)
- CObj (Ausdruck)
- CSByte (Ausdruck)
- CShort (Ausdruck)
- CSng (Ausdruck)
- CStr (Ausdruck)
- CUInt (Ausdruck)
- CULng (Ausdruck)
- CUShort (Ausdruck)

Parameter

Funktionsname	Bereich für das Ausdrucksargument
CBool	Jeder gültige Char oder String oder ein numerischer Ausdruck
CByte	0 bis 255 (ohne Vorzeichen); Bruchteile sind gerundet.
CChar	Jeder gültige Zeichen- oder Zeichenfolgenausdruck; Nur das erste Zeichen eines Strings wird konvertiert. Der Wert kann zwischen 0 und 65535 liegen (unsigniert).

Examples

Text des Textfelds in eine Ganzzahl konvertieren

Von MSDN

> Verwenden Sie die CInt-Funktion, um Konvertierungen von einem anderen Datentyp in einen Integer-Subtyp bereitzustellen. Beispielsweise erzwingt CInt eine Ganzzahlarithmetik, wenn normalerweise eine Währung mit einfacher Genauigkeit oder mit doppelter Genauigkeit auftreten würde.

Angenommen, Sie haben 1 Button und 2 Textbox. Wenn Sie Textbox1.text `5.5` und Textbox2.text `10` eingeben.

Wenn Sie diesen Code haben:

```
Dim result = textbox1.text + textbox2.text
MsgBox("Result: " & result)
'It will output
5.510
```

Um die Werte der `CInt(expression)` hinzuzufügen, müssen Sie ihre Werte mithilfe des `CInt(expression)` in `Int` konvertieren.

```
Dim result = CInt(textbox1.text) + CInt(textbox2.text)
MsgBox("Result: " & result)
'It will output
16
```

> Hinweis: Wenn der Bruchteil eines Werts genau 0,5 beträgt, rundet die CInt-Funktion auf die nächste gerade Zahl. Beispiel: **0,5 Runden auf 0** , **1,5 Runden auf 2 und 3,5 Runden auf 4** . Der Zweck der Rundung auf die nächste gerade Zahl ist die Kompensation einer Verzerrung, die sich addieren könnte, wenn viele Zahlen addiert werden.

Typumwandlung online lesen: https://riptutorial.com/de/vb-net/topic/4681/typumwandlung

Kapitel 44: Unit-Test in VB.NET

Bemerkungen

Dies ist das einfachste, aber beschreibende Beispiel für das Komponententestverfahren. Fühlen Sie sich frei, weitere Methoden hinzuzufügen, um verschiedene Datentypen zu überprüfen.

Examples

Unit-Test für Steuerberechnung

Dieses Beispiel ist in zwei Säulen unterteilt

- **Gehaltsberechnungsklasse** : Berechnung des Nettogehalts nach Steuerabzug
- **SalaryCalculationTests-Klasse** : Zum Testen der Methode, die den Nettolohn berechnet

Schritt 1: Erstellen Sie eine Klassenbibliothek, nennen Sie sie **WagesLibrary** oder einen beliebigen Namen. **Benennen Sie** dann die Klasse in **SalaryCalculation um**

" " " Class für Gehaltsberechnungen " 'Public Class SalaryCalculation

```
    ''' <summary>
    ''' Employee Salary
    ''' </summary>
    Public Shared Salary As Double

    ''' <summary>
    ''' Tax fraction (0-1)
    ''' </summary>
    Public Shared Tax As Double

    ''' <summary>
    ''' Function to calculate Net Salary
    ''' </summary>
    ''' <returns></returns>
    Public Shared Function CalculateNetSalary()
        Return Salary - Salary * Tax
    End Function
End Class
```

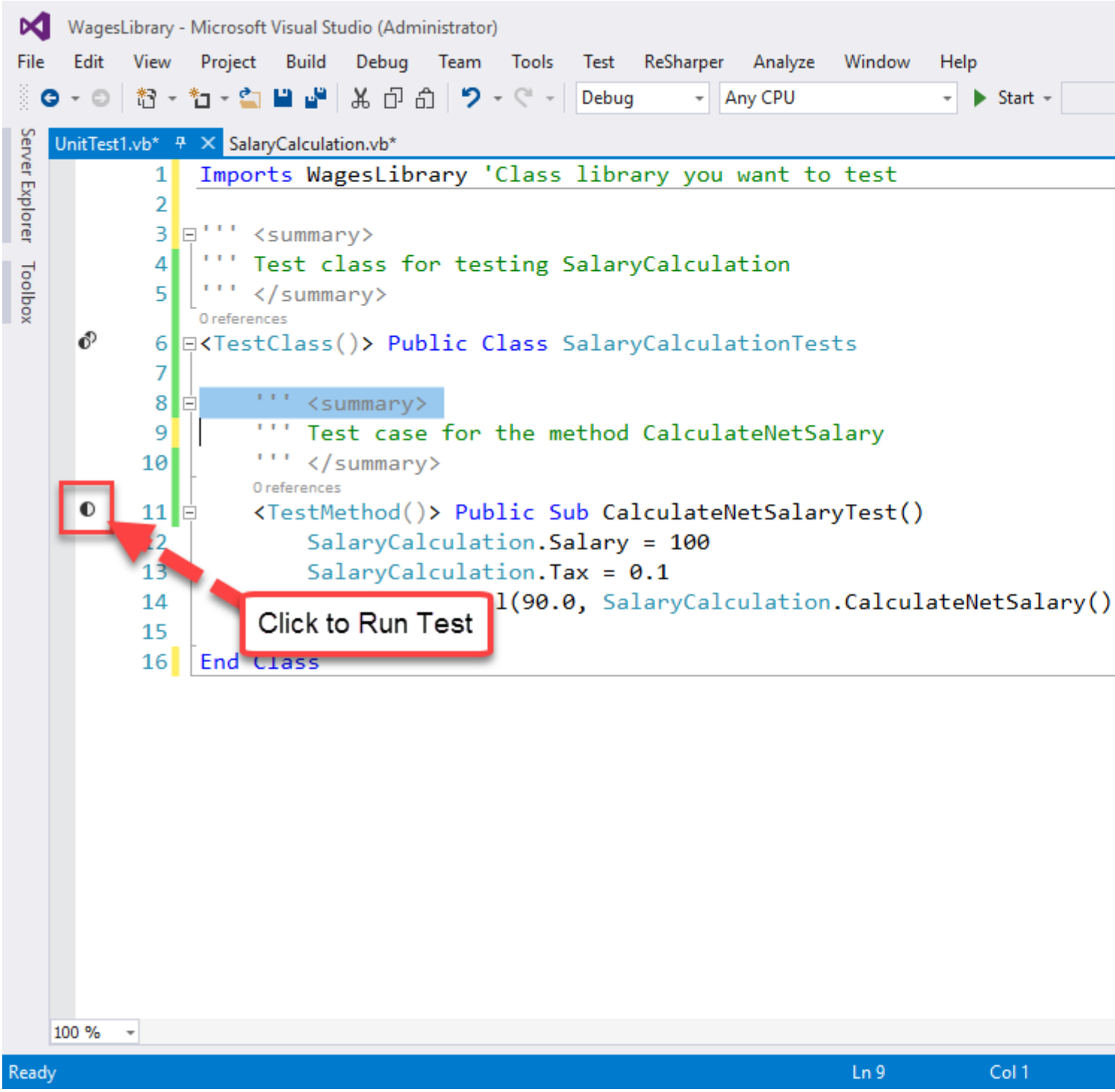
Schritt 2 : Erstellen Sie ein Unit-Test-Projekt. Fügen Sie der erstellten Klassenbibliothek einen Verweis hinzu und fügen Sie den folgenden Code ein

```
Imports WagesLibrary 'Class library you want to test

''' <summary>
''' Test class for testing SalaryCalculation
''' </summary>
<TestClass()> Public Class SalaryCalculationTests

    ''' <summary>
```

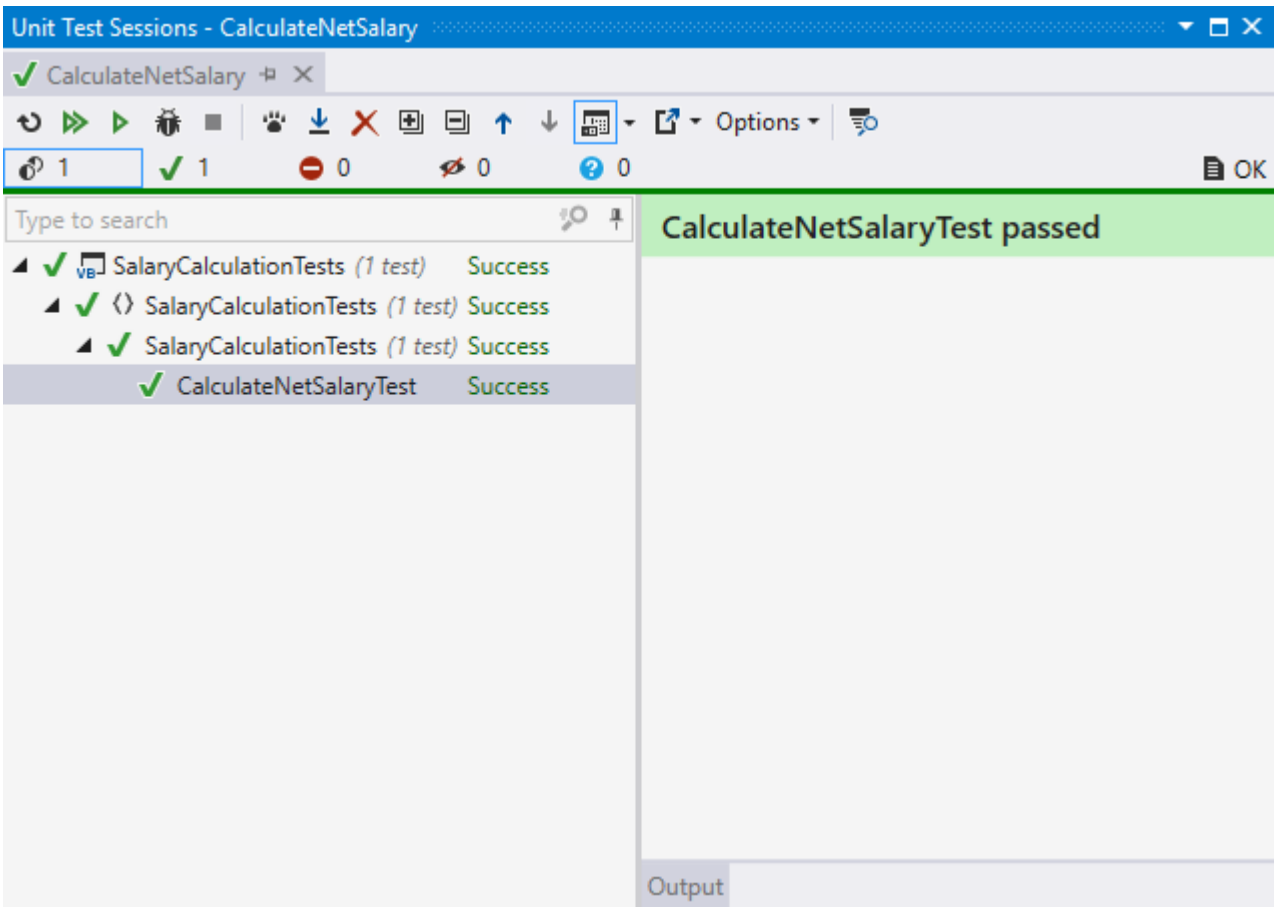
```
    ''' Test case for the method CalculateNetSalary
    ''' </summary>
    <TestMethod()> Public Sub CalculateNetSalaryTest()
        SalaryCalculation.Salary = 100
        SalaryCalculation.Tax = 0.1
        Assert.AreEqual(90.0, SalaryCalculation.CalculateNetSalary(), 0.1)
    End Sub
End Class
```

`Assert.Equal` überprüft den erwarteten Wert mit dem tatsächlich berechneten Wert. Der Wert `0.1` wird verwendet, um Toleranz oder Abweichung zwischen *erwartetem* und *tatsächlichem* Ergebnis zu ermöglichen.

Schritt 3: Führen Sie den Test der Methode aus, um das Ergebnis zu sehen



Testergebnis



Testen der von Mitarbeitern zugewiesenen und abgeleiteten Eigenschaften

In diesem Beispiel stehen mehr Tests für Komponententests zur Verfügung.

Employee.vb (Klassenbibliothek)

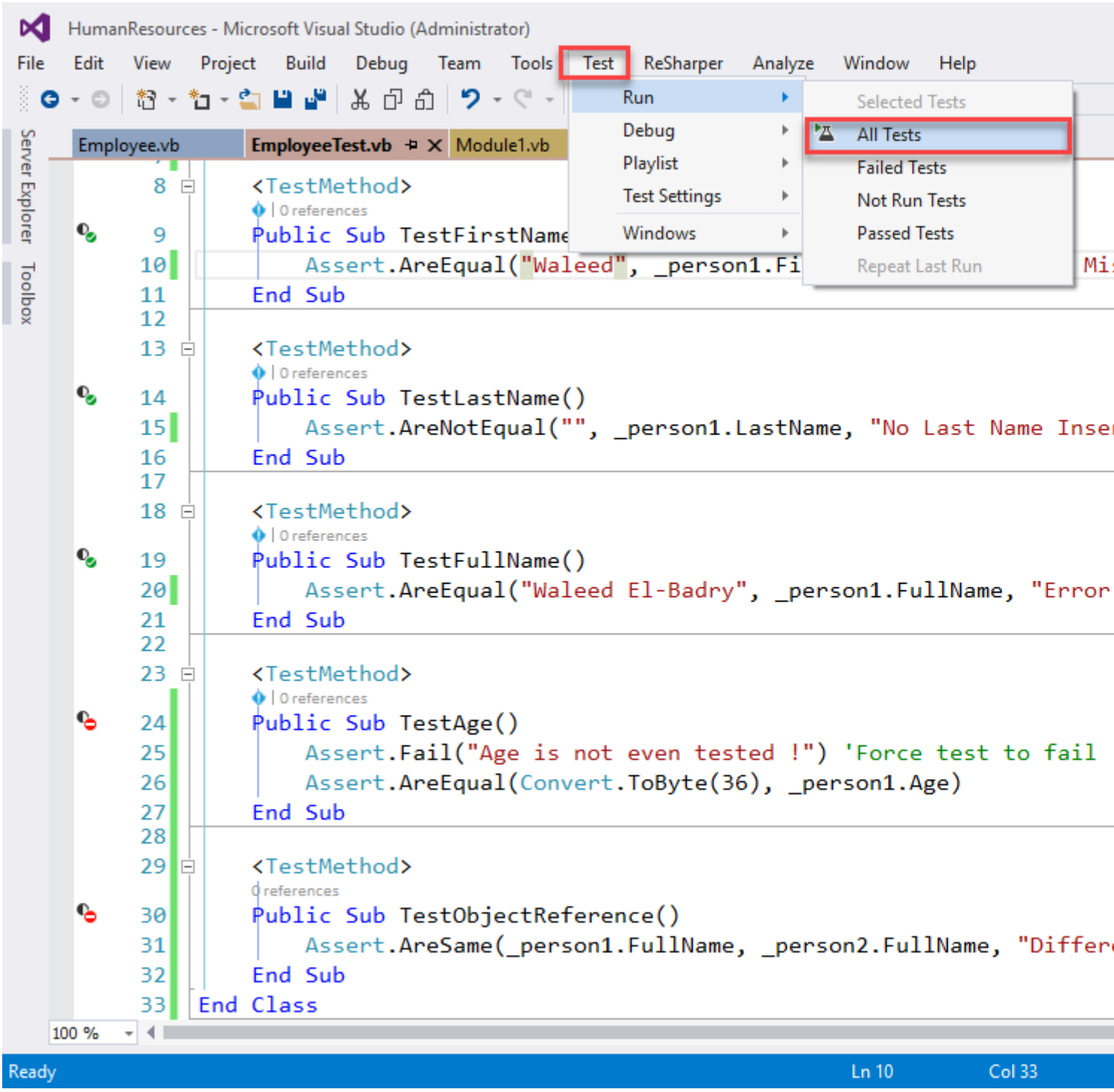
```
''' <summary>
''' Employee Class
''' </summary>
Public Class Employee

    ''' <summary>
    ''' First name of employee
    ''' </summary>
    Public Property FirstName As String = ""

    ''' <summary>
    ''' Last name of employee
    ''' </summary>
    Public Property LastName As String = ""

    ''' <summary>
    ''' Full name of employee
    ''' </summary>
    Public ReadOnly Property FullName As String = ""

    ''' <summary>
    ''' Employee's age
    ''' </summary>
```

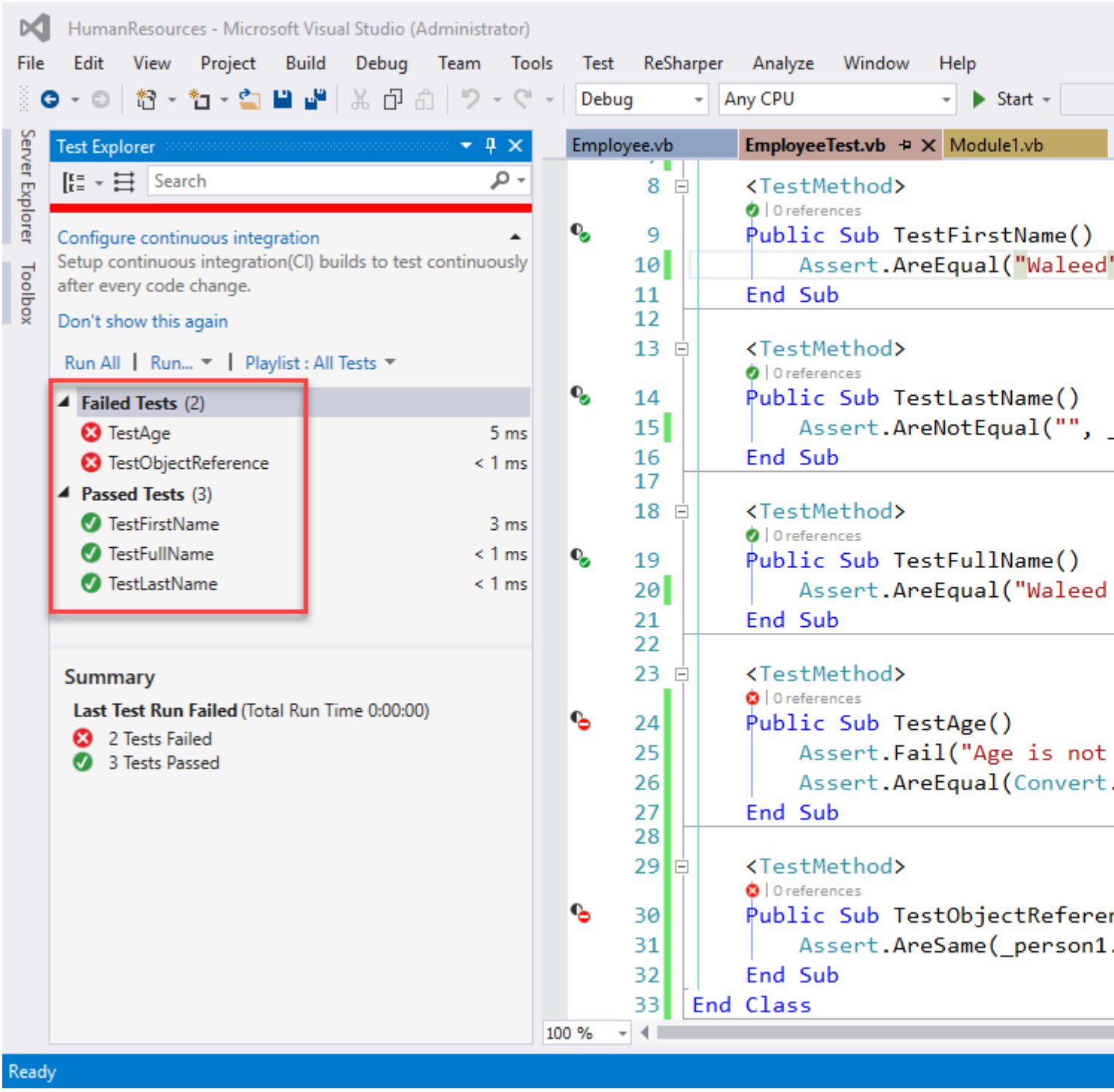
```
    Public Property Age As Byte

    ''' <summary>
    ''' Instantiate new instance of employee
    ''' </summary>
    ''' <param name="firstName">Employee first name</param>
    ''' <param name="lastName">Employee last name</param>
    Public Sub New(firstName As String, lastName As String, dateofbirth As Date)
        Me.FirstName = firstName
        Me.LastName = lastName
        FullName = Me.FirstName + " " + Me.LastName
        Age = Convert.ToByte(Date.Now.Year - dateofbirth.Year)
    End Sub
End Class
```

EmployeeTest.vb (Testprojekt)

```
Imports HumanResources

<TestClass()>
Public Class EmployeeTests
    ReadOnly _person1 As New Employee("Waleed", "El-Badry", New DateTime(1980, 8, 22))
    ReadOnly _person2 As New Employee("Waleed", "El-Badry", New DateTime(1980, 8, 22))

    <TestMethod>
    Public Sub TestFirstName()
        Assert.AreEqual("Waleed", _person1.FirstName, "First Name Mismatch")
    End Sub

    <TestMethod>
    Public Sub TestLastName()
        Assert.AreNotEqual("", _person1.LastName, "No Last Name Inserted!")
    End Sub

    <TestMethod>
    Public Sub TestFullName()
        Assert.AreEqual("Waleed El-Badry", _person1.FullName, "Error in concatination of
names")
    End Sub

    <TestMethod>
    Public Sub TestAge()
        Assert.Fail("Age is not even tested !") 'Force test to fail !
        Assert.AreEqual(Convert.ToByte(36), _person1.Age)
    End Sub

    <TestMethod>
    Public Sub TestObjectReference()
        Assert.AreSame(_person1.FullName, _person2.FullName, "Different objects with same
data")
    End Sub
End Class
```

Ergebnis nach laufenden Tests


HumanResources - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug Team Tools Test ReSharper Analyze Window Help
Run
Debug
Playlist
Test Settings
Windows
Selected Tests
All Tests
Failed Tests
Not Run Tests
Passed Tests
Repeat Last Run
Server Explorer
Toolbox
Employee.vb
EmployeeTest.vb
Module1.vb
<TestMethod>
0 references
Public Sub TestFirstName
Assert.AreEqual("Waleed", _person1.Fi
End Sub
<TestMethod>
0 references
Public Sub TestLastName()
Assert.AreNotEqual("", _person1.LastName, "No Last Name Inse
End Sub
<TestMethod>
0 references
Public Sub TestFullName()
Assert.AreEqual("Waleed El-Badry", _person1.FullName, "Error
End Sub
<TestMethod>
0 references
Public Sub TestAge()
Assert.Fail("Age is not even tested !") 'Force test to fail
Assert.AreEqual(Convert.ToByte(36), _person1.Age)
End Sub
<TestMethod>
0 references
Public Sub TestObjectReference()
Assert.AreSame(_person1.FullName, _person2.FullName, "Differ
End Sub
End Class
100 %
Ready
Ln 10
Col 33



Unit-Test in VB.NET online lesen: https://riptutorial.com/de/vb-net/topic/6843/unit-test-in-vb-net

Kapitel 45: Variablen deklarieren

Syntax

- Öffentlicher Zähler als Integer
- Privater _counter als ganze Zahl
- Dim-Zähler als Ganzzahl

Examples

Eine Variable mit einem primitiven Typ deklarieren und zuweisen

Variablen in Visual Basic werden mit dem Schlüsselwort `Dim` deklariert. Dies deklariert beispielsweise eine neue Variable namens `counter` mit dem Datentyp `Integer` :

```
Dim counter As Integer
```

Eine Variablendeklaration kann auch einen Zugriffsmodifizierer enthalten , z. B. " `Public` , " `Protected` , " `Friend` oder " `Private` . Dies hängt mit dem Gültigkeitsbereich der Variablen zusammen, um die Erreichbarkeit zu bestimmen.

Zugriffsmodifizierer	Bedeutung
Öffentlichkeit	Alle Typen, die auf den umgebenden Typ zugreifen können
Geschützt	Nur die umgebende Klasse und die, die davon erben
Freund	Alle Typen in derselben Assembly, die auf den umschließenden Typ zugreifen können
Geschützter Freund	Die umgebende Klasse und ihre Vererbung *oder* die Typen in derselben Assembly, die auf die umgebende Klasse zugreifen können
Privatgelände	Nur der umschließende Typ
Statisch	Nur bei lokalen Variablen und nur einmal initialisiert.

Das Kürzel `Dim` kann in der Deklaration der Variablen durch den Zugriffsmodifizierer ersetzt werden:

```
Public TotalItems As Integer
Private counter As Integer
```

Die unterstützten Datentypen sind in der folgenden Tabelle aufgeführt:

Art	Alias	Speicherzuweisung	Beispiel
SByte	N / A	1 Byte	`Dim example As SByte = 10`
Int16	Kurz	2 Bytes	`Dim example As Short = 10`
Int32	Ganze Zahl	4 Bytes	`Dim example As Integer = 10`
Int64	Lange	8 Bytes	`Dim example As Long = 10`
Single	N / A	4 Bytes	`Dim example As Single = 10.95`
Doppelt	N / A	8 Bytes	`Dim example As Double = 10.95`
Dezimal	N / A	16 Bytes	`Dim example As Decimal = 10.95`
Boolean	N / A	Durch Implementierung der Plattform vorgegeben	`Dim example As Boolean = True`
Verkohlen	N / A	2 Bytes	`Dim example As Char = "A"C`
String	N / A	$20 + \left\lfloor \frac{length}{2} \right\rfloor * 4bytes$ Quelle	`Dim example As String = "Stack Overflow"`
Terminzeit	Datum	8 Bytes	`Dim example As Date = Date.Now`
Byte	N / A	1 Byte	`Dim example As Byte = 10`
UInt16	UShort	2 Bytes	`Dim example As UShort = 10`
UInt32	UInteger	4 Bytes	`Dim example As UInteger = 10`
UInt64	ULong	8 Bytes	`Dim example As ULong = 10`
Objekt	N / A	4 Byte 32 Bit Architektur, 8 Byte 64 Bit Architektur	`Dim example As Object = Nothing`

Es gibt auch Datenbezeichner- und Literaltyp-Zeichen, die als Ersatz für den Texttyp und oder zum Erzwingen eines Literaltyps verwendet werden können:

Typ (oder Alias)	Kennungstypzeichen	Buchstabendes Zeichen
Kurz	N / A	`example = 10S`
Ganze Zahl	`Dim example%`	`example = 10%` oder `example = 10I`
Lange	`Dim example&`	`example = 10&` oder `example = 10L`

Typ (oder Alias)	Kennungstypzeichen	Buchstabendes Zeichen
Single	`Dim example!`	`example = 10!` oder `example = 10F`
Doppelt	`Dim example#`	`example = 10#` oder `example = 10R`
Dezimal	`Dim example@`	`example = 10@` oder `example = 10D`
Verkohlen	N / A	`example = "A"C`
String	`Dim example$`	N / A
UShort	N / A	`example = 10US`
UInteger	N / A	`example = 10UI`
ULong	N / A	`example = 10UL`

Die Integralsuffixe können auch mit hexadezimalen (& H) oder oktalen (& O) Präfixen verwendet werden:
`example = &H8000S` oder `example = &O77&`

Date (Time) -Objekte können auch mit der Literal-Syntax definiert werden:
`Dim example As Date = #7/26/2016 12:8 PM#`

Sobald eine Variable deklariert ist, ist sie im Gültigkeitsbereich des enthaltenen Typs, `Sub` oder `Function` deklariert. Beispiel:

```
Public Function IncrementCounter() As Integer
    Dim counter As Integer = 0
    counter += 1

    Return counter
End Function
```

Die Zählervariable existiert nur bis zur `End Function` und ist dann außerhalb des Gültigkeitsbereichs. Wenn diese Zählervariable außerhalb der Funktion benötigt wird, müssen Sie sie auf Klassen- / Struktur- oder Modulebene definieren.

```
Public Class ExampleClass

    Private _counter As Integer

    Public Function IncrementCounter() As Integer
       _counter += 1
       Return _counter
    End Function

End Class
```

Alternativ können Sie den `Static` (nicht mit `Shared` zu verwechseln) Modifikator verwenden, um zuzulassen, dass eine lokale Variable ihren Wert zwischen Aufrufen ihrer einschließenden

Methode behält:

```
Function IncrementCounter() As Integer
    Static counter As Integer = 0
    counter += 1

    Return counter
End Function
```

Deklarationsebenen - Lokale und Member-Variablen

Lokale Variablen - Die innerhalb einer Prozedur (Unterprogramm oder Funktion) einer Klasse (oder einer anderen Struktur) deklarierten **Variablen** . In diesem Beispiel ist `exampleLocalVariable` eine lokale Variable, die in `ExampleFunction() exampleLocalVariable` ist:

```
Public Class ExampleClass1

    Public Function ExampleFunction() As Integer
        Dim exampleLocalVariable As Integer = 3
        Return exampleLocalVariable
    End Function

End Class
```

Das Schlüsselwort `Static` ermöglicht die Beibehaltung einer lokalen Variablen und deren Beibehaltung des Werts nach der Beendigung. Normalerweise werden lokale Variablen nicht mehr angezeigt, wenn die enthaltende Prozedur beendet wird.

In diesem Beispiel ist die Konsole `024` . Bei jedem Aufruf von `ExampleSub()` von `Main()` behält die statische Variable den Wert bei, den sie am Ende des vorherigen Aufrufs hatte:

```
Module Module1

    Sub Main()
        ExampleSub()
        ExampleSub()
        ExampleSub()
    End Sub

    Public Sub ExampleSub()
        Static exampleStaticLocalVariable As Integer = 0
        Console.Write(exampleStaticLocalVariable.ToString)
        exampleStaticLocalVariable += 2
    End Sub

End Module
```

Member-Variablen - Wird außerhalb eines Verfahrens auf Klassen- oder anderer Strukturebene deklariert. Dies können **Instanzvariablen sein** , bei denen jede Instanz der enthaltenden Klasse über eine eigene Kopie dieser Variablen verfügt, oder `Shared` **Variablen** , die als einzelne, der Klasse selbst zugeordnete Variable unabhängig von jeder Instanz existieren.

Hier enthält `ExampleClass2` zwei `ExampleClass2` . Jede Instanz der `ExampleClass2` verfügt über eine

eigene `ExampleInstanceVariable` die über die Klassenreferenz zugegriffen werden kann. Auf die gemeinsam genutzte Variable `ExampleSharedVariable` jedoch unter Verwendung des Klassennamens zugegriffen:

```
Module Module1

    Sub Main()

        Dim instance1 As ExampleClass4 = New ExampleClass4
        instance1.ExampleInstanceVariable = "Foo"

        Dim instance2 As ExampleClass4 = New ExampleClass4
        instance2.ExampleInstanceVariable = "Bar"

        Console.WriteLine(instance1.ExampleInstanceVariable)
        Console.WriteLine(instance2.ExampleInstanceVariable)
        Console.WriteLine(ExampleClass4.ExampleSharedVariable)

    End Sub

    Public Class ExampleClass4

        Public ExampleInstanceVariable As String
        Public Shared ExampleSharedVariable As String = "FizzBuzz"

    End Class

End Module
```

Beispiel für Zugriffsmodifizierer

Stellen Sie sich im folgenden Beispiel vor, Sie haben eine Lösung, die zwei Projekte **hostet** : **ConsoleApplication1** und **SampleClassLibrary** . Das erste Projekt enthält die Klassen **SampleClass1** und **SampleClass2** . Die zweite enthält **SampleClass3** und **SampleClass4** . Mit anderen Worten, wir haben zwei Baugruppen mit jeweils zwei Klassen. **ConsoleApplication1** enthält eine Referenz auf **SampleClassLibrary** .

Sehen Sie, wie **SampleClass1.MethodA** mit anderen Klassen und Methoden interagiert.

SampleClass1.vb:

```
Imports SampleClassLibrary

Public Class SampleClass1
    Public Sub MethodA()
        'MethodA can call any of the following methods because
        'they all are in the same scope.
        MethodB()
        MethodC()
        MethodD()
        MethodE()

        'Sample2 is defined as friend. It is accessible within
        'the type itself and all namespaces and code within the same assembly.
        Dim class2 As New SampleClass2()
        class2.MethodA()
```

```
        'class2.MethodB() 'SampleClass2.MethodB is not accessible because
                          'this method is private. SampleClass2.MethodB
                          'can only be called from SampleClass2.MethodA,
                          'SampleClass2.MethodC, SampleClass2.MethodD
                          'and SampleClass2.MethodE
        class2.MethodC()
        'class2.MethodD() 'SampleClass2.MethodD is not accessible because
                          'this method is protected. SampleClass2.MethodD
                          'can only be called from any class that inherits
                          'SampleClass2, SampleClass2.MethodA, SampleClass2.MethodC,
                          'SampleClass2.MethodD and SampleClass2.MethodE
        class2.MethodE()

        Dim class3 As New SampleClass3() 'SampleClass3 resides in other
                                         'assembly and is defined as public.
                                         'It is accessible anywhere.
        class3.MethodA()
        'class3.MethodB() 'SampleClass3.MethodB is not accessible because
                          'this method is private. SampleClass3.MethodB can
                          'only be called from SampleClass3.MethodA,
                          'SampleClass3.MethodC, SampleClass3.MethodD
                          'and SampleClass3.MethodE

        'class3.MethodC() 'SampleClass3.MethodC is not accessible because
                          'this method is friend and resides in another assembly.
                          'SampleClass3.MethodC can only be called anywhere from the
                          'same assembly, SampleClass3.MethodA, SampleClass3.MethodB,
                          'SampleClass3.MethodD and SampleClass3.MethodE

        'class4.MethodD() 'SampleClass3.MethodE is not accessible because
                          'this method is protected friend. SampleClass3.MethodD
                          'can only be called from any class that resides inside
                          'the same assembly and inherits SampleClass3,
                          'SampleClass3.MethodA, SampleClass3.MethodB,
                          'SampleClass3.MethodC and SampleClass3.MethodD

        'Dim class4 As New SampleClass4() 'SampleClass4 is not accessible because
                                          'it is defined as friend and resides in
                                          'other assembly.
    End Sub

    Private Sub MethodB()
        'Doing MethodB stuff...
    End Sub

    Friend Sub MethodC()
        'Doing MethodC stuff...
    End Sub

    Protected Sub MethodD()
        'Doing MethodD stuff...
    End Sub

    Protected Friend Sub MethodE()
        'Doing MethodE stuff...
    End Sub
End Class
```

SampleClass2.vb:

```
Friend Class SampleClass2
    Public Sub MethodA()
        'Doing MethodA stuff...
    End Sub

    Private Sub MethodB()
        'Doing MethodB stuff...
    End Sub

    Friend Sub MethodC()
        'Doing MethodC stuff...
    End Sub

    Protected Sub MethodD()
        'Doing MethodD stuff...
    End Sub

    Protected Friend Sub MethodE()
        'Doing MethodE stuff...
    End Sub
End Class
```

SampleClass3.vb:

```
Public Class SampleClass3
    Public Sub MethodA()
        'Doing MethodA stuff...
    End Sub
    Private Sub MethodB()
        'Doing MethodB stuff...
    End Sub

    Friend Sub MethodC()
        'Doing MethodC stuff...
    End Sub

    Protected Sub MethodD()
        'Doing MethodD stuff...
    End Sub

    Protected Friend Sub MethodE()
        'Doing MethodE stuff...
    End Sub
End Class
```

SampleClass4.vb:

```
Friend Class SampleClass4
    Public Sub MethodA()
        'Doing MethodA stuff...
    End Sub
    Private Sub MethodB()
        'Doing MethodB stuff...
    End Sub

    Friend Sub MethodC()
        'Doing MethodC stuff...
    End Sub
```

```
    Protected Sub MethodD()
        'Doing MethodD stuff...
    End Sub

    Protected Friend Sub MethodE()
        'Doing MethodE stuff...
    End Sub
End Class
```

Variablen deklarieren online lesen: https://riptutorial.com/de/vb-net/topic/3366/variablen-deklarieren

Kapitel 46: Verbindungsabwicklung

Examples

Öffentliche Verbindungseigenschaft

```
Imports System.Data.OleDb

Private WithEvents _connection As OleDbConnection
Private _connectionString As String = "myConnectionString"

Public ReadOnly Property Connection As OleDbConnection
    Get
        If _connection Is Nothing Then
            _connection = New OleDbConnection(_connectionString)
            _connection.Open()
        Else
            If _connection.State <> ConnectionState.Open Then
                _connection.Open()
            End If
        End If
        Return _connection
    End Get
End Property
```

Verbindungsabwicklung online lesen: https://riptutorial.com/de/vb-net/topic/6398/verbindungsabwicklung

Kapitel 47: Visual Basic 14.0-Funktionen

Einführung

Visual Basic 14 ist die Version von Visual Basic, die als Teil von Visual Studio 2015 ausgeliefert wurde.

Diese Version wurde in rund 1,3 Millionen Zeilen VB von Grund auf neu geschrieben. Viele Funktionen wurden hinzugefügt, um häufige Irritationen zu beseitigen und gängige Codierungsmuster zu reinigen.

Die Versionsnummer von Visual Basic ging von 12 auf 14 über, wobei 13 übersprungen wurde. Dies wurde gemacht, um VB an die Versionsnummerierung von Visual Studio anzupassen.

Examples

Null bedingter Operator

Um eine ausführliche Nullprüfung zu vermeiden, das `?.` Operator wurde in der Sprache eingeführt.

Die alte verbose Syntax:

```
If myObject IsNot Nothing AndAlso myObject.Value >= 10 Then
```

Kann jetzt durch die Kurzfassung ersetzt werden:

```
If myObject?.Value >= 10 Then
```

Die `?` Operator ist besonders leistungsfähig, wenn Sie eine Kette von Eigenschaften haben. Folgendes berücksichtigen:

```
Dim fooInstance As Foo = Nothing
Dim s As String
```

Normalerweise müsste man so etwas schreiben:

```
If fooInstance IsNot Nothing AndAlso fooInstance.BarInstance IsNot Nothing Then
    s = fooInstance.BarInstance.Baz
Else
    s = Nothing
End If
```

Aber mit dem `?` Operator kann durch nur ersetzt werden:

```
s = fooInstance?.BarInstance?.Baz
```

NameOf-Operator

Der `NameOf` Operator löst Namespaces, Typen, Variablen und `NameOf` zur Kompilierzeit auf und ersetzt sie durch die entsprechende Zeichenfolge.

Einer der Anwendungsfälle:

```
Sub MySub(variable As String)
    If variable Is Nothing Then Throw New ArgumentNullException("variable")
End Sub
```

Die alte Syntax birgt das Risiko, dass die Variable umbenannt wird und der hartcodierte String den falschen Wert hat.

```
Sub MySub(variable As String)
    If variable Is Nothing Then Throw New ArgumentNullException(NameOf(variable))
End Sub
```

Bei `NameOf` führt das Umbenennen der Variablen nur zu einem Compiler-Fehler. Dadurch kann das Umbenennungstool beide mit einem einzigen Aufwand umbenennen.

Der `NameOf` Operator verwendet nur die letzte Komponente der Referenz in den Klammern. Dies ist wichtig, wenn Sie zum Beispiel Namespaces im `NameOf` Operator behandeln.

```
Imports System

Module Module1
    Sub WriteIO()
        Console.WriteLine(NameOf(IO)) 'displays "IO"
        Console.WriteLine(NameOf(System.IO)) 'displays "IO"
    End Sub
End Module
```

Der Operator verwendet auch den Namen der eingegebenen Referenz, ohne dass Importe mit Namensänderungen aufgelöst werden. Zum Beispiel:

```
Imports OldList = System.Collections.ArrayList

Module Module1
    Sub WriteList()
        Console.WriteLine(NameOf(OldList)) 'displays "OldList"
        Console.WriteLine(NameOf(System.Collections.ArrayList)) 'displays "ArrayList"
    End Sub
End Module
```

String-Interpolation

Diese neue Funktion macht die Verkettung von Strings lesbarer. Diese Syntax wird zu dem entsprechenden `String.Format` Aufruf kompiliert.

Ohne String Interpolation:

```
String.Format("Hello, {0}", name)
```

Mit Stringinterpolation:

```
$"Hello, {name}"
```

Die beiden Zeilen sind gleichwertig und beide werden zu einem Aufruf von `String.Format` .

Wie in `String.Format` können die Klammern jeden einzelnen Ausdruck enthalten (Aufruf einer Methode, Eigenschaft, eines Null-Koaleszenzoperators usw.).

String Interpolation ist die bevorzugte Methode gegenüber `String.Format` da einige Laufzeitfehler `String.Format` . Betrachten Sie die folgende `String.Format` Zeile:

```
String.Format("The number of people is {0}/{1}", numPeople)
```

Dies wird kompiliert, verursacht jedoch einen Laufzeitfehler, da der Compiler nicht überprüft, ob die Anzahl der Argumente mit den Platzhaltern übereinstimmt.

Schreibgeschützte Auto-Eigenschaften

Schreibgeschützte Eigenschaften waren in VB.NET immer in diesem Format möglich:

```
Public Class Foo

  Private _MyProperty As String = "Bar"

  Public ReadOnly Property MyProperty As String
    Get
      Return _MyProperty
    End Get
  End Property

End Class
```

Die neue Version von Visual Basic erlaubt eine kurze Hand für die Eigenschaftsdeklaration wie folgt:

```
Public Class Foo

  Public ReadOnly Property MyProperty As String = "Bar"

End Class
```

Die tatsächliche Implementierung, die der Compiler generiert, ist für beide Beispiele identisch. Die neue Methode zum Schreiben ist nur eine kurze Hand. Der Compiler generiert weiterhin ein privates Feld mit dem Format: `_<PropertyName>` , um die schreibgeschützte Eigenschaft zu sichern.

Teilmodule und Schnittstellen

Ähnlich wie bei Teilklassen kann die neue Version von Visual Basic jetzt mit Teilmodulen und Teilschnittstellen umgehen. Die Syntax und das Verhalten sind genau dieselben wie für Teilklassen.

Ein Teilmodulbeispiel:

```
Partial Module Module1
  Sub Main()
    Console.Write("Ping -> ")
    TestFunktion()
  End Sub
End Module

Partial Module Module1
  Private Sub TestFunktion()
    Console.WriteLine("Pong")
  End Sub
End Module
```

Und eine teilweise Schnittstelle:

```
Partial Interface Interface1
  Sub Methode1()
End Interface

Partial Interface Interface1
  Sub Methode2()
End Interface

Public Class Class1
  Implements Interface1
  Public Sub Methode1() Implements Interface1.Methode1
    Throw New NotImplementedException()
  End Sub

  Public Sub Methode2() Implements Interface1.Methode2
    Throw New NotImplementedException()
  End Sub
End Class
```

Wie bei Teilklassen müssen die Definitionen für die Teilmodule und Schnittstellen im selben Namensraum und in derselben Assembly liegen. Dies liegt daran, dass die Teilkomponenten der Module und Schnittstellen während der Kompilierung zusammengeführt werden und die kompilierte Assembly keine Hinweise darauf enthält, dass die ursprüngliche Definition des Moduls oder der Schnittstelle aufgeteilt wurde.

Mehrzeilige String-Literale

VB erlaubt jetzt String-Literale, die sich über mehrere Zeilen aufteilen.

Alte Syntax:

```
Dim text As String = "Line1" & Environment.NewLine & "Line2"
```

Neue Syntax:

```
Dim text As String = "Line 1
Line 2"
```

#Region Direktive Verbesserungen

Die #Region-Direktive kann jetzt in Methoden platziert werden und kann sogar Methoden, Klassen und Module umfassen.

```
#Region "A Region Spanning A Class and Ending Inside Of A Method In A Module"
   Public Class FakeClass
    'Nothing to see here, just a fake class.
   End Class

   Module Extensions

    ''' <summary>
    ''' Checks the path of files or directories and returns [TRUE] if it exists.
    ''' </summary>
    ''' <param name="Path">[Sting] Path of file or directory to check.</param>
    ''' <returns>[Boolean]</returns>
    <Extension>
    Public Function PathExists(ByVal Path As String) As Boolean
        If My.Computer.FileSystem.FileExists(Path) Then Return True
        If My.Computer.FileSystem.DirectoryExists(Path) Then Return True
        Return False
    End Function

    ''' <summary>
    ''' Returns the version number from the specified assembly using the assembly's strong
name.
    ''' </summary>
    ''' <param name="Assy">[Assembly] Assembly to get the version info from.</param>
    ''' <returns>[String]</returns>
    <Extension>
    Friend Function GetVersionFromAssembly(ByVal Assy As Assembly) As String
#End Region
        Return Split(Split(Assy.FullName, ",")(1), "=")(1)
    End Function
End Module
```

Kommentare nach impliziter Zeilenfortsetzung

Mit VB 14.0 können Kommentare nach der impliziten Zeilenfortsetzung hinzugefügt werden.

```
Dim number =
    From c As Char 'Comment
    In "dj58kwd92n4" 'Comment
    Where Char.IsNumber(c) 'Comment
    Select c 'Comment
```

Ausnahmebehandlung

Während der Codierung treten häufig unerwartete Fehler auf, die Debugging und Testen erfordern. Aber manchmal werden die Fehler tatsächlich erwartet und umgangen, gibt es den `Try..Catch..Throw..Finally..End Try` Block.

Um einen Fehler richtig zu behandeln, wird der Code in einen `Try..Catch` Block `Try..Catch` , wobei der `Catch` , wie der Name sagt, alle Ausnahmen fängt, die in diesem Block auftreten.

Und in Ausnahmefällen haben wir die Möglichkeit, den Fehler zu `Throw` , das heißt, den Benutzer zu benachrichtigen oder ihn intern im Code selbst zu verwalten.

Der `Finally` Teil ist der letzte Code, der , was auch immer das Ergebnis sein, wenn es eine Ausnahme ist oder nicht, wird der Code ausgeführt werden, bevor aus dem Block gehen.

Für den Fall, dass wir die Uhr herausspringen müssen, gibt es die `Exit Try` Anweisung, die verwendet werden kann. Aber auch hier ist der Code im `Finally` wird Abschnitt vor dem Ende ausgeführt werden.

Die Syntax ist einfach.

```
Try
    [ tryStatements ]
    [ Exit Try ]
[ Catch [ exception [ As type ] ] [ When expression ]
    [ catchStatements ]
    [ Exit Try ] ]
[ Catch ... ]
[ Finally
    [ finallyStatements ] ]
End Try
```

wo nur das `Try` and `End Try` obligatorisch ist. Der Rest kann ignoriert werden. Fügen Sie jedoch den `Finally` Teil hinzu, auch wenn er leer bleibt.

Bei der Ausnahme gibt es verschiedene Arten von Ausnahmen, die abgefangen werden können. Sie sind fertige Ausnahmen, die im .Net Framework verfügbar sind (siehe unten).

Ausnahmeklasse	Kurze Beschreibung
System.IO.IOException	Behandelt E / A-Fehler
System.IndexOutOfRangeException	Bezieht sich auf einen Array-Index außerhalb des gültigen Bereichs
System.ArrayTypeMismatchException	Wenn der Typ nicht mit dem Arraytyp übereinstimmt
System.NullReferenceException	Behandelt Fehler, die beim Referenzieren eines Nullobjekts generiert werden.
System.DivideByZeroException	Behandelt Fehler, die durch Dividieren einer Dividende mit Null generiert werden.

Ausnahmeklasse	Kurze Beschreibung
System.InvalidCastException	Behandelt Fehler, die während der Typumwandlung erzeugt werden.
System.OutOfMemoryException	Behandelt Fehler, die aus nicht genügend freiem Speicher stammen.
System.StackOverflowException	Behandelt Fehler, die vom Stapelüberlauf generiert werden.

Visual Basic 14.0-Funktionen online lesen: https://riptutorial.com/de/vb-net/topic/1501/visual-basic-14-0-funktionen

Kapitel 48: WinForms SpellCheckBox

Einführung

Beispiel zum Hinzufügen eines Kontrollkästchens für eine Rechtschreibprüfung zu einer WindowsForms-Anwendung. In diesem Beispiel ist weder eine Installation von Word noch eine Verwendung von Word erforderlich.

Es verwendet WPF-Interop mit dem ElementHost-Steuerelement, um ein WPF UserControl aus einer WPF-TextBox zu erstellen. WPF TextBox verfügt über eine integrierte Funktion zur Rechtschreibprüfung. Wir werden diese eingebaute Funktion nutzen, anstatt uns auf ein externes Programm zu verlassen.

Examples

ElementHost WPF TextBox

Dieses Beispiel ist einem Beispiel nachempfunden, das ich im Internet gefunden habe. Ich kann den Link nicht finden, oder ich würde dem Autor eine Anerkennung geben. Ich nahm das gefundene Muster und modifizierte es für meine Bewerbung.

1. Fügen Sie die folgenden Referenzen hinzu:

   System.Xaml, PresentationCore, PresentationFramework, WindowsBase und WindowsFormsIntegration

2. Erstellen Sie eine neue Klasse und fügen Sie diesen Code ein

```
Imports System
Imports System.ComponentModel
Imports System.ComponentModel.Design.Serialization
Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Forms.Integration
Imports System.Windows.Forms.Design

<Designer(GetType(ControlDesigner))> _
Class SpellCheckBox
Inherits ElementHost

Private box As TextBox

Public Sub New()
    box = New TextBox()
    MyBase.Child = box
    AddHandler box.TextChanged, AddressOf box_TextChanged
    box.SpellCheck.IsEnabled = True
    box.VerticalScrollBarVisibility = ScrollBarVisibility.Auto
    Me.Size = New System.Drawing.Size(100, 20)
End Sub
```

```
Private Sub box_TextChanged(ByVal sender As Object, ByVal e As EventArgs)
    OnTextChanged(EventArgs.Empty)
End Sub

<DefaultValue("")> _
Public Overrides Property Text() As String
    Get
        Return box.Text
    End Get
    Set(ByVal value As String)
        box.Text = value
    End Set
End Property

<DefaultValue(True)> _
Public Property MultiLine() As Boolean
    Get
        Return box.AcceptsReturn
    End Get
    Set(ByVal value As Boolean)
        box.AcceptsReturn = value
    End Set
End Property

<DefaultValue(True)> _
Public Property WordWrap() As Boolean
    Get
        Return box.TextWrapping <> TextWrapping.Wrap
    End Get
    Set(ByVal value As Boolean)
        If value Then
            box.TextWrapping = TextWrapping.Wrap
        Else
            box.TextWrapping = TextWrapping.NoWrap
        End If
    End Set
End Property

<DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)> _
Public Shadows Property Child() As System.Windows.UIElement
    Get
        Return MyBase.Child
    End Get
    Set(ByVal value As System.Windows.UIElement)
        '' Do nothing to solve a problem with the serializer !!
    End Set
End Property

End Class
```

3. Erstellen Sie die Lösung neu.

4. Fügen Sie ein neues Formular hinzu.

5. Durchsuchen Sie die Toolbox nach Ihrem Klassennamen. Dieses Beispiel ist "Rechtschreibprüfung". Es sollte unter "YourSoulutionName" -Komponenten aufgeführt sein.

6. Ziehen Sie das neue Steuerelement in Ihr Formular

7. Legen Sie alle zugeordneten Eigenschaften im Formularereignis zum Laden von Formularen fest

```
Private Sub form1_Load(sender As Object, e As EventArgs) Handles Me.Load
    spellcheckbox.WordWrap = True
    spellcheckbox.MultiLin = True
    'Add any other property modifiers here...
End Sub
```

7. Das letzte, was Sie tun müssen, ist die DPI-Kenntnis Ihrer Anwendung zu ändern. Dies ist darauf zurückzuführen, dass Sie die WinForms-Anwendung verwenden. Standardmäßig sind alle WinForms-Anwendungen DPI UNAWARE. Wenn Sie ein Steuerelement ausführen, das über einen Elementhost (WPF-Interop) verfügt, wird die Anwendung jetzt DPI AWARE. Dies kann mit Ihren Benutzeroberflächenelementen verwechseln. Die Lösung dafür ist, die Anwendung zu ERWERBEN, um DPI UNAWARE zu werden. Es gibt zwei Möglichkeiten, dies zu tun. Die erste ist durch die Manifestdatei und die zweite besteht darin, sie hart in Ihr Programm einzugeben. Wenn Sie OneClick zum Bereitstellen Ihrer Anwendung verwenden, müssen Sie sie hart codieren, nicht die Manifestdatei verwenden, da andernfalls Fehler auftreten.

Die beiden folgenden Beispiele finden Sie im Folgenden: WinForms-Skalierung bei großen DPI-Einstellungen - ist das überhaupt möglich? Vielen Dank an Telerik.com für die großartige Erklärung zu DPI.

> Fest codierter DPI-Code-Beispiel. Dies muss ausgeführt werden, bevor das erste Formular initialisiert wird. Ich platziere dies immer in der ApplicationEvents.vb-Datei. Sie können zu dieser Datei gelangen, indem Sie im Projektmappen-Explorer mit der rechten Maustaste auf Ihren Projektnamen klicken und "Öffnen" wählen. Wählen Sie dann die Anwendungsregisterkarte links aus und klicken Sie unten rechts neben dem Begrüßungsbildschirm auf "Anwendungsereignisse anzeigen".

```
Namespace My

    ' The following events are available for MyApplication:
    '
    ' Startup: Raised when the application starts, before the startup form is created.
    ' Shutdown: Raised after all application forms are closed.  This event is not raised if
the application terminates abnormally.
    ' UnhandledException: Raised if the application encounters an unhandled exception.
    ' StartupNextInstance: Raised when launching a single-instance application and the
application is already active.
    ' NetworkAvailabilityChanged: Raised when the network connection is connected or
disconnected.
    Partial Friend Class MyApplication

    Private Enum PROCESS_DPI_AWARENESS
        Process_DPI_Unaware = 0
        Process_System_DPI_Aware = 1
        Process_Per_Monitor_DPI_Aware = 2
    End Enum

    Private Declare Function SetProcessDpiAwareness Lib "shcore.dll" (ByVal Value As
PROCESS_DPI_AWARENESS) As Long
```

```
    Private Sub SetDPI()
        'Results from SetProcessDPIAwareness
        'Const S_OK = &H0&
        'Const E_INVALIDARG = &H80070057
        'Const E_ACCESSDENIED = &H80070005

        Dim lngResult As Long

        lngResult = SetProcessDpiAwareness(PROCESS_DPI_AWARENESS.Process_DPI_Unaware)

    End Sub

    Private Sub MyApplication_Startup(sender As Object, e As
ApplicationServices.StartupEventArgs) Handles Me.Startup
        SetDPI()
    End Sub

End Namespace
```

Manifestes Beispiel

```
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0"
xmlns:asmv3="urn:schemas-microsoft-com:asm.v3" >
   <asmv3:application>
        <asmv3:windowsSettings xmlns="http://schemas.microsoft.com/SMI/2005/WindowsSettings">
             <dpiAware>true</dpiAware>
        </asmv3:windowsSettings>
   </asmv3:application>
</assembly>
```

WinForms SpellCheckBox online lesen: https://riptutorial.com/de/vb-net/topic/8624/winforms-spellcheckbox

Kapitel 49: Wörterbücher

Einführung

Ein Wörterbuch repräsentiert eine Sammlung von Schlüsseln und Werten. Siehe MSDN Dictionary (Tkey, TValue) -Klasse .

Examples

Durchlaufen Sie ein Wörterbuch und drucken Sie alle Einträge

Jedes Paar im Wörterbuch ist eine Instanz von `KeyValuePair` mit denselben Typparametern wie das Wörterbuch. Wenn Sie mit `For Each` durch das Wörterbuch `For Each` , erhalten Sie bei jeder Wiederholung eines der im Wörterbuch gespeicherten Schlüsselwertpaare.

```
For Each kvp As KeyValuePair(Of String, String) In currentDictionary
  Console.WriteLine("{0}: {1}", kvp.Key, kvp.Value)
Next
```

Erstellen Sie ein mit Werten gefülltes Wörterbuch

```
Dim extensions As New Dictionary(Of String, String) _
  from { { "txt", "notepad" },
  { "bmp", "paint" },
  { "doc", "winword" } }
```

Dadurch wird ein Wörterbuch erstellt und sofort mit drei KeyValuePairs gefüllt.

Sie können neue Werte auch später mit der Add-Methode hinzufügen:

```
extensions.Add("png", "paint")
```

Beachten Sie, dass der Schlüssel (der erste Parameter) im Wörterbuch eindeutig sein muss. Andernfalls wird eine Ausnahme ausgelöst.

Einen Wörterbuchwert abrufen

Sie können den Wert eines Eintrags im Wörterbuch über die Eigenschaft 'Item' abrufen:

```
Dim extensions As New Dictionary(Of String, String) From {
    { "txt", "notepad" },
    { "bmp", "paint" },
    { "doc", "winword" }
}

Dim program As String = extensions.Item("txt") 'will be "notepad"
```

```
' alternative syntax as Item is the default property (a.k.a. indexer)
Dim program As String = extensions("txt") 'will be "notepad"

' other alternative syntax using the (rare)
' dictionary member access operator (a.k.a. bang operator)
Dim program As String = extensions!txt 'will be "notepad"
```

Wenn der Schlüssel nicht im Wörterbuch vorhanden ist, wird eine KeyNotFoundException ausgelöst.

Bereits im Wörterbuch nach Schlüssel suchen - Datenreduzierung

Mit der Methode `ContainsKey` , ob ein Schlüssel bereits im Wörterbuch vorhanden ist.

Dies ist praktisch für die Datenreduzierung. In dem folgenden Beispiel fügen wir jedes Mal, wenn wir auf ein neues Wort stoßen, es als Schlüssel im Wörterbuch hinzu, ansonsten erhöhen wir den Zähler für dieses bestimmte Wort.

```
Dim dic As IDictionary(Of String, Integer) = New Dictionary(Of String, Integer)

Dim words As String() = Split(<big text source>," ", -1, CompareMethod.Binary)

For Each str As String In words
    If dic.ContainsKey(str) Then
        dic(str) += 1
    Else
        dic.Add(str, 1)
    End If
Next
```

Beispiel für die XML-Reduzierung: Abrufen aller untergeordneten Knoten und Vorkommen in einem Zweig eines XML-Dokuments

```
Dim nodes As IDictionary(Of String, Integer) = New Dictionary(Of String, Integer)
Dim xmlsrc = New XmlDocument()
xmlsrc.LoadXml(<any text stream source>)

For Each xn As XmlNode In xmlsrc.FirstChild.ChildNodes 'selects the proper parent
    If nodes.ContainsKey(xn.Name) Then
        nodes(xn.Name) += 1
    Else
        nodes.Add(xn.Name, 1)
    End If
Next
```

Wörterbücher online lesen: https://riptutorial.com/de/vb-net/topic/2165/worterbucher

Kapitel 50: WPF-XAML-Datenbindung

Einführung

In diesem Beispiel wird gezeigt, wie ein ViewModel und eine View innerhalb des MVVM-Musters und der WPF erstellt und die beiden miteinander verbunden werden, sodass jedes aktualisiert wird, wenn das andere geändert wird.

Examples

Binden einer Zeichenfolge im ViewModel an ein Textfeld in der Ansicht

SampleViewModel.vb

```
'Import classes related to WPF for simplicity
Imports System.Collections.ObjectModel
Imports System.ComponentModel

Public Class SampleViewModel
    Inherits DependencyObject
    'A class acting as a ViewModel must inherit from DependencyObject

    'A simple string property
    Public Property SampleString as String
        Get
            Return CType(GetValue(SampleStringProperty), String)
        End Get

        Set(ByVal value as String)
            SetValue(SampleStringProperty, value)
        End Set
    End Property

    'The DependencyProperty that makes databinding actually work
    'for the string above
    Public Shared ReadOnly SampleStringProperty As DependencyProperty = _
                           DependencyProperty.Register("SampleString", _
                           GetType(String), GetType(SampleViewModel), _
                           New PropertyMetadata(Nothing))

End Class
```

Eine DependencyProperty kann einfach mit dem `wpfdp` hinzugefügt werden (geben Sie `wpfdp` , drücken `wpfdp` dann zweimal die `TAB` `wpfdp`). Der `wpfdp` ist jedoch nicht typsicher und wird unter `Option Strict On` nicht kompiliert.

SampleWindow.xaml

```
<Window x:Class="SampleWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

```
        xmlns:des="http://schemas.microsoft.com/expression/blend/2008"
        DataContext="{Binding}"
        Loaded="Window_Loaded">
    <Grid>
        <TextBox>
            <TextBox.Text>
                <Binding Path="SampleString" />
            </TextBox.Text>
        </TextBox>
    </Grid>
</Window>
```

SampleWindow.xaml.vb

```
Class SampleWindow

    Private WithEvents myViewModel As New SampleViewModel()

    Private Sub Window_Loaded(sender As Object, e As RoutedEventArgs)
        Me.DataContext = myViewModel
    End Sub
End Class
```

Beachten Sie, dass dies eine sehr rudimentäre Methode ist, um MVVM und Datenbindung zu implementieren. Eine robustere Vorgehensweise wäre, eine Plattform wie Unity zu verwenden, um das ViewModel in die Ansicht zu "injizieren".

WPF-XAML-Datenbindung online lesen: https://riptutorial.com/de/vb-net/topic/8177/wpf-xaml-datenbindung

Kapitel 51: Zufällig

Einführung

Die Random-Klasse wird verwendet, um nicht negative pseudozufällige Ganzzahlen zu generieren, die nicht wirklich zufällig sind, aber für allgemeine Zwecke nahe genug liegen.

Die Sequenz wird anhand einer Anfangsnummer (" **Seed"**) berechnet. In früheren Versionen von .net war diese Startnummer bei jeder Ausführung der Anwendung gleich. Es würde also passieren, dass Sie bei jeder Ausführung der Anwendung dieselbe Folge von Pseudo-Zufallszahlen erhalten würden. Nun basiert der Startwert auf der Zeit, zu der das Objekt deklariert ist.

Bemerkungen

Zum Schluss noch ein Hinweis zur Randomisierung. Wenn Sie, wie bereits erwähnt, eine Instanz von `Random` ohne Parameter deklarieren, verwendet der Konstruktor die aktuelle Uhrzeit als Teil der Berechnung, um die ursprüngliche Startnummer zu erstellen. Normalerweise ist das OK.

Jedoch. Wenn Sie innerhalb kürzester Zeit neue Instanzen erneut deklarieren, könnte die Zeit bei jeder Berechnung der Startnummer gleich sein. Betrachten Sie diesen Code.

```
For i As Integer = 1 To 100000
    Dim rnd As New Random
    x = rnd.Next
Next
```

Da Computer heutzutage sehr schnell sind, dauert die Ausführung dieses Codes nur einen Bruchteil einer Sekunde, und bei mehreren dequentiellen Iterationen der Schleife hat sich die Systemzeit nicht geändert. Die Startnummer ändert sich also nicht und die Zufallszahl ist gleich. Wenn Sie viele Zufallszahlen generieren möchten, deklarieren Sie in diesem einfachen Beispiel die Instanz des Zufalls außerhalb der Schleife.

```
Dim rnd As New Random
For i As Integer = 1 To 100000
    x = rnd.Next
Next
```

Als Faustregel gilt, dass der Zufallszahlengenerator nicht über kurze Zeiträume erneut instanziiert wird.

Examples

Instanz deklarieren

```
Dim rng As New Random()
```

Dies deklariert eine Instanz der Random-Klasse namens `rng` . In diesem Fall wird die aktuelle Zeit an dem Punkt, an dem das Objekt erstellt wird, zur Berechnung des Saatguts verwendet. Dies ist die häufigste Verwendung, hat jedoch ihre eigenen Probleme, wie wir später in den Ausführungen sehen werden

Anstatt zuzulassen, dass das Programm die aktuelle Zeit als Teil der Berechnung für die erste Startnummer verwendet, können Sie die ursprüngliche Startnummer angeben. Dies kann ein beliebiges 32-Bit-Integer-Literal, eine Konstante oder eine Variable sein. Beispiele finden Sie unten. Dies bedeutet, dass Ihre Instanz dieselbe Folge von Pseudozufallszahlen generiert, was in bestimmten Situationen hilfreich sein kann.

```
Dim rng As New Random(43352)
```

oder

```
Dim rng As New Random(x)
```

Dabei wurde `x` anderer Stelle in Ihrem Programm als Integer-Konstante oder -Variable deklariert.

Generieren Sie eine Zufallszahl aus einer Zufallsinstanz

Das folgende Beispiel deklariert eine neue Instanz der Random-Klasse und verwendet dann die Methode `.Next` , um die nächste Zahl in der Folge von Pseudozufallszahlen zu generieren.

```
Dim rnd As New Random
Dim x As Integer
x = rnd.Next
```

In der letzten Zeile wird die nächste Pseudozufallszahl generiert und `x` zugewiesen. Diese Nummer liegt im Bereich von 0 - 2147483647. Sie können jedoch auch den zu generierenden Nummernbereich angeben, wie im folgenden Beispiel gezeigt.

```
x = rnd.Next(15, 200)
```

Beachten Sie jedoch, dass bei Verwendung dieser Parameter der Nummernbereich zwischen 15 oder mehr und 199 oder darunter liegt.

Sie können auch Fließkommazahlen vom Typ Double erzeugen, indem `.NextDouble` zB `.NextDouble`

```
Dim rnd As New Random
Dim y As Double
y = rnd.NextDouble()
```

Sie können jedoch keinen Bereich dafür angeben. Sie liegt immer im Bereich von 0,0 bis weniger als 1,0.

Zufällig online lesen: https://riptutorial.com/de/vb-net/topic/10128/zufallig

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit Visual Basic .NET Language	Bjørn-Roger Kringsjå, Cary Bondoc, Community, Happypig375, Harjot, Jonathan Nixon, Martin Verjans, MDTech.us_MAN, Misaz, Natalie Orsi, Nico Agusta, Ryan Thomas, StardustGogeta, Virtual Anomaly
2	Anweisung verwenden	VV5198722
3	Array	BunkerMentality, djv, Drarig29, Luke Sheppard, Mark Hurd, MatVAD, Robert Columbia, Ryan Thomas, Sam Axe, Sehnsucht, Steven Doggart, TuxCopter, vbnet3d, VortixDev, zyabin101
4	Aufgabenbasiertes asynchrones Muster	Stefano d'Antonio
5	AxWindowsMediaPlayer in VB.Net verwenden	Berken Usar
6	BackgroundWorker	Jones Joseph, Shayan Toqraee
7	BackgroundWorker verwenden	MatVAD
8	Bedingungen	Allen Binuya, Chetan Sanghani, Nathan Tuggy, Robert Columbia
9	ByVal- und ByRef-Schlüsselwörter	Adam Zuckerman, Misaz, Sehnsucht
10	Datei- / Ordner-Komprimierung	Cody Gray, MatVAD, Misaz, vbnet3d
11	Dateibehandlung	Dan Granger, Luke Sheppard, Matt Wilko, Misaz, Shayan Toqraee, vbnet3d
12	Datenzugriff	MatVAD, Mike Robertson, Nico Agusta
13	Datum	Matt Wilko, Misaz
14	Debuggen Ihrer Anwendung	Martin Verjans
15	Einfädeln	BiscuitBaker, Stefano d'Antonio, Visual Vincent

16	Einführung in die Syntax	Bugs, Mark Hurd, Martin Verjans, mnoronha, Nat G., Nico Agusta, Sehnsucht
17	Einweggegenstände	KE0GSD, Mark Hurd, Martin Verjans, Matt Wilko, Misaz, Mithrandir, Sam Axe
18	Enum	4444, CiccioRocca, David Sdot, djv, ElektroStudios, kodkod, LogicalFlaps, Shog9, Steven Doggart, void
19	Erweiterungsmethoden	Fütemire, InteXX, Matt Wilko, Sam Axe, Stefano d'Antonio, void
20	Fehlerbehandlung	Adam Zuckerman, Bjørn-Roger Kringsjå, Happypig375, Luke Sheppard, MatVAD, Nico Agusta, Vishal
21	FTP-Server	Misaz
22	Funktionen	Berken Usar
23	GDI +	Dman
24	Generics	JDC
25	Google Maps in einem Windows-Formular	anonymous, Carlos Borau
26	Klassen	Darren Davies, Harjot
27	Komprimierte Textdatei im Handumdrehen lesen	Proger_Cbsk
28	Konsole	Bugs, InteXX, lucamauri, Martin Soles, Matthew Whited, Sam Axe, Slava Auer, StardustGogeta, vbnet3d, VortixDev
29	Kurzschlussoperatoren (AndAlso - OrElse)	Bart Jolling, CiccioRocca, Kendra, Sam Axe
30	LINQ	Dan Drews, Daz, Derek Tomes, Fütemire, H. Pauwelyn, Mark Hurd, Misaz, Sam Axe, Sehnsucht, Zev Spitz
31	Listen	Dman, DrDonut, Fütemire, Luke Sheppard, Robert Columbia, Seandk
32	Looping	CiccioRocca, debater, Imran Ali Khan, Mark, MatVAD, Sam Axe, Scott Mitchell, SilverShotBee, TyCobb, vbnet3d, void
33	Mit Windows Forms arbeiten	djv, SilverShotBee, vbnet3d
34	Multithreading	4444, Daz, MatVAD

35	NullReferenceException	Alessandro Mascolo, RamenChef, Sehnsucht
36	OOP-Schlüsselwörter	4444, David, JDC, Matt Wilko, Nat G.
37	Operatoren	Bjørn-Roger Kringsjå, Cary Bondoc, MatVAD, Mike Robertson, Pasilda, Robert Columbia, RoyalPotato, Sam Axe , Sree, varocarbas, void
38	Option explizit	Happypig375, Matt Wilko, sansknwoledge
39	Option Infer	Alex B., LogicalFlaps, Mark Hurd
40	Option Strict	Andrew Morton, Cary Bondoc, Matt Wilko, RussAwesome, Sam Axe, vbnet3d
41	Reflexion	Axarydax, Matt, Sam Axe, void
42	Rekursion	Robert Columbia
43	Typumwandlung	Cary Bondoc, LogicalFlaps, vbnet3d
44	Unit-Test in VB.NET	wbadry
45	Variablen deklarieren	Cody Gray, Darren Davies, Fütemire, glaubergft, keronconk, LogicalFlaps, MatVAD, RamenChef, Sehnsucht
46	Verbindungsabwicklung	Jonas_Hess
47	Visual Basic 14.0-Funktionen	Adam Zuckerman, Bjørn-Roger Kringsjå, Blackwood, Crazy Britt, Drake, Fütemire, Gridly, jColeson, liserdarts, Matt Wilko , Nadeem_MK, Nitram, Sam Axe, Stefano d'Antonio, yummypasta
48	WinForms SpellCheckBox	Nathan
49	Wörterbücher	DrDonut, Nathan, Proger_Cbsk, Sehnsucht, void
50	WPF-XAML-Datenbindung	Milliron X
51	Zufällig	David Wilson