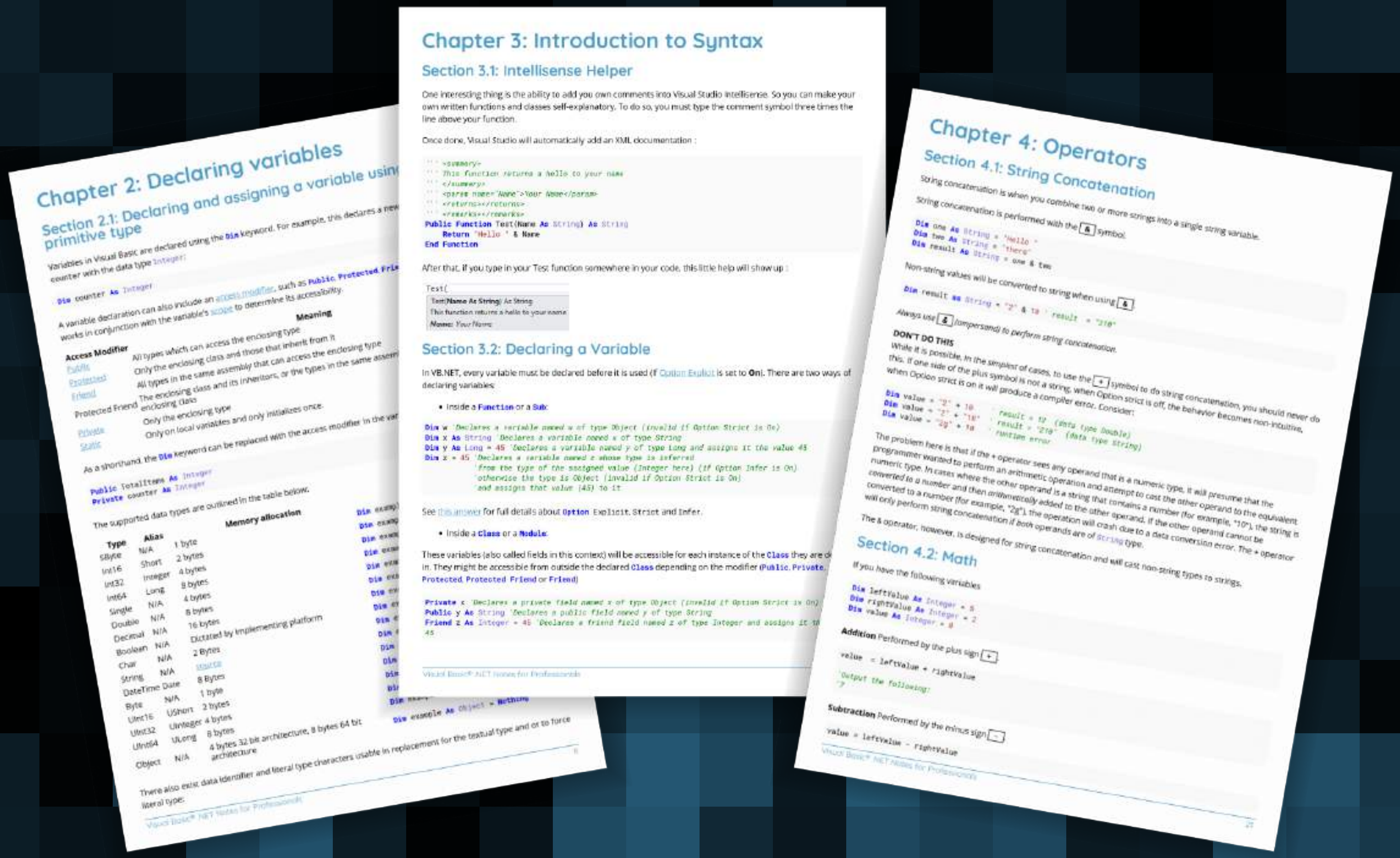
Visual Basic .NET

专业人士笔记

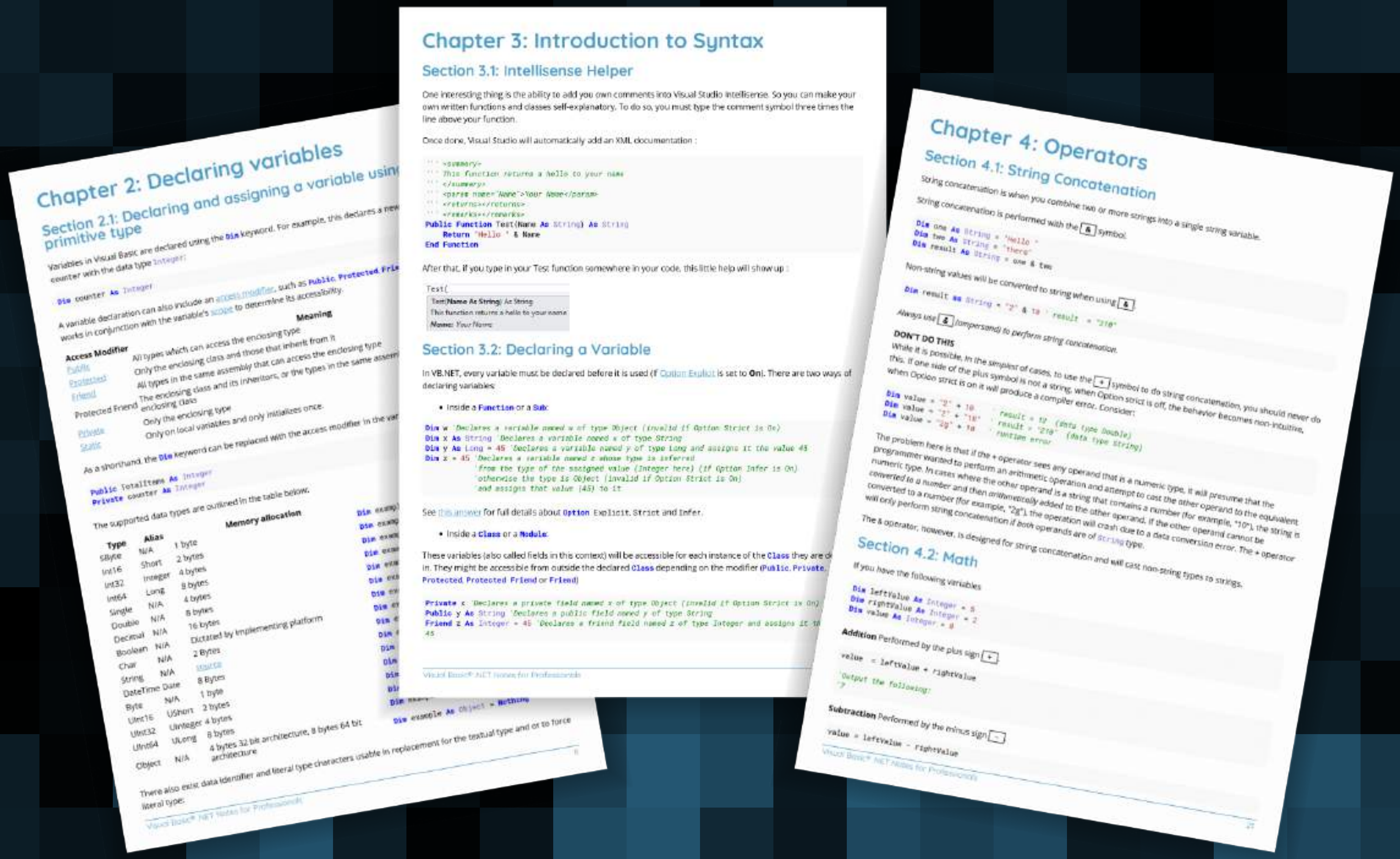
Visual Basic® .NET 专业人士笔记



100+ 页

专业提示和技巧

Visual Basic® .NET Notes for Professionals



100+ pages

of professional hints and tricks

目录

关于 . 1
第1章：Visual Basic .NET语言入门 . 2
第1.1节：你好，世界 . 2
第1.2节：点击按钮时在文本框中显示“Hello World” . 2
第1.3节：区域 . 3
第1.4节：创建一个简单的计算器以熟悉界面和代码 . 4
第2章：声明变量 . 8
第2.1节：使用原始类型声明和赋值变量 . 8
第2.2节：声明级别——局部变量和成员变量 . 10
第2.3节：访问修饰符示例 . 11
第3章：语法介绍 . 14
第3.1节：智能感知助手 . 14
第3.2节：声明变量 . 14
第3.3节：注释 . 15
第3.4节：修饰符 . 15
第3.5节：对象初始化器 . 16
第3.6节：集合初始化器 . 17
第3.7节：编写函数 . 19
第4章：运算符 . 21
第4.1节：字符串连接 . 21
第4.2节：数学 . 21
第4.3节：赋值 . 22
第4.4节：比较 . 23
第4.5节：按位操作 . 23
第5章：条件 . 25
第5.1节：If运算符 . 25
第5.2节：IF...Then...Else . 25
第6章：短路运算符（AndAlso - OrElse） . 27
第6.1节：OrElse的用法 . 27
第6.2节：AndAlso的用法 . 27
第6.3节：避免空引用异常（NullReferenceException） . 27
第7章：日期 . 30
第7.1节：将字符串转换（解析）为日期 . 30
第7.2节：将日期转换为字符串 . 30
第8章：数组 . 31
第8.1节：数组定义 . 31
第8.2节：空数组变量 . 31
第8.3节：数组初始化 . 32
第8.4节：声明一维数组并设置数组元素值 . 32
第8.5节：锯齿状数组初始化 . 32
第8.6节：非零下界 . 32
第8.7节：从两个变量引用同一数组 . 33
第8.8节：多维数组初始化 . 33
第9章：列表 . 34
第9.1节：向列表添加项目 . 34
第9.2节：检查列表中是否存在项目 . 34

Contents

About 1
Chapter 1: Getting started with Visual Basic .NET Language 2
Section 1.1: Hello World 2
Section 1.2: Hello World on a Textbox upon Clicking of a Button 2
Section 1.3: Region 3
Section 1.4: Creating a simple Calculator to get familiar with the interface and code 4
Chapter 2: Declaring variables 8
Section 2.1: Declaring and assigning a variable using a primitive type 8
Section 2.2: Levels of declaration – Local and Member variables 10
Section 2.3: Example of Access Modifiers 11
Chapter 3: Introduction to Syntax 14
Section 3.1: Intellisense Helper 14
Section 3.2: Declaring a Variable 14
Section 3.3: Comments 15
Section 3.4: Modifiers 15
Section 3.5: Object Initializers 16
Section 3.6: Collection Initializer 17
Section 3.7: Writing a function 19
Chapter 4: Operators 21
Section 4.1: String Concatenation 21
Section 4.2: Math 21
Section 4.3: Assignment 22
Section 4.4: Comparison 23
Section 4.5: Bitwise 23
Chapter 5: Conditions 25
Section 5.1: If operator 25
Section 5.2: IF...Then...Else 25
Chapter 6: Short-Circuiting Operators (AndAlso - OrElse) 27
Section 6.1: OrElse Usage 27
Section 6.2: AndAlso Usage 27
Section 6.3: Avoiding NullReferenceException 27
Chapter 7: Date 30
Section 7.1: Converting (Parsing) a String to a Date 30
Section 7.2: Converting a Date To A String 30
Chapter 8: Array 31
Section 8.1: Array definition 31
Section 8.2: Null Array Variables 31
Section 8.3: Array initialization 32
Section 8.4: Declare a single-dimension array and set array element values 32
Section 8.5: Jagged Array Initialization 32
Section 8.6: Non-zero lower bounds 32
Section 8.7: Referencing Same Array from Two Variables 33
Section 8.8: Multidimensional Array initialization 33
Chapter 9: Lists 34
Section 9.1: Add items to a List 34
Section 9.2: Check if item exists in a List 34

第9.3节：遍历列表中的项目 . 34
第9.4节：创建列表 . 35
第9.5节：从列表中移除项目 . 36
第9.6节：从列表中获取项目 . 36
第10章：枚举 . 38
第10.1节：GetNames() . 38
第10.2节：HasFlag() . 38
第10.3节：枚举定义 . 39
第10.4节：成员初始化 . 39
第10.5节：Flags属性 . 39
第10.6节：GetValues() . 40
第10.7节：字符串解析 . 40
第10.8节：ToString() . 41
第10.9节：确定枚举是否指定了FlagsAttribute . 41
第10.10节：遍历每个标志（标志迭代） . 42
第10.11节：确定标志组合中标志的数量 . 42
第10.12节：查找枚举中最接近的值 . 43
第11章：字典 . 45
第11.1节：创建一个填充了值的字典 . 45
第11.2节：遍历字典并打印所有条目 . 45
第11.3节：检查键是否已在字典中——数据简化 . 45
第11.4节：获取字典的值 . 46
第12章：循环 . 47
第12.1节：For...Next . 47
第12.2节：For Each...Next循环，用于遍历集合中的项目 . 48
第12.3节：短路 . 48
第12.4节：While循环，在某条件为真时迭代 . 50
第12.5节：嵌套循环 . 50
第12.6节：Do...循环 . 51
第13章：文件处理 . 53
第13.1节：向文件写入数据 . 53
第13.2节：读取文件的全部内容 . 53
第13.3节：使用StreamWriter逐行写入文本文件 . 53
第14章：文件/文件夹压缩 . 54
第14.1节：向项目添加文件压缩 . 54
第14.2节：从目录创建zip压缩包 . 54
第14.3节：将zip压缩包解压到目录 . 54
第14.4节：动态创建zip压缩包 . 54
第15章：连接处理 . 55
第15.1节：公共连接属性 . 55
第16章：数据访问 . 56
第16.1节：从数据库读取字段 . 56
第16.2节：从数据库读取并返回为数据表的简单函数 . 57
第17章：类型转换 . 58
第17.1节：将文本框中的文本转换为整数 . 58
第18章：ByVal 和 ByRef 关键字 . 59
第18.1节：ByRef 关键字 . 59
第18.2节：ByVal 关键字 . 59
第19章：控制台 . 61

Section 9.3: Loop through items in list 34
Section 9.4: Create a List 35
Section 9.5: Remove items from a List 36
Section 9.6: Retrieve items from a List 36
Chapter 10: Enum 38
Section 10.1: GetNames() 38
Section 10.2: HasFlag() 38
Section 10.3: Enum definition 39
Section 10.4: Member initialization 39
Section 10.5: The Flags attribute 39
Section 10.6: GetValues() 40
Section 10.7: String parsing 40
Section 10.8: ToString() 41
Section 10.9: Determine whether a Enum has FlagsAttribute specified or not 41
Section 10.10: For-each flag (flag iteration) 42
Section 10.11: Determine the amount of flags in a flag combination 42
Section 10.12: Find the nearest value in a Enum 43
Chapter 11: Dictionaries 45
Section 11.1: Create a dictionary filled with values 45
Section 11.2: Loop through a dictionary and print all entries 45
Section 11.3: Checking for key already in dictionary - data reduction 45
Section 11.4: Getting a dictionary value 46
Chapter 12: Looping 47
Section 12.1: For...Next 47
Section 12.2: For Each...Next loop for looping through collection of items 48
Section 12.3: Short Circuiting 48
Section 12.4: While loop to iterate while some condition is true 50
Section 12.5: Nested Loop 50
Section 12.6: Do...Loop 51
Chapter 13: File Handling 53
Section 13.1: Write Data to a File 53
Section 13.2: Read All Contents of a File 53
Section 13.3: Write Lines Individually to a Text File using StreamWriter 53
Chapter 14: File/Folder Compression 54
Section 14.1: Adding File Compression to your project 54
Section 14.2: Creating zip archive from directory 54
Section 14.3: Extracting zip archive to directory 54
Section 14.4: Create zip archive dynamicaly 54
Chapter 15: Connection Handling 55
Section 15.1: Public connection property 55
Chapter 16: Data Access 56
Section 16.1: Read field from Database 56
Section 16.2: Simple Function to read from Database and return as DataTable 57
Chapter 17: Type conversion 58
Section 17.1: Converting Text of The Textbox to an Integer 58
Chapter 18: ByVal and ByRef keywords 59
Section 18.1: ByRef keyword 59
Section 18.2: ByVal keyword 59
Chapter 19: Console 61

第19.1节：Console.ReadLine() . 61
第19.2节：Console.Read() . 61
第19.3节：Console.ReadKey() . 61
第19.4节：命令行提示符原型 . 61
第19.5节：Console.WriteLine() . 62
第20章：函数 . 63
第20.1节：定义函数 . 63
第20.2节：定义函数 #2 . 63
第21章：递归 . 64
第21.1节：计算第n个斐波那契数 . 64
第22章：随机数 . 65
第22.1节：声明一个实例 . 65
第22.2节：从Random实例生成随机数 . 65
第23章：类 . 67
第23.1节：抽象类 . 67
第23.2节：创建类 . 67
第24章：泛型 . 69
第24.1节：创建泛型类 . 69
第24.2节：泛型类的实例 . 69
第24.3节：定义“泛型”类 . 69
第24.4节：使用泛型类 . 69
第24.5节：限制给定的可能类型 . 70
第24.6节：创建给定类型的新实例 . 70
第25章：可释放对象 . 71
第25.1节：IDisposable的基本概念 . 71
第25.2节：在一个Using中声明多个对象 . 71
第26章：空引用异常 . 73
第26.1节：空返回 . 73
第26.2节：未初始化变量 . 73
第27章：Using语句 . 74
第27.1节：参见可释放对象下的示例 . 74
第28章：Option Strict . 75
第28.1节：为什么使用它？ . 75
第28.2节：如何开启它 . 75
第29章：Option Explicit . 77
第29.1节：它是什么？ . 77
第29.2节：如何开启它？ . 77
第30章：Option Infer . 78
第30.1节：如何启用/禁用它 . 78
第30.2节：它是什么？ . 78
第30.3节：何时使用类型推断 . 79
第31章：错误处理 81
第31.1节：Try...Catch...Finally语句 . 81
第31.2节：创建自定义异常并抛出 . 81
第31.3节：数据库操作中的Try Catch . 82
第31.4节：无法捕获的异常 . 82
第31.5节：关键异常 . 82
第32章：面向对象编程关键字 . 84

Section 19.1: Console.ReadLine() 61
Section 19.2: Console.Read() 61
Section 19.3: Console.ReadKey() 61
Section 19.4: Prototype of command line prompt 61
Section 19.5: Console.WriteLine() 62
Chapter 20: Functions 63
Section 20.1: Defining a Function 63
Section 20.2: Defining a Function #2 63
Chapter 21: Recursion 64
Section 21.1: Compute nth Fibonacci number 64
Chapter 22: Random 65
Section 22.1: Declaring an instance 65
Section 22.2: Generate a random number from an instance of Random 65
Chapter 23: Classes 67
Section 23.1: Abstract Classes 67
Section 23.2: Creating classes 67
Chapter 24: Generics 69
Section 24.1: Create a generic class 69
Section 24.2: Instance of a Generic Class 69
Section 24.3: Define a 'generic' class 69
Section 24.4: Use a generic class 69
Section 24.5: Limit the possible types given 70
Section 24.6: Create a new instance of the given type 70
Chapter 25: Disposable objects 71
Section 25.1: Basic concept of IDisposable 71
Section 25.2: Declaring more objects in one Using 71
Chapter 26: NullReferenceException 73
Section 26.1: Empty Return 73
Section 26.2: Uninitialized variable 73
Chapter 27: Using Statement 74
Section 27.1: See examples under Disposable objects 74
Chapter 28: Option Strict 75
Section 28.1: Why Use It? 75
Section 28.2: How to Switch It On 75
Chapter 29: Option Explicit 77
Section 29.1: What is it? 77
Section 29.2: How to switch it on? 77
Chapter 30: Option Infer 78
Section 30.1: How to enable/disable it 78
Section 30.2: What is it? 78
Section 30.3: When to use type inference 79
Chapter 31: Error Handling 81
Section 31.1: Try...Catch...Finally Statement 81
Section 31.2: Creating custom exception and throwing 81
Section 31.3: Try Catch in Database Operation 82
Section 31.4: The Un-catchable Exception 82
Section 31.5: Critical Exceptions 82
Chapter 32: OOP Keywords 84

第32.1节：定义类 . 84
第32.2节：继承修饰符（针对类） . 84
第32.3节：继承修饰符（针对属性和方法） . 85
第32.4节：MyBase . 86
第32.5节：Me与MyClass . 87
第32.6节：重载 . 88
第32.7节：遮蔽 . 88
第32.8节：接口 . 90
第33章：扩展方法 . 91
第33.1节：创建扩展方法 . 91
第33.2节：使用扩展方法使语言更具函数式 . 91
第33.3节：从强名称获取程序集版本 . 91
第33.4节：数字填充 . 92
第34章：反射 . 94
第34.1节：获取类实例的属性 . 94
第34.2节：获取方法并调用它 . 94
第34.3节：创建泛型类型的实例 . 94
第34.4节：获取类型的成员 . 94
第35章：Visual Basic 14.0 特性 . 96
第35.1节：空条件运算符 . 96
第35.2节：字符串插值 . 96
第35.3节：只读自动属性 . 97
第35.4节：NameOf 运算符 . 97
第35.5节：多行字符串字面量 . 98
第35.6节：部分模块和接口 . 98
第35.7节：隐式换行后的注释 . 99
第35.8节：#Region 指令改进 . 99
第36章：LINQ . 101
第36.1节：带简单条件的数组选择 . 101
第36.2节：通过Select子句映射数组 . 101
第36.3节：排序输出 . 101
第36.4节：从IEnumerable生成字典 . 101
第36.5节：投影 . 102
第36.6节：获取不同值（使用Distinct方法） . 102
第37章：FTP服务器 . 103
第37.1节：从FTP服务器下载文件 . 103
第37.2节：登录时从FTP服务器下载文件 . 103
第37.3节：上传文件到FTP服务器 . 103
第37.4节：登录时上传文件到FTP服务器 . 103
第38章：使用Windows窗体 . 104
第38.1节：使用默认的表单实例 . 104
第38.2节：在表单之间传递数据 . 104
第39章：Windows窗体中的谷歌地图 . 106
第39.1节：如何在Windows窗体中使用谷歌地图 . 106
第40章：GDI+ . 115
第40.1节：绘制图形 . 115
第40.2节：填充图形 . 115
第40.3节：文本 . 116
第40.4节：创建图形对象 . 116

Section 32.1: Defining a class 84
Section 32.2: Inheritance Modifiers (on classes) 84
Section 32.3: Inheritance Modifiers (on properties and methods) 85
Section 32.4: MyBase 86
Section 32.5: Me vs MyClass 87
Section 32.6: Overloading 88
Section 32.7: Shadows 88
Section 32.8: Interfaces 90
Chapter 33: Extension methods 91
Section 33.1: Creating an extension method 91
Section 33.2: Making the language more functional with extension methods 91
Section 33.3: Getting Assembly Version From Strong Name 91
Section 33.4: Padding Numerics 92
Chapter 34: Reflection 94
Section 34.1: Retrieve Properties for an Instance of a Class 94
Section 34.2: Get a method and invoke it 94
Section 34.3: Create an instance of a generic type 94
Section 34.4: Get the members of a type 94
Chapter 35: Visual Basic 14.0 Features 96
Section 35.1: Null conditional operator 96
Section 35.2: String interpolation 96
Section 35.3: Read-Only Auto-Properties 97
Section 35.4: NameOf operator 97
Section 35.5: Multiline string literals 98
Section 35.6: Partial Modules and Interfaces 98
Section 35.7: Comments after implicit line continuation 99
Section 35.8: #Region directive improvements 99
Chapter 36: LINQ 101
Section 36.1: Selecting from array with simple condition 101
Section 36.2: Mapping array by Select clause 101
Section 36.3: Ordering output 101
Section 36.4: Generating Dictionary From IEnumerable 101
Section 36.5: Projection 102
Section 36.6: Getting distinct values (using the Distinct method) 102
Chapter 37: FTP server 103
Section 37.1: Download file from FTP server 103
Section 37.2: Download file from FTP server when login required 103
Section 37.3: Upload file to FTP server 103
Section 37.4: Upload file to FTP server when login required 103
Chapter 38: Working with Windows Forms 104
Section 38.1: Using the default Form instance 104
Section 38.2: Passing Data From One Form To Another 104
Chapter 39: Google Maps in a Windows Form 106
Section 39.1: How to use a Google Map in a Windows Form 106
Chapter 40: GDI+ 115
Section 40.1: Draw Shapes 115
Section 40.2: Fill Shapes 115
Section 40.3: Text 116
Section 40.4: Create Graphic Object 116

第41章：WinForms 拼写复选框 . 118
第41.1节：ElementHost WPF 文本框 . 118
第42章：在 VB.Net 中使用 axWindowsMediaPlayer . 122
第42.1节：添加 axWindowsMediaPlayer . 122
第42.2节：播放多媒体文件 . 123
第43章：WPF XAML 数据绑定 . 124
第43.1节：在视图模型中将字符串绑定到视图中的文本框 . 124
第44章：动态读取压缩文本文件 . 126
第44.1节：逐行读取.gz文本文件 . 126
第45章：多线程 . 127
第45.1节：使用Control.Invoke()执行线程安全调用 . 127
第45.2节：使用Async/Await执行线程安全调用 . 127
第46章：多线程 . 129
第46.1节：使用Thread类进行多线程 . 129
第47章：BackgroundWorker . 131
第47.1节：使用BackgroundWorker . 131
第47.2节：在BackgroundWorker中访问GUI组件 . 132
第48章：使用BackgroundWorker . 133
第48.1节：后台工作者类的基本实现 . 133
第49章：基于任务的异步模式 . 134
第49.1节：Async/Await的基本用法 . 134
第49.2节：在LINQ中使用TAP . 134
第50章：调试你的应用程序 . 135
第50.1节：在控制台中调试 . 135
第50.2节：缩进你的调试输出 . 135
第50.3节：在文本文件中调试 . 136
第51章：VB.NET中的单元测试 . 137
第51.1节：税务计算的单元测试 . 137
第51.2节：测试员工类的赋值和派生属性 . 138
鸣谢 . 141
你可能也喜欢 . 143

Chapter 41: WinForms SpellCheckBox 118
Section 41.1: ElementHost WPF TextBox 118
Chapter 42: Using axWindowsMediaPlayer in VB.Net 122
Section 42.1: Adding the axWindowsMediaPlayer 122
Section 42.2: Play a Multimedia File 123
Chapter 43: WPF XAML Data Binding 124
Section 43.1: Binding a String in the ViewModel to a TextBox in the View 124
Chapter 44: Reading compressed textfile on-the-fly 126
Section 44.1: Reading .gz textfile line after line 126
Chapter 45: Threading 127
Section 45.1: Performing thread-safe calls using Control.Invoke() 127
Section 45.2: Performing thread-safe calls using Async/Await 127
Chapter 46: Multithreading 129
Section 46.1: Multithreading using Thread Class 129
Chapter 47: BackgroundWorker 131
Section 47.1: Using BackgroundWorker 131
Section 47.2: Accessing GUI components in BackgroundWorker 132
Chapter 48: Using BackgroundWorker 133
Section 48.1: Basic implementation of Background worker class 133
Chapter 49: Task-based asynchronous pattern 134
Section 49.1: Basic usage of Async/Await 134
Section 49.2: Using TAP with LINQ 134
Chapter 50: Debugging your application 135
Section 50.1: Debug in the console 135
Section 50.2: Indenting your debug output 135
Section 50.3: Debug in a text file 136
Chapter 51: Unit Testing in VB.NET 137
Section 51.1: Unit Testing for Tax Calculation 137
Section 51.2: Testing Employee Class assigned and derived Properties 138
Credits 141
You may also like 143

关于

About

第1章：开始使用Visual Basic .NET语言

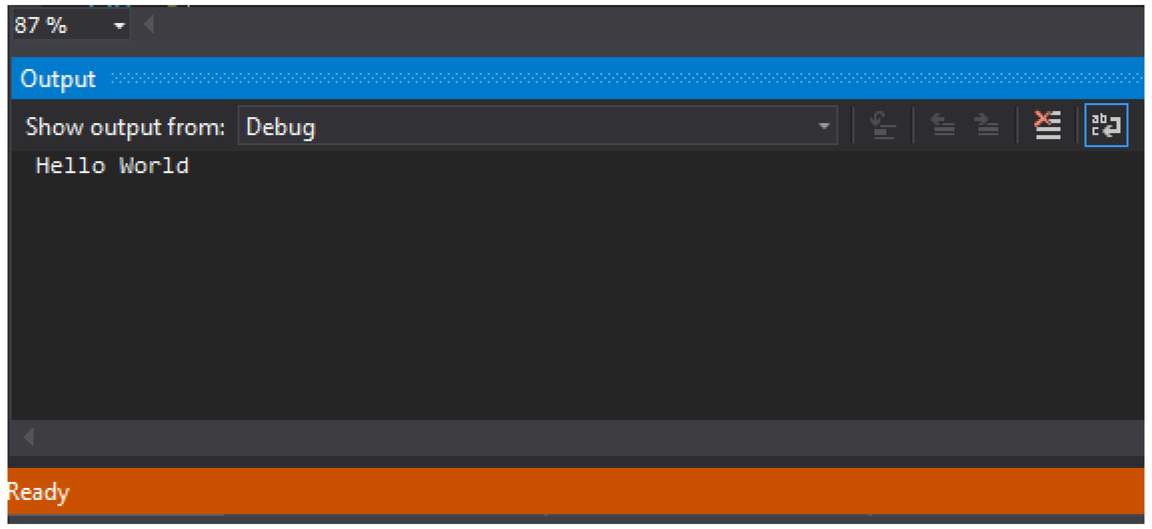
VB.NET 版本	Visual Studio 版本	.NET 框架版本	发布日期
7.0	2002	1.0	2002-02-13
7.1	2003	1.1	2003-04-24
8.0	2005	2.0 / 3.0	2005-10-18
9.0	2008	3.5	2007-11-19
10.0	2010	4.0	2010-04-12
11.0	2012	4.5	2012-08-15
12.0	2013	4.5.1 / 4.5.2	2013-10-17
14.0	2015	4.6.0 ~ 4.6.2	2015-07-20
15.0	2017	4.7	2017-03-07

第1.1节：你好，世界

首先，安装一个版本的Microsoft Visual Studio，包括免费的Community版本。然后，创建一个类型为控制台应用程序的Visual Basic控制台应用程序项目，以下代码将把字符串“Hello World”打印到控制台：

```
模块 模块1

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

End Module
```

然后，保存并按下 F5 键盘上的键（或进入*调试*菜单，然后点击*无调试运行*或*运行*）来编译并运行程序。控制台窗口中应显示`'Hello World'`。



```
Hello World
```

第1.2节：点击按钮后在文本框中显示Hello World

拖动1个文本框和1个按钮

Chapter 1: Getting started with Visual Basic .NET Language

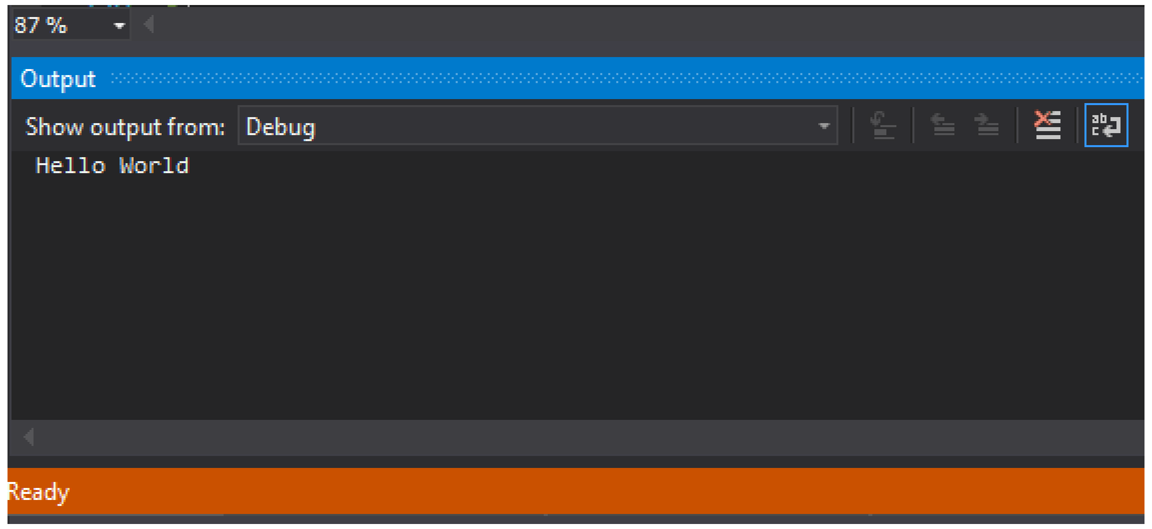
VB.NET Version	Visual Studio Version	.NET Framework Version	Release Date
7.0	2002	1.0	2002-02-13
7.1	2003	1.1	2003-04-24
8.0	2005	2.0 / 3.0	2005-10-18
9.0	2008	3.5	2007-11-19
10.0	2010	4.0	2010-04-12
11.0	2012	4.5	2012-08-15
12.0	2013	4.5.1 / 4.5.2	2013-10-17
14.0	2015	4.6.0 ~ 4.6.2	2015-07-20
15.0	2017	4.7	2017-03-07

Section 1.1: Hello World

First, install a version of Microsoft Visual Studio, including the free Community edition. Then, create a Visual Basic Console Application project of type *Console Application*, and the following code will print the string `'Hello World'` to the Console:

```
Module Module1

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

End Module
```

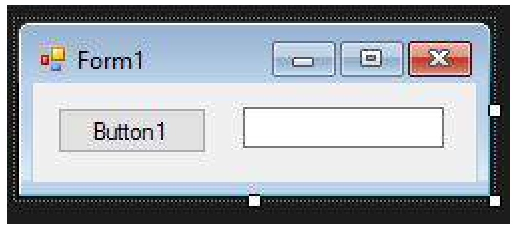
Then, save and press F5 on the keyboard (or go to the *Debug* menu, then click *Run without Debug* or *Run*) to compile and run the program. `'Hello World'` should appear in the console window.



```
Hello World
```

Section 1.2: Hello World on a Textbox upon Clicking of a Button

Drag 1 textbox and 1 button



双击button1，你将被转到Button1_Click事件

```
公共类 Form1
    私有子程序 Button1_Click(sender 作为 对象, e 作为 事件参数) 处理 Button1.Click

    结束子程序
结束类
```

输入你想要操作的对象名称，在本例中是textbox1。 .Text 是我们想要使用的属性，如果我们想在上面放置文本。

属性 Textbox.Text，获取或设置当前文本在文本框中。现在，我们有了Textbox1.Text我们需要设置该Textbox1.Text的值，所

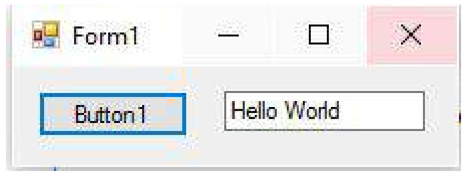
以我们将使用=符号。我们想放入Textbox1.Text的值是Hello World。总体来说，这就是将Hello World赋值给

Textbox1.Text

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

将该代码添加到button1的点击事件

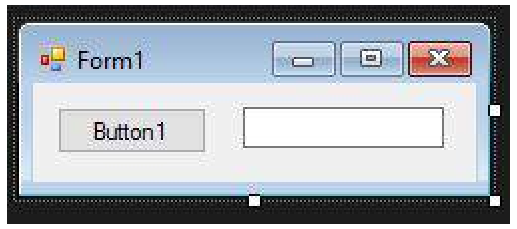
```
公共类 Form1
    私有子程序 Button1_Click(sender 作为 对象, e 作为 事件参数) 处理 Button1.Click
       TextBox1.Text = "Hello World"
    End Sub
结束类
```



第1.3节：地区

为了提高可读性，这对初学者阅读VB代码以及全职开发人员维护代码都很有帮助，我们可以使用“Region”来设置同一组事件、函数或变量的区域：

```
#Region "事件"
    受保护的子程序 txtPrice_TextChanged(...) 处理 txtPrice.TextChanged
        '在这里执行操作...
    End Sub

    受保护的子程序 txtTotal_TextChanged(...) 处理 txtTotal.TextChanged
        '在这里执行操作...
    End Sub

    '其他一些事件....

#End Region
```



Double click the button1 and you will be transferred to the `Button1_Click` **event**

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

    End Sub
End Class
```

Type the name of the object that you want to target, in our case it is the `textbox1`. `.Text` is the property that we want to use if we want to put a text on it.

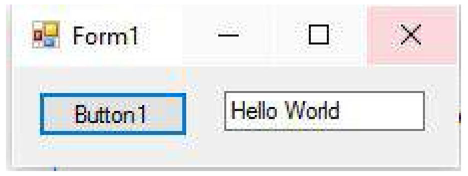
`Property Textbox.Text, gets or sets the current text in the TextBox`. Now, we have `Textbox1.Text`

We need to set the value of that `Textbox1.Text` so we will use the `=` sign. The value that we want to put in the `Textbox1.Text` is `Hello World`. Overall, this is the total code for putting a value of `Hello World` to the `Textbox1.Text`

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

Adding that code to the `clicked` **event** of `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
```

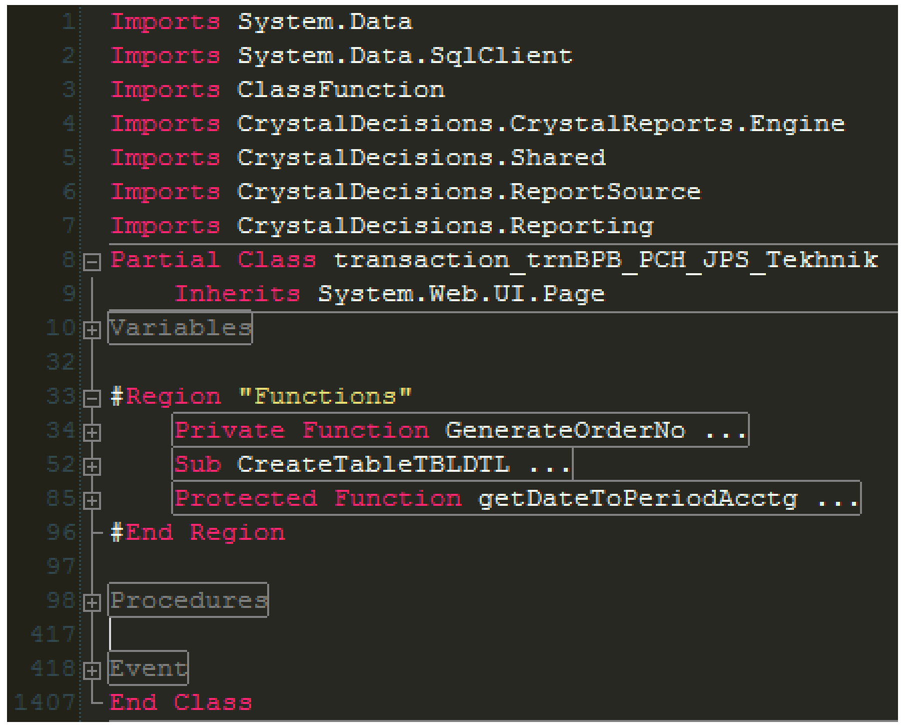


Section 1.3: Region

For the sake of readability, which will be useful for beginners when reading VB code as well for full time developers to maintain the code, we can use "Region" to set a region of the same set of events, functions, or variables:

```
#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
```

当代码行数超过1000时，这个区域块可以折叠，以获得视觉上的帮助。这也能节省你的滚动操作。

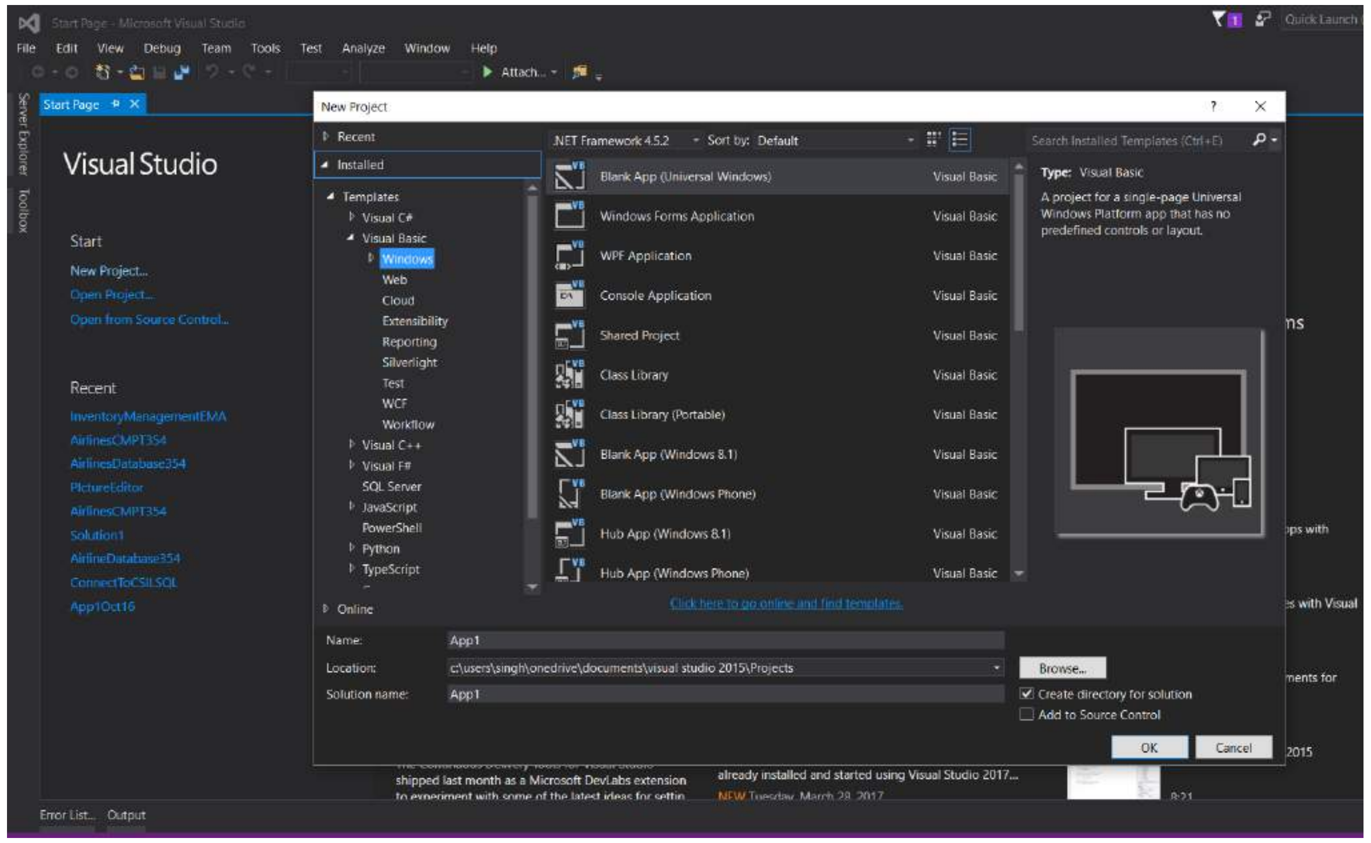


```
Imports System.Data
Imports System.Data.SqlClient
Imports ClassFunction
Imports CrystalDecisions.CrystalReports.Engine
Imports CrystalDecisions.Shared
Imports CrystalDecisions.ReportSource
Imports CrystalDecisions.Reporting
Partial Class transaction_trnBPB_PCH_JPS_Tekhnik
    Inherits System.Web.UI.Page
Variables

#Region "Functions"
    Private Function GenerateOrderNo ...
    Sub CreateTableTBLDTL ...
    Protected Function getDateToPeriodAcctg ...
#End Region

Procedures

Event
End Class
```

已在VS 2005、2008、2010、2015和2017上测试。

第1.4节：创建一个简单的计算器以熟悉界面和代码

1. 安装Visual Studio后，访问https://www.visualstudio.com/downloads/，开始一个新项目。

2. 

Start Page - Microsoft Visual Studio

New Project

Installed > Templates > Visual Basic > Windows

.NET Framework 4.5.2 · Sort by: Default

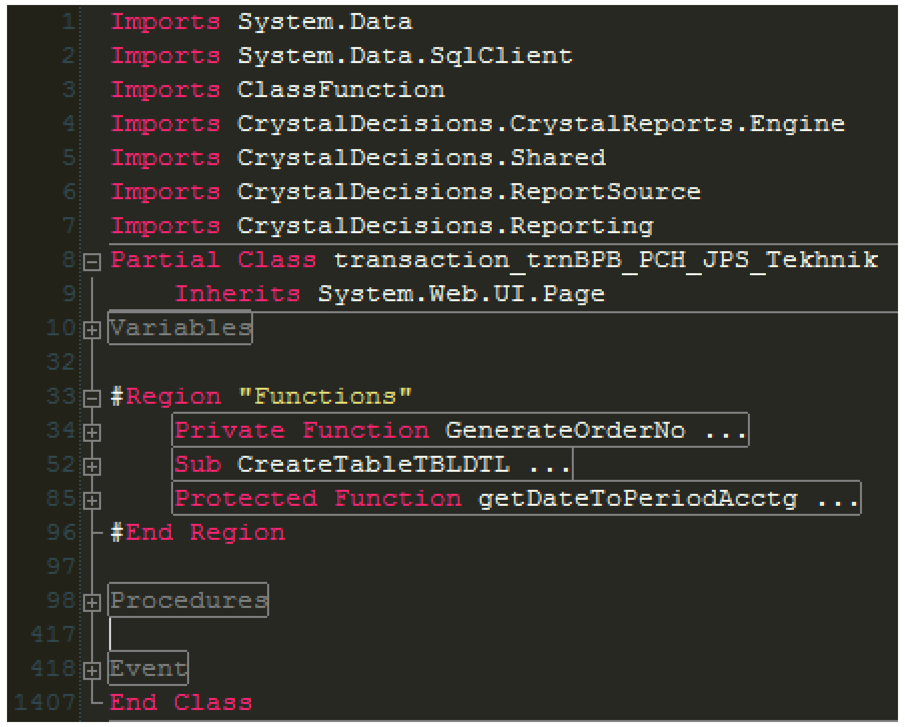
Blank App (Universal Windows)	Visual Basic
Windows Forms Application	Visual Basic
WPF Application	Visual Basic
Console Application	Visual Basic
Shared Project	Visual Basic
Class Library	Visual Basic
Class Library (Portable)	Visual Basic
Blank App (Windows 8.1)	Visual Basic
Blank App (Windows Phone)	Visual Basic
Hub App (Windows 8.1)	Visual Basic
Hub App (Windows Phone)	Visual Basic

Type: Visual Basic
A project for a single-page Universal Windows Platform app that has no predefined controls or layout.

Name: App1
Location: c:\users\singh\onedrive\documents\visual studio 2015\Projects
Solution name: App1
☑ Create directory for solution
☐ Add to Source Control

3. 从 Visual Basic 选项卡中选择“Windows 窗体应用程序”。如果需要，可以在此处重命名它。

This region block could be collapsed to gain some visual help when the code row goes to 1000+. It is also save your scroll efforts.

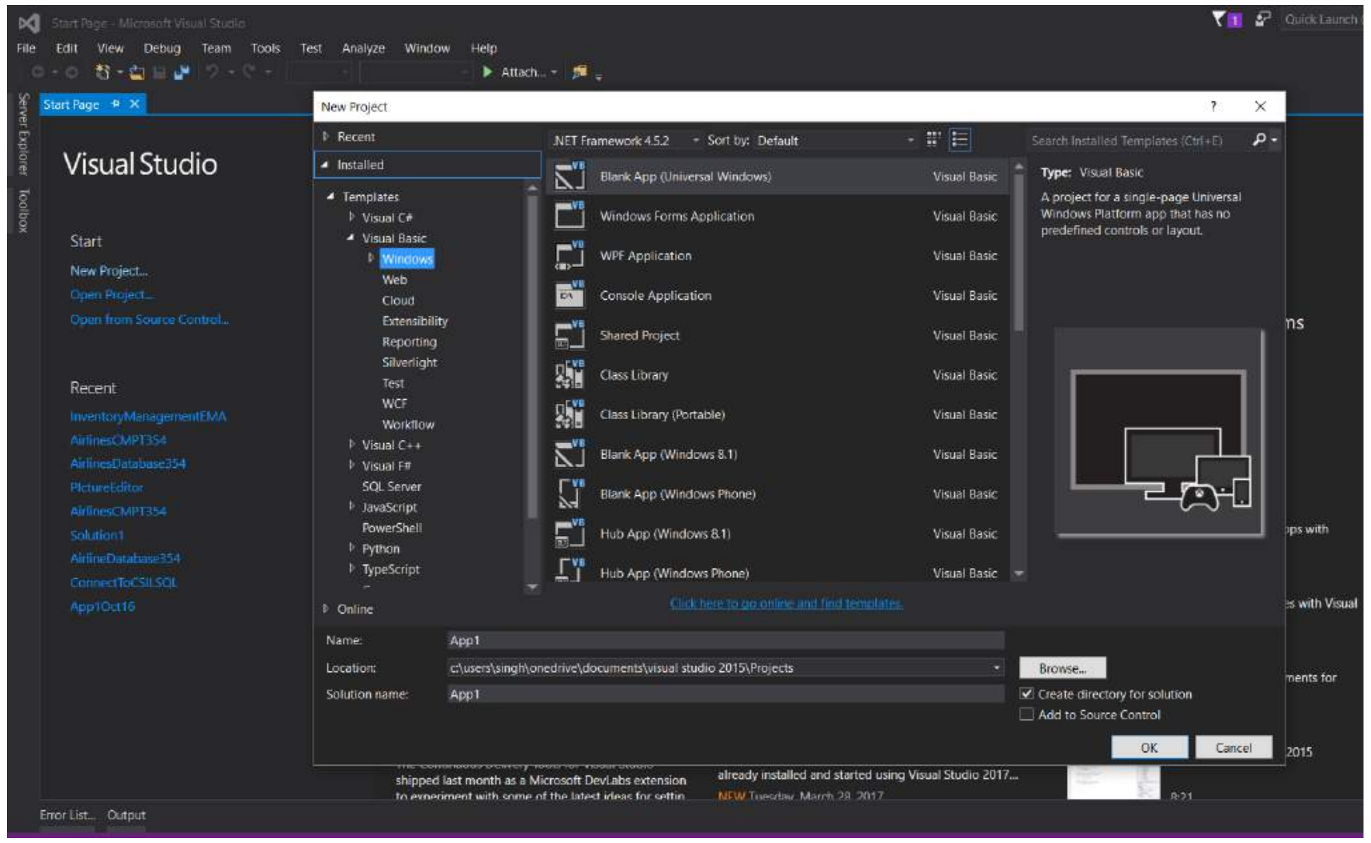


```
Imports System.Data
Imports System.Data.SqlClient
Imports ClassFunction
Imports CrystalDecisions.CrystalReports.Engine
Imports CrystalDecisions.Shared
Imports CrystalDecisions.ReportSource
Imports CrystalDecisions.Reporting
Partial Class transaction_trnBPB_PCH_JPS_Tekhnik
    Inherits System.Web.UI.Page
Variables

#Region "Functions"
    Private Function GenerateOrderNo ...
    Sub CreateTableTBLDTL ...
    Protected Function getDateToPeriodAcctg ...
#End Region

Procedures

Event
End Class
```

Tested on VS 2005, 2008 2010, 2015 and 2017.

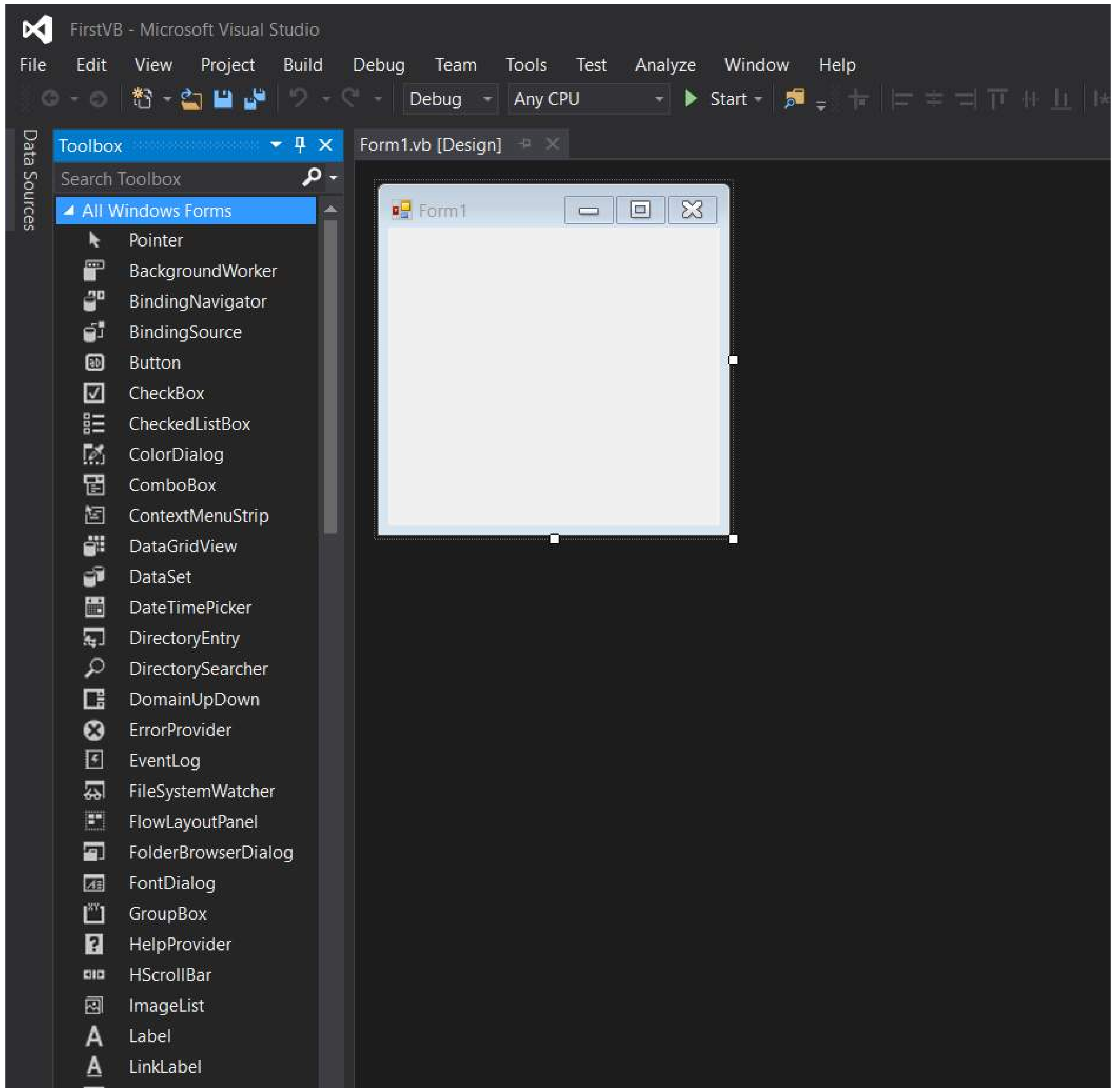
Section 1.4: Creating a simple Calculator to get familiar with the interface and code

1. Once you have installed Visual Studio from https://www.visualstudio.com/downloads/, start a new project.

2. 

3. Select 'Windows Forms Application' from Visual Basic Tab. You can rename it here if you need to.

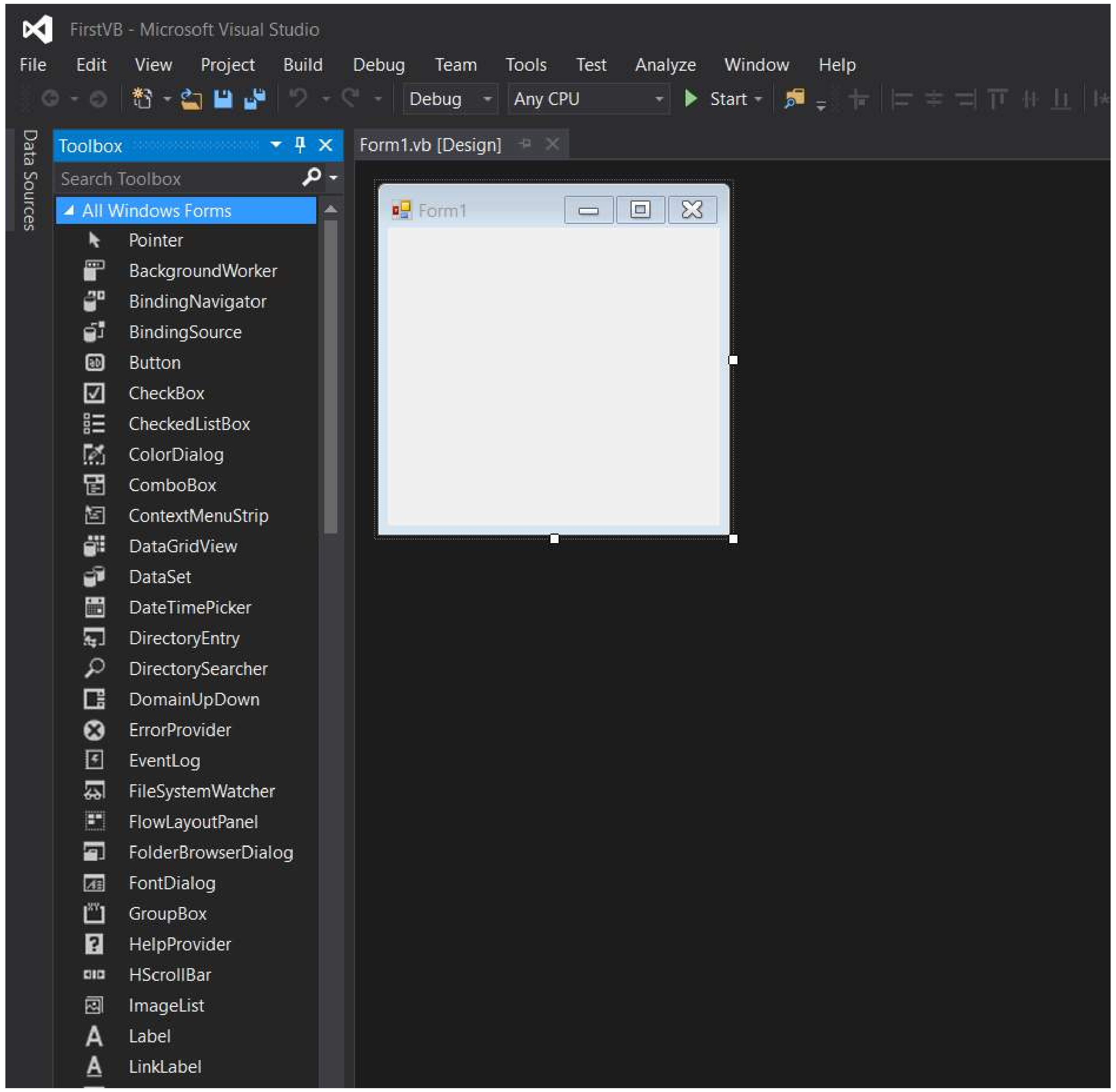
4. 点击“确定”后，你将看到此窗口：



5. 点击左侧的“工具箱”选项卡。工具栏默认启用了“自动隐藏”选项。要禁用此选项，请点击工具箱窗口右上角“向下箭头”符号和“×”符号之间的小图标。

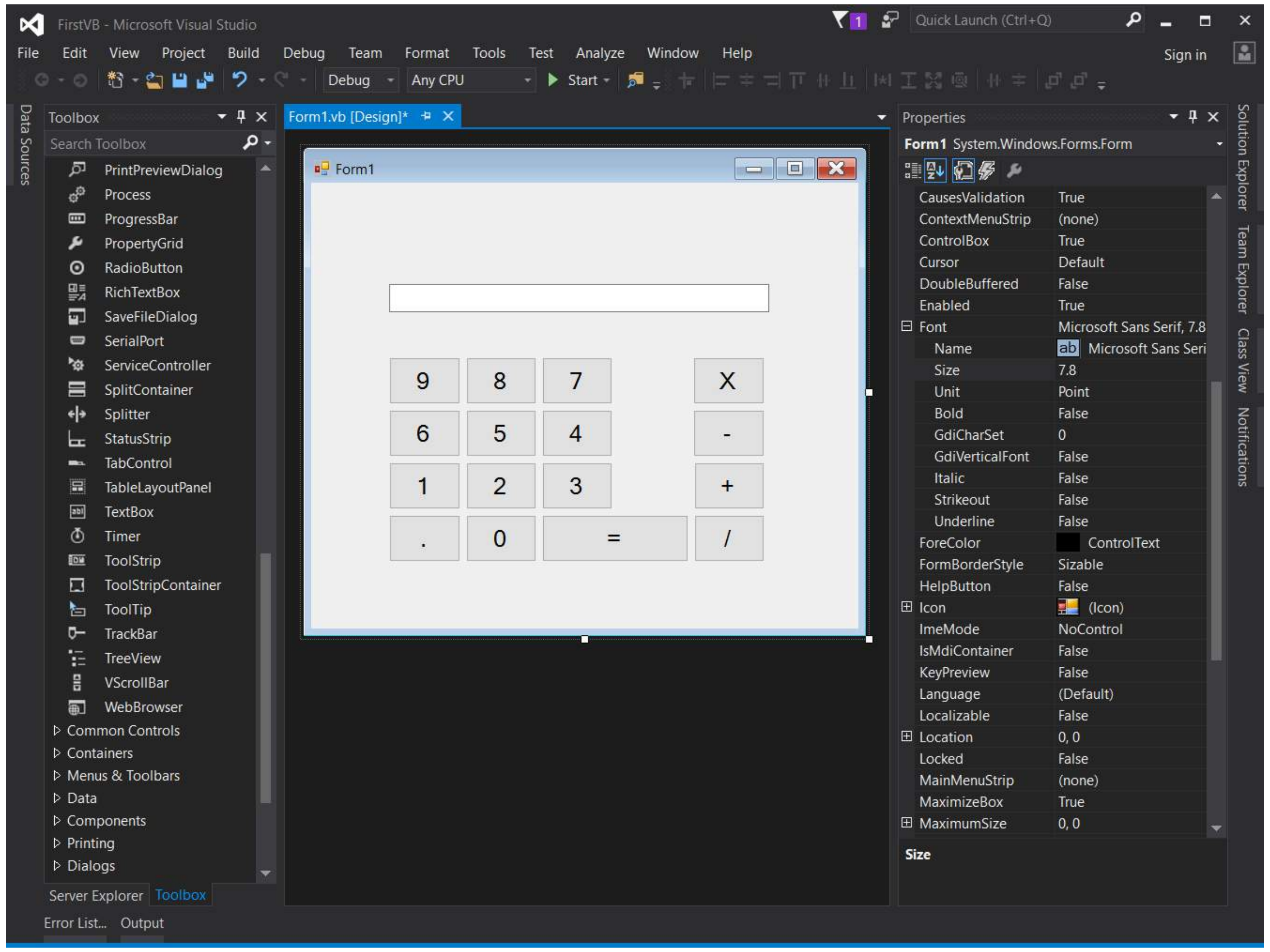
6. 熟悉工具箱中提供的工具。我使用按钮和文本框制作了一个计算器界面。

4. Once you click 'OK', you will see this window:



5. Click on the 'Toolbox' tab on the left. The toolbar has 'auto-hide' option enabled by default. To disable this option, click the small symbol between the 'down arrow' symbol and the 'x' symbol, on the top-right corner of Toolbox window.

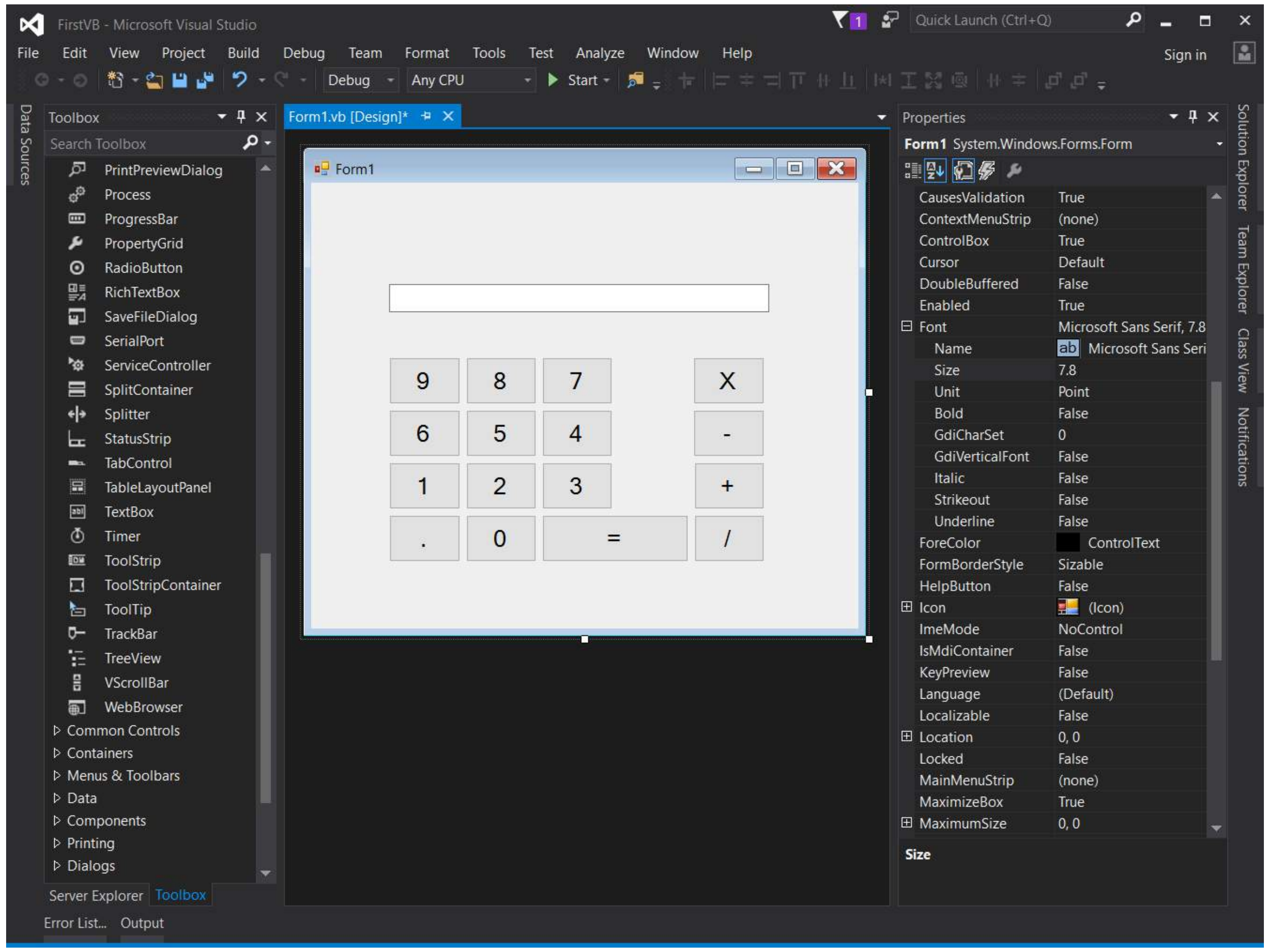
6. Get yourself familiar with the tools provided in the box. I have made a calculator interface by using buttons and a Textbox.


FirstVB - Microsoft Visual Studio
Toolbox
Form1.vb [Design]*
Form1
9
8
7
X
6
5
4
-
1
2
3
+
.
0
=
/
Properties
Form1 System.Windows.Forms.Form

CausesValidation	True
ContextMenuStrip	(none)
ControlBox	True
Cursor	Default
DoubleBuffered	False
Enabled	True
Font	Microsoft Sans Serif, 7.8
Name	Microsoft Sans Seri
Size	7.8
Unit	Point
Bold	False
GdiCharSet	0
GdiVerticalFont	False
Italic	False
Strikeout	False
Underline	False
ForeColor	ControlText
FormBorderStyle	Sizable
HelpButton	False
Icon	(Icon)
ImeMode	NoControl
IsMdiContainer	False
KeyPreview	False
Language	(Default)
Localizable	False
Location	0, 0
Locked	False
MainMenuStrip	(none)
MaximizeBox	True
MaximumSize	0, 0

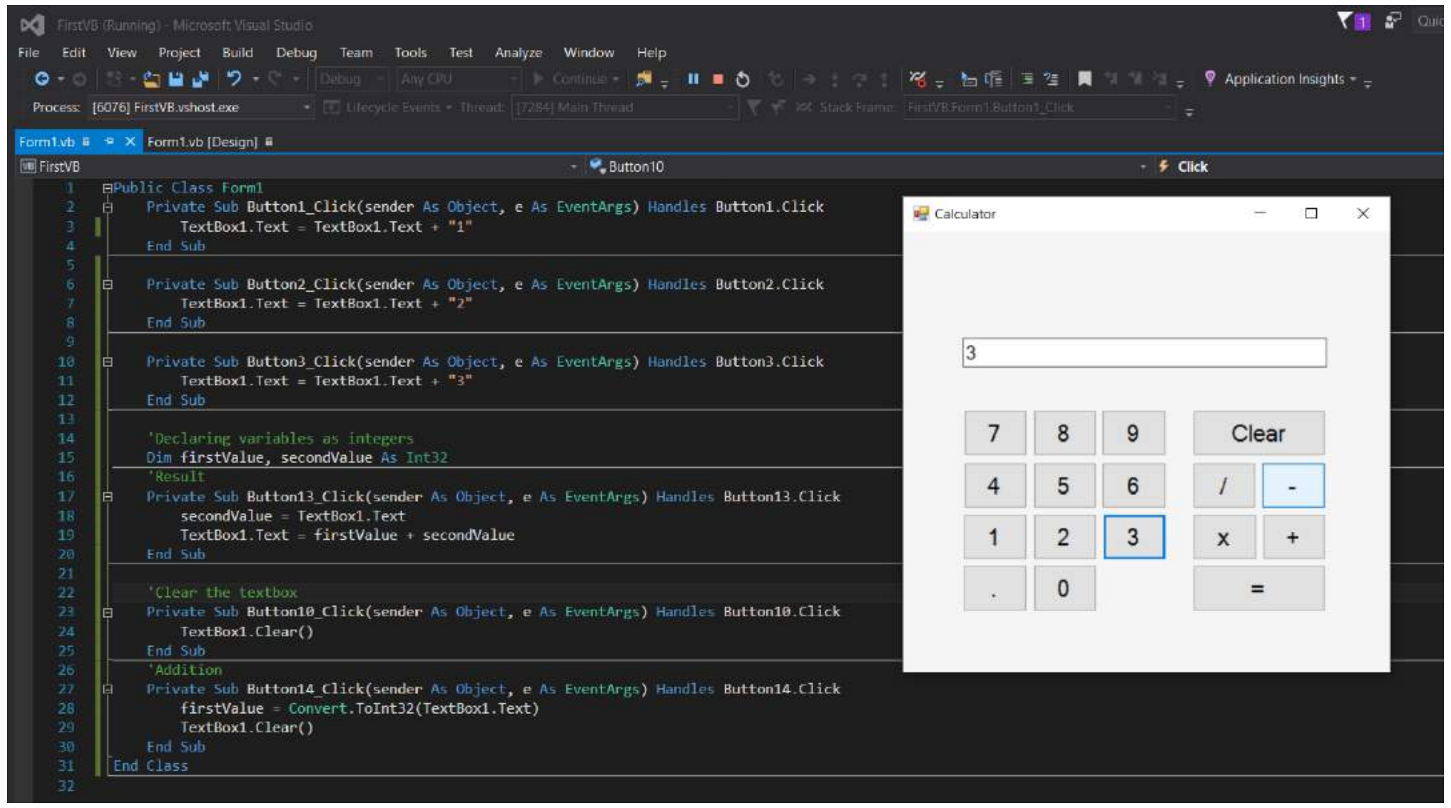
7. 点击属性选项卡（位于编辑器右侧）。你可以更改按钮的文本属性，以及用于重命名它们的文本框。Font属性可用于更改控件的字体。

8. 要为某个事件（例如点击按钮）编写具体操作，双击控件。代码窗口将会打开。



7. Click on the *Properties* tab (It is on the right side of the editor). You can change the *Text* property of a button, and the textbox to rename them. *Font* property can be used to alter the font of the controls.

8. To write the specific action for an event(eg. clicking on a button), double click on the control. Code window will open.

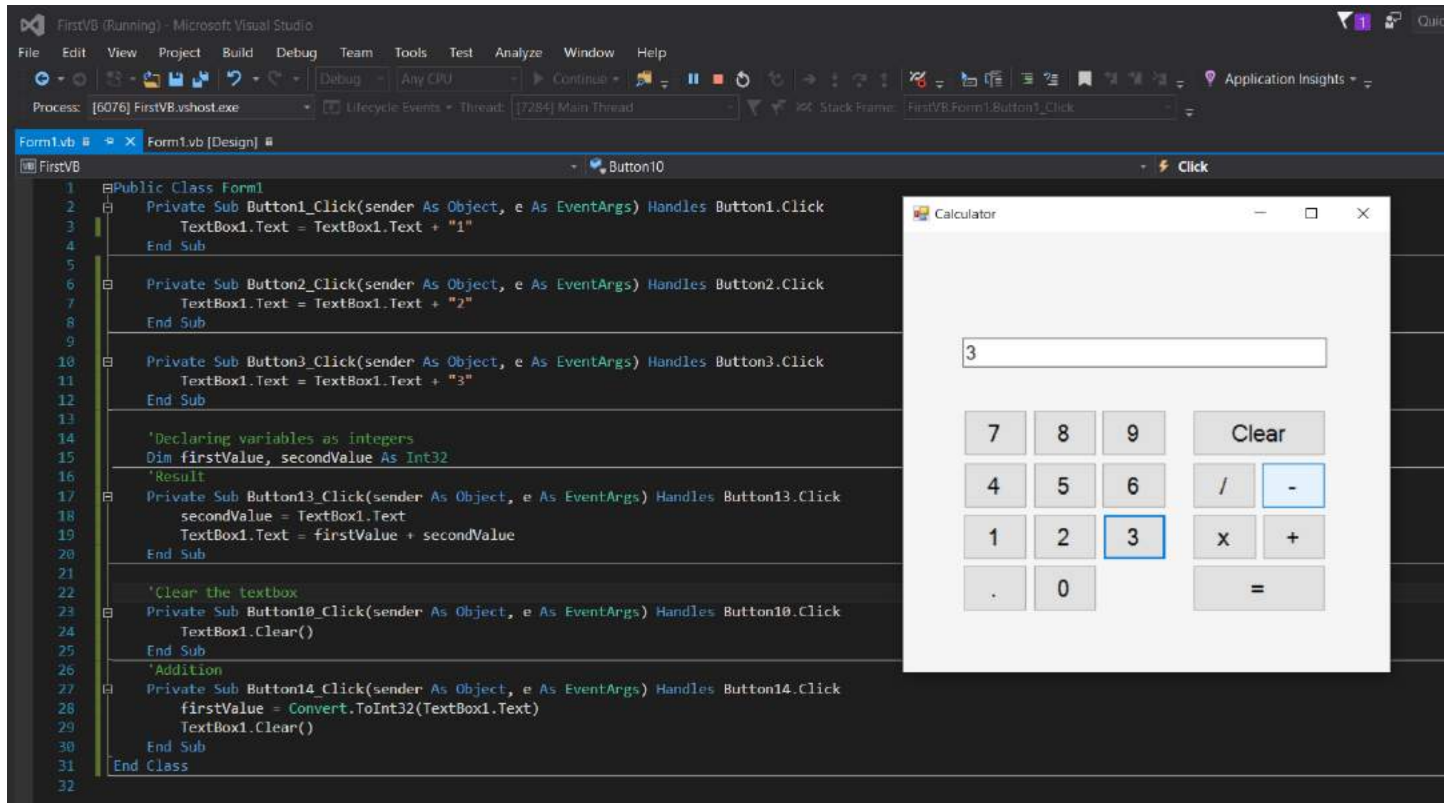


```vb
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        TextBox1.Text = TextBox1.Text + "1"
    End Sub

    Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
        TextBox1.Text = TextBox1.Text + "2"
    End Sub

    Private Sub Button3_Click(sender As Object, e As EventArgs) Handles Button3.Click
        TextBox1.Text = TextBox1.Text + "3"
    End Sub

    'Declaring variables as integers
    Dim firstValue, secondValue As Int32
    'Result
    Private Sub Button13_Click(sender As Object, e As EventArgs) Handles Button13.Click
        secondValue = TextBox1.Text
        TextBox1.Text = firstValue + secondValue
    End Sub

    'Clear the textbox
    Private Sub Button10_Click(sender As Object, e As EventArgs) Handles Button10.Click
        TextBox1.Clear()
    End Sub
    'Addition
    Private Sub Button14_Click(sender As Object, e As EventArgs) Handles Button14.Click
        firstValue = Convert.ToInt32(TextBox1.Text)
        TextBox1.Clear()
    End Sub
End Class
```

9. VB.Net 是一种为快速开发设计的强大语言。高封装性和抽象性是其代价。你不需要添加分号来表示语句结束，没有括号，并且大多数情况下，它会自动纠正字母的大小写。

10. 图片中提供的代码应该很容易理解。 Dim 是用于初始化变量的关键字，而 new 用于分配内存。你在文本框中输入的任何内容默认都是 string 类型。若要将值用作不同类型，则需要进行类型转换。

享受你在 VB.Net 中的第一个创作吧！



```vb
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        TextBox1.Text = TextBox1.Text + "1"
    End Sub

    Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
        TextBox1.Text = TextBox1.Text + "2"
    End Sub

    Private Sub Button3_Click(sender As Object, e As EventArgs) Handles Button3.Click
        TextBox1.Text = TextBox1.Text + "3"
    End Sub

    'Declaring variables as integers
    Dim firstValue, secondValue As Int32
    'Result
    Private Sub Button13_Click(sender As Object, e As EventArgs) Handles Button13.Click
        secondValue = TextBox1.Text
        TextBox1.Text = firstValue + secondValue
    End Sub

    'Clear the textbox
    Private Sub Button10_Click(sender As Object, e As EventArgs) Handles Button10.Click
        TextBox1.Clear()
    End Sub
    'Addition
    Private Sub Button14_Click(sender As Object, e As EventArgs) Handles Button14.Click
        firstValue = Convert.ToInt32(TextBox1.Text)
        TextBox1.Clear()
    End Sub
End Class
```

9. VB.Net is a powerful language designed for fast development. High encapsulation and abstraction is cost for it. You do not need to add *semicolon* to indicate the end of a statement, there are no brackets, and most of the time, it auto-corrects the case of the alphabets.
10. Code provided in the picture should be simple to understand. *Dim* is the keyword used to initialize a variable, and *new* allocates memory. Anything you type in the textbox is of type *string* by default. Casting is required to use the value as a different type.

Enjoy your first creation in VB.Net!

第二章：声明变量

第2.1节：使用原始类型声明和赋值变量

在 Visual Basic 中，变量使用 Dim 关键字声明。例如，下面声明了一个名为counter的新变量，数据类型为Integer：

```
Dim counter As Integer
```

变量声明也可以包括一个访问修饰符，例如**Public**、**Protected**、**Friend**或**Private**。它与变量的作用域配合使用，以确定其可访问性。

访问修饰符	含义
Public	所有可以访问封闭类型的类型
Protected	只有封闭类及其继承类
Friend	同一程序集中的所有类型可以访问封闭类型
受保护的友元	包含类及其继承者，或同一程序集中的类型可以访问该成员封闭类
私有	仅限封闭类型
静态	仅用于局部变量且只初始化一次。

作为简写，Dim关键字可以在变量声明中被访问修饰符替代：

```
Public TotalItems As Integer
Private counter As Integer
```

支持的数据类型列于下表：

类型	别名	内存分配	示例
SByte	不适用	1 字节	`Dim example As SByte = 10`
Int16	Short	2 字节	`Dim example As Short = 10`
Int32	整数4字节		`Dim example As Integer = 10`
Int64	长整型	8字节	`Dim example As Long = 10`
单精度浮点数	不适用	4字节	`Dim example As Single = 10.95`
双精度浮点数	不适用	8字节	`Dim example As Double = 10.95`
Decimal	N/A	16 字节	`Dim example As Decimal = 10.95`
Boolean	N/A	由实现平台决定	`Dim example As Boolean = True`
Char	不适用	2 字节	`Dim example As Char = "A"C`
字符串	不适用	源	`Dim example As String = "Stack Overflow"`
DateTime	日期	8 字节	`Dim example As Date = Date.Now`
字节	不适用	1 字节	`Dim example As Byte = 10`
无符号16位整数	UShort	2 字节	`Dim example As UShort = 10`
UInt32	无符号整数	4 字节	`Dim example As UInteger = 10`
UInt64	无符号长整数	8字节	`Dim example As ULong = 10`
对象	不适用	4 字节 32 位架构，8 字节 64 位架构	`Dim example As Object = Nothing`

还存在可用于替代文本类型和/或强制字面量类型的数据标识符和字面量类型字符：

Chapter 2: Declaring variables

Section 2.1: Declaring and assigning a variable using a primitive type

Variables in Visual Basic are declared using the **Dim** keyword. For example, this declares a new variable called `counter` with the data type `Integer`:

```
Dim counter As Integer
```

A variable declaration can also include an access modifier, such as **Public**, **Protected**, **Friend**, or **Private**. This works in conjunction with the variable's scope to determine its accessibility.

Access Modifier	Meaning
Public	All types which can access the enclosing type
Protected	Only the enclosing class and those that inherit from it
Friend	All types in the same assembly that can access the enclosing type
Protected Friend	The enclosing class and its inheritors, *or* the types in the same assembly that can access the enclosing class
Private	Only the enclosing type
Static	Only on local variables and only initializes once.

As a shorthand, the **Dim** keyword can be replaced with the access modifier in the variable's declaration:

```
Public TotalItems As Integer
Private counter As Integer
```

The supported data types are outlined in the table below:

Type	Alias	Memory allocation	Example
SByte	N/A	1 byte	`Dim example As SByte = 10`
Int16	Short	2 bytes	`Dim example As Short = 10`
Int32	Integer	4 bytes	`Dim example As Integer = 10`
Int64	Long	8 bytes	`Dim example As Long = 10`
Single	N/A	4 bytes	`Dim example As Single = 10.95`
Double	N/A	8 bytes	`Dim example As Double = 10.95`
Decimal	N/A	16 bytes	`Dim example As Decimal = 10.95`
Boolean	N/A	Dictated by implementing platform	`Dim example As Boolean = True`
Char	N/A	2 Bytes	`Dim example As Char = "A"C`
String	N/A	source	`Dim example As String = "Stack Overflow"`
DateTime	Date	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`
Object	N/A	4 bytes 32 bit architecture, 8 bytes 64 bit architecture	`Dim example As Object = Nothing`

There also exist data identifier and literal type characters usable in replacement for the textual type and or to force literal type:

类型（或别名）	标识符类型字符	字面量类型字符
Short	不适用	示例 = 10S
整数	Dim 示例%	示例 = 10% 或 示例 = 10I
长整数	Dim 示例&	示例 = 10& 或 示例 = 10L
单精度	Dim 示例!	示例 = 10! 或 示例 = 10F
双精度	Dim example#	example = 10# 或者 example = 10R
十进制	Dim example@	example = 10@ 或者 example = 10D
字符型	不适用	example = "A"C
字符串	Dim example$	不适用
无符号短整型	不适用	example = 10US
无符号整型	不适用	example = 10UI
无符号长整数	不适用	example = 10UL

整数后缀也可用于带有十六进制（&H）或八进制（&O）前缀的数值：

`example = &H8000S` 或 `example = &O77&`

Date(Time) 对象也可以使用字面量语法定义：

`Dim example As Date = #7/26/2016 12:8 PM#`

一旦变量被声明，它将在包含的类型、Sub或Function声明的作用域内存在，
例如：

```
Public Function IncrementCounter() As Integer
    Dim counter As Integer = 0
counter += 1

    Return counter
End Function
```

counter 变量只会存在到End Function为止，之后将超出作用域。如果需要在函数外使用该 counter 变量，则必须在类/结构或模块级别定义它。

```
Public Class ExampleClass

    Private _counter As Integer

    Public Function IncrementCounter() As Integer
       _counter += 1
       返回_counter
    结束函数

结束类
```

或者，你可以使用**Static**（不要与**Shared**混淆）修饰符，使局部变量在其包含方法的多次调用之间保留其
值：

```
函数 IncrementCounter() 作为 整数
    Static counter 作为 整数 = 0
    counter += 1

    Return counter
End Function
```

Type (or Alias)	Identifier type character	Literal type character
Short	N/A	example = 10S
Integer	Dim example%	example = 10% or example = 10I
Long	Dim example&	example = 10& or example = 10L
Single	Dim example!	example = 10! or example = 10F
Double	Dim example#	example = 10# or example = 10R
Decimal	Dim example@	example = 10@ or example = 10D
Char	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

The integral suffixes are also usable with hexadecimal (&H) or octal (&O) prefixes:

`example = &H8000S` or `example = &O77&`

Date(Time) objects can also be defined using literal syntax:

`Dim example As Date = #7/26/2016 12:8 PM#`

Once a variable is declared it will exist within the Scope of the containing type, **Sub** or **Function** declared, as an example:

```
Public Function IncrementCounter() As Integer
    Dim counter As Integer = 0
    counter += 1

    Return counter
End Function
```

The counter variable will only exist until the **End Function** and then will be out of scope. If this counter variable is needed outside of the function you will have to define it at class/structure or module level.

```
Public Class ExampleClass

    Private _counter As Integer

    Public Function IncrementCounter() As Integer
       _counter += 1
       Return _counter
    End Function

End Class
```

Alternatively, you can use the **Static** (not to be confused with **Shared**) modifier to allow a local variable to retain it's value between calls of its enclosing method:

```
Function IncrementCounter() As Integer
    Static counter As Integer = 0
    counter += 1

    Return counter
End Function
```

第2.2节：声明级别——局部变量和成员变量

局部变量 - 在类（或其他结构）的过程（子程序或函数）内声明的变量。在此示例中，exampleLocalVariable 是在ExampleFunction()内声明的局部变量：

```
公共类 ExampleClass1

    公共函数 ExampleFunction() 作为 整数
        Dim exampleLocalVariable 作为 整数 = 3
        返回 exampleLocalVariable
    结束函数

结束类
```

Static 关键字允许局部变量被保留，并在过程结束后保持其值（通常情况下，局部变量在包含它们的过程终止时会消失）。

在此示例中，控制台输出为024。每次从Main()调用ExampleSub()时，静态变量都会保留上一次调用结束时的值：

```
模块 模块1

    子程序 主程序()
       示例子程序()
       示例子程序()
       示例子程序()
    End Sub

    Public Sub ExampleSub()
        Static exampleStaticLocalVariable As Integer = 0
       Console.Write(exampleStaticLocalVariable.ToString)
        exampleStaticLocalVariable += 2
    End Sub

End Module
```

成员变量 - 在任何过程之外、类（或其他结构）级别声明。它们可以是**实例变量**，每个包含类的实例都有自己独立的该变量副本，或者共享变量，作为与类本身关联的单一变量存在，独立于任何实例。

这里，ExampleClass2 包含两个成员变量。每个 ExampleClass2 的实例都有一个单独的 ExampleInstanceVariable，可以通过类引用访问。共享变量 ExampleSharedVariable 则是通过类名访问的：

```
模块 模块1

    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)
```

Section 2.2: Levels of declaration - Local and Member variables

Local variables - Those declared within a procedure (subroutine or function) of a class (or other structure). In this example, exampleLocalVariable is a local variable declared within ExampleFunction():

```
Public Class ExampleClass1

    Public Function ExampleFunction() As Integer
        Dim exampleLocalVariable As Integer = 3
        Return exampleLocalVariable
    End Function

End Class
```

The Static keyword allows a local variable to be retained and keep its value after termination (where usually, local variables cease to exist when the containing procedure terminates).

In this example, the console is 024. On each call to ExampleSub() from Main() the static variable retains the value it had at the end of the previous call:

```
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 variables - Declared outside of any procedure, at the class (or other structure) level. They may be **instance variables**, in which each instance of the containing class has its own distinct copy of that variable, or **Shared variables**, which exist as a single variable associated with the class itself, independent of any instance.

Here, ExampleClass2 contains two member variables. Each instance of the ExampleClass2 has an individual ExampleInstanceVariable which can be accessed via the class reference. The shared variable ExampleSharedVariable however is accessed using the class name:

```
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"

    结束类

End Module
```

第2.3节：访问修饰符示例

在以下示例中，假设您有一个解决方案，包含两个项目：**ConsoleApplication1**和**SampleClassLibrary**。第一个项目将包含类**SampleClass1**和**SampleClass2**。第二个项目将包含**SampleClass3**和**SampleClass4**。换句话说，我们有两个程序集，每个程序集包含两个类。**ConsoleApplication1引用了SampleClassLibrary。**

请看SampleClass1.MethodA如何与其他类和方法交互。

SampleClass1.vb：

```
Imports SampleClassLibrary

Public Class SampleClass1
    Public Sub MethodA()
        'MethodA 可以调用以下任意方法，因为
        它们都在同一作用域内。
MethodB()
        MethodC()
        MethodD()
        MethodE()

        “Sample2”被定义为友元。它在类型本身以及同一程序集中的所有命名
        空间和代码中都是可访问的。
        Dim class2 As New SampleClass2()
        class2.MethodA()
        'class2.MethodB() 'SampleClass2.MethodB 不可访问，因为
                         'this method is private. SampleClass2.MethodB
                         '只能从 SampleClass2.MethodA、
                         'SampleClass2.MethodC、SampleClass2.MethodD
                         '和 SampleClass2.MethodE 调用
        class2.MethodC()
        'class2.MethodD() 'SampleClass2.MethodD 不可访问，因为
                         'this method is protected. SampleClass2.MethodD
                         '只能从继承 SampleClass2 的任何类、SampleClass2.Method
                         A、SampleClass2.MethodC、'SampleClass2.MethodD 和 SampleClass2.
                         MethodE 调用
        class2.MethodE()

        Dim class3 As New SampleClass3() 'SampleClass3 位于其他
                                         '程序集并定义为 public。
                                         '它在任何地方都可访问。
        class3.MethodA()
        'class3.MethodB() 'SampleClass3.MethodB 不可访问，因为
                         'this method is private. SampleClass3.MethodB 只能
                         '从 SampleClass3.MethodA 调用，
                         'SampleClass3.MethodC，SampleClass3.MethodD
                         '和 SampleClass3.MethodE
```

```
    End Sub

    Public Class ExampleClass4

        Public ExampleInstanceVariable As String
        Public Shared ExampleSharedVariable As String = "FizzBuzz"

    End Class

End Module
```

Section 2.3: Example of Access Modifiers

In the following example consider you have a solution hosting two projects: **ConsoleApplication1** and **SampleClassLibrary**. The first project will have the classes **SampleClass1** and **SampleClass2**. The second one will have **SampleClass3** and **SampleClass4**. In other words we have two assemblies with two classes each. **ConsoleApplication1** has a reference to **SampleClassLibrary**.

See how **SampleClass1.MethodA** interacts with other classes and methods.

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 不可访问，因为
                          '该方法是友元方法且位于另一个程序集内。
                          'SampleClass3.MethodC 只能在
                          '同一程序集内调用，SampleClass3.MethodA，SampleClass3.MethodB，
                          'SampleClass3.MethodD 和 SampleClass3.MethodE

        'class4.MethodD() 'SampleClass3.MethodE 不可访问，因为'该方法是受保护的友元
                          方法。SampleClass3.MethodD'只能从位于同一程序集内且继承 Samp
                          leClass3 的任何类中调用，

                          'SampleClass3.MethodA，SampleClass3.MethodB，
                          'SampleClass3.MethodC 和 SampleClass3.MethodD

        'Dim class4 As New SampleClass4() 'SampleClass4 不可访问，因为
                                          '它被定义为友元且位于
                                          '另一个程序集内。
    End Sub

    Private Sub MethodB()
        '执行 MethodB 的操作...
    End Sub

    Friend Sub MethodC()
        '执行 MethodC 的操作...
    End Sub

    Protected Sub MethodD()
        '执行 MethodD 的操作...
    End Sub

    受保护的 Friend 子程序 MethodE()
        '执行 MethodE 的操作...
    结束子程序
结束类
```

SampleClass2.vb：

```
Friend 类 SampleClass2
    Public 子程序 MethodA()
        '执行 MethodA 的操作...
    End Sub

    Private Sub MethodB()
        '执行 MethodB 的操作...
    End Sub

    Friend Sub MethodC()
        '执行 MethodC 的操作...
    End Sub

    Protected Sub MethodD()
        '执行 MethodD 的操作...
    End Sub

    受保护的 Friend 子程序 MethodE()
        '执行 MethodE 的操作...
    结束子程序
结束类
```

SampleClass3.vb：

```
        '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 类 SampleClass3
    Public 子程序 MethodA()
        '执行 MethodA 的操作...
    End Sub
    Private Sub MethodB()
        '执行 MethodB 的操作...
    End Sub

    Friend Sub MethodC()
        '执行 MethodC 的操作...
    End Sub

    Protected Sub MethodD()
        '执行 MethodD 的操作...
    End Sub

    受保护的 Friend 子程序 MethodE()
        '执行 MethodE 的操作...
    结束子程序
结束类
```

SampleClass4.vb：

```
Friend 类 SampleClass4
    Public 子程序 MethodA()
        '执行 MethodA 的操作...
    End Sub
    Private Sub MethodB()
        '执行 MethodB 的操作...
    End Sub

    Friend Sub MethodC()
        '执行 MethodC 的操作...
    End Sub

    Protected Sub MethodD()
        '执行 MethodD 的操作...
    End Sub

    受保护的 Friend 子程序 MethodE()
        '执行 MethodE 的操作...
    结束子程序
结束类
```

```
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
```

第3章：语法介绍

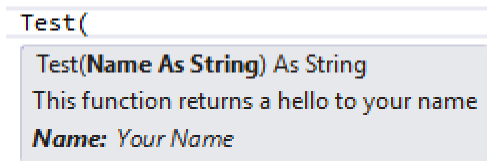
第3.1节：智能感知助手

一个有趣的功能是能够将你自己的注释添加到 Visual Studio 智能感知中。这样你就可以让自己编写的函数和类具有自解释性。要做到这一点，你必须在函数上方的行输入三次注释符号。

完成后，Visual Studio 会自动添加 XML 文档注释：

```
''' <summary>
''' 此函数返回对你的名字的问候
''' </summary>
''' <param name="Name">你的名字</param>
''' <returns></returns>
''' <remarks></remarks>
Public Function Test(Name As String) As String
    Return "Hello " & Name
结束函数
```

之后，如果你在代码中的某处输入 Test 函数，这个小提示就会出现：



```
Test(
```

Test(**Name As String**) As String
This function returns a hello to your name
Name: *Your Name*

第3.2节：声明变量

在 VB.NET 中，每个变量在使用前必须声明（如果Option Explicit设置为On）。声明变量有两种方式：

- 在Function或Sub内部：

```
Dim w 声明一个名为 w 的 Object 类型变量（如果 Option Strict 开启则无效）
Dim x As String '声明一个名为 x 的字符串类型变量
Dim y As Long = 45 '声明一个名为 y 的长整型变量并赋值为 45
Dim z = 45 '声明一个名为 z 的变量，其类型由赋值推断
           '根据赋值的类型（此处为整数类型）（如果 Option Infer 开启）
           '否则类型为 Object（如果 Option Strict 开启则无效）
           '并将该值（45）赋给它
```

详见this answer关于Option Explicit、Strict 和 Infer 的完整说明。

- 在Class或Module内部：

这些变量（在此上下文中也称为字段）对于它们声明的`Class`的每个实例都是可访问的
它们是否能从声明的`Class`外部访问取决于修饰符（`Public`、`Private`、`Protected`、`Protected Friend`或`Friend`）

```
Private x '声明一个名为 x 的私有字段，类型为 Object（如果 Option Strict 开启则无效）
Public y As String '声明一个名为 y 的公共字段，类型为字符串
Friend z As Integer = 45 '声明一个名为 z 的 Friend 字段，类型为整数并赋值为 45
45
```

Chapter 3: Introduction to Syntax

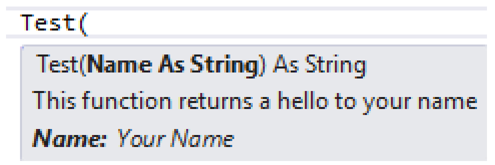
Section 3.1: Intellisense Helper

One interesting thing is the ability to add you own comments into Visual Studio Intellisense. So you can make your own written functions and classes self-explanatory. To do so, you must type the comment symbol three times the line above your function.

Once done, Visual Studio will automatically add an XML documentation :

```
''' <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
```

After that, if you type in your Test function somewhere in your code, this little help will show up :



```
Test(
```

Test(**Name As String**) As String
This function returns a hello to your name
Name: *Your Name*

Section 3.2: Declaring a Variable

In VB.NET, every variable must be declared before it is used (If Option Explicit is set to **On**). There are two ways of declaring variables:

- Inside a `Function` or a `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
```

See this answer for full details about `Option Explicit`, `Strict` and `Infer`.

- Inside a `Class` or a `Module`:

These variables (also called fields in this context) will be accessible for each instance of the `Class` they are declared in. They might be accessible from outside the declared `Class` depending on the modifier (`Public`, `Private`, `Protected`, `Protected Friend` or `Friend`)

```
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
```

这些字段也可以用Dim声明，但含义取决于其所在的封闭类型：

```
类 SomeClass
    Dim z As Integer = 45 ' 与 Private z As Integer = 45 含义相同
End Class

结构 SomeStructure
    Dim y As String ' 与 Public y As String 含义相同
End Structure
```

第3.3节：注释

首先要了解的有趣内容是如何编写注释。

在 VB .NET 中，你可以通过写撇号 ' 或写 REM 来编写注释。这意味着该行剩余部分不会被编译器考虑。

```
'整行都是注释
Dim x As Integer = 0 '这里的注释表示我们给 x 赋值为 0

REM 没有多行注释这种东西
'所以我们必须每行都以撇号或 REM 开头
```

第3.4节：修饰符

修饰符是一种指示外部对象如何访问对象数据的方式。

- Public

表示任何对象都可以无限制地访问此内容

- 私有

表示只有声明该内容的对象可以访问和查看此内容

- Protected

表示只有声明该内容的对象及其继承的任何对象可以访问和查看此内容。

- Friend

表示只有声明该内容的对象、继承自它的任何对象以及同一命名空间中的任何对象可以访问
和查看此内容。

```
公共类 MyClass
    私有 x 作为 整数

    友元属性 Hello 作为 字符串

    公共子程序 New()
    End Sub

    受保护函数 Test() 作为 整数
        返回 0
    结束函数
结束类
```

These fields can also be declared with **Dim** but the meaning changes depending on the enclosing type:

```
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
```

Section 3.3: Comments

The first interesting thing to know is how to write comments.

In VB .NET, you write a comment by writing an apostrophe ' or writing **REM**. This means the rest of the line will not be taken into account by the compiler.

```
'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
```

Section 3.4: Modifiers

Modifiers are a way to indicate how external objects can access an object's data.

- Public

Means any object can access this without restriction

- Private

Means only the declaring object can access and view this

- Protected

Means only the declaring object and any object that inherits from it can access and view this.

- Friend

Means only the delcaring object, any object that inherits from it and any object in the same namespace can access and view this.

```
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
```

第3.5节：对象初始化器

- 命名类型

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

等同于

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

- *匿名类型（必须开启 Option Infer）*

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

虽然类似的anonymousInstance类型与someInstance

成员名称在匿名类型中必须唯一，且可以取自变量或其他对象
成员名称

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

每个成员前面可以加上 Key 关键字。带有该关键字的成员将是 只读 属性，未带关键字的成员将是可读写属
性

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

两个具有相同成员（名称、类型、是否带有 Key 以及顺序）的匿名实例将具有相同的匿名类型。

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

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

Section 3.5: Object Initializers

- Named Types

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

Is equivalent to

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

- Anonymous Types *(Option Infer must be On)*

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

Although similar `anonymousInstance` doesn't have same type as `someInstance`

Member name must be unique in the anonymous type, and can be taken from a variable or another object member name

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

Each member can be preceded by the `Key` keyword. Those members will be **ReadOnly** properties, those without will be read/write properties

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

Two anonymous instance defined with the same members (name, type, presence of `Key` and order) will have the same anonymous type.

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

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

匿名类型在结构上是可比较的。两个相同匿名类型的实例只要至少有一个具有相同 Key 属性值的对象将被视为相等。你必须使用 Equals 方法来测试，使用 = 不会通过编译，而 Is 会比较对象引用。

```
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 尽管非 Key 的 Salary 不同
```

命名类型和匿名类型的初始化器都可以嵌套和混合使用

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

第3.6节：集合初始化器

- 数组

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

- 容器（列表（的 T），字典（的 TKey，TValue）等）

```
Dim names 作为新的 List(Of String) 从 {
    "Foo",
    "Bar"
    '...
}

Dim indexedDays 作为新的 Dictionary(Of Integer, String) 从 {
    {0, "Sun"},
    {1, "Mon"}
    '...
}
```

等同于

```
Dim indexedDays 作为新的 Dictionary(Of Integer, String)
indexedDays.Add(0, "Sun")
indexedDays.Add(1, "Mon")
'...
```

项可以是构造函数、方法调用、属性访问的结果。它也可以与对象初始化器混合使用。

```
Dim someList 作为新的 List(Of SomeClass) 从 {
    New SomeClass(参数),
```

Anonymous types are structurally equatable. Two instance of the same anonymous types having at least one Key property with the same Key values will be equal. You have to use Equals method to test it, using = won't compile and Is will compare the object reference.

```
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
```

Both Named and Anonymous types initializer can be nested and mixed

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

Section 3.6: Collection Initializer

- Arrays

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

- Containers (List(Of T), Dictionary(Of TKey, TValue), etc.)

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

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

Is equivalent to

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

Items can be the result of a constructor, a method call, a property access. It can also be mixed with Object initializer.

```
Dim someList As New List(Of SomeClass) From {
    New SomeClass(argument),
    New SomeClass 带有 { .Member = 值 },
    otherClass.PropertyReturningSomeClass,
   FunctionReturningSomeClass(参数)
    '...
}
```

不能同时对同一个对象使用对象初始化器语法和集合初始化器语法。例如，以下这两个都不能工作

```
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
}
```

- 自定义类型

我们也可以通过提供自定义类型来允许集合初始化器语法。
它必须实现`IEnumerable`接口，并且根据重载规则拥有一个可访问且兼容的`Add`方法
（实例方法、共享方法甚至扩展方法）

牵强的示例：

```
类 Person
    实现 IEnumerable(Of Person) ' 继承自 IEnumerable

    Private ReadOnly relationships 作为 List(Of Person)

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

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

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

        For Each relation In relationships
          Yield relation
        Next
    结束函数

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

        Return GetEnumerator()
    End Function
结束类

' 用法
```

```
    New SomeClass With { .Member = value },
    otherClass.PropertyReturningSomeClass,
    FunctionReturningSomeClass(arguments)
    '...
}
```

It is not possible to use Object initializer syntax **AND** collection initializer syntax for the same object at the same time. For example, these **won't** work

```
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
}
```

- Custom Type

We can also allow collection initializer syntax by providing for a custom type.
It must implement `IEnumerable` and have an accessible and compatible by overload rules `Add` method (instance, Shared or even extension method)

Contrived example :

```
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 作为 New Person("name") From {
    "FriendName",
    "CoWorkerName"
    '...
}
```

如果我们想通过只在集合初始化器中放入名称来将 Person 对象添加到 `List(Of Person)` 中（但我们不能修改 List(Of Person) 类），我们可以使用扩展方法

```
' 在模块内部
<Runtime.CompilerServices.Extension>
Sub Add(target As List(Of Person), name As String)
    target.Add(New Person(name))
End Sub

' 用法
Dim people As New List(Of Person) From {
    "Name1", ' 这里不需要创建 Person 对象
    "Name2"
}
```

第3.7节：编写函数

函数是一段将在执行过程中被多次调用的代码块。与其一次又一次地编写相同的代码，不如将这段代码写在函数中，并在需要时调用该函数。

函数：

- 必须在类或模块中声明返回一个值（由返
- 回类型指定）具有修饰符
-
- 可以接受参数以进行处理

```
Private Function AddNumbers(X As Integer, Y As Integer) As Integer
    Return X + Y
结束函数
```

函数名，可以用作返回语句

```
函数 sealBarTypeValidation() 作为布尔值
    Dim err 作为布尔值 = False

    如果 rbSealBarType.SelectedValue = "" 那么
        err = True
    结束如果

    返回 err
结束函数
```

与……完全相同

```
函数 sealBarTypeValidation() 作为布尔值
    sealBarTypeValidation = False

    如果 rbSealBarType.SelectedValue = "" 那么
        sealBarTypeValidation = True
    结束如果
```

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

If we wanted to add Person object to a `List(Of Person)` by just putting the name in the collection initializer (but we can't modify the List(Of Person) class) we can use an Extension method

```
' 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"
}
```

Section 3.7: Writing a function

A function is a block of code that will be called several times during the execution. Instead of writing the same piece of code again and again, one can write this code inside a function and call that function whenever it is needed.

A function :

- Must be declared in a *class* or a *module*
- Returns a value (specified by the return type)
- Has a *modifier*
- Can take parameters to do its processing

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

A Function Name, could be used as the return statement

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

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

    Return err
End Function
```

is just the same as

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

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

```
结束函数
```

```
End Function
```

第4章：运算符

第4.1节：字符串连接

字符串连接是将两个或多个字符串合并成一个字符串变量。

字符串连接是通过 `&` 符号来完成的。

```
Dim one As String = "Hello "
Dim two As String = "there"
Dim result As String = one & two
```

非字符串值在使用 `&`.

```
Dim result as String = "2" & 10 ' result  = "210"
```

始终使用 `&` （&）符号来执行字符串连接。

不要这样做
虽然在最简单的情况下，可以使用 `+` 使用加号符号进行字符串连接时，您绝不应该这样做。如果加号符号的一侧不是字符串，当 Option Strict 关闭时，行为会变得不直观；当 Option Strict 打开时，会产生编译错误。请考虑：

```
Dim value = "2" + 10    ' 结果 = 12  （数据类型 Double）
Dim value = "2" + "10"  ' 结果 = "210"  （数据类型 String）
Dim value = "2g" + 10   ' 运行时错误
```

这里的问题是，如果+运算符遇到任何一个操作数是数值类型，它会假定程序员想要执行算术运算，并尝试将另一个操作数转换为等效的数值类型。在另一个操作数是包含数字的字符串（例如，“10”）的情况下，字符串会被转换为数字，然后与另一个操作数进行算术加法运算。如果另一个操作数无法转换为数字（例如，“2g”），操作将因数据转换错误而崩溃。只有当+运算符的两个操作数都是String类型时，才会执行字符串连接。

然而，& 运算符是为字符串连接设计的，并且会将非字符串类型转换为字符串。

第4.2节：数学

如果你有以下变量

```
Dim leftValue As Integer = 5
Dim rightValue As Integer = 2
Dim value As Integer = 0
```

加法 由加号执行 `+`.

```
value  = leftValue + rightValue

输出以下内容：
7
```

减法 由减号执行 `-`.

```
value = leftValue - rightValue
```

Chapter 4: Operators

Section 4.1: String Concatenation

String concatenation is when you combine two or more strings into a single string variable.

String concatenation is performed with the `&` symbol.

```
Dim one As String = "Hello "
Dim two As String = "there"
Dim result As String = one & two
```

Non-string values will be converted to string when using `&`.

```
Dim result as String = "2" & 10 ' result  = "210"
```

Always use `&` (ampersand) to perform string concatenation.

DON'T DO THIS
While it is possible, in the *simplest* of cases, to use the `+` symbol to do string concatenation, you should never do this. If one side of the plus symbol is not a string, when Option strict is off, the behavior becomes non-intuitive, when Option strict is on it will produce a compiler error. Consider:

```
Dim value = "2" + 10    ' result = 12  (data type Double)
Dim value = "2" + "10"  ' result = "210"  (data type String)
Dim value = "2g" + 10   ' runtime error
```

The problem here is that if the + operator sees any operand that is a numeric type, it will presume that the programmer wanted to perform an arithmetic operation and attempt to cast the other operand to the equivalent numeric type. In cases where the other operand is a string that contains a number (for example, "10"), the string is *converted to a number* and then *arithmetically* added to the other operand. If the other operand cannot be converted to a number (for example, "2g"), the operation will crash due to a data conversion error. The + operator will only perform string concatenation if *both* operands are of `String` type.

The & operator, however, is designed for string concatenation and will cast non-string types to strings.

Section 4.2: Math

If you have the following variables

```
Dim leftValue As Integer = 5
Dim rightValue As Integer = 2
Dim value As Integer = 0
```

Addition Performed by the plus sign `+`.

```
value  = leftValue + rightValue

'Output the following:
'7
```

Subtraction Performed by the minus sign `-`.

```
value = leftValue - rightValue
```

```
输出以下内容：
3
```

乘法 由星号执行 `*`.

```
value = leftValue * rightValue

输出以下内容：
'10
```

除法 由斜杠符号执行 `/`.

```
value = leftValue / rightValue

输出以下内容：
'2.5
```

整数除法 由反斜杠符号执行 `\`.

```
value = leftValue \ rightValue

输出以下内容：
'2
```

取模 由 `Mod` 关键字。

```
value = leftValue Mod rightValue

输出以下内容：
'1
```

乘方运算 由以下执行 `^` 符号来完成的。

```
value = leftValue ^ rightValue

输出以下内容：
'25
```

第4.3节：赋值

VB中只有一个赋值运算符。

- 等号 `=` 既用于相等比较，也用于赋值。
  `Dim value = 5`

注意事项
注意赋值与相等比较的区别。

```
Dim result = leftValue = rightValue
```

在此示例中，你可以看到等号既用作比较运算符，也用作赋值运算符，这与其他语言不同。在这种情况下，result 的类型将是 Boolean，且包含 leftValue 和 rightValue 之间相等比较的结果值。

相关内容：使用 Option Strict On 正确声明变量

```
'Output the following:
'3
```

Multiplication Performed by the star symbol `*`.

```
value = leftValue * rightValue

'Output the following:
'10
```

Division Performed by the forward slash symbol `/`.

```
value = leftValue / rightValue

'Output the following:
'2.5
```

Integer Division Performed by the backslash symbol `\`.

```
value = leftValue \ rightValue

'Output the following:
'2
```

Modulus Performed by the `Mod` keyword.

```
value = leftValue Mod rightValue

'Output the following:
'1
```

Raise to a Power of Performed by the `^` symbol.

```
value = leftValue ^ rightValue

'Output the following:
'25
```

Section 4.3: Assignment

There is a single assignment operator in VB.

- The equal sign `=` is used both for equality comparison and assignment.
  `Dim value = 5`

Notes
Watch out for assignment vs. equality comparison.

```
Dim result = leftValue = rightValue
```

In this example you can see the equal sign being used as both a comparison operator and an assignment operator, unlike other languages. In this case, `result` will be of type `Boolean` and will contain the value of the equality comparison between `leftValue` and `rightValue`.

Related: Using Option Strict On to declare variables properly

第4.4节：比较

比较运算符比较两个值，并返回一个布尔值（True 或 False）作为结果。

相等

- 等号 `=`= 既用于相等比较，也用于赋值。
  如果 leftValue = rightValue 那么 ...

不等式

- 左尖括号紧挨右尖括号 `<>`执行不等比较。
  如果 leftValue <> rightValue 那么 ...

大于

- 左尖括号 `<`执行大于比较。
  如果 leftValue < rightValue 那么 ...

大于或等于

- 等号位于左尖括号旁边 `=>`执行大于或等于比较。
  如果 leftValue =< rightValue 则 ...

小于

- 右尖括号 `>`执行小于比较。
  如果 leftValue > rightValue 则 ...

小于或等于

- 等号位于右尖括号旁边 `=>`执行大于或等于比较。
  如果 leftValue => rightValue 那么 ...

类似于

- 该 `类似于`运算符测试字符串与搜索模式的相等性。
- 该 `类似于`运算符依赖于Option Compare语句
- 下表列出了可用的模式。来源：
  https://msdn.microsoft.com/en-us/library/swf8kaxw.aspx（备注部分）

模式中的字符Pattern	字符串中的匹配
?	任意单个字符
*	零个或多个字符
#	任意单个数字（0 - 9）
[字符列表]	任意单个字符，位于字符列表中
[!字符列表]	任意单个字符，不在字符列表中

- 有关MSDN的更多信息请参见备注部分。
  如果stringLikepatternThen...

第4.5节：按位操作

这些是VB.NET中的按位运算符：And、Or、Xor、Not

Section 4.4: Comparison

Comparison operators compare two values and return to you a boolean (`True` or `False`) as the result.

Equality

- The equal sign `=` is used both for equality comparison and assignment.
  `If leftValue = rightValue Then` ...

Inequality

- The left angle bracket nest to the right angle bracket `<>` performs an unequal comparison.
  `If leftValue <> rightValue Then` ...

Greater Than

- The left angle bracket `<` performs a greater than comparison.
  `If leftValue < rightValue Then` ...

Greater Than Or Equal

- The equal sign nest to the left angle bracket `=>` performs a greater than or equals comparison.
  `If leftValue =< rightValue Then` ...

Less Than

- The right angle bracket `>` performs a less than comparison.
  `If leftValue > rightValue Then` ...

Less Than Or Equal

- The equal sign nest to the right angle bracket `=>` performs a greater than or equals comparison.
  `If leftValue => rightValue Then` ...

Like

- The `Like` operator tests the equality of a string and a search pattern.
- The `Like` operator relies on the Option Compare Statement
- The following table lists the available patterns. Source:
  https://msdn.microsoft.com/en-us/library/swf8kaxw.aspx (Remarks section)

Characters in the *Pattern*	Matches in the *String*
?	Any single character
*	Zero or more characters
#	Any single digit (0 - 9)
[charlist]	Any single character in *charlist*
[!charlist]	Any single character not in *charlist*

- See further info on MSDN in the remarks section.
  `If string Like pattern Then ...`

Section 4.5: Bitwise

These are the bitwise operators in VB.NET : And, Or, Xor, Not

And按位运算示例

```
Dim a 作为Integer
a = 3 And 5
```

a的值将是1。结果是在二进制形式下比较3和5后得到的。3的二进制形式是011，5的二进制形式是101。And运算符当两个位都是1时结果为1，如果任一位为0，则结果为0。

```
3 And 5 将是 011
                101
                 ---
                001
```

所以二进制结果是001，当转换为十进制时，答案是1。

或运算符当两个或其中一个位为1时，结果为1

```
   3 或 5 将是 011
                101
                ---
                111
```

异或运算符当且仅当其中一个位为1（而非两个都为1）时，结果为1

```
   3 异或 5 将是 011
                  101
                  ---
                  110
```

非运算符会反转包括符号位在内的所有位

```
非 5 将是 - 010
```

Example of And bitwise operation

```
Dim a as Integer
a = 3 And 5
```

The value of a will be 1. The result is obtained after comparing 3 and 5 in binary for. 3 in binary form is 011 and 5 in binary form is 101. The And operator places 1 if both bits are 1. If any of the bits are 0 then the value will be 0

```
3 And 5 will be  011
                 101
                 ---
                 001
```

So the binary result is 001 and when that is converted to decimal, the answer will be 1.

Or operator places 1 if both or one bit is 1

```
   3 Or 5 will be 011
                  101
                  ---
                  111
```

Xor operator places 1 if only one of the bit is 1 (not both)

```
   3 Xor 5 will be  011
                    101
                    ---
                    110
```

Not operator reverts the bits including sign

```
Not 5 will be - 010
```

第5章：条件

第5.1节：If运算符

版本 ≥ 9.0

```
If(条件 > 值, "True", "False")
```

我们可以使用If运算符来代替If...Then...Else..End If语句块。

考虑以下示例：

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

等同于

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

If()使用*短路*求值，这意味着它只会计算所使用的参数。如果条件为false（或为Nullable且值为Nothing），则第一个选项根本不会被计算，其任何副作用也不会被观察到。这实际上与C#中的三元运算符condition?a:b形式相同。

这在避免异常时尤其有用：

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

我们都知道除以零会抛出异常，但If()这里通过短路机制防止了这种情况，只计算条件已经确保有效的表达式。

另一个例子：

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

如果varString <> "N/A"的结果为False，则会将varDate赋值为Now.Date，而不会计算第一个表达式。

版本 < 9.0

较旧版本的VB没有If()运算符，只能使用内置函数IIf()。由于它是函数而非运算符，不支持短路；所有表达式都会被计算，可能带来副作用，包括性能损失、状态改变和抛出异常。（上述避免异常的例子如果用IIf实现都会抛出异常。）如果这些副作用会造成问题，就无法使用内联条件表达式，只能像往常一样使用If..Then代码块。

第5.2节：IF...Then...Else

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

If count = 0 Then
```

Chapter 5: Conditions

Section 5.1: If operator

Version ≥ 9.0

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

We can use the **If** operator instead of **If...Then...Else..End If** statement blocks.

Consider the following example:

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

is the same as

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

If() uses *short-circuit* evaluation, which means that it will only evaluate the arguments it uses. If the condition is false (or a Nullable that is Nothing), the first alternative will not be evaluated at all, and none of its side effects will be observed. This is effectively the same as C#'s ternary operator in the form of condition?a:b.

This is especially useful in avoiding exceptions:

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

We all know that dividing by zero will throw an exception, but If() here guards against this by short-circuiting to only the expression that the condition has already ensured is valid.

Another example:

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

If varString <> "N/A" evaluates to False, it will assign varDate's value as Now.Date without evaluating the first expression.

Version < 9.0

Older versions of VB do not have the If() operator and have to make do with the IIf() built-in function. As it's a function, not an operator, it does *not* short-circuit; all expressions are evaluated, with all possible side-effects, including performance penalties, changing state, and throwing exceptions. (Both of the above examples that avoid exceptions would throw if converted to IIf.) If any of these side effects present a problem, there's no way to use an inline conditional; instead, rely on If..Then blocks as usual.

Section 5.2: IF...Then...Else

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

If count = 0 Then
```

```
message = "没有项目。"
ElseIf count = 1 Then
    message = "有 1 个项目。"
Else
message = "有 " & count & " 个项目。"
结束如果
```

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

第6章：短路运算符 (AndAlso - OrElse)

参数	详细信息
结果	必需。任何布尔表达式。结果是两个表达式比较的布尔结果。
expression1	必需。任何布尔表达式。
expression2	必需。任何布尔表达式。

第6.1节：OrElse 的用法

```
“OrElse”运算符是“AndAlso”的对应运算符。它允许我们执行布尔比较，只有当第一个条件为假时才会计算
第二个条件。

如果 testFunction(5) = True 或者 otherFunction(4) = True 那么
    ' 如果 testFunction(5) 为 True，则不会调用 otherFunction(4)。
    ' 插入要执行的代码。
结束如果
```

第6.2节：AndAlso 的用法

```
' 有时我们不需要对 if 语句的布尔检查中的所有条件进行求值。

' 假设我们有一个字符串列表：

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

' 我们想要检查列表中的第一个值：

如果 MyCollection.Count > 0 并且 MyCollection(0).Equals("Somevalue")
    Console.WriteLine("是的，我在集合中找到了 Somevalue！")
结束如果

' 如果 MyCollection 为空，运行时会抛出异常。
' 这是因为它无论第一个条件的结果如何，都会对 if 语句的第一个和第二个条件都进行求值。
'

' 现在让我们使用 AndAlso 运算符

如果 MyCollection.Count > 0 并且 MyCollection(0).Equals("Somevalue")
    Console.WriteLine("是的，我在集合中找到了Somevalue！")
结束如果

' 这不会抛出任何异常，因为编译器只会计算第一个条件。
' 如果第一个条件返回False，第二个表达式根本不会被计算。
```

第6.3节：避免NullReferenceException（空引用异常）

版本 ≥ 7.0

OrElse

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

    Dim average As Double = AverageElementsOrElse(elements)
   Console.WriteLine(average) ' 输出0到控制台
```

Chapter 6: Short-Circuiting Operators (AndAlso - OrElse)

Parameter	Details
result	Required. Any Boolean expression. The result is the Boolean result of comparison of the two expressions.
expression1	Required. Any Boolean expression.
expression2	Required. Any Boolean expression.

Section 6.1: OrElse Usage

```
' 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
```

Section 6.2: AndAlso Usage

```
' 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.
```

Section 6.3: Avoiding NullReferenceException

Version ≥ 7.0

OrElse

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

    Dim average As Double = AverageElementsOrElse(elements)
    Console.WriteLine(average) ' Writes 0 to Console
    Try
        抛出 ArgumentNullException
        average = AverageElementsOr(elements)
    捕获 ex 作为 ArgumentNullException
        Console.WriteLine(ex.Message)
    结束 Try
End Sub

公共函数 AverageElementsOrElse(ByVal elements 作为 IEnumerable(Of Integer)) 作为 Double
    ' 如果 elements 为 Nothing，则不会调用 elements.Count，因此不会崩溃
    如果 (elements 是 Nothing 或者 elements.Count = 0) 那么
        返回 0
    否则
        返回 elements.Average()
    结束 If
结束函数

公共函数 AverageElementsOr(ByVal elements 作为 IEnumerable(Of Integer)) 作为 Double
    ' 总是调用 elements.Count，因此如果 elements 是 Nothing 可能会崩溃
    如果 (elements 是 Nothing 或 elements.Count = 0) 那么
        返回 0
    否则
        返回 elements.Average()
    结束 If
结束函数
```

版本 ≥ 7.0

并且

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

    定义 average 作为 Double = AverageElementsAndAlso(elements)
    Console.WriteLine(average) ' 输出 0 到控制台

    Try
        抛出 ArgumentNullException
        average = AverageElementsAnd(elements)
    捕获 ex 作为 ArgumentNullException
        Console.WriteLine(ex.Message)
    结束 Try
End Sub

Public Function AverageElementsAndAlso(ByVal elements As IEnumerable(Of Integer)) As Double
    ' 如果 elements 为 Nothing，则不会调用 elements.Count，因此不会崩溃
    If (Not elements Is Nothing AndAlso elements.Count > 0) Then
        Return elements.Average()
    否则
        Return 0
    End If
结束函数

Public Function AverageElementsAnd(ByVal elements As IEnumerable(Of Integer)) As Double
    ' elements.Count 总是被调用，因此如果 elements 为 Nothing 可能会崩溃
    If (Not elements Is Nothing And elements.Count > 0) Then
        Return elements.Average()
    否则
        Return 0
    End If
结束函数
```

版本 ≥ 14.0

Visual Basic 14.0 引入了空条件运算符，使得函数可以以更简洁的方式重写，

```
    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
```

Version ≥ 7.0

AndAlso

```
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
```

Version ≥ 14.0

Visual Basic 14.0 introduced the null conditional operator, allowing to rewrite the functions in a cleaner way,

模仿示例中 AndAlso 版本的行为。

mimicking the behavior of the `AndAlso` version of the example.

第七章：日期

第7.1节：将字符串转换（解析）为日期

如果你知道要转换（解析）的字符串格式，应使用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)
```

如果你不确定字符串的格式，可以使用`DateTime.TryParseExact`并测试结果以判断是否解析成功：

```
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
    '解析成功，dateValue变量现在保存了解析后的日期时间，因为它是通过ByRef传递的

否则
    '解析失败
结束如果
```

第7.2节：将日期转换为字符串

只需使用DateTime对象的.ToString重载即可获得所需的格式：

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

Chapter 7: Date

Section 7.1: Converting (Parsing) a String to a Date

If you know the format of the string you are converting (parsing) you should use `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)
```

If you are not certain for the format of the string, you can use `DateTime.TryParseExact` and test the result to see if parsed or not:

```
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
```

Section 7.2: Converting a Date To A String

Simply use the `.ToString` overload of a `DateTime` object to get the format you require:

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

第8章：数组

第8.1节：数组定义

```
Dim array(9) As Integer ' 定义一个包含10个整数元素（0-9）的数组变量。

Dim array = New Integer(10) {} ' 定义一个包含11个整数元素（0-10）的数组变量
                                       '使用New关键字。

Dim array As Integer() = {1, 2, 3, 4} ' 定义一个整数数组变量并初始化'使用数组字面量。数组包含4个元素。


ReDim Preserve array(10) ' 重新定义现有数组变量的大小，同时保留数组中已有的值。'数组现在包含11个整数元素（
                           0-10）。


ReDim array(10) ' 重新定义现有数组变量的大小，丢弃数组中已有的值。'数组现在包含11个整数元素（0-
                  10）。
```

基于零

VB.NET 中的所有数组都是从零开始的。换句话说，VB.NET 数组中第一个元素的索引（下界）始终是0。较早版本的 VB，如 VB6 和 VBA，默认是从一开始的，但它们提供了一种方法来覆盖默认的边界。在那些早期的 VB 版本中，下界和上界可以被明确指定（例如Dim array(5 To 10)）。在 VB.NET 中，为了保持与其他 .NET 语言的兼容性，这种灵活性被移除，下界0现在始终被强制执行。然而，VB.NET 中仍然可以使用To语法，这可能使范围更加明确。例如，以下示例都等同于上面列出的示例：

```
Dim array(0 到 9) 作为 整数

Dim array = 新建 整数(0 到 10) {}

ReDim 保留 array(0 到 10)

ReDim array(0 到 10)
```

嵌套数组声明

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

第8.2节：空数组变量

由于数组是引用类型，数组变量可以为null。要声明一个空数组变量，必须声明它不带大小：

```
Dim array() 作为 整数
```

或者

```
Dim array 作为 整数()
```

Chapter 8: Array

Section 8.1: 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).
```

Zero-Based

All arrays in VB.NET are zero-based. In other words, the index of the first item (the lower bound) in a VB.NET array is always `0`. Older versions of VB, such as VB6 and VBA, were one-based by default, but they provided a way to override the default bounds. In those earlier versions of VB, the lower and upper bounds could be explicitly stated (e.g. `Dim array(5 To 10)`). In VB.NET, in order to maintain compatibility with other .NET languages, that flexibility was removed and the lower bound of `0` is now always enforced. However, the `To` syntax can still be used in VB.NET, which may make the range more explicitly clear. For instance, the following examples are all equivalent to the ones listed above:

```
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)
```

Nested Array Declarations

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

Section 8.2: Null Array Variables

Since arrays are reference types, an array variable can be null. To declare a null array variable, you must declare it without a size:

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

Or

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

要检查数组是否为null，测试它是否Is Nothing：

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

要将现有的数组变量设置为 null，只需将其设置为 Nothing：

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

或者使用 Erase，它的作用相同：

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

第8.3节：数组初始化

```
Dim array() As Integer = {2, 0, 1, 6}                   ''初始化一个包含四个整数的数组。
Dim strings() As String = {"this", "is", "an", "array"} ''初始化一个包含四个字符串的数组。
Dim floats() As Single = {56.2, 55.633, 1.2, 5.7743, 22.345}
              初始化一个包含五个单精度浮点数的数组，单精度浮点数在C#中与float相同。
Dim miscellaneous() 作为 Object = { New Object(), "Hello", New List(of String) }
              ''初始化一个包含三个引用的数组，引用类型可以是任意类型
              ''并将它们指向三种不同类型的对象。
```

第8.4节：声明一维数组并设置数组元素值

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

或者

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

第8.5节：锯齿形数组初始化

注意括号以区分锯齿形数组和多维数组，子数组长度可以不同

```
Dim jaggedArray()() 作为 Integer = { ({1, 2, 3}), ({4, 5, 6}), ({7}) }
' jaggedArray(0) 是 {1, 2, 3}，因此 jaggedArray(0)(0) 是 1
' jaggedArray(1) 是 {4, 5, 6}，因此 jaggedArray(1)(0) 是 4
' jaggedArray(2) 是 {7}，因此 jaggedArray(2)(0) 是 7
```

第8.6节：非零下界

使用`Option Strict On`时，虽然.NET框架允许创建具有非零
下界的单维数组，但它们不是"向量"，因此与VB.NET类型化数组不兼容。这意味着它们只能被视为Array，
因此不能使用普通的数组(index)引用。

To check if an array is null, test to see if it `Is Nothing`:

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

To set an existing array variable to null, simply set it to `Nothing`:

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

Or use `Erase`, which does the same thing:

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

Section 8.3: Array initialization

```
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.
```

Section 8.4: Declare a single-dimension array and set array element values

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

or

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

Section 8.5: Jagged Array Initialization

Note the parenthesis to distinguish between a jagged array and a multidimensional array SubArrays can be of different length

```
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
```

Section 8.6: Non-zero lower bounds

With `Option Strict On`, although the .NET Framework allows the creation of single dimension arrays with non-zero lower bounds they are not "vectors" and so not compatible with VB.NET typed arrays. This means they can only be seen as `Array` and so cannot use normal array `(index)` references.

```
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
```

除了使用`Option Strict Off`外，你还可以通过将数组视为IList来恢复(index)语法，但这样它就不是数组了，因此不能对该变量名使用LBound和UBound（而且你仍然无法避免装箱）：

```
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
```

多维非零下界数组与VB.NET多维类型化数组兼容：

```
Dim nza(,) As Integer = DirectCast(Array.CreateInstance(GetType(Integer),
                                      {4, 3}, {1, -1}), Integer(,))
对于 y = LBound(nza) 到 UBound(nza)
    对于 w = LBound(nza, 2) 到 UBound(nza, 2)
       nza(y, w) = -y * w + nza(UBound(nza) - y + LBound(nza),
                                         UBound(nza, 2) - w + LBound(nza, 2))
    Next
Next
对于 y = LBound(nza) 到 UBound(nza)
    定义 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 参考：Array.CreateInstance

第8.7节：从两个变量引用同一个数组

由于数组是引用类型，因此可以有多个变量指向同一个数组对象。

```
定义 array1() 为 整数 = {1, 2, 3}
定义 array2() 为 整数 = array1
array1(0) = 4
Console.WriteLine(String.Join(", ", array2))  ' 输出 "4, 2, 3"
```

第8.8节：多维数组初始化

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

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

```
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
```

As well as by using `Option Strict Off`, you can get the `(index)` syntax back by treating the array as an `IList`, but then it's not an array, so you can't use `LBound` and `UBound` on that variable name (and you're still not avoiding boxing):

```
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
```

Multi-dimensional non-zero lower bounded arrays *are* compatible with VB.NET multi-dimensional typed arrays:

```
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 reference: Array.CreateInstance

Section 8.7: Referencing Same Array from Two Variables

Since arrays are reference types, it is possible to have multiple variables pointing to the same array object.

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

Section 8.8: Multidimensional Array initialization

```
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
```

第9章：列表

第9.1节：向列表添加项目

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

要一次添加多个项目，请使用AddRange。始终添加到列表末尾

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


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

若要向列表中间添加项目，请使用**Insert**

Insert会将项目放置在指定索引处，并重新编号剩余项目

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

新输出：

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

第9.2节：检查列表中是否存在某项

```
    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
```

产生以下输出：

```

False
True
False

```

第9.3节：遍历列表中的项

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

Chapter 9: Lists

Section 9.1: Add items to a List

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

To add more than one item at a time use **AddRange**. Always adds to the end of the list

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


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

In order to add items to the middle of the list use **Insert**

Insert will place the item at the index, and renumber the remaining items

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

New Output:

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

Section 9.2: Check if item exists in a List

```
    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
```

Produces the following output:

```

False
True
False

```

Section 9.3: Loop through items in list

```
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
```

产生以下输出：

```
one
two
three
```

另一种选择是，使用每个元素的索引进行循环：

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

For i = 0 to aList.Count - 1 '我们使用"- 1"是因为列表采用0基索引。
    System.Console.WriteLine(aList(i))
Next
```

第9.4节：创建列表

列表可以根据需要填充任何数据类型，格式为

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

例如：

创建一个新的空字符串列表

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

创建一个新的字符串列表，并填充一些数据

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"})
```

注意：

如果运行代码时收到以下信息：

```
aList.Add("one")
aList.Add("two")
aList.Add("three")

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

Produces the following output:

```
one
two
three
```

Another option, would be to loop through using the index of each element:

```
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
```

Section 9.4: Create a List

Lists can populated with any data type as necessary, with the format

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

For example:

Create a new, empty list of Strings

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

Create a new list of strings, and populate with some data

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"})
```

NOTE:

If you are receiving the following when the code is ran:

> 对象引用未设置为对象的实例。

确保你要么声明为New，例如Dim aList as New List(Of String)，
要么如果声明时没有New，确保你将列表设置为一个新列表 - Dim aList as List(Of String) = New List(Of String)

第9.5节：从列表中移除项目

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

'从索引1处移除列表中的项目
aList.RemoveAt(1)

'从列表中移除一段范围的项目，从索引0开始，数量为1)
'这将移除索引0和1！
aList.RemoveRange(0, 1)

清空整个列表
alist.Clear()
```

第9.6节：从列表中检索项目

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

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

output:

> Hello, World

如果不知道项目的索引或只知道字符串的一部分，则使用Find或FindAll方法

```
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:

> Hello, World

FindAll方法返回一个新的字符串列表（List）

```
Dim aList 作为 新 列表(字符串)
aList.Add("Hello, Test")
aList.Add("Hello, World")
aList.Add("Test")
```

> Object reference not set to an instance of an object.

Make sure you either declare as **New** i.e. `Dim aList as New List(Of String)` or if declaring without the **New**, make sure you set the list to a new list - `Dim aList as List(Of String) = New List(Of String)`

Section 9.5: Remove items from a List

```
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()
```

Section 9.6: Retrieve items from a List

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

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

output:

> Hello, World

If you do not know the index of the item or only know part of the string then use the **Find** or **FindAll** method

```
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:

> Hello, World

The **FindAll** method returns a new List (of String)

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

```
Dim output 作为 字符串 = aList.FindAll(函数(x) x.Contains("Test"))
```

> output(0) = "Hello, Test"

> output(1) = "Test"

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

> output(0) = "Hello, Test"

> output(1) = "Test"

第10章：枚举

第10.1节：GetNames()

返回指定枚举中常量的名称，作为字符串数组：

```
模块 模块1

  Enum Size
    Small
    Medium
    Large
  End Enum

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

    对于每个 size 在 sizes
      Console.WriteLine(size)
      下一步
  End Sub

End Module
```

输出：

```
小

中

大
```

第10.2节：HasFlag()

可以使用HasFlag()方法来检查标志是否被设置。

```
模块 模块1

    <Flags>
    枚举 材料
        木材 = 1
        塑料 = 2
        金属 = 4
        石头 = 8
    结束枚举

    Sub Main()
        Dim houseMaterials 作为 材料 = 材料.木材 或 材料.石头

        如果 houseMaterials.HasFlag(材料.石头) 那么
            Console.WriteLine("房子是用石头建造的")
        否则
Console.WriteLine("the house is not made of stone")
        结束 If
    结束 Sub
```

Chapter 10: Enum

Section 10.1: GetNames()

Returns the names of constants in the specified Enum as a string array:

```
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
```

Output:

```
Small

Medium

Large
```

Section 10.2: HasFlag()

The `HasFlag()` method can be used to check if a flag is set.

```
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
```

有关 Flags 属性及其使用方法的更多信息，请参见微软官方文档。

第10.3节：枚举定义

枚举是一组逻辑相关的常量。

```
枚举 大小
小
中
大
结束枚举

公共子程序 Order(shirtSize 作为 大小)
    选择情况 shirtSize
        机壳 尺寸.小
            ' ...
        Case 中号
            ' ...
        Case 大号
            ' ...
    End Select
End Sub
```

第10.4节：成员初始化

每个枚举成员都可以被初始化为一个值。如果没有为成员指定值，默认情况下它会被初始化为0（如果它是成员列表中的第一个成员），或者初始化为比前一个成员的值大1的值。

```
模块 模块1

    Enum 大小
       小号
       中号 = 3
       大号
    结束枚举

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

        ' 等待用户按下任意键
        Console.ReadKey()
    End Sub

End Module
```

第10.5节：Flags属性

使用<Flags>属性，枚举变成一组标志。该属性允许为枚举变量分配多个值。标志枚举的成员应初始化为2的幂（1、2、4、8...）。

```
模块 模块1

    <Flags>
```

```
End Module
```

For more information about the Flags-attribute and how it should be used see the official Microsoft documentation.

Section 10.3: Enum definition

An enum is a set of logically related constants.

```
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
```

Section 10.4: Member initialization

Each of the enum members may be initialized with a value. If a value is not specified for a member, by default it's initialized to 0 (if it's the first member in the member list) or to a value greater by 1 than the value of the preceding member.

```
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
```

Section 10.5: The Flags attribute

With the **<Flags>** attribute, the enum becomes a set of flags. This attribute enables assigning multiple values to an enum variable. The members of a flags enum should be initialized with powers of 2 (1, 2, 4, 8...).

```
Module Module1

    <Flags>
    枚举 材料
       木材 = 1
       塑料 = 2
       金属 = 4
       石头 = 8
    结束枚举

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

End Module
```

第10.6节：GetValues()

' 此方法对于遍历枚举值非常有用 '

```
Enum Animal
    Dog = 1
    Cat = 2
    Frog = 4
结束枚举

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

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

打印结果：

```
1

2

4
```

第10.7节：字符串解析

可以通过解析枚举的字符串表示来创建一个枚举实例。

```
模块 模块1

    枚举 大小
       小
       中
       大
    结束枚举

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

```
    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
```

Section 10.6: GetValues()

' This method is useful for iterating Enum values '

```
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
```

Prints:

```
1

2

4
```

Section 10.7: String parsing

An Enum instance can be created by parsing a string representation of the Enum.

```
Module Module1

    Enum Size
        Small
        Medium
        Large
    End Enum

    Sub Main()
        Dim shirtSize As Size =  DirectCast([Enum].Parse(GetType(Size), "Medium"), Size)
        ' 输出 'Medium'
        Console.WriteLine(shirtSize.ToString())

        ' 等待用户按下任意键
        Console.ReadKey()
    End Sub

End Module
```

另见：在 VB.NET 中将字符串解析为枚举值

第 10.8 节：ToString()

枚举的 ToString 方法返回枚举的字符串名称。例如：

```
模块 Module1
    枚举 大小
       小
       中
       大
    结束枚举

    Sub Main()
        Dim shirtSize As Size = Size.Medium
        Dim output As String = shirtSize.ToString()
        Console.WriteLine(output)  ' 输出 "Medium"
    结束子程序
End Module
```

但是，如果需要枚举实际整数值的字符串表示，可以先将枚举转换为
整数，然后调用 ToString：

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

第10.9节：确定枚举是否指定了 FlagsAttribute

下面的示例可用于确定枚举是否指定了FlagsAttribute。所用的方法基于反射。

此示例将返回True结果：

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

此示例将返回False结果：

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

我们可以设计一个像这样的通用扩展方法：

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

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

End Module
```

See also: Parse a string to an Enum value in VB.NET

Section 10.8: ToString()

The ToString method on an enum returns the string name of the enumeration. For instance:

```
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
```

If, however, the string representation of the actual integer value of the enum is desired, you can cast the enum to an Integer and then call ToString:

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

Section 10.9: Determine whether a Enum has FlagsAttribute specified or not

The next example can be used to determine whether a enumeration has the **FlagsAttribute** specified. The methodology used is based on **Reflection**.

This example will give a **True** result:

```
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)
```

This example will give a **False** result:

```
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)
```

We can design a generic usage extension method like this one:

```
<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
```

使用示例：

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

第10.10节：For-each标志（标志迭代）

在一些非常特定的场景中，我们需要对源枚举的每个标志执行特定操作。

我们可以编写一个简单的泛型扩展方法来实现此任务。

```
<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
```

使用示例：

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

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

第10.11节：确定标志组合中标志的数量

下面的示例旨在计算指定标志组合中标志的数量。

该示例作为扩展方法提供：

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

使用示例：

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

```
<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
```

Usage Example:

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

Section 10.10: For-each flag (flag iteration)

In some very specific scenarios we would feel the need to perform a specific action for each flag of the source enumeration.

We can write a simple *Generic* extension method to realize this task.

```
<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
```

Usage Example:

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

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

Section 10.11: Determine the amount of flags in a flag combination

The next example is intended to count the amount of flags in the specified flag combination.

The example is provided as a extension method:

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

Usage Example:

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

```
Console.WriteLine(count)
```

第10.12节：查找枚举中的最近值

下面的代码演示如何查找枚举（Enum）的最近值。

首先我们定义这个枚举，用于指定搜索条件（搜索方向）

```
公共枚举 EnumFindDirection 作为 整数
    最近的 = 0
小于 = 1
    小于或等于 = 2
    大于 = 3
    大于或等于 = 4
结束枚举
```

现在我们实现搜索算法：

```
<DebuggerStepThrough>
公共共享函数 FindNearestEnumValue(泛型 T)(按值传递 value 作为 长整型,
                                                    按值传递 direction 作为 EnumFindDirection) 作为 T

    选择 direction

        情况 EnumFindDirection.Nearest
            返回 (从 enumValue 作为 T 在 [Enum].GetValues(GetType(T)).Cast(泛型 T)()
  按 Math.Abs(value - Convert.ToInt64(enumValue))排序).FirstOrDefault


        情况 EnumFindDirection.Less
            如果 value < Convert.ToInt64([Enum].GetValues(GetType(T)).Cast(泛型 T).First) 那么
                返回 [Enum].GetValues(GetType(T)).Cast(泛型 T).FirstOrDefault

            否则
                返回 (从 enumValue 作为 T 在 [Enum].GetValues(GetType(T)).Cast(Of T)()
                        其中 Convert.ToInt64(enumValue) < value
                        ).LastOrDefault
            结束如果

        情况 EnumFindDirection.LessOrEqual
            如果 value < Convert.ToInt64([Enum].GetValues(GetType(T)).Cast(泛型 T).First) 那么
                返回 [Enum].GetValues(GetType(T)).Cast(泛型 T).FirstOrDefault

            否则
                返回 (从 enumValue 作为 T 在 [Enum].GetValues(GetType(T)).Cast(Of T)()
                        其中 Convert.ToInt64(enumValue) <= value
                        ).LastOrDefault
            结束如果

        情况 EnumFindDirection.Greater
            如果 value > Convert.ToInt64([Enum].GetValues(GetType(T)).Cast(Of T).Last) 则
                返回 [Enum].GetValues(GetType(T)).Cast(Of T).LastOrDefault

            否则
                返回 (从 enumValue 作为 T 在 [Enum].GetValues(GetType(T)).Cast(Of T)()
                        其中 Convert.ToInt64(enumValue) > value
                        ).FirstOrDefault
            结束如果

        情况 EnumFindDirection.GreaterOrEqual
```

```
Console.WriteLine(count)
```

Section 10.12: Find the nearest value in a Enum

The next code illustrates how to find the nearest value of a **Enum**.

First we define this **Enum** that will serve to specify search criteria (search direction)

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

And now we implement the search algorithm:

```
<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
```

```
            如果 value > Convert.ToInt64([Enum].GetValues(GetType(T)).Cast(Of T).Last) 则
                返回 [Enum].GetValues(GetType(T)).Cast(Of T).LastOrDefault

            否则
                返回 (从 enumValue 作为 T 在[Enum].GetValues(GetType(T)).Cast(Of T)()
                        其中 Convert.ToInt64(enumValue) >= value
                        ).FirstOrDefault
            结束如果

    结束选择

结束函数
```

使用示例：

```
公共枚举 比特率 作为 整数
   Kbps128 = 128
Kbps192 = 192
    Kbps256 = 256
    Kbps320 = 320
结束枚举

定义 nearestValue 作为 比特率 = 查找最接近的枚举值(Of 比特率)(224,
EnumFindDirection.大于或等于)
```

```
            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
```

Usage Example:

```
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)
```

第11章：字典

字典表示键和值的集合。参见 MSDN Dictionary(Tkey, TValue) 类。

第11.1节：创建一个填充了值的字典

```
定义 extensions 作为 新的 字典(Of 字符串, 字符串) _
  来自 { { "txt", "记事本" },
  { "bmp", "画图" },
  { "doc", "Word" } }
```

这将创建一个字典并立即用三个键值对填充它。

您也可以稍后使用 Add 方法添加新值：

```
extensions.Add("png", "paint")
```

请注意，键（第一个参数）在字典中必须是唯一的，否则会抛出异常。

第11.2节：遍历字典并打印所有条目

字典中的每个键值对都是一个 KeyValuePair 实例，其类型参数与字典相同。当您使用 For Each 遍历字典时，每次迭代都会给出字典中存储的一个键值对。

```
For Each kvp As KeyValuePair(Of String, String) In currentDictionary
  Console.WriteLine("{0}: {1}", kvp.Key, kvp.Value)
Next
```

第11.3节：检查字典中是否已有键 - 数据简化

ConstainsKey 方法是判断字典中是否已存在某个键的方式。

这在数据简化中非常有用。下面的示例中，每当遇到一个新单词时，我们将其作为字典中的键添加，否则就增加该单词对应的计数器。

```
 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
     如果 dic.ContainsKey(str) 则
        dic(str) += 1
     否则
dic.Add(str, 1)
     End If
 Next
```

XML缩减示例：获取XML文档某分支中所有子节点的名称及出现次数

```
Dim nodes As IDictionary(Of String, Integer) = New Dictionary(Of String, Integer)
Dim xmlsrc = New XmlDocument()
xmlsrc.LoadXml(<any text stream source>)
```

Chapter 11: Dictionaries

A dictionary represents a collection of keys and values. See MSDN Dictionary(Tkey, TValue) Class.

Section 11.1: Create a dictionary filled with values

```
Dim extensions As New Dictionary(Of String, String) _
  from { { "txt", "notepad" },
  { "bmp", "paint" },
  { "doc", "winword" } }
```

This creates a dictionary and immediately fills it with three KeyValuePairs.

You can also add new values later on by using the Add method:

```
extensions.Add("png", "paint")
```

Note that the key (the first parameter) needs to be unique in the dictionary, otherwise an Exception will be thrown.

Section 11.2: Loop through a dictionary and print all entries

Each pair in the dictionary is an instance of `KeyValuePair` with the same type parameters as the Dictionary. When you loop through the dictionary with **For Each**, each iteration will give you one of the Key-Value Pairs stored in the dictionary.

```
For Each kvp As KeyValuePair(Of String, String) In currentDictionary
  Console.WriteLine("{0}: {1}", kvp.Key, kvp.Value)
Next
```

Section 11.3: Checking for key already in dictionary - data reduction

The `ConstainsKey` method is the way to know if a key already exists in the Dictionary.

This come in handy for data reduction. In the sample below, each time we encountner a new word, we add it as a key in the dictionary, else we increment the counter for this specific word.

```
 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
```

XML reduction example : getting all the child nodes names and occurrence in an branch of an XML document

```
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 '选择合适的父节点
    If nodes.ContainsKey(xn.Name) Then
nodes(xn.Name) += 1
    否则
nodes.Add(xn.Name, 1)
    End If
Next
```

第11.4节：获取字典中的值

您可以使用“Item”属性获取字典中条目的值：

```
Dim extensions 作为新的 Dictionary(Of String, String) 从 {
    { "txt", "notepad" },
    { "bmp", "paint" },
    { "doc", "winword" }
}

Dim program 作为 String = extensions.Item("txt") '将是 "notepad"

' 另一种语法，因为 Item 是默认属性（也称为索引器）
Dim program 作为 String = extensions("txt") '将是 "notepad"

' 另一种使用（罕见的）
' 字典成员访问操作符（也称为感叹号操作符）的语法
Dim program 作为 String = extensions!txt '将是 "notepad"
```

如果字典中不存在该键，将抛出 KeyNotFoundException 异常。

```
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
```

Section 11.4: Getting a dictionary value

You can get the value of an entry in the dictionary using the 'Item' property:

```
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"
```

If the key is not present in the dictionary, a KeyNotFoundException will be thrown.

第12章：循环

第12.1节：For...Next

For...Next 循环用于重复执行相同操作有限次数。以下循环中的语句将执行11次。第一次，i 的值为0，第二次值为1，最后一次值为10。

```
For i 作为 Integer = 0 到 10
    '执行操作
    Console.Writeline(i.ToString)
Next
```

任何整数表达式都可以用来参数化循环。允许但不要求在Next之后也声明控制变量（在本例中为i）。控制变量可以提前声明，而不是在For语句中声明。

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

For i = StartIndex To EndIndex - 1
    '执行操作
    Console.Writeline(i.ToString)
Next i
```

能够定义起始和结束整数允许创建直接引用其他对象的循环，例如：

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

这将遍历DataGridView1中的每一行，并执行将第1列的值写入控制台的操作。（-1 是因为计数的行的第一行是1，而不是0）

也可以定义控制变量必须如何递增。

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

输出结果为：

```
1 3 5 7 9
```

也可以递减控制变量（倒计数）。

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

输出结果为：

Chapter 12: Looping

Section 12.1: For...Next

For...**Next** loop is used for repeating the same action for a finite number of times. The statements inside the following loop will be executed 11 times. The first time, `i` will have the value 0, the second time it will have the value 1, the last time it will have the value 10.

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

Any integer expression can be used to parameterize the loop. It is permitted, but not required, for the control variable (in this case `i`) to also be stated after the **Next**. It is permitted for the control variable to be declared in advance, rather than within the **For** statement.

```
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
```

Being able to define the Start and End integers allows loops to be created that directly reference other objects, such as:

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

This would then loop through every row in `DataGridView1` and perform the action of writing the value of Column 1 to the Console. *(The -1 is because the first row of the counted rows would be 1, not 0)*

It is also possible to define how the control variable must increment.

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

This outputs:

```
1 3 5 7 9
```

It is also possible to decrement the control variable (count down).

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

This outputs:

```
10 9 8 7 6 5 4 3 2 1
```

不应尝试在循环外使用（读取或更新）控制变量。

第12.2节：用于遍历项目集合的 For Each...Next 循环

您可以使用 For Each...Next 循环来遍历任何 IEnumerable 类型。这包括数组、列表以及任何其他可能是 IEnumerable 类型或返回 IEnumerable 的对象。

遍历 DataTable 的 Rows 属性的示例代码如下：

```
For Each row As DataRow In DataTable1.Rows
    '每次循环时，row 将是 Rows 中的下一个项目
    '这里我们打印 row 变量的第一列的值。
Debug.Print(Row.Item(0))
Next
```

一个重要的注意事项是，在`For Each`循环中，集合不能被修改。否则会导致
抛出带有以下信息的`System.InvalidOperationException`异常：

> 集合已被修改；枚举操作可能无法执行。

第12.3节：短路

任何循环都可以通过使用Exit或Continue语句在任意点提前终止或继续。

退出

你可以通过提前退出来停止任何循环。为此，你可以使用关键字Exit以及循环的名称。

Loop	Exit语句
For	`退出For`
For Each	`退出For`
Do While	`退出Do`
While	`退出While`

提前退出循环是提升性能的好方法，只需循环满足应用需求的必要次数即可。下面的示例中，循环将在找到数字2时退出。

```
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
    结束如果
Next
Debug.Print(SoughtIndex)
```

继续

```
10 9 8 7 6 5 4 3 2 1
```

You should not attempt to use (read or update) the control variable outside the loop.

Section 12.2: For Each...Next loop for looping through collection of items

You can use a `For Each...Next` loop to iterate through any `IEnumerable` type. This includes arrays, lists, and anything else that may be of type IEnumerable or returns an IEnumerable.

An example of looping through a DataTable's Rows property would look like this:

```
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
```

An important thing to note is that the collection must not be modified while in a `For Each` loop. Doing so will cause a `System.InvalidOperationException` with the message:

> Collection was modified; enumeration operation may not execute.

Section 12.3: Short Circuiting

Any loop may be terminated or continued early at any point by using the `Exit` or `Continue` statements.

Exiting

You can stop any loop by exiting early. To do this, you can use the keyword `Exit` along with the name of the loop.

Loop	Exit Statement
For	`Exit For`
For Each	`Exit For`
Do While	`Exit Do`
While	`Exit While`

Exiting a loop early is a great way to boost performance by only looping the necessary number of times to satisfy the application's needs. Below is example where the loop will exit once it finds the number 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)
```

Continuing

除了提前退出，你还可以选择直接进入下一次循环迭代。这可以通过使用Continue语句轻松实现。和Exit一样，它前面也跟着循环名称。

LoopContinue语句	
For	`Continue For`
For Each	`Continue For`
Do While	`Continue Do`
While	`继续执行条件`

这是一个防止将偶数加入总和的示例。

```
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
```

使用建议

有两种替代技术可以用来代替使用Exit或Continue。

你可以声明一个新的布尔变量，初始化为一个值，并在循环内部根据条件将其设置为另一个值；然后你使用基于该变量的条件语句（例如If）来避免在后续迭代中执行循环内的语句。

```
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
    结束如果
Next
```

对这种技术的一个反对意见是它可能效率不高。例如，如果在上述示例中N是1000000，并且数组A的第一个元素等于SoughtValue，循环将再迭代999999次而没有做任何有用的事情。然而，这种技术在某些情况下可能具有更高的清晰度优势。

您可以使用GoTo语句跳出循环。请注意，您不能使用GoTo跳入循环。

```
    Dim FoundIndex As Integer
    For i As Integer = 0 To N - 1
        If A(i) = SoughtValue Then
            FoundIndex = i
            GoTo Found
        结束如果
    Next
    Debug.Print("未找到")
找到：
    Debug.Print(FoundIndex)
```

这种技术有时是跳出循环并避免执行一个或多个紧接在循环自然结束后执行的语句的最简洁方法。

您应考虑所有备选方案，并根据您的需求选择最合适的方案，考虑诸如

Along with exiting early, you can also decide that you need to just move on to the next loop iteration. This is easily done by using the `Continue` statement. Just like `Exit`, it is proceeded by the loop name.

Loop	Continue Statement
For	`Continue For`
For Each	`Continue For`
Do While	`Continue Do`
While	`Continue While`

Here's an example of preventing even numbers from being added to the sum.

```
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
```

Usage Advice

There are two alternative techniques that can be used instead of using `Exit` or `Continue`.

You can declare a new Boolean variable, initializing it to one value and conditionally setting it to the other value inside the loop; you then use a conditional statement (e.g. `If`) based on that variable to avoid execution of the statements inside the loop in subsequent iterations.

```
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
```

One of the objections to this technique is that it may be inefficient. For example, if in the above example `N` is 1000000 and the first element of the array `A` is equal to `SoughtValue`, the loop will iterate a further 999999 times without doing anything useful. However, this technique can have the advantage of greater clarity in some cases.

You can use the `GoTo` statement to jump out of the loop. Note that you cannot use `GoTo` to jump *into* a loop.

```
    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)
```

This technique can sometimes be the neatest way to jump out of the loop and avoid one or more statements that are executed just after the natural end of the loop.

You should consider all of the alternatives, and use whichever one best fits your requirements, considering such

诸如效率、代码编写速度和可读性（从而可维护性）等方面。

在那些使用GoTo是最佳选择的情况下，不要回避使用它。

第12.4节：当某个条件为真时使用while循环进行迭代

一个While循环开始时会先评估一个条件。如果条件为真，则执行循环体。在循环体执行完毕后，会再次评估While条件，以确定是否重新执行循环体。

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

  iteration += 1
End While
```

输出结果为：

```
1 2 3 4 5 6 7 8 9 10
```

警告： While 循环可能导致 无限循环。请考虑如果移除增加iteration 的代码行会发生什么。在这种情况下，条件将永远不会为真，循环将无限期地继续下去。

第12.5节：嵌套循环

> 嵌套循环是指循环中的循环，即外层循环体内的内层循环。其工作原理是
> 外层循环的第一次执行触发内层循环，内层循环执行完毕。然后
> 外层循环的第二次执行再次触发内层循环。这个过程重复，直到外层循环结束。无论是内层还是外层循环中的
> break 都会中断此过程。

For Next 嵌套循环的结构是：

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

    For counter2=startNumber to endNumber (Step increment)

一条或多条 VB 语句

    下一步 counter2

下一步 counter1
```

示例：

```
    对于 firstCounter = 1 到 5

打印 "第一次 循环 " + firstCounter

    对于 secondCounter= 1 到 4

打印 "第二次 循环 " + secondCounter

    下一步 secondCounter
```

things as efficiency, speed of writing the code, and readability (thus maintainability).

Do not be put off using `GoTo` on those occasions when it is the best alternative.

Section 12.4: While loop to iterate while some condition is true

A `While` loop starts by evaluating a condition. If it is true, the body of the loop is executed. After the body of the loop is executed, the `While` condition is evaluated again to determine whether to re-execute the body.

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

  iteration += 1
End While
```

This outputs:

```
1 2 3 4 5 6 7 8 9 10
```

Warning: A `While` loop can lead to an *infinite loop*. Consider what would happen if the line of code that increments `iteration` were removed. In such a case the condition would never be True and the loop would continue indefinitely.

Section 12.5: Nested Loop

> 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.

The Structure of a For Next nested loop is :

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

    For counter2=startNumber to endNumber (Step increment)

        One or more VB statements

    Next counter2

Next  counter1
```

Example :

```
    For  firstCounter = 1 to 5

      Print "First Loop of " + firstCounter

    For   secondCounter= 1 to 4

      Print "Second Loop of " + secondCounter

    Next secondCounter
```

```
下一步 firstCounter
```

第12.6节：Do...Loop

使用Do...Loop来重复一段语句块，While或Until条件为真时停止，条件可以在循环开始时或结束时检查。

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

或者

```
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
```

或者

```
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 可用于跳过当前循环的剩余部分，直接进入下一次循环：

```
Dim x As Integer = 0
Do While x < 10
    x += 1
    如果 x Mod 2 = 0 则
        继续循环
    结束如果
Console.Write(x & " ")
循环
```

```
1 3 5 7 9
```

```
Next firstCounter
```

Section 12.6: Do...Loop

Use `Do...Loop` to repeat a block of statements `While` or `Until` a condition is true, checking the condition either at the beginning or at the end of the loop.

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

or

```
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
```

or

```
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` can be used to skip to the next iteration of the loop:

```
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
```

你可以使用`Exit Do`终止循环——注意在此示例中，缺少任何条件否则会导致无限循环：

```
Dim x As Integer = 0
Do
Console.Write(x & " ")
    x += 1
    如果 x = 10 则
        退出循环
    结束如果
Loop
```

```
0 1 2 3 4 5 6 7 8 9
```

You can terminate the loop with `Exit Do` - note that in this example, the lack of any condition would otherwise cause an infinite loop:

```
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
```

第13章：文件处理

第13.1节：写入数据到文件

将字符串内容写入文件的方法：

```
Dim toWrite As String = "这将被写入文件。"
System.IO.File.WriteAllText("filename.txt", toWrite)
```

WriteAllText 会打开指定的文件，写入数据，然后关闭文件。如果目标文件存在，则会被覆盖。如果目标文件
不存在，则会被创建。

将数组内容写入文件：

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

WriteAllLines 会打开指定的文件，将数组中的每个值写入新的一行，然后关闭文件。如果目标文件存在，则会被覆盖。
如果目标文件不存在，则会被创建。

第13.2节：读取文件的全部内容

将文件内容读取到字符串变量中：

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

ReadAllText 会打开指定的文件，读取到文件末尾的数据，然后关闭文件。

读取文件，将每一行分隔为数组的一个元素：

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

ReadAllLines 会打开指定的文件，将文件的每一行读取到数组的新索引中，直到文件末尾，然后关闭文件。

第13.3节：使用

StreamWriter逐行写入文本文件

```
使用 sw 作为新的 System.IO.StreamWriter("patho\file.txt")
    sw.WriteLine("Hello world")
结束 Using
```

建议在使用实现了IDisposable接口的对象时使用Using代码块，这是一种良好的编程习惯

Chapter 13: File Handling

Section 13.1: Write Data to a File

To write the contents of a string to a file:

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

`WriteAllText` will open the specified file, write the data, and then close the file. If the target file exists, it is overwritten. If the target file does not exist, it is created.

To write the contents of an array to a file:

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

`WriteAllLines` will open the specified file, write each value of the array on a new line, and then close the file. If the target file exists, it is overwritten. If the target file does not exist, it is created.

Section 13.2: Read All Contents of a File

To read the contents to a file into a string variable:

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

`ReadAllText` will open the specified file, read data to the end, then close the file.

To read a file, separating it into an array element for each line:

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

`ReadAllLines` will open the specified file, read each line of the file into a new index in an array until the end of the file, then close the file.

Section 13.3: Write Lines Individually to a Text File using StreamWriter

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

The use of a **`Using`** block is recommended good practice when using an object that Implements `IDisposable`

第14章：文件/文件夹压缩

第14.1节：向项目添加文件压缩功能

1. 在解决方案资源管理器中，找到你的项目，右键点击引用，然后选择添加引用...
2. 搜索 Compression 并选择System.IO.Compression.FileSystem，然后点击确定。
3. 在代码文件顶部（任何类或模块之前，与其他

Imports语句一起）添加Imports System.IO.Compression。

```
Option Explicit On
Option Strict On

Imports System.IO.Compression

Public Class Foo

    ...

结束类
```

请注意，此类（ZipArchive）仅从 .NET 版本 4.5 开始可用

第14.2节：从目录创建压缩档案

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

创建包含myfolder中所有文件的archive.zip文件。示例中的路径相对于程序工作目录。您也可以指定绝对路径。

第14.3节：将压缩档案解压到目录

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

将archive.zip解压到myfolder目录。示例中的路径相对于程序工作目录。您也可以指定绝对路径。

第14.4节：动态创建压缩档案

```
' 创建指向文件的文件流
Using fileStream = New IO.FileStream("archive.zip", IO.FileMode.Create)
    ' 从流中打开zip存档
    Using archive = New System.IO.Compression.ZipArchive(fileStream,
IO.Compression.ZipArchiveMode.Create)
        ' 在存档中创建 file_in_archive.txt 文件
        Dim zipfile = archive.CreateEntry("file_in_archive.txt")

        ' 向存档中的 file_in_archive.txt 写入 Hello world
        Using sw As New IO.StreamWriter(zipfile.Open())
            sw.WriteLine("Hello world")
        End Using

    End Using
End Using
```

Chapter 14: File/Folder Compression

Section 14.1: Adding File Compression to your project

1. In *Solution Explorer* go to your project, right click on *References* then *Add reference*…
2. Search for Compression and select *System.IO.Compression.FileSystem* then press OK.
3. Add **Imports** `System.IO.Compression` to the top of your code file (before any class or module, with the other **Imports** statements).

```
Option Explicit On
Option Strict On

Imports System.IO.Compression

Public Class Foo

    ...

End Class
```

Plese note that this class (ZipArchive) is only available from .NET verison 4.5 onwards

Section 14.2: Creating zip archive from directory

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

Create archive.zip file containing files which are in `myfolder`. In example paths are relative to program working directory. You can specify absolute paths.

Section 14.3: Extracting zip archive to directory

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

Extracts archive.zip to myfolder directory. In example paths are relative to program working directory. You can specify absolute paths.

Section 14.4: Create zip archive dynamicaly

```
' 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
```

第15章：连接处理

第15.1节：公共连接属性

```
    Imports System.Data.OleDb

    Private WithEvents _connection As OleDbConnection
    Private _connectionString As String = "myConnectionString"

    Public ReadOnly Property Connection As OleDbConnection
        Get
            如果 _connection 为 Nothing 则
_connection = New OleDbConnection(_connectionString)
             _connection.Open()
            否则
                如果 _connection.State <> ConnectionState.Open 则
                    _connection.Open()
                结束 如果
            结束如果
            返回 _connection
        结束 获取
    结束 属性
```

Chapter 15: Connection Handling

Section 15.1: Public connection property

```
    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
```

第16章：数据访问

第16.1节：从数据库读取字段

```
公共函数 GetUserFirstName(UserName 作为 字符串) 作为 字符串
    Dim Firstname 作为 字符串 = ""

    '指定您想使用的包含参数的SQL
    Dim SQL 作为 字符串 = "select firstname from users where username=@UserName"

    提供数据源
    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)

    提供一个或多个参数
dbCommand.Parameters.AddWithValue("@UserName", UserName)

    可选的超时设置
dbCommand.CommandTimeout = 600

    Dim reader As SqlDataReader
    Dim previousConnectionState As ConnectionState = dbConn.State
    Try
        If dbConn.State = ConnectionState.Closed Then
          dbConn.Open()
        结束如果
reader = dbCommand.ExecuteReader
        Using reader
            带有读卡器
                如果 .HasRows 则
                    读取第一条记录
reader.Read()
                    读取必填字段
Firstname = .Item("FirstName").ToString
                结束 如果

            结束 With

        结束 Using

    捕获
        在此处处理错误
    最终
        如果 previousConnectionState = ConnectionState.Closed 则
          dbConn.Close()
        结束如果
dbConn.Dispose()
      dbCommand.Dispose()

    结束 Try
    '从函数传回数据
    返回 名字

结束函数
```

使用上述函数很简单：

Chapter 16: Data Access

Section 16.1: Read field from Database

```
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
```

Using the above function is simply:

```
Dim UserFirstName 作为字符串=GetUserFirstName(UserName)
```

第16.2节：从数据库读取并返回为DataTable的简单函数

此简单函数将执行指定的Select SQL命令并将结果作为数据集返回。

```
Public Function ReadFromDatabase(ByVal DBConnectionString 作为 String, ByVal SQL 作为 String) 作为
DataTable
    Dim dtReturn 作为 新的 DataTable
    Try
        '使用连接字符串打开连接
        Using conn 作为 新的 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
        结束 Using
    Catch ex As Exception
        '处理异常
    End Try


    '返回结果数据集
    Return dtReturn
End Function
```

现在你可以通过以下代码执行上述函数

```
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
```

上述示例假设你的 SQL Express 实例“SQLEXPRESS”当前安装在“MYDEVPC”上，且你的数据库“MyDatabase”包含“Customers”、“Suppliers”和“Employees”表，且“sa”用户的密码是“pwd22”。请根据你的环境更改这些值以获得所需结果。

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

Section 16.2: Simple Function to read from Database and return as DataTable

This simple function will execute the specified Select SQL command and return the result as data set.

```
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
```

Now you can execute the above function from below codes

```
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
```

The above example expects that your SQL Express instance "SQLEXPRESS" is currently installed on "MYDEVPC" and your database "MyDatabase" contains "Customers", "Suppliers" and "Employees" tables and the "sa" user password is "pwd22". Please change these values as per your setup to get the desired results.

第17章：类型转换

函数名	表达式参数范围
CBool	任何有效的字符、字符串或数值表达式
CByte	0到255（无符号）；小数部分四舍五入。
CChar	任何有效的字符或字符串表达式；仅转换字符串的第一个字符；值范围为0到65535（无符号）。

第17.1节：将文本框中的文本转换为整数

来自MSDN

> 使用 CInt 函数可以将任何其他数据类型转换为 Integer 子类型。例如，当通常会进行货币、单精度或双精度运算时，CInt 会强制进行整数运算。

假设你有 1 个按钮和 2 个文本框。如果你在 textbox1.text 中输入5.5，在 textbox2.text 中输入10。

如果你有以下代码：

```
Dim result = textbox1.text + textbox2.text
MsgBox("结果: " & result)
'它将输出
5.510
```

为了将两个文本框的值相加，你需要使用 CInt 将它们的值转换为Int，方法是 CInt(expression)。

```
Dim result = CInt(textbox1.text) + CInt(textbox2.text)
MsgBox("结果: " & result)
'它将输出
16
```

> 注意：当数值的小数部分正好是 0.5 时，CInt 函数会四舍五入到最接近的偶数。例如，0.5 会四舍五入到 0，而1.5 会四舍五入到 2，3.5 会四舍五入到 4。四舍五入到最接近的偶数的目的是为了补偿在累加许多数值时可能产生的偏差。

Chapter 17: Type conversion

Function name	Range for Expression argument
CBool	Any valid Char or String or numeric expression
CByte	0 through 255 (unsigned); fractional parts are rounded.
CChar	Any valid Char or String expression; only first character of a String is converted; value can be 0 through 65535 (unsigned).

Section 17.1: Converting Text of The Textbox to an Integer

From MSDN

> Use the CInt function to provide conversions from any other data type to an Integer subtype. For example, CInt forces integer arithmetic when currency, single-precision, or double-precision arithmetic would normally occur.

Assuming that you have 1 button and 2 textbox. If you type on textbox1.text `5.5` and on textbox2.text `10`.

If you have this code:

```
Dim result = textbox1.text + textbox2.text
MsgBox("Result: " & result)
'It will output
5.510
```

In order to add the values of the 2 textboxes you need to convert their values to `Int` by using the `CInt(expression)`.

```
Dim result = CInt(textbox1.text) + CInt(textbox2.text)
MsgBox("Result: " & result)
'It will output
16
```

> Note: When the fractional part of a value is exactly 0.5, the CInt function rounds to the closest even number. For example, **0.5 rounds to 0**, while **1.5 rounds to 2, and 3.5 rounds to 4**. The purpose of rounding to the closest even number is to compensate for a bias that could accumulate when many numbers are added together.

第18章：ByVal 和 ByRef 关键字

第18.1节：ByRef 关键字

方法参数前的 ByRef 关键字表示参数将以允许方法更改（赋予新值）该参数所对应变量的方式传递。

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

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

       Foo (someInstance)
        ' 此处 someInstance 不为 Nothing
        ' 但 someInstance.Member 为 -42

        Bar(someInstance)
        ' 此处 someInstance 为 Nothing
    End Sub

    Sub Foo(ByVal arg As SomeClass)
        arg.Member = -arg.Member ' 更改参数内容
        arg = Nothing ' 更改（重新赋值）参数
    End Sub

    Sub Bar(ByRef param As Integer)
        arg.Member = -arg.Member ' 更改参数内容
        arg = Nothing ' 更改（重新赋值）参数
    End Sub
End Module
```

第18.2节：ByVal关键字

方法参数前的ByVal关键字（或无关键字时默认视为ByVal）表示参数将以一种不允许方法更改（赋新值）参数所对应变量的方式传递。
如果参数是类的实例，这并不阻止其内容（或状态）被更改。

```
类 SomeClass
    Public Property Member As Integer
结束类

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

       Foo (someInstance)
        ' 此处someInstance不为Nothing（仍是同一个对象）
        ' 但someInstance.Member是-42（内部状态仍可更改）

        Dim number As Integer = 42
        Foo(number)
        ' 此处number仍是42
    End Sub

    Sub Foo(ByVal arg As SomeClass)
```

Chapter 18: ByVal and ByRef keywords

Section 18.1: ByRef keyword

ByRef keyword before method parameter says that parameter will be sent in a way allowing the method to change (assign a new value) the variable underlying the parameter.

```
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
```

Section 18.2: ByVal keyword

ByVal keyword before method parameter (or no keyword as ByVal is assumed by default) says that parameter will be sent in a way **not** allowing the method to change (assign a new value) the variable underlying the parameter.
It doesn't prevent the content (or state) of the argument to be changed if it's a class.

```
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 ' 更改参数内容
        arg = Nothing ' 更改（重新赋值）参数
    End Sub

    子程序 Foo(arg 作为 整数) ' 没有 ByVal 或 ByRef 关键字，默认是 ByVal
        arg = -arg
    End Sub
End Module
```

```
        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
```

第19章：控制台

第19.1节：Console.ReadLine()

```
Dim 输入 作为 字符串 = Console.ReadLine()
```

Console.ReadLine() 会读取用户的控制台输入，直到检测到下一个换行符（通常是在按下回车键时）。当前线程的代码执行会暂停，直到提供换行符为止。
之后，将执行下一行代码。

第19.2节：Console.Read()

```
Dim 输入代码 作为 整数 = Console.Read()
```

Console.Read() 等待用户输入，接收后返回一个整数值，该值对应于输入字符的字符代码。如果在获取输入之前输入流以某种方式结束，则返回 -1。

第19.3节：Console.ReadKey()

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

Console.ReadKey()等待用户输入，接收后返回一个ConsoleKeyInfo类的对象，该对象包含用户输入字符的相关信息。有关提供信息的详细内容，访问MSDN文档。

第19.4节：命令行提示符的原型

```
模块 MainPrompt
Public Const PromptSymbol 作为字符串 = "TLA > "
Public Const ApplicationTitle 作为字符串 = GetType(Project.BaseClass).Assembly.FullName
REM 或者你可以使用自定义的字符串
REM Public Const ApplicationTitle 作为字符串 = "应用程序的简称"

Sub Main()
    Dim Statement 作为字符串
    Dim BrokenDownStatement 作为字符串()
    Dim Command 作为字符串
    Dim Args 作为字符串()
    Dim Result 作为字符串

Console.ForegroundColor = ConsoleColor.Cyan
    Console.Title = ApplicationTitle & " 命令行控制台"

    Console.WriteLine("欢迎使用 " & ApplicationTitle & " 控制台前端")
    Console.WriteLine("此软件包版本为 " &
GetType(Project.BaseClass).Assembly.GetName().Version.ToString)
   Console.WriteLine()
    Console.Write(PromptSymbol)

    当条件为真时循环
语句 = Console.ReadLine()
      拆分后的语句 = 语句.Split(" ")
        重新定义 参数(BrokenDownStatement.Length - 1)
      命令 = 拆分后的语句(0)
```

Chapter 19: Console

Section 19.1: Console.ReadLine()

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

`Console.ReadLine()` will read the console input from the user, up until the next newline is detected (usually upon pressing the Enter or Return key). Code execution is paused in the current thread until a newline is provided. Afterwards, the next line of code will be executed.

Section 19.2: Console.Read()

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

`Console.Read()` awaits input from the user and, upon receipt, returns an integer value corresponding with the character code of the entered character. If the input stream is ended in some way before input can be obtained, -1 is returned instead.

Section 19.3: Console.ReadKey()

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

`Console.ReadKey()` awaits input from the user and, upon receipt, returns an object of class `ConsoleKeyInfo`, which holds information relevant to the character which the user provided as input. For detail regarding the information provided, visit the MSDN documentation.

Section 19.4: Prototype of command line prompt

```
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)
        对于 i = 1 到 拆分后的语句.Length - 1
          参数(i - 1) = 拆分后的语句(i)
        Next

        选择情况 命令.ToLower
            情况 "example"
结果 = 执行某操作(Example)
            情况 "exit", "quit"
                退出循环
            情况 "ver"
结果 = "此包的版本是 " &
GetType(Project.BaseClass).Assembly.GetName().Version.ToString
            其他情况
结果 = "无法识别的命令: -" & 命令 & "-"
        结束选择
Console.WriteLine(" " & Result)
        Console.Write(PromptSymbol)
    Loop

Console.WriteLine("我正在退出，时间是 " & DateTime.Now.ToString("u"))
    Console.WriteLine("再见")
Environment.Exit(0)
End Sub
End Module
```

该原型生成一个基本的命令行解释器。

它会自动获取应用程序名称和版本以便与用户通信。对于每一行输入，它识别命令和任意数量的参数，所有参数均以空格分隔。

作为一个基本示例，该代码理解ver、quit和exit命令。

参数Project.BaseClass是项目中的一个类，用于设置程序集的详细信息。

第19.5节：Console.WriteLine()

```
Dim x As Int32 = 128
Console.WriteLine(x) ' 变量 '
Console.WriteLine(3) ' 整数 '
Console.WriteLine(3.14159) ' 浮点数 '
Console.WriteLine("Hello, world") ' 字符串 '
Console.WriteLine(myObject) ' 输出调用 myObject.ToString() 的值
```

Console.WriteLine() 方法会打印给定的参数，并在末尾附加一个换行符。这将打印任何提供的对象，包括但不限于字符串、整数、变量、浮点数。

当写入未被各种 WriteLine 重载显式调用的对象时（也就是说，你使用的是期望 Object 类型值的重载），WriteLine 会使用 .ToString() 方法生成一个字符串来实际写入。你的自定义对象应重写 .ToString 方法，生成比默认实现（通常只是写出完全限定的类型名称）更有意义的内容。

```
        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
```

This prototype generate a basic command line interpreter.

It automatically get the application name and version to communicate to the user. For each input line, it recognize the command and an arbitrary list of arguments, all separated by space.

As a basic example, this code understand *ver*, *quit* and *exit* commands.

The parameter *Project.BaseClass* is a class of your project where the Assembly details are set.

Section 19.5: 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()
```

The `Console.WriteLine()` method will print the given argument(s) **with** a newline attached at the end. This will print any object supplied, including, but not limited to, strings, integers, variables, floating-point numbers.

When writing objects that are not explicitly called out by the various `WriteLine` overloads (that is, you are using the overload that expects a value of type `Object`, WriteLine will use the `.ToString()` method to generate a `String` to actually write. Your custom objects should OverRide the `.ToString` method and produce something more meaningful than the default implementation (which typically just writes the fully qualified type name).

第20章：函数

函数就像子程序（sub），但函数会返回一个值。函数可以接受单个或多个参数。

第20.1节：定义函数

定义函数非常简单。

```
Function GetAreaOfARectangle(ByVal Edge1 As Integer, ByVal Edge2 As Integer) As Integer
    Return Edge1 * Edge2
结束函数
```

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

第20.2节：定义函数 #2

```
Function Age(ByVal YourAge As Integer) As String
    Select Case YourAge
        案例是 < 18
            返回("你未满18岁！你是青少年！")
        情况 18 到 64
            返回("你超过18岁但未满65岁！你是成年人！")
        情况 是 >= 65
            返回("你超过65岁！你是老年人！")
    结束选择
结束函数
```

```
Console.WriteLine(Age(48)) '输出：你超过18岁但未满65岁！你是成年人！
```

Chapter 20: Functions

The function is just like sub. But function returns a value. A function can accept single or multiple parameters.

Section 20.1: Defining a Function

It's really easy to define the functions.

```
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
```

Section 20.2: Defining a Function #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!
```

第21章：递归

第21.1节：计算第n个斐波那契数

Visual Basic.NET，像大多数语言一样，允许递归，即函数在某些

条件下调用自身的过程。

这是一个用Visual Basic .NET编写的基本函数，用于计算斐波那契数。

```
''' <summary>
''' 获取第n个斐波那契数
''' </summary>
                        ''' <param name="n">您希望获取的斐波那契数列的1起始索引序号。
前提条件：必须大于或等于1。</param>
''' <returns>第n个斐波那契数。如果违反前提条件则抛出异常。</returns>
Public Shared Function Fibonacci(ByVal n as Integer) as Integer
    If n<1
        Throw New ArgumentOutOfRangeException("n必须大于或等于1。")
    End If
    If (n=1) or (n=2)
                        ''基本情况。根据定义，前两个斐波那契数（n=1和n=2）均为1。
        Return 1
    结束如果
    ''递归情况。
    ''通过递归获取前两个斐波那契数，将它们相加并返回结果。
    Return Fibonacci(n-1) + Fibonacci(n-2)
End Function
```

该函数的工作原理是首先检查函数调用时参数 n 是否等于1或2。根据定义，斐波那契数列的前两个值是1和1，因此无需进一步计算即可确定这一点。如果 n 大于2，我们就无法轻易查找对应的值，但我们知道任何这样的斐波那契数都是前两个数之和，因此我们通过递归（调用我们自己的斐波那契函数）来获取这两个数。由于连续的递归调用会通过减1和减2传入越来越小的数字，我们知道最终它们会达到小于2的数字。一旦达到这些条件（称为基例），调用栈开始回退，我们就得到了最终结果。

Chapter 21: Recursion

Section 21.1: Compute nth Fibonacci number

Visual Basic.NET, like most languages, permits recursion, a process by which a function calls *itself* under certain conditions.

Here is a basic function in Visual Basic .NET to compute Fibonacci numbers.

```
''' <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
```

This function works by first checking if the function has been called with the parameter n equal to 1 or 2. By definition, the first two values in the Fibonacci sequence are 1 and 1, so no further computation is necessary to determine this. If n is greater than 2, we cannot look up the associated value as easily, but we know that any such Fibonacci number is equal to the sum of the prior two numbers, so we request those via *recursion* (calling our own Fibonacci function). Since successive recursive calls get called with smaller and smaller numbers via decrements of -1 and -2, we know that eventually they will reach numbers that are smaller than 2. Once those conditions (called *base cases*) are reached, the stack unwinds and we get our final result.

第22章：随机

Random类用于生成非负的伪随机整数，这些整数不是真正的随机数，但对于一般用途来说足够接近。

序列是使用一个初始数字（称为种子（Seed））计算的。在早期版本的.net中，每次运行应用程序时这个种子数字都是相同的。因此，每次执行应用程序时都会得到相同的伪随机数序列。现在，种子基于对象声明的时间。

第22.1节：声明实例

```
Dim rng As New Random()
```

这声明了一个名为 rng 的Random类实例。在这种情况下，使用对象创建时的当前时间来计算种子。这是最常见的用法，但正如我们稍后在备注中将看到的，它也有自身的问题。

你可以指定初始种子数字，而不是让程序使用当前时间作为初始种子数字计算的一部分。这个种子可以是任何32位整数字面量、常量或变量。下面有示例。这样做意味着你的实例将生成相同的伪随机数序列，这在某些情况下是有用的。

```
Dim rng As New Random(43352)
```

或者

```
Dim rng As New Random(x)
```

其中 x 已在程序的其他地方声明为整数常量或变量。

第22.2节：从Random类的实例生成随机数

下面的示例声明了Random类的新实例，然后使用.Next方法生成伪随机数序列中的下一个数字。

```
Dim rnd As New Random
Dim x As Integer
x = rnd.Next
```

上面最后一行将生成下一个伪随机数并赋值给 x。该数字的范围是0到2147483647。然而，你也可以像下面的示例中那样指定生成数字的范围。

```
x = rnd.Next(15, 200)
```

但请注意，使用这些参数时，生成的数字范围是15或以上，且不超过199。

你也可以使用.NextDouble生成Double类型的浮点数，例如

Chapter 22: Random

The Random class is used to generate non-negative pseudo-random integers that are not truly random, but are for general purposes close enough.

The sequence is calculated using an initial number (called the **Seed**) In earlier versions of .net, this seed number was the same every time an application was run. So what would happen was that you would get the same sequence of pseudo-random numbers every time the application was executed. Now, the seed is based on the time the object is declared.

Section 22.1: Declaring an instance

```
Dim rng As New Random()
```

This declares an instance of the Random class called `rng`. In this case, the current time at the point where the object is created is used to calculate the seed. This is the most common usage, but has its own problems as we shall see later in the remarks

Instead of allowing the program to use the current time as part of the calculation for the initial seed number, you can specify the initial seed number. This can be any 32 bit integer literal, constant or variable. See below for examples. Doing this means that your instance will generate the same sequence of pseudo-random numbers, which can be useful in certain situations.

```
Dim rng As New Random(43352)
```

or

```
Dim rng As New Random(x)
```

where `x` has been declared elsewhere in your program as an Integer constant or variable.

Section 22.2: Generate a random number from an instance of Random

The following example declares a new instance of the Random class and then uses the method `.Next` to generate the next number in the sequence of pseudo-random numbers.

```
Dim rnd As New Random
Dim x As Integer
x = rnd.Next
```

The last line above will generate the next pseudo-random number and assign it to `x`. This number will be in the range of 0 - 2147483647. However, you can also specify the range of numbers to be generated as in the example below.

```
x = rnd.Next(15, 200)
```

Please note however, that using these parameters, range of numbers will be between 15 or above and 199 or below.

You can also generate floating point numbers of the type Double by using `.NextDouble` e.g

```
Dim rnd As New Random
Dim y As Double
y = rnd.NextDouble()
```

但是，您无法指定此范围。它始终在0.0到小于1.0的范围内。

```
Dim rnd As New Random
Dim y As Double
y = rnd.NextDouble()
```

You cannot however specify a range for this. It will always be in the range of 0.0 to less than 1.0.

第23章：类

类将不同的函数、方法、变量和属性组合在一起，这些称为它的成员。类封装了这些成员，可以通过类的实例（称为对象）来访问。类对程序员非常有用，因为它们使任务变得方便且快速，具有模块化、可重用性、可维护性和代码可读性等特点。

类是面向对象编程语言的构建模块。

第23.1节：抽象类

如果类共享公共功能，可以将其归组到基类或抽象类中。抽象类可以包含部分实现或完全不实现，并允许派生类型重写基类的实现。

在VisualBasic.NET中，抽象类必须声明为MustInherit，且不能被实例化。

```
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
     结束函数
结束类
```

子类型可以继承此抽象类，如下所示：

```
公共类 车
    继承 车辆
结束类
```

车将继承车辆中声明的所有类型，但只能根据底层访问修饰符访问它们。

```
声明 车 作为新 车()
车.显示轮数()
```

在上述示例中，创建了一个新的车实例。然后调用显示轮数()方法，该方法将调用基类车辆的实现。

第23.2节：创建类

类提供了一种在.NET框架内创建自定义类型的方法。在类定义中，您可以包含以下内容：

- 字段
- 属性
- 方法
- 构造函数

Chapter 23: Classes

A class groups different functions, methods, variables, and properties, which are called its members. A class encapsulates the members, which can be accessed by an instance of the class, called an object. Classes are extremely useful for the programmer, as they make the task convenient and fast, with characteristics such as modularity, re-usability, maintainability, and readability of the code.

Classes are the building blocks of object-oriented programming languages.

Section 23.1: Abstract Classes

If classes share common functionality you can group this in a base or abstract class. Abstract classes can contain partial or no implementation at all and allow the derived type to override the base implementation.

Abstract classes within VisualBasic.NET must be declared as `MustInherit` and cannot be instantiated.

```
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
```

A sub type can then `inherit` this abstract class as shown below:

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

Car will inherit all of the declared types within vehicle, but can only access them based upon the underlying access modifier.

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

In the above example a new Car instance is created. The `DisplayWheelCount()` method is then invoked which will call the base class `Vehicles` implementation.

Section 23.2: Creating classes

Classes provide a way of creating your own types within the .NET framework. Within a class definition you may include the following:

- Fields
- Properties
- Methods
- Constructors

- 事件

要声明一个类，您可以使用以下语法：

```
公共类 车辆
结束类
```

其他 .NET 类型可以封装在类中并相应地暴露，如下所示：

```
公共类 车辆
     私有属性 _车轮数量 作为 整数
     私有属性 _发动机排量 作为 整数

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

     Public Function DisplayWheelCount() As Integer
         Return _numberOfWheels
     结束函数
结束类
```

- Events

To declare a class you use the following syntax:

```
Public Class Vehicle
End Class
```

Other .NET types can be encapsulated within the class and exposed accordingly, as shown below:

```
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
```

第24章：泛型

第24.1节：创建泛型类

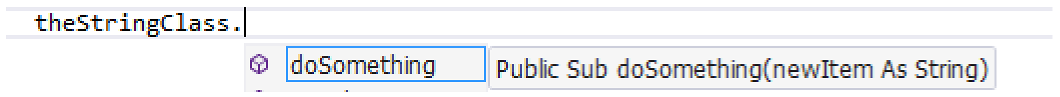
创建泛型类型以适应，使相同的功能可以用于不同的数据类型。

```
公共类 某类(泛型参数 T)
    公共子程序 执行操作(新项目 作为 T)
        定义 临时项目 作为 T
        ' 插入处理类型为 T 的数据项的代码。
    结束子程序
结束类
```

第24.2节：泛型类的实例

通过创建同一类的不同类型实例，类的接口会根据所给类型而变化。

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



```
theStringClass.
doSomething   Public Sub doSomething(newItem As String)
```

第24.3节：定义“泛型”类

泛型类是一种适应后续给定类型的类，以便为不同类型提供相同功能。

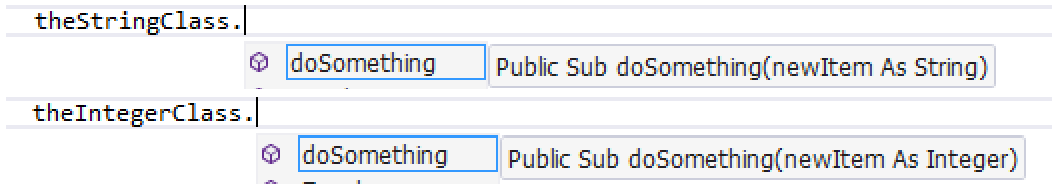
在这个基本示例中创建了一个泛型类。它有一个使用泛型类型T的子程序。在编写此类时，我们不知道T的类型。在这种情况下，T具有Object的所有特性。

```
公共类 某类(泛型参数 T)
    公共子程序 执行操作(新项目 作为 T)
        定义 临时项目 作为 T
        ' 插入处理类型为 T 的数据项的代码。
    结束子程序
结束类
```

第24.4节：使用泛型类

在此示例中，创建了SomeClass类的两个实例。根据所给类型，这两个实例具有不同的接口：

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



```
theStringClass.
doSomething   Public Sub doSomething(newItem As String)
theIntegerClass.
doSomething   Public Sub doSomething(newItem As Integer)
```

Chapter 24: Generics

Section 24.1: Create a generic class

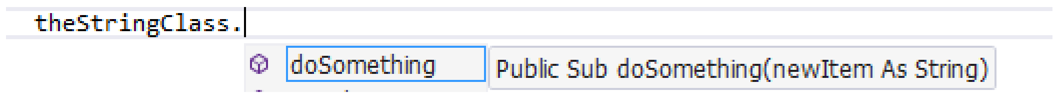
A generic type is created to adapt so that the same functionallity can be accessible for different data types.

```
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
```

Section 24.2: Instance of a Generic Class

By creating an instance of the same class with a different type given, the interface of the class changes depending on the given type.

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



```
theStringClass.
doSomething   Public Sub doSomething(newItem As String)
```

Section 24.3: Define a 'generic' class

A generic class is a class who adapts to a later-given type so that the same functionality can be offered to different types.

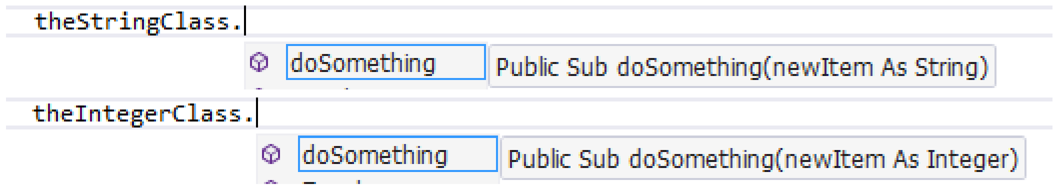
In this basic example a generic class is created. It has a sub who uses the generic type T. While programming this class, we don't know the type of T. In this case T has all the characteristics of Object.

```
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
```

Section 24.4: Use a generic class

In this example there are 2 instances created of the SomeClass Class. Depending on the type given the 2 instances have a different interface:

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



```
theStringClass.
doSomething   Public Sub doSomething(newItem As String)
theIntegerClass.
doSomething   Public Sub doSomething(newItem As Integer)
```

最著名的泛型类是List(of)

第24.5节：限制可给定的类型

传递给SomeClass新实例的可能类型必须继承SomeBaseClass。这也可以是一个接口。在此类定义中可以访问SomeBaseClass的特性。

```
公共类 SomeClass(泛型 T 继承自 SomeBaseClass)
    公共子程序 DoSomething(newItem 类型为 T)
        newItem.DoSomethingElse()
        ' 插入处理类型为 t 的数据项的代码。
    End Sub
结束类

公共类 SomeBaseClass
    公共子程序 DoSomethingElse()
    结束子程序
结束类
```

第24.6节：创建给定类型的新实例

创建泛型类型的新实例可以在编译时完成/检查。

```
公共类 SomeClass(泛型 T 作为 {新建})
    公共函数 GetInstance() 作为 T
        返回 新建 T
    结束函数
结束类
```

或者限定类型：

```
公共类 SomeClass(泛型 T 作为 {新建, SomeBaseClass})
    公共函数 GetInstance() 作为 T
        返回 新建 T
    结束函数
结束类

公共类 SomeBaseClass
结束类
```

基类（如果未指定，则为 Object）必须有一个无参数的构造函数。

这也可以通过反射在运行时完成

The most famous generic class is List(of)

Section 24.5: Limit the possible types given

The possible types passed to a new instance of SomeClass must inherit SomeBaseClass. This can also be an interface. The characteristics of SomeBaseClass are accessible within this class definition.

```
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
```

Section 24.6: Create a new instance of the given type

Creating a new intance of a generic type can be done/checed at compile time.

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

Or with limited types:

```
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
```

The baseClass (if none given it is Object) must have a parameter less constructor.

This can also be done at runtime through reflection

第25章：一次性对象

第25.1节：IDisposable的基本概念

每当你实例化一个实现了IDisposable接口的类时，使用完毕后应调用该类的.Dispose方法。这允许该类清理它可能使用的任何托管或非托管依赖项。不这样做可能会导致内存泄漏。

Using关键字确保会调用.Dispose，而无需你显式调用它。

例如，没有使用Using时：

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

现在使用Using：

```
Using sr As New StreamReader("C:\foo.txt")
    Dim line = sr.ReadLine
End Using '此处会为你调用.Dispose
```

使用的一个主要优点是当抛出异常时，因为它确保调用.Dispose。

考虑以下情况。如果抛出异常，你需要记得调用.Dispose，但你可能还需要检查对象的状态，以确保不会出现空引用错误等。

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

使用using块意味着你不必记得去做这些操作，并且你可以在try内部声明你的对象：

```
    Try
        Using sr As New StreamReader("C:\foo.txt")
            Dim line = sr.ReadLine
        结束 Using
    Catch ex As Exception
        '此时sr已被释放
    End Try
```

1 我是否总是需要对我的DbContext对象调用Dispose()？不需要

第25.2节：在一个Using中声明多个对象

有时，你需要连续创建两个Disposable对象。有一种简单的方法可以避免嵌套Using块。

这段代码

Chapter 25: Disposable objects

Section 25.1: Basic concept of IDisposable

Any time you instantiate a class that Implements `IDisposable`, you should call `.Dispose`1 on that class when you have finished using it. This allows the class to clean up any managed or unmanaged dependencies that it may be using. Not doing this could cause a memory leak.

The **Using** keyword ensures that `.Dispose` is called, without you having to *explicitly* call it.

For example without **Using**:

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

Now with **Using**:

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

One major advantage **Using** has is when an exception is thrown, because it *ensures* `.Dispose` is called.

Consider the following. If an exception is thrown, you need to need to remember to call .Dispose but you might also have to check the state of the object to ensure you don't get a null reference error, etc.

```
    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
```

A using block means you don't have to remember to do this and you can declare your object inside the **try**:

```
    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 Do I always have to call Dispose() on my DbContext objects? Nope

Section 25.2: Declaring more objects in one Using

Sometimes, you have to create two `Disposable` objects in a row. There is an easy way to avoid nesting **Using** blocks.

This code

```
使用 File 作为新的 FileStream("MyFile", FileMode.Append)
    使用 Writer 作为新的 BinaryWriter(File)
        你的代码在这里
Writer.Write("Hello")
    End Using
End Using
```

可以简化成这一段。主要优点是你减少了一个缩进层级：

```
使用 File 作为新的 FileStream("MyFile", FileMode.Append), Writer 作为新的 BinaryWriter(File)
    你的代码在这里
Writer.Write("Hello")
结束 Using
```

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

can be shortened into this one. The main advantage is that you gain one indentation level:

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

第26章：NullReferenceException（空引用异常）

第26.1节：空返回

```
函数 TestFunction() 作为 TestClass
    返回无
结束函数
```

错误代码

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

正确代码

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

版本 = 14.0

空条件运算符

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

第26.2节：未初始化变量

错误代码

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

正确代码

```
Dim f As System.Windows.Forms.Form = New System.Windows.Forms.Form
' Dim f As New System.Windows.Forms.Form ' 另一种语法
f.ShowModal()
```

更好的代码（确保正确处理 IDisposable 对象 更多信息）

```
使用 f 作为 System.Windows.Forms.Form = 新建 System.Windows.Forms.Form
' 使用 f 作为新的 System.Windows.Forms.Form ' 另一种语法
    f.ShowModal()
结束 Using
```

Chapter 26: NullReferenceException

Section 26.1: Empty Return

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

BAD CODE

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

GOOD CODE

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

Version = 14.0

Null Conditional Operator

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

Section 26.2: Uninitialized variable

BAD CODE

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

GOOD 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()
```

EVEN BETTER CODE (Ensure proper disposal of IDisposable object more info)

```
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
```

第27章：Using 语句

第27.1节：参见可释放对象下的示例

IDisposable 的基本概念

Chapter 27: Using Statement

Section 27.1: See examples under Disposable objects

Basic concept of IDisposable

第28章：Option Strict

第28.1节：为什么使用它？

Option Strict 开启 防止以下三种情况发生：

1. 隐式窄化转换错误

它防止你将值赋给精度较低或容量较小的变量（缩小转换）
而不进行显式转换。这样做会导致数据丢失。

```
Dim d As Double = 123.4
Dim s As Single = d '启用 Option Strict 时此行无法编译
```

2. 迟绑定调用

不允许迟绑定。这是为了防止拼写错误导致代码能编译但运行时失败

```
Dim obj As New Object
obj.Foo '启用 Option Strict 时此行无法编译
```

3. 隐式对象类型错误

这防止变量被推断为 Object 类型，而实际上它们应该被声明为具体类型

```
Dim something = Nothing. '启用 Option Strict 时此行无法编译
```

结论

除非需要进行晚绑定，否则应始终启用Option Strict On，因为它会将上述错误生成编译时错误，而非运行时异常。

如果必须进行晚绑定，可以选择

- 将所有晚绑定调用封装到一个类/模块中，并在代码文件顶部使用`Option Strict Off`（这是首选方法，因为它减少了其他文件中拼写错误的可能性），或者
- 指定晚绑定不会导致编译失败（项目属性 > 编译选项卡 > 警告配置）

第28.2节：如何启用

- 可以通过在代码文件顶部放置指令，在模块/类级别启用它。

```
Option Strict On
```

- 也可以通过Visual Studio菜单在项目级别启用

> 项目 > [项目] 属性 > 编译选项卡 > Option Strict > 开启

Chapter 28: Option Strict

Section 28.1: Why Use It?

`Option Strict On` prevents three things from happening:

1. Implicit Narrowing Conversion Errors

It prevents you from assigning to a variable that has *less precision or smaller capacity* (a narrowing conversion) without an explicit cast. Doing so would result in data loss.

```
Dim d As Double = 123.4
Dim s As Single = d 'This line does not compile with Option Strict On
```

2. Late Binding Calls

Late binding is not allowed. This is to prevent typos that would compile, but fail at runtime

```
Dim obj As New Object
obj.Foo 'This line does not compile with Option Strict On
```

3. Implicit Object Type Errors

This prevents variable being inferred as an Object when in fact they should have been declared as a type

```
Dim something = Nothing. 'This line does not compile with Option Strict On
```

Conclusion

Unless you need to do late binding, you should always have `Option Strict On` as it will cause the mentioned errors to generate compile time errors instead of runtime exceptions.

If you *have* to do late binding, you can *either*

- Wrap all your late binding calls into one class/module and use `Option Strict Off` at the top of the code file (this is the preferred method as it reduces the likelihood of a typos in other files), *or*
- Specify that Late Binding does not cause a compilation failure (`Project Properties > Compile Tab > Warning Configuration`)

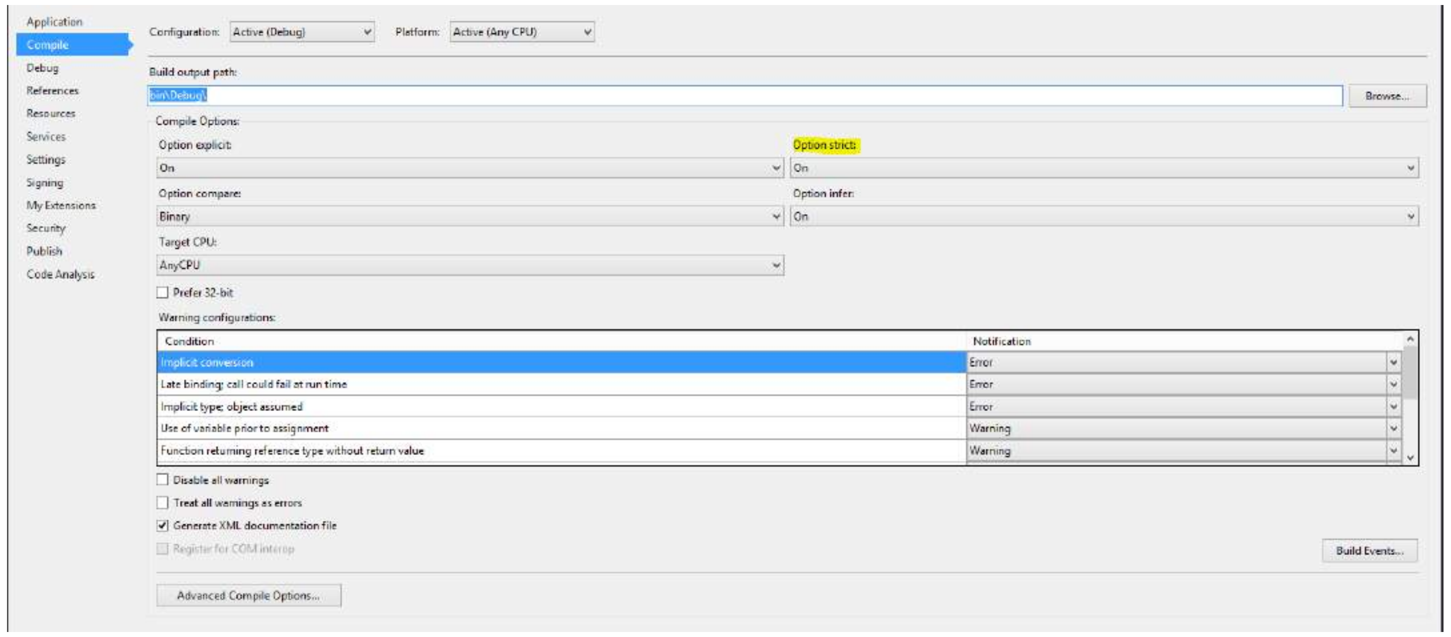
Section 28.2: How to Switch It On

- You can switch it On at the Module/Class Level by placing the directive at the top of the code file.

```
Option Strict On
```

- You can switch it on at the project level via the menu in Visual Studio

> Project > [Project] Properties > Compile Tab > Option Strict > On



Configuration: Active (Debug)
Platform: Active (Any CPU)

Build output path: bin\Debug\

Compile Options:

Option explicit: On
Option strict: On
Option compare: Binary
Option infer: On
Target CPU: AnyCPU

☐ Prefer 32-bit

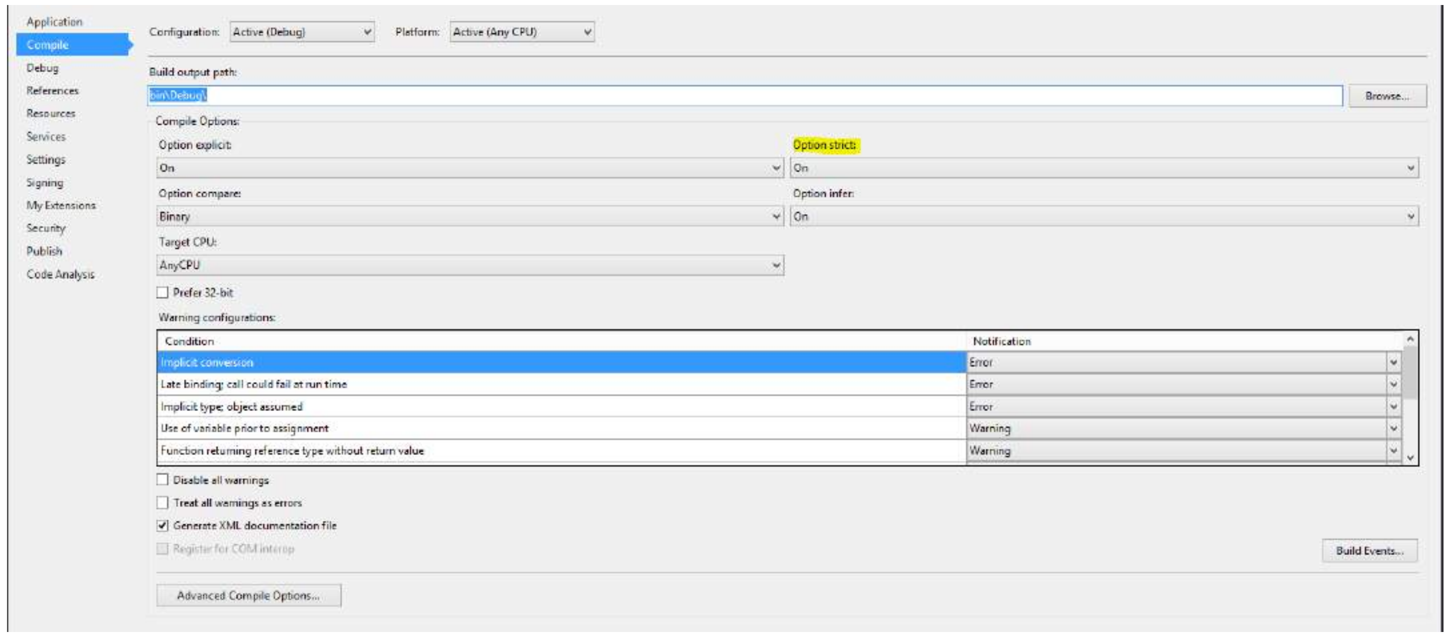
Warning configurations:

Condition	Notification
Implicit conversion	Error
Late binding; call could fail at run time	Error
Implicit type; object assumed	Error
Use of variable prior to assignment	Warning
Function returning reference type without return value	Warning

☐ Disable all warnings
☐ Treat all warnings as errors
☑ Generate XML documentation file
☐ Register for COM interop

- 您可以通过选择以下选项将其默认开启，适用于所有新项目：

> 工具 > 选项 > 项目和解决方案 > VB 默认值 > Option Strict
> 将其设置为开启。



Configuration: Active (Debug)
Platform: Active (Any CPU)

Build output path: bin\Debug\

Compile Options:

Option explicit: On
Option strict: On
Option compare: Binary
Option infer: On
Target CPU: AnyCPU

☐ Prefer 32-bit

Warning configurations:

Condition / Notification
Implicit conversion — Error
Late binding; call could fail at run time — Error
Implicit type; object assumed — Error
Use of variable prior to assignment — Warning
Function returning reference type without return value — Warning

☐ Disable all warnings
☐ Treat all warnings as errors
☑ Generate XML documentation file
☐ Register for COM interop

- You can switch it On by default for all new Projects by selecting:

> Tools > Options > Projects and Solutions > VB defaults > Option Strict
> Set it to On.

第29章：Option Explicit

第29.1节：它是什么？

它强制您显式声明所有变量。

显式声明变量和隐式声明变量有什么区别？

显式声明变量：

```
Dim anInteger As Integer = 1234
```

隐式声明变量：

```
'未使用 Dim 声明 aNumber
aNumber = 1234
```

结论

因此，你应该始终启用`Option Explicit On`，因为在赋值时可能会拼写错误变量，
这会导致程序行为异常。

第29.2节：如何开启？

文档级别

默认情况下是开启的，但你可以通过在代码文件顶部放置Option Explicit On来增加一层保护。该选项将应用于整个文档。

项目级别

你可以通过Visual Studio菜单开启它：

> 项目 > [项目] 属性 > 编译选项卡 > Option Explicit

在下拉菜单中选择On。该选项将应用于整个文档。

所有新项目

您可以通过选择以下选项将其默认开启，适用于所有新项目：

> 工具 > 选项 > 项目和解决方案 > VB 默认值 > 明确声明变量

在下拉菜单中选择开启。

Chapter 29: Option Explicit

Section 29.1: What is it?

It forces you to explicitly declare all variables.

What is the difference between explicitly declaring and implicitly declaring a variable?

Explicitly declaring a variable:

```
Dim anInteger As Integer = 1234
```

Implicitly declaring a variable:

```
'Did not declare aNumber using Dim
aNumber = 1234
```

Conclusion

Therefore, you should always have `Option Explicit On` as you could misspel a variable during assignment, which cause your program to behave unexpectedly.

Section 29.2: How to switch it on?

Document level

It is on by default, but you can have an extra layer of protection by placing `Option Explicit On` at the top of the code file. The option will apply to the whole document.

Project level

You can switch it on via the menu in Visual Studio:

> Project > [Project] Properties > Compile Tab > Option Explicit

Choose `On` in the drop-down menu. The option will apply to the whole document.

All new projects

You can switch it On by default for all new Projects by selecting:

> Tools > Options > Projects and Solutions > VB defaults > Option Explicit

Choose `On` in the drop-down menu.

第30章：Option Infer

第30.1节：如何启用/禁用

文档级别

默认情况下是开启的，但你可以通过在代码文件顶部放置`Option Infer On|Off`来设置。该选项将应用于整个文档。

项目级别

你可以通过Visual Studio中的菜单来切换开关：

> 项目 > [项目] 属性 > 编译选项卡 > Option infer

在下拉菜单中选择开启|关闭。该选项将应用于整个文档。

所有新项目

您可以通过选择以下选项将其默认开启，适用于所有新项目：

> 工具 > 选项 > 项目和解决方案 > VB 默认值 > Option Infer

在下拉菜单中选择开启|关闭。

第30.2节：它是什么？

启用在声明变量时使用局部类型推断。

什么是类型推断？

您可以声明局部变量而无需显式指定数据类型。编译器根据变量初始化表达式的类型推断变量的数据类型。

Option Infer 开启:

```
Dim aString  = "1234" '--> 编译器将其视为 String 类型
Dim aNumber  = 4711   '--> 编译器将其视为 Integer 类型
```

与显式类型声明相比：

```
'显式声明类型
Dim aString as String = "1234"
Dim aNumber as Integer = 4711
```

Option Infer 关闭：
Option Infer 关闭时，编译器的行为取决于已在此处记录的 Option Strict 设置。

- **关闭选项推断 - 关闭严格选项**
  所有未显式声明类型的变量均声明为Object类型。

Chapter 30: Option Infer

Section 30.1: How to enable/disable it

Document level

It is on by default, but you can set it by placing `Option Infer On|Off` at the top of the code file. The option will apply to the whole document.

Project level

You can switch it on/off via the menu in Visual Studio:

> Project > [Project] Properties > Compile Tab > Option infer

Choose `On|Off` in the drop-down menu. The option will apply to the whole document.

All new projects

You can switch it On by default for all new Projects by selecting:

> Tools > Options > Projects and Solutions > VB defaults > Option Infer

Choose `On|Off` in the drop-down menu.

Section 30.2: What is it?

Enables the use of local type inference in declaring variables.

What is type inference?

You can declare local variables without explicitly stating a data type. The compiler infers the data type of a variable from the type of its initialization expression.

Option Infer On:

```
Dim aString  = "1234" '--> Will be treated as String by the compiler
Dim aNumber  = 4711   '--> Will be treated as Integer by the compiler
```

vs. explicit type declaration:

```
'State a type explicitly
Dim aString as String = "1234"
Dim aNumber as Integer = 4711
```

Option Infer Off:
The compiler behavior with `Option Infer Off` depends on the `Option Strict` setting which is already documented here.

- **Option Infer Off - Option Strict Off**
  All variables without explicit type declarations are declared as `Object`.

```
Dim aString  = "1234" '--> 编译器将其视为Object类型
```

- **Option Infer Off - Option Strict On**
  编译器不会允许你声明一个没有显式类型的变量。

```
'Dim aString  = "1234" '--> 由于声明中缺少类型，无法编译
```

第30.3节：何时使用类型推断

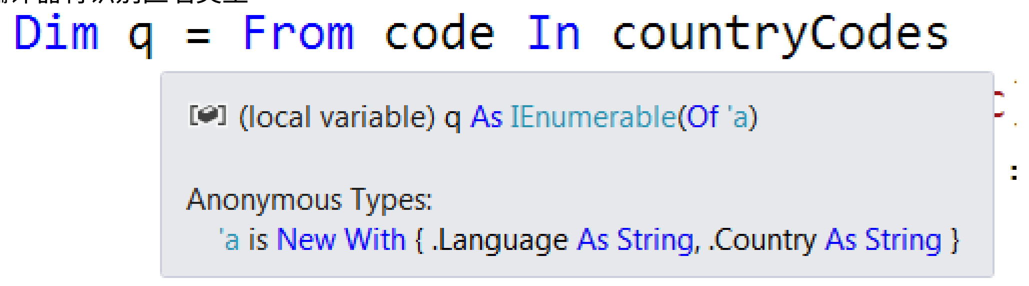
基本上，只要可能，你都可以使用类型推断。
但是，当Option Infer Off和Option Strict Off同时使用时要小心，因为这可能导致不期望的运行时行为：

```
Dim someVar = 5
someVar.GetType.ToString() '--> System.Int32
someVar = "abc"
someVar.GetType.ToString() '--> System.String
```

匿名类型
匿名类型只能在Option Infer On 时声明。
它们通常用于处理 LINQ 时：

```
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 On**
  编译器将识别匿名类型：



```
Dim q = From code In countryCodes
(local variable) q As IEnumerable(Of 'a)

Anonymous Types:
  'a is New With { .Language As String, .Country As String }
```

- **Option Infer 关闭**
  编译器要么会抛出错误（当Option Strict On时），要么会将q视为object类型（当Option Strict Off时）。
  这两种情况都会导致无法使用匿名类型的结果。

双精度/十进制数
带小数的数值变量默认会被推断为Double类型：

```
Dim aNumber = 44.11 '--> 编译器会将其视为`Double`类型
```

如果需要其他类型如Decimal，则初始化变量的值需要加以标记：

```
Dim aString  = "1234" '--> Will be treated as Object by the compiler
```

- **Option Infer Off - Option Strict On**
  The compiler won´t let you declare a variable without an explicit type.

```
'Dim aString  = "1234" '--> Will not compile due to missing type in declaration
```

Section 30.3: When to use type inference

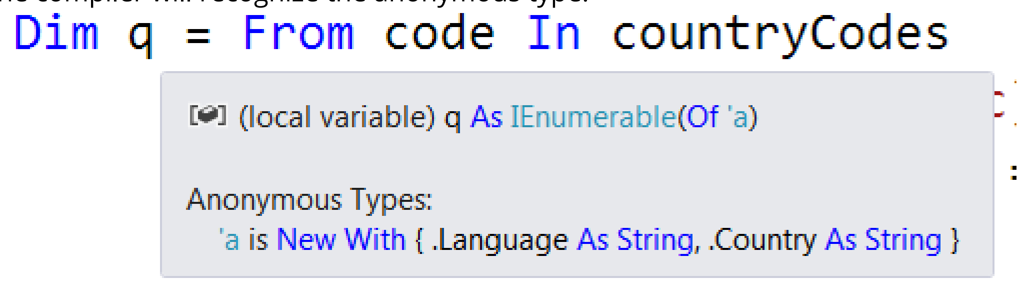
Basically you can use type inference whenever it is possible.
However, be careful when combining `Option Infer Off` and `Option Strict Off`, as this can lead to undesired run-time behavior:

```
Dim someVar = 5
someVar.GetType.ToString() '--> System.Int32
someVar = "abc"
someVar.GetType.ToString() '--> System.String
```

Anonymous Type
Anonymous types can **only** be declared with `Option Infer On`.
They are often used when dealing with LINQ:

```
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 On**
  The compiler will recognize the anonymous type:



```
Dim q = From code In countryCodes
(local variable) q As IEnumerable(Of 'a)

Anonymous Types:
  'a is New With { .Language As String, .Country As String }
```

- **Option Infer Off**
  The compiler will either throw an error (with `Option Strict On`)
  or will consider q as type `object` (with `Option Strict Off`).
  Both cases will produce the outcome that you cannot use the anonymous type.

Doubles/Decimals
Numeric variables with decimal places will be infered as `Double` by default:

```
Dim aNumber = 44.11 '--> Will be treated as type `Double` by the compiler
```

If another type like `Decimal` is desired the value which initialized the variable needs to be marked:

```
Dim mDecimal = 47.11D '--> 编译器会将其视为`Decimal`类型
```

```
Dim mDecimal = 47.11D '--> Will be treated as type `Decimal` by the compiler
```

第31章：错误处理

第31.1节：Try...Catch...Finally语句

结构：

```
Try
    您的程序将尝试运行此代码块中的代码。
    如果抛出任何异常，Catch 块中的代码将被执行，
    且不会执行导致异常的那一行之后的代码。
Catch ex As System.IO.IOException
    如果在处理 Try 块时发生异常，将按文本顺序检查每个 Catch 语句，
    以确定哪个语句处理该异常。
    例如，此 Catch 块处理 IOException。
Catch ex As Exception
    此 Catch 块处理所有 Exception 类型的异常。
    异常的详细信息在此情况下存储于变量“ex”中。
    您可以使用以下代码行在消息框中显示错误。
    MessageBox.Show(ex.Message)
最终
    “Finally”块总是会被执行，无论是否发生异常。
结束 Try
```

示例代码：

```
Try
    Dim obj = Nothing
    Dim prop = obj.Name '这行会抛出 NullReferenceException

    Console.WriteLine("Test.") ' 这行不会被执行
Catch ex As System.IO.IOException
    ' 处理 IOException 的代码。
Catch ex As NullReferenceException
    ' 处理 NullReferenceException 的代码
   Console.WriteLine("NullReferenceException: " & ex.Message)
   Console.WriteLine("堆栈跟踪: " & ex.StackTrace)
Catch ex As Exception
    ' 处理其他异常的代码。
Finally
    ' 无论是否抛出异常，这部分代码都会执行。
Console.WriteLine("完成")
结束 Try
```

第31.2节：创建自定义异常并抛出

您可以创建自定义异常并在函数执行过程中抛出。一般来说，只有当函数无法实现其定义的功能时，才应抛出异常。

```
私有函数 OpenDatabase(Byval Server as String, Byval User as String, Byval Pwd as String)
    if Server.trim="" then
        抛出新 异常("服务器名称不能为空")
    elseif User.trim ="" then
        抛出新 异常("用户名不能为空")
    elseif Pwd.trim="" then
        抛出新 异常("密码不能为空")
    endif

    '此处添加连接服务器的代码
```

Chapter 31: Error Handling

Section 31.1: Try...Catch...Finally Statement

Structure:

```
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
```

Example Code:

```
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
```

Section 31.2: Creating custom exception and throwing

You can create a custom exception and throw them during the execution of your function. As a general practice you should only throw an exception when your function could not achieve its defined functionality.

```
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
```

```
结束函数
```

第31.3节：数据库操作中的Try Catch

您可以使用Try..Catch通过将回滚语句放在Catch段来回滚数据库操作。

```
    尝试
        '执行数据库操作...        '
xCmd.CommandText = "INSERT into ...."
        xCmd.ExecuteNonQuery()

objTrans.Commit()
        conn.Close()
    Catch ex As Exception
        '当出现异常时的回滚操作
        objTrans.Rollback()
conn.Close()
    结束 Try
```

第31.4节：无法捕获的异常

虽然Catch ex As Exception 声称它可以处理所有异常——但有一个例外（无意双关）。

```
Imports System
Static Sub StackOverflow() ' 同样无意双关
    StackOverflow()
End Sub
Static Sub Main()
    Try
        StackOverflow()
    Catch ex As Exception
Console.WriteLine("捕获异常！")
    Finally
Console.WriteLine("Finally 块")
    End Try
End Sub
```

哎呀......发生了未捕获的System.StackOverflowException，但控制台甚至没有打印出任何内容！
根据MSDN，

> 从.NET Framework 2.0开始，你可以通过try/catch块捕获StackOverflowException对象，但对应的进程默认会被终止。因此，你应该编写代码来检测并防止堆栈溢出。

所以，System.StackOverflowException是不可捕获的。请注意！

第31.5节：关键异常

通常大多数异常并不那么严重，但有一些非常严重的异常你可能无法处理，比如著名的System.StackOverflowException。然而，还有一些可能被Catch ex As Exception隐藏的异常，比如System.OutOfMemoryException，

System.BadImageFormatException 和 System.InvalidProgramException。 如果无法正确处理它们，最好编程时将其排除。要过滤这些异常，我们需要一个辅助方法：

```
End function
```

Section 31.3: Try Catch in Database Operation

You can use Try..Catch to rollback database operation by placing the rollback statement at the Catch Segment.

```
    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
```

Section 31.4: The Un-catchable Exception

Although `Catch ex As Exception` claims that it can handle all exceptions - there are one exception (no pun intended).

```
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
```

Oops... There is an un-caught `System.StackOverflowException` while the console didn't even print out anything!
According to MSDN,

> Starting with the .NET Framework 2.0, you can't catch a StackOverflowException object with a try/catch block, and the corresponding process is terminated by default. Consequently, you should write your code to detect and prevent a stack overflow.

So, `System.StackOverflowException` is un-catchable. Beware of that!

Section 31.5: Critical Exceptions

Generally most of the exceptions are not that critical, but there are some really serious exceptions that you might not be capable to handle, such as the famous `System.StackOverflowException`. However, there are others that might get hidden by `Catch ex As Exception`, such as `System.OutOfMemoryException`, `System.BadImageFormatException` and `System.InvalidProgramException`. It is a good programming practice to leave these out if you cannot correctly handle them. To filter out these exceptions, we need a helper method:

```
公共共享函数 IsCritical(ex 作为 Exception) 作为 布尔值
    返回 TypeOf ex 是 OutOfMemoryException 或否则
           TypeOf ex 是 AppDomainUnloadedException 或否则
           TypeOf ex 是 AccessViolationException 或否则
           TypeOf ex 是 BadImageFormatException 或否则
           TypeOf ex 是 CannotUnloadAppDomainException 或否则
           TypeOf ex 是 ExecutionEngineException 或否则 ' 已废弃，但最好包含
           TypeOf ex 是 InvalidProgramException 或否则
           TypeOf ex 是 System.Threading.ThreadAbortException
结束函数
```

用法：

```
Try
    SomeMethod()
捕获 ex 作为 Exception 当 不 IsCritical(ex)
    Console.WriteLine("捕获异常: " & ex.Message)
结束尝试
```

```
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
```

Usage:

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

第32章：面向对象编程关键字

第32.1节：定义类

类是面向对象编程的重要方面。类就像对象的“蓝图”。对象具有类的属性，但这些特征并不在类本身中定义。由于每个对象可能不同，它们定义自己的特征。

```
公共类 人员
结束类

公共类 客户
结束类
```

一个类也可以包含*子类*。子类继承其父类的相同属性和行为，但可以拥有自己独特的属性和类。

第32.2节：继承修饰符（针对类）

继承

指定基类（或父类）

```
公共类 人员
结束类

公共类 客户
    继承 人员

结束类

'单行表示法
公共类 学生 : 继承 人员
结束类
```

可能的对象：

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

不可继承

防止程序员将该类用作基类。

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

可能的对象：

```
Dim p As New Person
```

必须继承

指定该类仅用于作为基类。（抽象类）

```
Public MustInherit Class Person
```

Chapter 32: OOP Keywords

Section 32.1: Defining a class

Classes are vital aspects of OOP. A class is like the "blueprint" of an object. An object has the properties of a class, but the characteristics are not defined within the class itself. As each object can be different, they define their own characteristics.

```
Public Class Person
End Class

Public Class Customer
End Class
```

A class can also contain *subclasses*. A subclass inherits the same properties and behaviors as its parent class, but can have its own unique properties and classes.

Section 32.2: Inheritance Modifiers (on classes)

Inherits

Specifies the base (or parent) class

```
Public Class Person
End Class

Public Class Customer
    Inherits Person

End Class

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

Possible objects:

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

NotInheritable

Prevents programmers from using the class as a base class.

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

Possible objects:

```
Dim p As New Person
```

MustInherit

Specifies that the class is intended for use as a base class only. (Abstract class)

```
Public MustInherit Class Person
```

```
结束类

Public Class Customer
    Inherits Person
End Class
```

可能的对象：

```
Dim c As New Customer
```

第32.3节：继承修饰符（用于属性和方法）

可重写

允许在派生类中重写类中的属性或方法。

```
公共类 Person（人）
    公共可重写子程序 DoSomething()
        Console.WriteLine("Person（人）")
    结束子程序
结束类
```

重写

重写基类中定义的可重写属性或方法。

```
公共类 客户
    继承 人员

    '基类必须是可重写的'
    公共重写子程序 DoSomething()
        Console .WriteLine（Customer（客户）")
    结束子程序
结束类
```

不可重写

防止属性或方法在继承类中被重写。默认行为。只能在重写方法上声明

```
公共类 Person（人）

    公共可重写子程序 DoSomething()
        Console.WriteLine("Person（人）")
    结束子程序

结束类

公共类 客户
    继承 人员

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

结束类

公共类 详细客户
    继承 客户
```

```
End Class

Public Class Customer
    Inherits Person
End Class
```

Possible objects:

```
Dim c As New Customer
```

Section 32.3: Inheritance Modifiers (on properties and methods)

Overridable

Allows a property or method in a class to be overridden in a derived class.

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

Overrides

Overrides an Overridable property or method defined in the base class.

```
Public Class Customer
    Inherits Person

    'Base Class must be Overridable
    Public Overrides Sub DoSomething()
        Console.WriteLine("Customer")
    End Sub
End Class
```

NotOverridable

Prevents a property or method from being overridden in an inheriting class. Default behaviour. Can only be declared on **overrides methods**

```
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 不能被重写
结束类
```

示例用法：

```
Dim p 作为新 人员
p.DoSomething()

Dim c 作为新 客户
c.DoSomething()

Dim d 作为新 详细客户
d.DoSomething()
```

输出：

```
人员
客户
客户
```

必须重写

要求派生类重写该属性或方法。

必须重写的方法必须声明在必须继承类中。

```
Public MustInherit Class Person

    公共必须重写子程序 DoSomething()
    '此处无方法定义

结束类

公共类 客户
    继承 人员

    必须重写 DoSomething
    公共重写子程序 DoSomething()
        Console .WriteLine (Customer (客户) ")
    结束子程序

结束类
```

示例用法：

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

输出：

```
客户
```

第 32.4 节：MyBase

MyBase 关键字的行为类似于一个对象变量，指向当前类实例的基类。

```
Public Class Person
    Public Sub DoSomething()
```

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

Example Usage:

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

Dim c As New Customer
c.DoSomething()

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

Output:

```
Person
Customer
Customer
```

MustOverride

Requires that a derived class override the property or method.

MustOverride methods must be declared in **MustInherit classes.**

```
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
```

Example Usage:

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

Output:

```
Customer
```

Section 32.4: MyBase

The MyBase keyword behaves like an object variable that refers to the base class of the current instance of a class.

```
Public Class Person
    Public Sub DoSomething()
```

```
Console.WriteLine("Person")
    End Sub
结束类

公共类 客户
    继承 人员

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

结束类
```

使用示例：

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

Console.WriteLine("----")

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

输出：

```
Person
----
Person
Person
```

第32.5节：Me 与 MyClass

Me 使用当前对象实例。

MyClass 使用调用成员所在类中的成员定义

```
类 Person
    公共可重写子程序 DoSomething()
        Console.WriteLine("Person（人）")
    结束子程序

    Public Sub useMe()
        Me.DoSomething()
    End Sub

    Public Sub useMyClass()
        MyClass.DoSomething()
    End Sub
结束类

Class Customer
    Inherits Person

    公共重写子程序 DoSomething()
        Console .WriteLine（"Customer（客户）")
    结束子程序
结束类
```

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

Public Class Customer
    Inherits Person

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

End Class
```

Usage example:

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

Console.WriteLine("----")

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

Output:

```
Person
----
Person
Person
```

Section 32.5: Me vs MyClass

Me uses the current object instance.

MyClass uses the memberdefinition in the class where the member is called

```
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
```

示例用法：

```
Dim c As New Customer
c.useMe()
c.useMyClass()
```

输出：

```
Customer
Person
```

第32.6节：重载

重载是在一个类中创建多个过程、实例构造函数或属性，这些成员具有相同的名称但参数类型不同。

```
类 Person
    重载子程序 Display(ByVal theChar As Char)
        ' 添加显示字符数据的代码。
    End Sub

    重载子程序 Display(ByVal theInteger As Integer)
        ' 添加显示整数数据的代码。
    End Sub

    重载子程序 Display(ByVal theDouble As Double)
        ' 添加显示双精度浮点数数据的代码。
    结束子程序
结束类
```

第32.7节：遮蔽

它重新声明一个不可重写的成员。只有对实例的调用会受到影响。基类内部的代码不会受到影响。

```
公共类 Person
    公共子程序 DoSomething()
        Console.WriteLine("Person")
    结束子程序


    公共子程序 UseMe()
        Me.DoSomething()
    结束子程序
结束类
公共类 客户
    继承 人员
    公共遮蔽子程序 DoSomething()
        Console.WriteLine("Customer")
    结束子程序

结束类
```

示例用法：

```
Dim p 作为新的 Person
Dim c 作为新的 Customer
```

Example Usage:

```
Dim c As New Customer
c.useMe()
c.useMyClass()
```

Output:

```
Customer
Person
```

Section 32.6: Overloading

Overloading is the creation of more than one procedure, instance constructor, or property in a class with the same name but different argument types.

```
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
```

Section 32.7: Shadows

It redeclares a member that is not overridable. Only calls to the instance will be affected. Code inside the base classes will not be affected by this.

```
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
```

Example usage:

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

```
p.UseMe()
c.UseMe()
Console.WriteLine("----")
p.DoSomething()
c.DoSomething()
```

输出：

```
Person
Person
----
Person
Customer
```

陷阱:

示例1，通过泛型创建一个新对象。将使用哪个函数？

```
Public Sub CreateAndDoSomething(Of T 作为 {Person, New})()
    Dim obj 作为新的 T
    obj.DoSomething()
End Sub
```

示例用法：

```
Dim p As New Person
p.DoSomething()
Dim s As New Student
s.DoSomething()
Console.WriteLine("----")
CreateAndDoSomething(Of Person)()
CreateAndDoSomething(Of Student)()
```

输出：直觉上结果应该相同，但事实并非如此。

```
Person
Student
----
Person
Person
```

示例 2：

```
Dim p As Person
Dim s As New Student
p = s
p.DoSomething()
s.DoSomething()
```

输出：直觉上你可能会认为 p 和 s 是相等的，并且表现也会相同。但事实并非如此。

```
人物
学生
```

在这个简单的例子中，很容易理解阴影的奇怪行为。但在现实生活中，它带来了许多

```
p.UseMe()
c.UseMe()
Console.WriteLine("----")
p.DoSomething()
c.DoSomething()
```

Output:

```
Person
Person
----
Person
Customer
```

Pitfalls:

Example1, Creating a new object through a generic. Which function will be used??

```
Public Sub CreateAndDoSomething(Of T As {Person, New})()
    Dim obj As New T
    obj.DoSomething()
End Sub
```

example usage:

```
Dim p As New Person
p.DoSomething()
Dim s As New Student
s.DoSomething()
Console.WriteLine("----")
CreateAndDoSomething(Of Person)()
CreateAndDoSomething(Of Student)()
```

Output: By intuition the result should be the same. Yet that is not true.

```
Person
Student
----
Person
Person
```

Example 2:

```
Dim p As Person
Dim s As New Student
p = s
p.DoSomething()
s.DoSomething()
```

Output: By intuition you could think that p and s are equal and will behave equal. Yet that is not true.

```
Person
Student
```

In this simple examples it is easy to learn the strange behaviour of Shadows. But in real-life it brings a lot of

惊喜。建议避免使用阴影。应尽可能使用其他替代方案
（重写等..）

第32.8节：接口

```
公共接口 IPerson
    子程序 DoSomething()
接口结束

公共类 客户
    实现 IPerson
    公共子程序 DoSomething() 实现 IPerson.DoSomething
        Console.WriteLine("客户")
    结束子程序

结束类
```

surprises. It is advisably to prevent the usage of shadows. One should use other alternatives as much as possible (overrides etc..)

Section 32.8: Interfaces

```
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
```

第33章：扩展方法

第33.1节：创建扩展方法

扩展方法对于扩展我们不拥有的库的行为非常有用。

由于编译器的语法糖，它们的使用方式类似于实例方法：

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

    '扩展方法直接在对象上调用。
stringBuilder.AppendIf(true, "条件为真")

    '扩展方法作为常规方法调用。这违背了扩展方法的初衷
    '但应注意这是可能的。
AppendIf(stringBuilder, true, "条件为真")

End Sub

<Extension>
公共函数 AppendIf(stringBuilder 作为 StringBuilder, condition 作为 布尔值, text 作为 字符串) 作为
StringBuilder
    如果(条件) 则 stringBuilder.Append(text)

    返回 stringBuilder
结束函数
```

为了拥有一个可用的扩展方法，该方法需要Extension属性并且需要声明在一个
模块中。

第33.2节：通过扩展方法使语言更具函数式

扩展方法的一个良好用法是使语言更具函数式

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

    字符串.Join(Environment.NewLine).Print()
End Sub

<Extension>
公共函数 Join(字符串 作为 IEnumerable(Of String), 分隔符 作为 String) 作为 String
    返回 String.Join(分隔符, 字符串)
结束函数

<Extension>
公共子程序 Print(文本 作为 String)
    Console.WriteLine(文本)
结束子程序
```

第33.3节：从强名称获取程序集版本

作为扩展方法和常规方法调用扩展方法的示例。

```
public Class MyClass
```

Chapter 33: Extension methods

Section 33.1: Creating an extension method

Extension methods are useful to extend the behaviour of libraries we don't own.

They are used similar to instance methods thanks to the compiler's syntactic sugar:

```
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
```

To have a usable extension method, the method needs the `Extension` attribute and needs to be declared in a **`Module`**.

Section 33.2: Making the language more functional with extension methods

A good use of extension method is to make the language more functional

```
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
```

Section 33.3: Getting Assembly Version From Strong Name

Example of calling an extension method as an extension and as a regular method.

```
public Class MyClass
```

```
  Sub Main()

          '扩展方法直接在对象上调用。
          Dim Version = Assembly.GetExecutingAssembly.GetVersionFromAssembly()

          '作为常规方法调用。
          Dim Ver = GetVersionFromAssembly(SomeOtherAssembly)

     结束子程序
结束类
```

模块中的扩展方法。如果扩展被编译为dll并将在另一个程序集引用，则将模块设为Public。

```
Public Module Extensions
     ''' <summary>
                                        ''' 使用程序集的强名称返回指定程序集的版本号。
     '''
     ''' <param name="Assy">[Assembly] 要获取版本信息的程序集。</param>
     ''' <returns>[String]</returns>
     <Extension>
     Friend Function GetVersionFromAssembly(ByVal Assy As Assembly) As String
         Return Split(Split(Assy.FullName,",")(1),"=")(1)
     结束函数
结束模块
```

第33.4节：数字填充

```
公共模块 用法
  公共子程序 LikeThis()
    定义 iCount 为 整数
    定义 sCount 为 字符串

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

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



公共模块 填充
  <Extension>
  公共函数 PadLeft(值 为 整数, 长度 为 整数) 为 字符串
    返回 值.PadLeft(长度, Space(长度))
  结束函数



  <Extension>
  公共函数 PadRight(值 为 整数, 长度 为 整数) 为 字符串
    返回 值.PadRight(长度, Space(长度))
  结束函数



  <Extension>
  公共函数 PadLeft(值 为 整数, 长度 为 整数, 字符 为 字符)为 字符串
```

```
  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
```

The Extension Method in a Module. Make the Module Public if extensions are compiled to a dll and will be referenced in another assembly.

```
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
```

Section 33.4: 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
```

```
    返回 CStr(值).PadLeft(长度, 字符)
  结束函数



  <Extension>
  Public Function PadRight(Value As Integer, Length As Integer, Character As Char) As String
    Return CStr(Value).PadRight(Length, Character)
  结束函数
结束模块
```

```
    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
```

第34章：反射

第34.1节：获取类实例的属性

```
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
    结束函数

    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
```

GetProperties 的参数定义了函数将返回哪种类型的属性。由于我们传递了
Public 和 Instance，该方法将仅返回既是公共的又是非共享的属性。有关如何组合标志枚举的说明，请参见 Flags
属性。

第34.2节：获取方法并调用它

静态方法：

```
Dim parseMethod = GetType(Integer).GetMethod("Parse",{GetType(String)})
Dim result = DirectCast(parseMethod.Invoke(Nothing,{"123"}), Integer)
```

实例方法：

```
 Dim instance = "hello".ToUpper
 Dim method = Gettype(String).GetMethod("ToUpper",{})
 Dim result = method.Invoke(instance,{})
Console.WriteLine(result)  'HELLO
```

第34.3节：创建泛型类型的实例

```
    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")
```

第34.4节：获取某个类型的成员

```
Dim flags = BindingFlags.Static Or BindingFlags.Public Or BindingFlags.Instance
Dim members = GetType(String).GetMembers(flags)
```

Chapter 34: Reflection

Section 34.1: Retrieve Properties for an Instance of a Class

```
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
```

The Parameter of GetProperties defines which kinds of Properties will be returned by the function. Since we pass Public and Instance, the method will return only properties that are both public and non-shared. See The Flags attribute for and explanation on how Flag-enums can be combined.

Section 34.2: Get a method and invoke it

Static method:

```
Dim parseMethod = GetType(Integer).GetMethod("Parse",{GetType(String)})
Dim result = DirectCast(parseMethod.Invoke(Nothing,{"123"}), Integer)
```

Instance method:

```
 Dim instance = "hello".ToUpper
 Dim method = Gettype(String).GetMethod("ToUpper",{})
 Dim result = method.Invoke(instance,{})
 Console.WriteLine(result) 'HELLO
```

Section 34.3: Create an instance of a generic type

```
    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")
```

Section 34.4: Get the members of a type

```
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
```

```
For Each member In members
    Console.WriteLine($"{member.Name}, ({member.MemberType})")
Next
```

第35章：Visual Basic 14.0 特性

Visual Basic 14 是作为 Visual Studio 2015 一部分发布的 Visual Basic 版本。

该版本从头重写，约有130万行 VB 代码。添加了许多功能以消除常见的烦恼，并使常见的编码模式更简洁。

Visual Basic 的版本号直接从12跳到14，跳过了13。这是为了使 VB 的版本号与 Visual Studio 本身的版本号保持一致。

第35.1节：空条件运算符

为了避免冗长的空值检查，语言中引入了?.操作符。

旧的冗长语法：

```
如果 myObject 不为 Nothing 并且 myObject.Value >= 10 则
```

现在可以用简洁的语法替代：

```
如果 myObject?.Value >= 10 则
```

? 操作符在你有一连串属性时特别强大。考虑以下情况：

```
声明 fooInstance 为 Foo = Nothing
声明 s 为 字符串
```

通常你需要写成这样：

```
如果 fooInstance 不为 Nothing 并且 fooInstance.BarInstance 不为 Nothing 则
    s = fooInstance.BarInstance.Baz
否则
s = Nothing
结束如果
```

但使用? 操作符，这可以简化为：

```
s = fooInstance?.BarInstance?.Baz
```

第35.2节：字符串插值

这个新特性使字符串拼接更易读。该语法将被编译为等效的
String.Format调用。

没有字符串插值：

```
String.Format("Hello, {0}", name)
```

使用字符串插值：

```
$"Hello, {name}"
```

这两行代码是等价的，都会被编译成对String.Format的调用。

Chapter 35: Visual Basic 14.0 Features

Visual Basic 14 is the version of Visual Basic that was shipped as part of Visual Studio 2015.

This version was rewritten from scratch in about 1.3 million lines of VB. Many features were added to remove common irritations and to make common coding patterns cleaner.

The version number of Visual Basic went straight from 12 to 14, skipping 13. This was done to keep VB in line with the version numbering of Visual Studio itself.

Section 35.1: Null conditional operator

To avoid verbose null checking, the `?.` operator has been introduced in the language.

The old verbose syntax:

```
If myObject IsNot Nothing AndAlso myObject.Value >= 10 Then
```

Can be now replaced by the concise:

```
If myObject?.Value >= 10 Then
```

The ? operator is particularly powerful when you have a chain of properties. Consider the following:

```
Dim fooInstance As Foo = Nothing
Dim s As String
```

Normally you would have to write something like this:

```
If fooInstance IsNot Nothing AndAlso fooInstance.BarInstance IsNot Nothing Then
    s = fooInstance.BarInstance.Baz
Else
    s = Nothing
End If
```

But with the ? operator this can be replaced with just:

```
s = fooInstance?.BarInstance?.Baz
```

Section 35.2: String interpolation

This new feature makes the string concatenation more readable. This syntax will be compiled to its equivalent `String.Format` call.

Without string interpolation:

```
String.Format("Hello, {0}", name)
```

With string interpolation:

```
$"Hello, {name}"
```

The two lines are equivalent and both get compiled to a call to `String.Format`.

与String.Format中一样，括号内可以包含任何单一表达式（方法调用、属性、空合并运算符等）。

字符串插值是优于String.Format的方法，因为它可以防止某些运行时错误的发生。请考虑以下String.Format语句：

```
String.Format("人数是 {0}/{1}", numPeople)
```

这段代码可以编译，但会导致运行时错误，因为编译器不会检查参数数量是否与占位符匹配。

第35.3节：只读自动属性

在VB.NET中，只读属性一直可以用以下格式实现：

```
公共类 Foo

  私有 _MyProperty 作为 字符串 = "Bar"

  公共只读属性 MyProperty 作为 字符串
    获取
      返回 _MyProperty
    结束获取
  结束 属性

结束类
```

新版Visual Basic允许用简写方式声明属性，如下所示：

```
公共类 Foo

  公共只读属性 MyProperty 作为 字符串 = "Bar"

结束类
```

编译器生成的实际实现对于这两个例子完全相同。新的写法只是简写。编译器仍然会生成一个私有字段，格式如下
：
_<PropertyName> 用于支持只读属性。

第35.4节：NameOf 操作符

NameOf 操作符在编译时解析命名空间、类型、变量和成员名称，并将其替换为字符串等价物。

使用场景之一：

```
Sub MySub(variable As String)
    If variable Is Nothing Then Throw New ArgumentNullException("variable")
End Sub
```

旧语法存在重命名变量但硬编码字符串未更改导致错误值的风险。

```
Sub MySub(variable As String)
    If variable Is Nothing Then Throw New ArgumentNullException(NameOf(variable))
End Sub
```

As in String.Format, the brackets can contain any single expression (call to a method, property, a null coalescing operator et cetera).

String Interpolation is the preferred method over String.Format because it prevents some runtime errors from occurring. Consider the following String.Format line:

```
String.Format("The number of people is {0}/{1}", numPeople)
```

This will compile, but will cause a runtime error as the compiler does not check that the number of arguments match the placeholders.

Section 35.3: Read-Only Auto-Properties

Read-only properties were always possible in VB.NET in this format:

```
Public Class Foo

  Private _MyProperty As String = "Bar"

  Public ReadOnly Property MyProperty As String
    Get
      Return _MyProperty
    End Get
  End Property

End Class
```

The new version of Visual Basic allows a short hand for the property declaration like so:

```
Public Class Foo

  Public ReadOnly Property MyProperty As String = "Bar"

End Class
```

The actual implementation that is generated by the compiler is exactly the same for both examples. The new method to write it is just a short hand. The compiler will still generate a private field with the format: _<PropertyName> to back the read-only property.

Section 35.4: NameOf operator

The NameOf operator resolves namespaces, types, variables and member names at compile time and replaces them with the string equivalent.

One of the use cases:

```
Sub MySub(variable As String)
    If variable Is Nothing Then Throw New ArgumentNullException("variable")
End Sub
```

The old syntax will expose the risk of renaming the variable and leaving the hard-coded string to the wrong value.

```
Sub MySub(variable As String)
    If variable Is Nothing Then Throw New ArgumentNullException(NameOf(variable))
End Sub
```

使用NameOf时，仅重命名变量会引发编译错误。这也允许重命名工具一次性重命名两者。

NameOf 操作符只使用括号中引用的最后一个组件。这在处理NameOf操作符中的命名空间时非常重要。

```
Imports System

模块 模块1
    子程序 WriteIO()
Console.WriteLine(NameOf(IO)) '显示 "IO"
        Console.WriteLine(NameOf(System.IO)) '显示 "IO"
    End Sub
End Module
```

运算符还使用输入的引用名称，而不解析任何名称更改的导入。例如：

```
Imports OldList = System.Collections.ArrayList

Module Module1
    Sub WriteList()
Console.WriteLine(NameOf(OldList)) '显示 "OldList"
        Console.WriteLine(NameOf(System.Collections.ArrayList)) '显示 "ArrayList"
    End Sub
End Module
```

第35.5节：多行字符串字面量

VB现在允许跨多行的字符串字面量。

旧语法：

```
Dim text As String = "Line1" & Environment.NewLine & "Line2"
```

新语法：

```
Dim text As String = "Line 1
Line 2"
```

第35.6节：部分模块和接口

类似于部分类，Visual Basic的新版本现在能够处理部分模块和部分
接口。语法和行为与部分类完全相同。

部分模块示例：

```
部分模块 Module1
  子程序 Main()
Console.Write("Ping -> ")
   TestFunktion()
  End Sub
End Module

部分模块 Module1
  私有子程序 TestFunktion()
   Console.WriteLine("Pong")
```

With NameOf, renaming the variable only will raise a compiler error. This will also allow the renaming tool to rename both with a single effort.

The NameOf operator only uses the last component of the reference in the brackets. This is important when handling something like namespaces in the NameOf operator.

```
Imports System

Module Module1
    Sub WriteIO()
        Console.WriteLine(NameOf(IO)) 'displays "IO"
        Console.WriteLine(NameOf(System.IO)) 'displays "IO"
    End Sub
End Module
```

The operator also uses the name of the reference that is typed in without resolving any name changing imports. For example:

```
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
```

Section 35.5: Multiline string literals

VB now allows string literals that split over multiple lines.

Old syntax:

```
Dim text As String = "Line1" & Environment.NewLine & "Line2"
```

New syntax:

```
Dim text As String = "Line 1
Line 2"
```

Section 35.6: Partial Modules and Interfaces

Similar to partial classes the new version of Visual Basic is now able to handle partial modules and partial interfaces. The syntax and behaviour is exactly the same as it would be for partial classes.

A partial module example:

```
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
```

以及一个部分接口：

```
部分接口 Interface1
  子程序 Methode1 ()
接口结束

部分接口 Interface1
  子程序 Methode2 ()
接口结束

公共类 Class1
  实现 Interface1
  公共子程序 Methode1() 实现 Interface1.Methode1
    抛出新 NotImplementedException()
  End Sub

  公共子程序 Methode2() 实现 Interface1.Methode2
    抛出新 NotImplementedException()
  结束子程序
结束类
```

就像部分类一样，部分模块和接口的定义必须位于相同的命名空间和相同的程序集内。这是因为模块和接口的部分在编译过程中会被合并，编译后的程序集不包含任何表明模块或接口原始定义被拆分的标记。

第35.7节：隐式换行后的注释

VB 14.0引入了在隐式换行后添加注释的功能。

```
Dim number =
    From c 作为字符 '注释
    在 "dj58kwd92n4" '注释
    Where Char.IsNumber(c) '注释
    Select c '注释
```

第35.8节：#Region指令的改进

#Region指令现在可以放置在方法内部，甚至可以跨越方法、类和模块。

```
#Region "跨越类并在模块的方法内部结束的区域"
   公共类 FakeClass
    '这里没什么可看的，只是一个假类。
   结束类

   模块 Extensions

    ''' <summary>
    ''' 检查文件或目录的路径，若存在则返回[TRUE]。
    '''
    ''' <param name="Path">[字符串] 要检查的文件或目录路径。</param>
    ''' <returns>[布尔值]</returns>
    <Extension>
    公共函数 PathExists(ByVal Path As String) As Boolean
        如果 My.Computer.FileSystem.FileExists(Path) 则返回 True
```

```
  End Sub
End Module
```

And a partial interface:

```
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
```

Just like for partial classes the definitions for the partial modules and interfaces have to be located in the same namespace and the same assembly. This is because the partial parts of the modules and interfaces are merged during the compilation and the compiled assembly does not contain any indication that the original definition of the module or interface was split.

Section 35.7: Comments after implicit line continuation

VB 14.0 introduces the ability to add comments after implicit line continuation.

```
Dim number =
    From c As Char 'Comment
    In "dj58kwd92n4" 'Comment
    Where Char.IsNumber(c) 'Comment
    Select c 'Comment
```

Section 35.8: #Region directive improvements

#Region directive can now be placed inside methods and can even span over methods, classes and modules.

```
#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
        如果 My.Computer.FileSystem.DirectoryExists(Path) 则返回 True
        返回 False
    结束函数

    ''' <summary>
                              ''' 使用程序集的强名称返回指定程序集的版本号。
    '''
    ''' <param name="Assy">[Assembly] 要获取版本信息的程序集。</param>
    ''' <returns>[String]</returns>
    <Extension>
    Friend 函数 GetVersionFromAssembly(ByVal Assy As Assembly) As String
#End Region
        返回 Split(Split(Assy.FullName, “,”)(1), “=”)(1)
    结束函数
End Module
```

```
        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
```

第36章：LINQ

LINQ（语言集成查询）是一种从数据源检索数据的表达式。LINQ通过提供一个一致的模型来简化这一情况，使得可以跨各种类型的数据源和格式处理数据。在LINQ查询中，你始终处理的是对象。你使用相同的基本编码模式来查询和转换XML文档、SQL数据库、ADO.NET数据集、.NET集合以及任何有LINQ提供程序支持的其他格式中的数据。

第36.1节：使用简单条件从数组中选择

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

查询将是一个可枚举对象，包含Stack Overflow和Super User。查询中的x是迭代变量，Where子句将检查的每个对象存储在这里。

第36.2节：通过Select子句映射数组

```
Dim sites() As String = {"Stack Overflow",
                         "Super User",
                         "Ask Ubuntu",
                         "Hardware Recommendations"}
Dim query = From x In sites Select x.Length
' 结果 = 14, 10, 10, 24
```

查询结果将是一个可枚举对象，包含输入数组中字符串的长度。在此示例中，这些值分别是14、10、10、24。查询中的x是迭代变量，用于存储输入数组中的每个对象。

第36.3节：输出排序

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

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

' 结果为 "Super User", "Ask Ubuntu", "Stack Overflow", "Hardware Recommendations"
```

OrderBy子句根据子句返回的值对输出进行排序。在此示例中，排序依据是每个字符串的长度。默认输出顺序为升序。如果需要降序，可以在子句后指定Descending关键字。

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

第36.4节：从IEnumerable生成字典

```
' 仅设置示例
公共类 A
    Public Property ID 作为整数
    Public Property Name 作为字符串
```

Chapter 36: LINQ

LINQ (Language Integrated Query) is an expression that retrieves data from a data source. LINQ simplifies this situation by offering a consistent model for working with data across various kinds of data sources and formats. In a LINQ query, you are always working with objects. You use the same basic coding patterns to query and transform data in XML documents, SQL databases, ADO.NET Datasets, .NET collections, and any other format for which a LINQ provider is available.

Section 36.1: Selecting from array with simple condition

```
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"
```

Query will be enumerable object containing `Stack Overflow` and `Super User`. `x` in the query is iterating variable where will be stored each object checked by `Where` clause.

Section 36.2: Mapping array by Select clause

```
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
```

Query result will be enumerable object containing lengths of strings in input array. In this example this would be values 14, 10, 10, 24. x in the query is iterating variable where will be stored each object from the input array.

Section 36.3: Ordering output

```
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"
```

OrderBy clause orders the output by the value returned from the clause. In this example it is Length of each string. Default output order is ascending. If you need descending you could specify `Descending` keyword after clause.

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

Section 36.4: Generating Dictionary From IEnumerable

```
' Just setting up the example
Public Class A
    Public Property ID as integer
    Public Property Name as string
    Public Property OtherValue 作为对象
结束类

Public Sub 示例()
    '设置项目列表
    Dim originalList 作为新 List(Of A)
originalList.Add(New A() 带有 {.ID = 1, .Name = "项目 1", .OtherValue = "项目 1 值"})
    originalList.Add(New A() 带有 {.ID = 2, .Name = "项目 2", .OtherValue = "项目 2 值"})
    originalList.Add(New A() 带有 {.ID = 3, .Name = "项目 3", .OtherValue = "项目 3 值"})

    '根据 ID 将列表转换为字典
    Dim dict 作为 Dictionary(Of Integer, A) = originalList.ToDictionary(function(c) c.ID, function(c)
c)

    '从字典访问值
console.Write(dict(1).Name) ' 输出 "项目 1"
    console.Write(dict(1).OtherValue) ' 输出 "项目 1 值"
End Sub
```

第36.5节：投影

```
示例数据
Dim sample = {1, 2, 3, 4, 5}

使用“查询语法”
Dim squares = From number In sample Select number * number

使用“方法语法”的同样操作
Dim squares = sample.Select (Function (number) number * number)
```

我们也可以一次投影多个结果

```
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})
```

第36.6节：获取不同值（使用 Distinct 方法）

```
Dim duplicateFruits = New List(Of String) From {"葡萄", "苹果", "葡萄", "苹果", "葡萄"}
此时，duplicateFruits.Length = 5

Dim uniqueFruits = duplicateFruits.Distinct();
现在，uniqueFruits.Count() = 2
如果此时遍历它，将包含各一个“葡萄”和“苹果”
```

```
    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
```

Section 36.5: Projection

```
' 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)
```

We can project multiple result at once too

```
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})
```

Section 36.6: Getting distinct values (using the Distinct method)

```
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"
```

第37章：FTP服务器

第37.1节：从FTP服务器下载文件

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

此命令从名为server.my的服务器下载myfile.txt文件，并将其保存为donwloaded_file.txt到工作目录。你可以为下载的文件指定绝对路径。

第37.2节：登录验证时从FTP服务器下载文件

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

此命令从名为srv.my的服务器下载myfile.txt文件，并将其保存为donwload.txt到工作目录。文件由用户Peter使用密码1234下载。

第37.3节：上传文件到FTP服务器

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

此命令将工作目录中的example.txt文件（如果需要，可以指定绝对路径）上传到名为server.my的服务器。存储在服务器上的文件名将为server_example.txt。

第37.4节：登录时上传文件到FTP服务器

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

此命令将工作目录中的doc.txt文件（如果需要，可以指定绝对路径）上传到名为server.my的服务器。存储在服务器上的文件名将为server_example.txt。文件由用户Peter使用密码1234发送到服务器。

Chapter 37: FTP server

Section 37.1: Download file from FTP server

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

This command download myfile.txt file from server named server.my and saves it as donwloaded_file.txt into working directory. You can specify absolute path for downloaded file.

Section 37.2: Download file from FTP server when login required

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

This command download myfile.txt file from server named srv.my and saves it as donwload.txt into working directory. You can specify absolute path for downloaded file. File is download by user Peter with password 1234.

Section 37.3: Upload file to FTP server

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

This command upload example.txt file from working directory (you could specify absolute path if you want) to server named server.my. File stored on the server will be named server_example.txt.

Section 37.4: Upload file to FTP server when login required

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

This command upload doc.txt file from working directory (you could specify absolute path if you want) to server named server.my. File stored on the server will be named server_example.txt. Fill is send on the server by user Peter and password 1234.

第38章：使用Windows窗体

第38.1节：使用默认的窗体实例

VB.NET提供默认的窗体实例。开发者无需创建实例，因为它在后台自动创建。然而，除非是最简单的程序，不建议使用默认实例。

```
公共类 Form1

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

结束类

模块 模块1

    Public Sub Main()
        ' 默认实例
        Form1.Foo()
        ' 新实例
        Dim myForm1 As Form1 = New Form1()
       myForm1.Foo()

    End Sub

End Module
```

另请参见：

- 在 VB.NET 中是否必须显式创建窗体实例？
- 为什么 VB.Net 中每个窗体都有默认实例，而 C# 中没有？

第38.2节：从一个窗体向另一个窗体传递数据

有时你可能想将一个窗体中生成的信息传递给另一个窗体以供额外使用。这对于显示搜索工具或设置页面的窗体等多种用途非常有用。

假设你想在已经打开的窗体（主窗体）和一个新窗体（新窗体）之间传递一个DataTable：

在主窗体中：

```
  私有子程序打开新窗体()
       Dim NewInstanceOfForm 作为新 NewForm(DataTable1)
       NewInstanceOfForm.ShowDialog()
    结束子程序
```

在新窗体中

```
公共类 NewForm
    Dim NewDataTable 作为 DataTable

    公共子程序 New（传入的DataTable 作为 DataTable）
       InitializeComponent()
      NewDataTable = 传入的DataTable
    结束子程序
```

Chapter 38: Working with Windows Forms

Section 38.1: Using the default Form instance

VB.NET offers default Form instances. The developer does not need to create the instance as it is created behind the scenes. However, *it is* ***not*** *preferable* to use the default instance all but the simplest programs.

```
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
```

See also:

- Do you have to explicitly create instance of form in VB.NET?
- Why is there a default instance of every form in VB.Net but not in C#?

Section 38.2: Passing Data From One Form To Another

Sometimes you might want to pass information that has been generated in one form, to another form for additional use. This is useful for forms that display a search tool, or a settings page among many other uses.

Let's say you want to pass a `DataTable` between a form that is already open *(MainForm)* and a new form *(NewForm)*:

In The MainForm:

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

In The NewForm

```
Public Class NewForm
    Dim NewDataTable as Datatable

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

```
结束类
```

现在当新窗体被打开时，它会从主窗体接收DataTable1并存储为新数据表在新窗体中供该窗体使用。

当尝试在窗体之间传递大量信息时，这非常常有用，尤其是当
将所有信息合并到一个`ArrayList`中并将该`ArrayList`传递给新窗体时。

```
End Class
```

Now when the *NewForm* is opened, it is passed `DataTable1` from *MainForm* and stored as `NewDataTable` in *NewForm* for use by that form.

This can be extremely useful when trying to pass large amounts of information between forms, especially when combining all of the information in to a single `ArrayList` and passing the `ArrayList` to the new form.

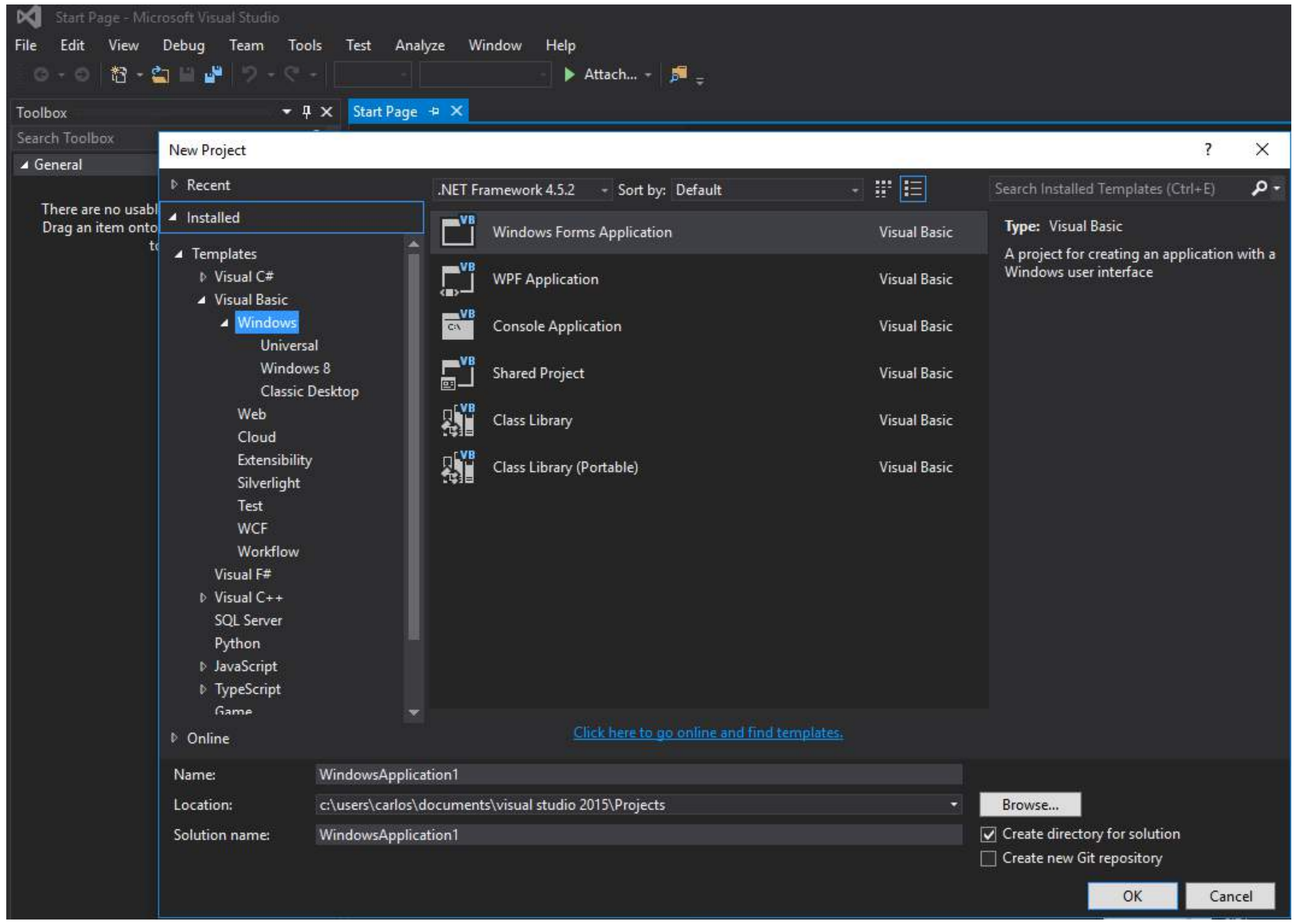
第39章：Windows窗体中的谷歌地图

第39.1节：如何在Windows窗体中使用谷歌地图

本示例的第一部分说明如何实现它。第二部分将解释其工作原理。此示例力求通用。地图模板（见步骤3）和示例函数均可完全自定义。

################################## 实现 ##################################

步骤1. 首先，创建一个新项目并选择Windows窗体应用程序。名称保持为“Form1”。


New Project

.NET Framework 4.5.2 Sort by: Default

Windows Forms Application Visual Basic
WPF Application Visual Basic
Console Application Visual Basic
Shared Project Visual Basic
Class Library Visual Basic
Class Library (Portable) Visual Basic

Type: Visual Basic
A project for creating an application with a Windows user interface

Name: WindowsApplication1
Location: c:\users\carlos\documents\visual studio 2015\Projects
Solution name: WindowsApplication1
[✓] Create directory for solution
[ ] Create new Git repository

步骤2. 向Form1添加一个WebBrowser控件（用于承载地图）。命名为“wbmap”。

步骤3. 使用你喜欢的文本编辑器创建一个名为“googlemap_template.html”的.html文件，并粘贴以下代码：

googlemap_template.html

```
&lt;!DOCTYPE html&gt;
&lt;html&gt;
&lt;head&gt;
&lt;meta charset="UTF-8"&gt;
&lt;meta http-equiv="X-UA-Compatible" content="IE=edge"/&gt;
    &lt;style type="text/css"&gt;
      html, body {
```

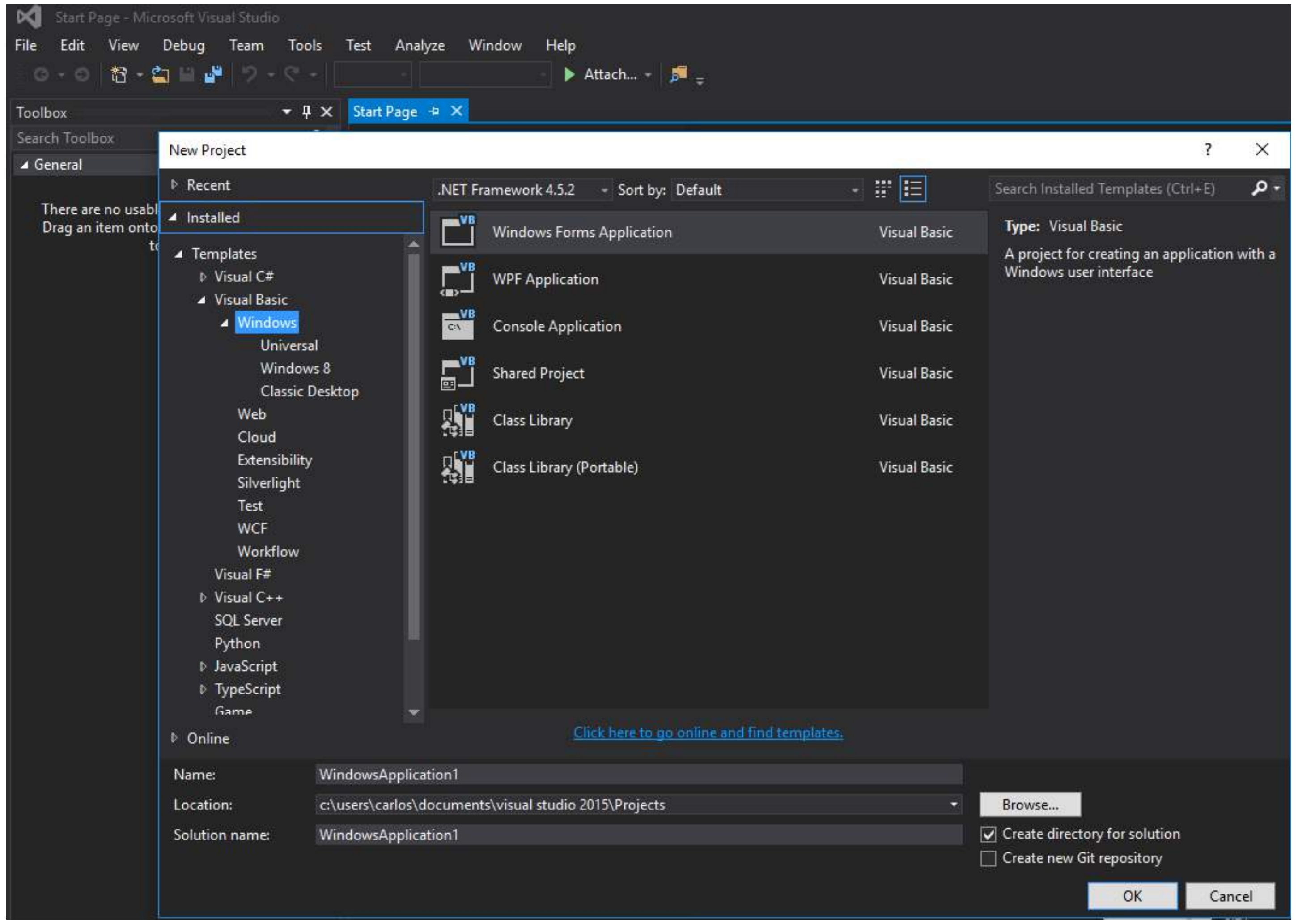
Chapter 39: Google Maps in a Windows Form

Section 39.1: How to use a Google Map in a Windows Form

The first part of this example explains how to implement it. In the second, I will explain how it works. This tries to be a general example. The template for the map (see step 3) and the example functions are fully customizable.

################################# IMPLEMENTATION #################################

Step 1. Firstly, create a new project and select Windows Form Application. Let's leave its name as "Form1".


New Project

.NET Framework 4.5.2 Sort by: Default

Windows Forms Application Visual Basic
WPF Application Visual Basic
Console Application Visual Basic
Shared Project Visual Basic
Class Library Visual Basic
Class Library (Portable) Visual Basic

Type: Visual Basic
A project for creating an application with a Windows user interface

Name: WindowsApplication1
Location: c:\users\carlos\documents\visual studio 2015\Projects
Solution name: WindowsApplication1
[✓] Create directory for solution
[ ] Create new Git repository

Step 2. Add a WebBrowser control (which will hold your map) to your Form1. Let's call it "wbmap"

Step 3. Create a .html file named "googlemap_template.html" with your favourite text editor and paste the following code:

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() {
            //使用 window.X 替代 var X 以使变量全局可用
          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());
                    // 调用WF中编写的函数
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">
        // 从WF触发的无参数函数
        function showJavascriptHelloWorld() {
            alert("Hello world in HTML from WF");
```

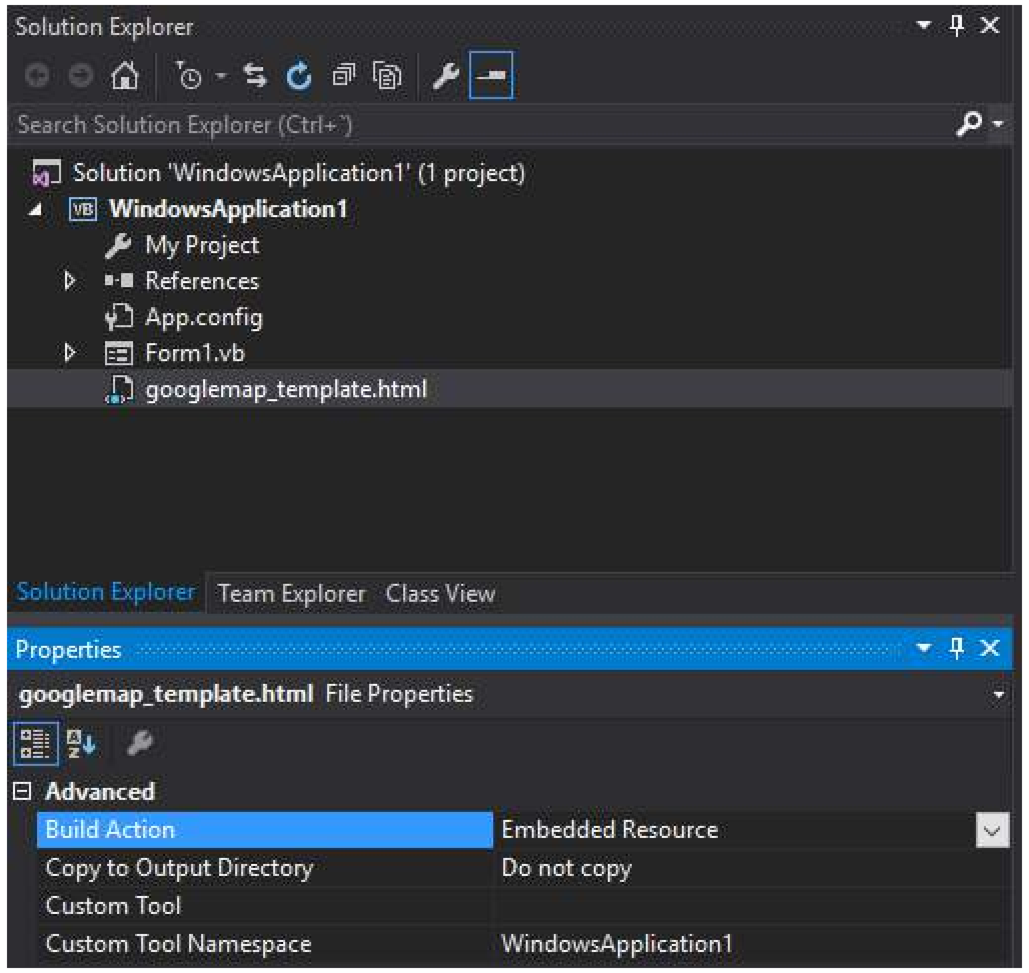
```
        height: 100%;
        margin: 0;
        padding: 0;
      }
      #gmap {
        height: 100%;
      }
     </style>
    <script type="text/javascript" src="http://maps.google.com/maps/api/js?sensor=false"></script>
    <script type="text/javascript">
        function initialize() {
            //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">
        // 从工作流触发的带字符串参数的函数
        function focusMarkerFromIdx(idx) {
            google.maps.event.trigger(markers[idx], 'click');
        }
     </script>
  </head>
  <body>
<div id="gmap"></div>
  </body>
</html>
```

这将作为我们的地图模板。我稍后会解释它的工作原理。

步骤4. 将 googlemap_template.html 文件添加到你的项目中（右键点击项目->添加->现有项）

步骤5. 当它出现在解决方案资源管理器中时，设置其属性为：

- 生成操作 -> 嵌入的资源
- 自定义工具命名空间 -> 写入项目名称


Solution 'WindowsApplication1' (1 project)
WindowsApplication1
My Project
References
App.config
Form1.vb
googlemap_template.html

googlemap_template.html File Properties

Advanced

Build Action	Embedded Resource
Copy to Output Directory	Do not copy
Custom Tool	
Custom Tool Namespace	WindowsApplication1

步骤6. 添加一个新类（右键点击你的项目->添加->类）。在我的示例中，我将其命名为GoogleMapHelper。

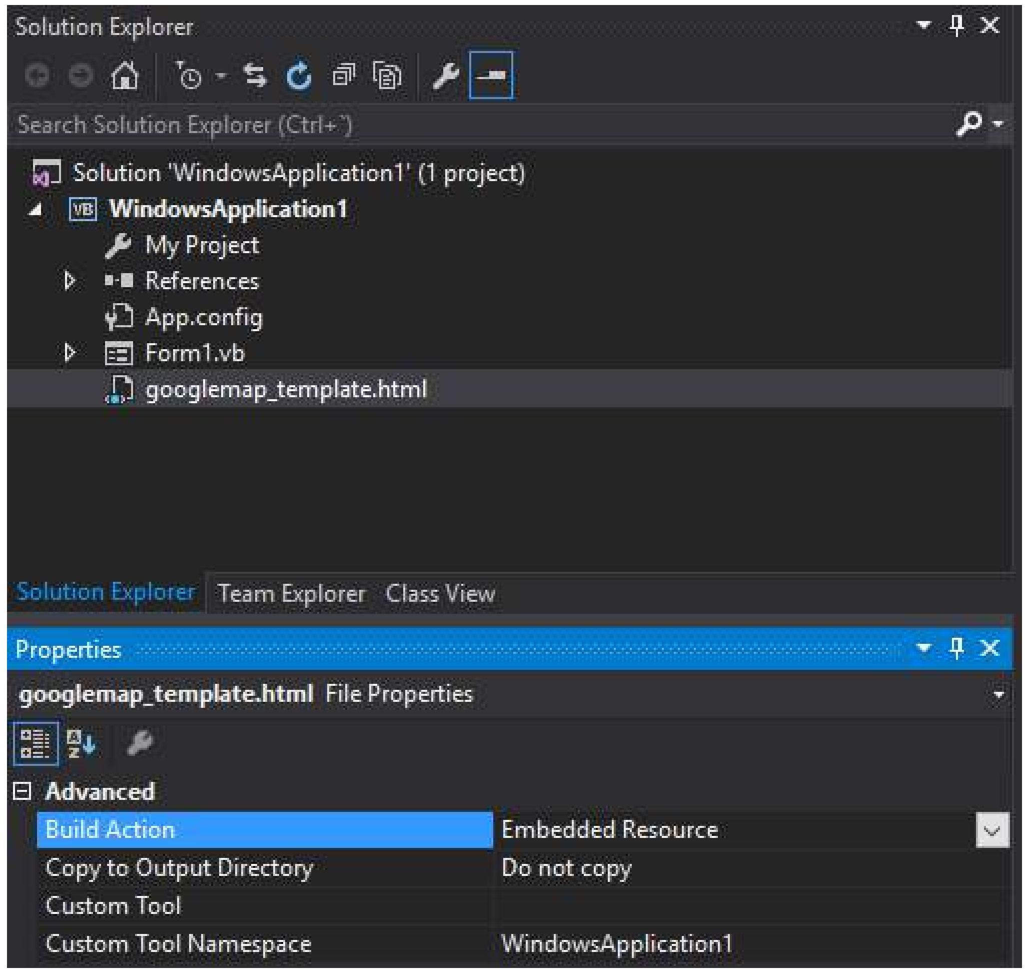
```
        }
    </script>
     <script type="text/javascript">
        // Function triggered from the WF with a String argument
        function focusMarkerFromIdx(idx) {
            google.maps.event.trigger(markers[idx], 'click');
        }
     </script>
  </head>
  <body>
    <div id="gmap"></div>
  </body>
</html>
```

This will serve as our map template. I will explain how it works later.

Step 4. Add the googlemap_template.hmtl file to your project (right click on your project->add->existing item)

Step 5. Once it appears in your Solution Explorer, set its properties to:

- Build Action -> Embedded Resource
- Custom Tool Namespace -> write the name of the project

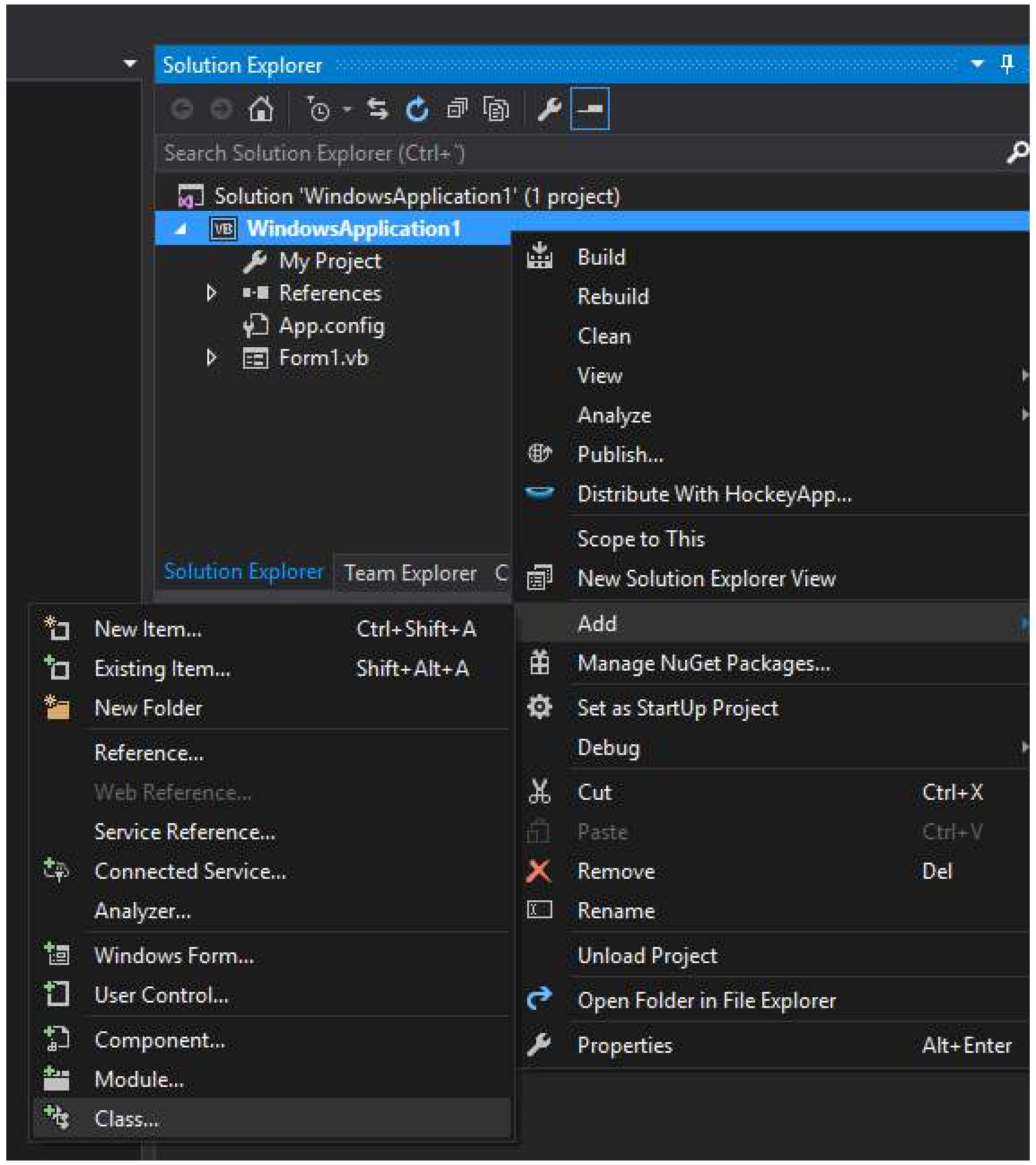

Solution 'WindowsApplication1' (1 project)
WindowsApplication1
My Project
References
App.config
Form1.vb
googlemap_template.html

googlemap_template.html File Properties

Advanced

Build Action	Embedded Resource
Copy to Output Directory	Do not copy
Custom Tool	
Custom Tool Namespace	WindowsApplication1

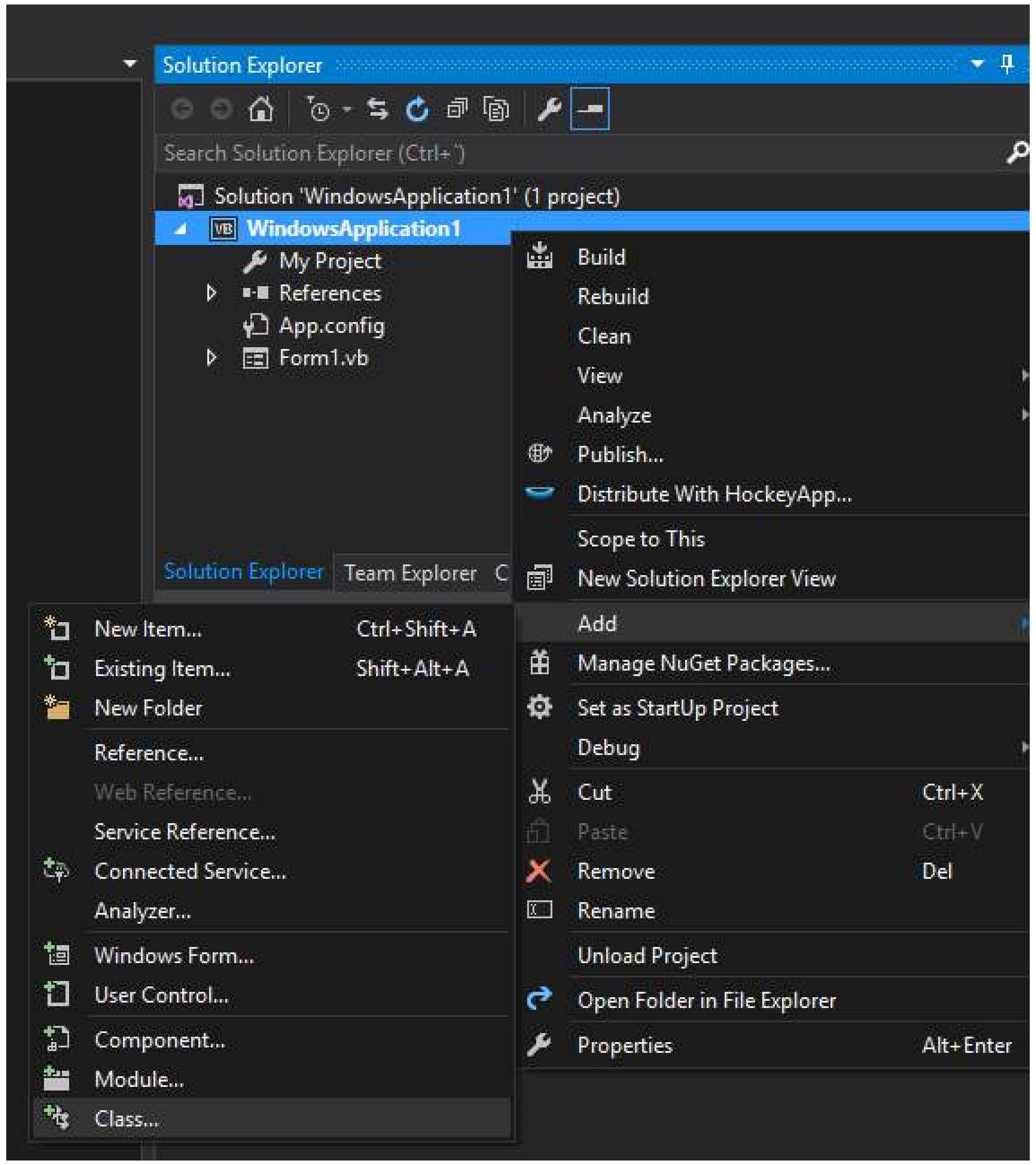
Step 6. Add a new class (right click on your project->add->class). In my example I'll call it GoogleMapHelper.



步骤7. 将以下代码粘贴到你的类中：

GoogleMapHelper.vb

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

Public Class GoogleMapHelper

' 1- googlemap_template.html 必须复制到主项目文件夹中
' 2- 在Visual Studio解决方案资源管理器中添加该文件（添加现有文件）
' 3- 设置该文件的属性为：
'                          生成操作 -> 嵌入的资源
'                          自定义工具命名空间 -> 写入项目名称

Private Const ICON_FOLDER As String = "marker_icons/" '图片必须存放在Debug/Release文件夹内的一个文件夹中

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"
```



Step 7. Paste the following code into your class:

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

    '标记位置
    Private mPositions As Double(,) '纬度，经度
    ' 标记数据允许不同格式，包括纬度、经度，且可选标题和图标：
    ' 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("标记数据尺寸不正确。它必须有2、3或4列")
            Return Nothing
        结束如果

        Dim htmlTemplate As New StringBuilder()
        Dim tmpFolder As String = Environment.GetEnvironmentVariable("TEMP")
        Dim dataSize As Integer = mMarkerData.GetLength(1) '列数
        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 += ","
            结束如果
            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
```

```
    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
```

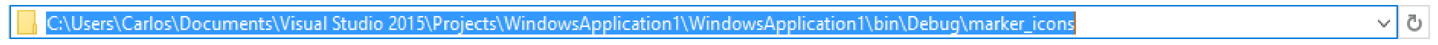
```
            End If
        Next

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])
        结束 Try

        If existsMapFile Then
            Return tmpHtmlMapFile
        Else
            Return Nothing
        End If
    结束函数

    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
    结束函数

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

        Using stream 作为 Stream = assem.GetManifestResourceStream(resourceName)
            Try
                Using reader 作为 新的 StreamReader(stream)
                    Return reader.ReadToEnd()
                结束 Using
            Catch e 作为 Exception
                Throw New Exception((Convert.ToString("访问资源错误 '") &
resourceName) + "'" & vbCr & vbLf + e.ToString())
            结束 Try
        结束 Using
    结束函数

    Private Function createTxtFile(mFile 作为 String, content 作为 StringBuilder) 作为 Boolean
        Dim mPath 作为 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 作为 StreamWriter = File.CreateText(mFile)
        sw.Write(content.ToString())
       sw.Close()
        Return True
    End Function
    End Class
```

注意： MAP_TEMPLATE 常量必须包含您的项目名称

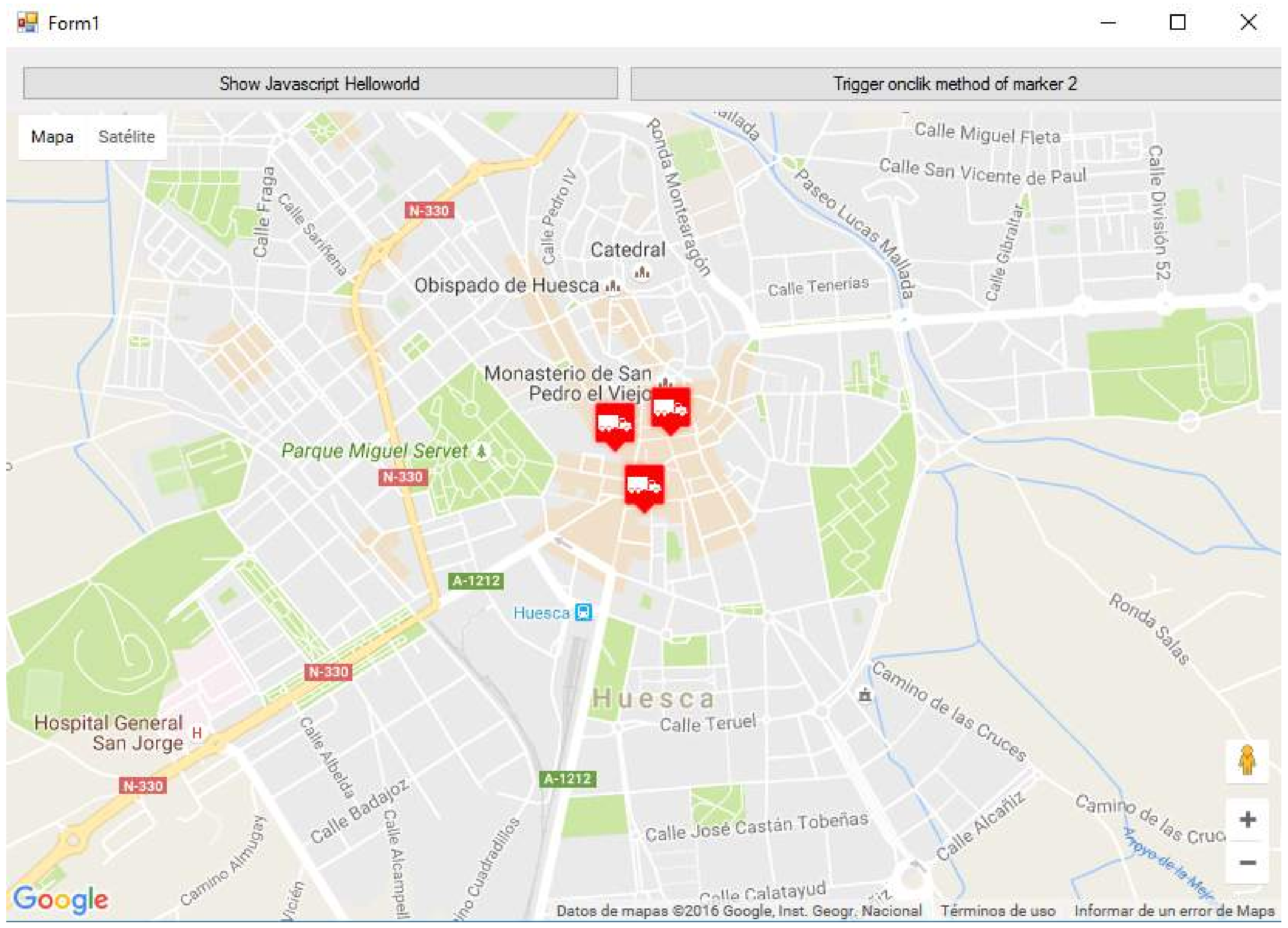
```
            End If
        Next

        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
```

Note: The MAP_TEMPLATE constant must include the name of your project

步骤 8. 现在我们可以使用 GoogleMapHelper 类通过简单地创建一个实例并调用其 loadMap() 方法，将地图加载到我们的网页浏览器中。如何构建您的 markerData 由您决定。在此示例中，为了说明，我手动编写了它们。定义标记数据有 3 种选项（参见 GoogleMapHelper 类注释）。请注意，如果您使用第三种选项（包括标题和图标），则必须在您的 Debug/Release 文件夹中创建一个名为 "marker_icons"（或您在 GoogleMapHelper 常量 ICON_FOLDER 中定义的名称）的文件夹，并将您的 .png 文件放在那里。以我的情况为例：


C:\Users\Carlos\Documents\Visual Studio 2015\Projects\WindowsApplication1\WindowsApplication1\bin\Debug\marker_icons

我在 Form1 中创建了两个按钮，用于演示地图与 Windows 窗体的交互。界面如下所示：

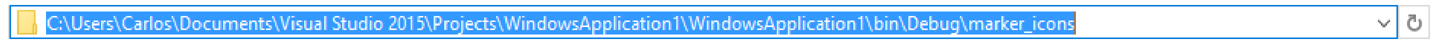

Form1
Show Javascript Helloworld
Trigger onclik method of marker 2
Mapa
Satélite
Catedral
Obispado de Huesca
Monasterio de San Pedro el Viejo
Parque Miguel Servet
Huesca
Hospital General San Jorge
Google
Datos de mapas ©2016 Google, Inst. Geogr. Nacional   Términos de uso   Informar de un error de Maps

代码如下：

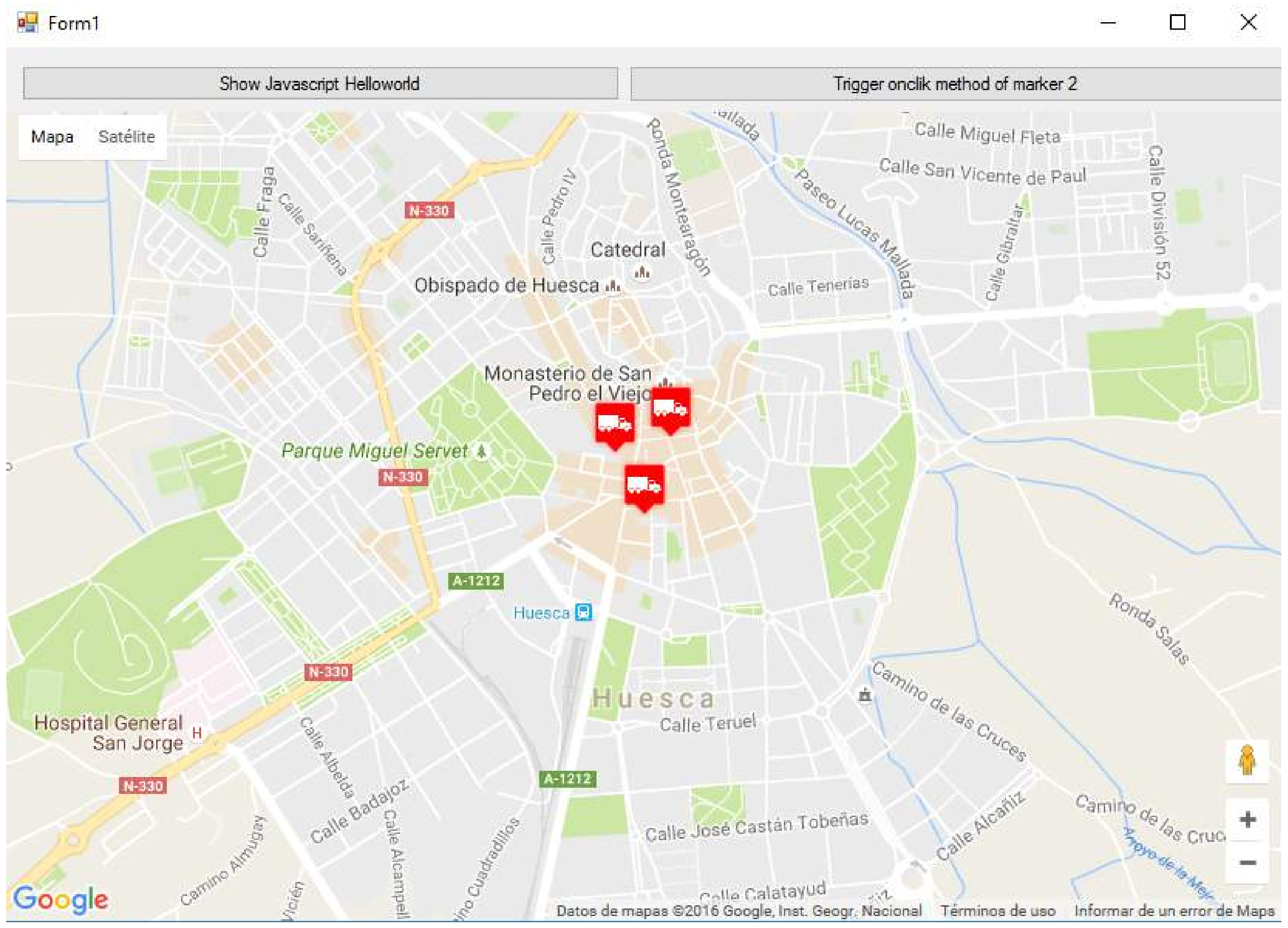
Form1.vb

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

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

    Me.wbmap.ObjectForScripting = Me
```

Step 8. Now we can use our GoogleMapHelper class to load the map into our webbrowser by simply creating and instance and calling its loadMap() method. How you build your markerData is up to you. In this example, for clarification, I write them by hand. There are 3 options to define the marker data (see GoogleMapHelper class comments). Note that if you use the third option (including title and icons) you must create a folder called "marker_icons" (or whatever you define in the GoogleMapHelper constant ICON_FOLDER) in your Debug/Release folder and place there your .png files. In my case:


C:\Users\Carlos\Documents\Visual Studio 2015\Projects\WindowsApplication1\WindowsApplication1\bin\Debug\marker_icons

I created two buttons in my Form1 to illustrate how the map and the WF interact. Here is how it looks:


Form1
Show Javascript Helloworld
Trigger onclik method of marker 2
Mapa
Satélite
Catedral
Obispado de Huesca
Monasterio de San Pedro el Viejo
Parque Miguel Servet
Huesca
Hospital General San Jorge
Google
Datos de mapas ©2016 Google, Inst. Geogr. Nacional   Términos de uso   Informar de un error de Maps

And here is the 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

'################################ 调用 JAVASCRIPT 方法 ################################
'这些方法调用 googlemap_template.html 中编写的方法
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
'####################################################################################

'################################ 从 JAVASCRIPT 调用的方法 ############################'这些
方法由 googlemap_template.html 中定义的 javascript 在某些事件触发时调用

Public Sub getMarkerDataFromJavascript(title As String, idx As String)
    MsgBox("标题: " & title & " 索引: " & idx)
End Sub

Public Sub showVbHelloWorld()
    MsgBox("来自HTML的WF中的Hello world")
End Sub
结束类
```

重要提示： 别忘了在你的Form1类定义之前添加这些行：

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

它们的作用是告诉.NET框架我们需要完全信任，并使类对COM可见，因此Form1
对JavaScript是可见的。

另外别忘了在你的Form1加载函数中添加这段代码：

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

它将你的Form1类暴露给googlemap_template.html页面上的JavaScript。

现在你可以执行，应该可以正常工作了

################################### 工作原理 ###################################

基本上，我们的GoogleMapHelper类所做的是读取googlemap_template.html，制作一个临时副本，替换与标记相关的代码（[[MARKER_DATA]]），并在窗体的网页浏览器控件中执行该页面。该HTML会遍历所有标记，并为每个标记分配一个“点击”监听器。这个点击函数显然是完全可定制的。在示例中，如果标记有标题，它会打开信息窗口，并将地图中心定位于

```
    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
```

IMPORTANT : don't forget to add these lines before your class Form1 definition:

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

What they do is to tell the .NET Framework that we want fulltrust and make the class visible to COM so Form1 is visible to JavaScript.

Also don't forget this in your Form1 load function:

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

It exposes your Form1 class to the JavaScript on the googlemap_template.hmtl page.

Now you can execute and it should be working

################################### HOW IT WORKS###################################

Basically, what our GoogleMapHelper class does is to read our googlemap_template.html, make a temporal copy, replace the code related to the markers ([[MARKER_DATA]]) and execute the page in the web browser control of our form. This html loops through all the markers and assigns a 'click' listener to each one. This click function is obviously fully customizable. In the example it opens an infowindow if the marker has a title, centers the map in

该标记并调用我们Form1类中定义的两个外部函数。

另一方面，我们可以在此HTML中定义其他JavaScript函数（带参数或不带参数），以便从我们的Windows窗体调用（通过使用wbmap.Document.InvokeScript）。

such marker and calls two external functions that are defined in our Form1 class.

On the other hand, we can define other javascript functions (with or without arguments) in this html to be called from our Windows Form (by using wbmap.Document.InvokeScript).

第40章：GDI+

第40.1节：绘制图形

要开始绘制图形，您需要定义一个画笔对象Pen接受两个参数：

1.画笔颜色或画刷
2.画笔宽度

画笔对象用于创建您想绘制对象的**轮廓**

定义画笔后，您可以设置特定的画笔属性

```
    Dim pens As New Pen(Color.Purple)
pens.DashStyle = DashStyle.Dash '画笔将以虚线绘制
    pens.EndCap = LineCap.ArrowAnchor '线条将以箭头结束
    pens.StartCap = LineCap.Round '线条绘制将以圆形开始
    '*注意* - 如果绘制封闭图形，起始和结束端帽将不会显示
```

然后使用您创建的graphics对象来绘制图形

```
   Private Sub GraphicForm_Paint(sender As Object, e As PaintEventArgs) Handles MyBase.Paint
     Dim pen As New Pen(Color.Blue, 15) '使用宽度为15的蓝色画笔
     Dim point1 As New Point(5, 15) '线的起点
     Dim point2 As New Point(30, 100) '线的终点
     e.Graphics.DrawLine(pen, point1, point2)

    e.Graphics.DrawRectangle(pen, 60, 90, 200, 300) '绘制矩形轮廓
```

默认情况下，画笔的宽度等于1

```
     Dim pen2 as New Pen(Color.Orange) '使用宽度为1的橙色画笔
     Dim origRect As New Rectangle(90, 30, 50, 60) '定义弧线的边界
     e.Graphics.DrawArc(pen2, origRect, 20, 180) '在矩形边界内绘制弧线

 End Sub
```

第40.2节：填充图形

Graphics.FillShapes 绘制图形并用指定颜色填充。填充图形可以使用

1. Brush 工具 - 用纯色填充图形

```
    Dim rect As New Rectangle(50, 50, 50, 50)
    e.Graphics.FillRectangle(Brushes.Green, rect) '绘制一个填充为绿色的矩形e.Graphics.FillPie(Brushes.Silver, rect, 0,

     180) '绘制一个填充为银色的半圆
```

2. HatchBrush 工具 - 用图案填充图形

```
Dim hBrush As New HatchBrush(HatchStyle.ZigZag, Color.SkyBlue, Color.Gray)
'创建一个背景色为天蓝色，前景色为灰色的HatchBrush工具，
'并将以之字形图案填充
Dim rectan 作为新 矩形(100, 100, 100, 100)
```

Chapter 40: GDI+

Section 40.1: Draw Shapes

To start drawing a shape you need to define a pen object The Pen accepts two parameters:

1. Pen Color or Brush
2. Pen Width

The Pen Object is used to create an **outline** of the object you want to draw

After Defining the Pen you can set specific Pen Properties

```
    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
```

Then use the graphics object you created to draw the shape

```
  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
```

By default, the pen's width is equal to 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
```

Section 40.2: Fill Shapes

Graphics.FillShapes draws a shape and fills it in with the color given. Fill Shapes can use

1. Brush Tool - to fill shape with a solid color

```
    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 Tool - to fill shape with a pattern

```
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 - 用于用渐变填充形状

```
Dim lBrush 作为新 LinearGradientBrush(point1, point2, Color.MediumVioletRed, Color.PaleGreen)
 Dim rect 作为新 矩形(50, 50, 200, 200)
 e.Graphics.FillRectangle(lBrush, rect)
```

4. TextureBrush - 用于用图片填充形状

你可以从资源、已定义的Bitmap或文件名中选择图片

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

无论是Hatch Brush Tool还是LinearGradientBrush都需要导入以下语句：Imports **System.Drawing.Drawing2D**

第40.3节：文本

要在窗体上绘制文本，请使用DrawString方法

绘制字符串时，可以使用上述4种画刷中的任意一种

```
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))
'这将在指定点使用线性渐变画刷绘制单词"Hello"'
```

由于无法定义文本的宽度或高度，使用Measure Text来检查文本大小

```
Dim lBrush 作为新 LinearGradientBrush(point1, point2, Color.MediumVioletRed, Color.PaleGreen)
        Dim TextSize =  e.Graphics.MeasureString("HELLO", New Font("Impact", 60, FontStyle.Bold), lBrush)
'使用TextSize来确定字符串的位置，或判断字体是否需要缩小'
```

> 例如：你需要在窗体顶部绘制单词"Test"。窗体宽度为120。使用此循环逐渐减小字体大小，直到适合窗体宽度

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

第40.4节：创建图形对象

创建图形对象有三种方法

1. **从Paint事件**

每次控件重绘（调整大小、刷新等）时都会调用此事件，如果你希望控件能够

```
e.Graphics.FillRectangle(hBrush, rectan)
```

3. `LinearGradientBrush` - to fill shape with a gradient

```
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` - to fill shape with a picture

You can choose a picture from resources, an already defined Bitmap, or from a file name

```
   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)
```

Both the `Hatch Brush Tool` and `LinearGradientBrush` import the following statement : **Imports System.Drawing.Drawing2D**

Section 40.3: Text

To draw text onto the form use the `DrawString` Method

When you draw a string you can use any of the 4 brushes listed above

```
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
```

Since you can't define the width or height of the text use `Measure Text` to check text size

```
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
```

> Ex: You need to draw the word "Test" on top of the form. The form's width is 120. Use this loop to decrease the font size till it will fit into the forms width

```
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
```

Section 40.4: Create Graphic Object

There are three ways to create a graphics object

1. From the **Paint Event**

Every time the control is redrawn (resized, refreshed...) this event is called, use this way if you want the control to

持续地在控件上绘制，则使用此方法

```
'这适用于任何对象的paint事件，而不仅仅是窗体
Private Sub Form1_Paint(sender as Object, e as PaintEventArgs) Handles Me.Paint
  Dim gra as Graphics
  gra = e.Graphics
End Sub
```

2.创建图形

当您想在控件上创建一次性图形，或者不希望控件重新绘制自身时，最常使用此方法

```
Dim btn 作为新 按钮
Dim g 作为 图形 = btn.CreateGraphics
```

3. 从现有图形

当您想绘制并更改现有图形时使用此方法

```
'现有图像可以来自文件名、流或 Drawing.Graphic
Dim image = 新 Bitmap("C:\TempBit.bmp")
Dim gr 作为 图形 = Graphics.FromImage(image)
```

consistently draw on the control

```
'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. **Create Graphic**

This is most often used when you want to create a one time graphic on the control, or you don't want the control to repaint itself

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

3. From an **Existing Graphic**

Use this method when you want to draw and change an existing graphic

```
'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)
```

第41章：WinForms 拼写检查框

示例说明如何向 WindowsForms 应用程序添加拼写检查框。此示例不需要安装 Word，也不以任何方式使用 Word。

它使用 ElementHost 控件进行 WPF 互操作，从 WPF 文本框创建一个 WPF 用户控件。WPF 文本框具有内置的拼写检查功能。我们将利用这个内置功能，而不是依赖外部程序。

第41.1节：ElementHost WPF 文本框

此示例是基于我在网上找到的一个示例进行建模的。我找不到链接，否则我会给作者致谢。我拿到示例后进行了修改，使其适用于我的应用程序。

1. 添加以下引用：

> System.Xaml、PresentationCore、PresentationFramework、WindowsBase 和 WindowsFormsIntegration

2. 创建一个新类并粘贴以下代码

```
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
继承 ElementHost

Private box As TextBox

Public Sub New()
box = New TextBox()
    MyBase.Child = box
    添加处理程序 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
结束 属性
```

Chapter 41: WinForms SpellCheckBox

Example on how to add a spell check box to a WindowsForms application. This example DOES NOT require Word to be installed nor does it use Word in any way.

It uses WPF Interop using the ElementHost control to create a WPF UserControl from a WPF TextBox. WPF TextBox has a built in function for spell check. We are going to leverage this built in function rather than relying on an external program.

Section 41.1: ElementHost WPF TextBox

This example is was modeled after an example that I found on the internet. I can't find the link or I would give the author credit. I took the sample that I found and modified it to work for my application.

1. Add the following references:

> System.Xaml, PresentationCore, PresentationFramework, WindowsBase, and WindowsFormsIntegration

2. Create a new Class and past this code

```
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
    设置(ByVal value As Boolean)
        box.AcceptsReturn = value
    结束设置
结束 属性

<DefaultValue(True)> _
公共属性 WordWrap() As Boolean
    获取
        返回 box.TextWrapping <> TextWrapping.Wrap
    结束获取
    设置(ByVal value As Boolean)
        如果 value 则
            box.TextWrapping = TextWrapping.Wrap
        否则
            box.TextWrapping = TextWrapping.NoWrap
        结束如果
    结束设置
结束 属性

<DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)> _
公共隐藏属性 Child() As System.Windows.UIElement
    获取
        返回 MyBase.Child
    End Get
    Set(ByVal value As System.Windows.UIElement)
        '' 不做任何操作以解决序列化器的问题！！
    End Set
End Property

End Class
```

3. 重建解决方案。
4. 添加一个新窗体。
5. 在工具箱中搜索你的类名。此示例为“SpellCheck”。它应该列在‘YourSoulutionName’组件下。
6. 将新控件拖到你的窗体上
7. 在窗体加载事件中设置任何映射的属性

```
Private Sub form1_Load(sender As Object, e As EventArgs) Handles Me.Load
    spellcheckbox.WordWrap = True
spellcheckbox.MultiLin = True
    '在此添加其他属性修改器...
End Sub
```

7. 你需要做的最后一件事是更改应用程序的DPI感知设置。这是因为你正在使用WinForms应用程序。默认情况下，所有WinForms应用程序都是DPI不感知的。一旦你执行了包含元素宿主（WPF互操作）的控件，应用程序将变为DPI感知。这可能会影响你的UI元素，也可能不会。解决方案是强制应用程序变为DPI不感知。有两种方法：第一种是通过清单文件，第二种是硬编码到程序中。如果你使用OneClick部署应用程序，必须硬编码，而不能使用清单文件，否则错误不可避免。

```
<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. Rebuild the solution.
4. Add a new form.
5. Search the toolbox for your Class name. This example is "SpellCheck". It should be listed under 'YourSoulutionName' Components.
6. Drag the new control to your form
7. Set any of the mapped properties in the forms load event

```
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. The last thing that you need to do is to change the DPI Awareness of your application. This is because you are using WinForms application. By default all WinForms applications are DPI UNAWARE. Once you execute a control that has an element host (WPF Interop), the application will now become DPI AWARE. This may or may not mess with your UI Elements. The solution to this is to FORCE the application to become DPI UNAWARE. There are 2 ways to do this. The first is through the manifest file and the second is to hard code it in to your program. If you are using OneClick to deploy your application, you must hard code it, not use the manifest file or errors will be inevitable.

以下两个示例均可在以下位置找到：WinForms 在大 DPI 设置下的缩放——这甚至可能吗？感谢 Telerik.com 对 DPI 的精彩解释。

> 硬编码的DPI感知代码示例。此代码必须在第一个窗体初始化之前执行。我通常将其放在ApplicationEvents.vb文件中。你可以通过在解决方案资源管理器中右键点击你的项目名称并选择“打开”来访问此文件。然后选择左侧的应用程序选项卡，接着点击右下角的“查看应用程序事件”，该按钮位于启动画面下拉菜单旁边。

```
命名空间 My

    ' 以下事件可用于 MyApplication：
    '
    ' 启动：当应用程序启动且启动窗体创建之前触发。
    ' 关闭：当所有应用程序窗体关闭后触发。如果应用程序异常终止，则不会触发此事件。

    ' 未处理异常：当应用程序遇到未处理的异常时触发。
    ' 启动下一个实例：当启动单实例应用程序且应用程序已处于活动状态时触发。

                           ' 网络可用性更改：当网络连接连接或断开时触发。
    部分友元类 MyApplication

    私有枚举 PROCESS_DPI_AWARENESS
       Process_DPI_Unaware = 0
       Process_System_DPI_Aware = 1
       Process_Per_Monitor_DPI_Aware = 2
    结束枚举

    Private Declare Function SetProcessDpiAwareness Lib "shcore.dll" (ByVal Value As
PROCESS_DPI_AWARENESS) As Long

    Private Sub SetDPI()
        '来自 SetProcessDPIAwareness 的结果
        '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
```

> 清单示例

```
<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>
```

Both of the following examples can be found at the following: WinForms Scaling at Large DPI Settings - Is It Even Possible? Thanks to Telerik.com for the great explanation on DPI.

> Hard coded DPI Aware code example. This MUST be executed before the first form is initialized. I always place this in the ApplicationEvents.vb file. You can get to this file by right clicking on your project name in the solution explorer and choosing "Open". Then choose the application tab on the left and then click on "View Application Events" on the lower right next to the splash screen drop down.

```
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
```

> Manifest Example

```
<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>
```

```
        </asmv3:windowsSettings>
    </asmv3:application>
</assembly>
```

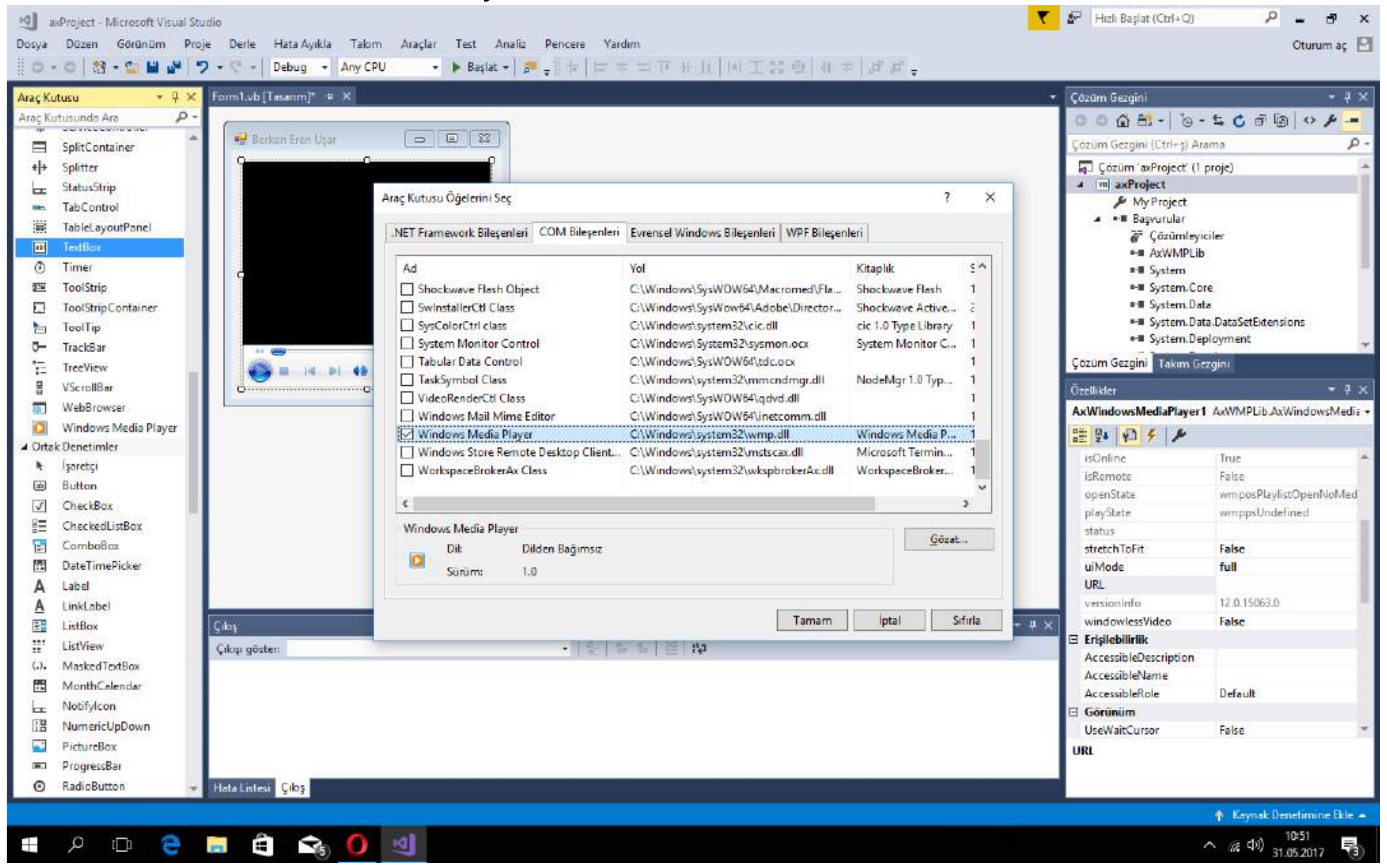
第42章：在VB.Net中使用axWindowsMediaPlayer

axWindowsMediaPlayer是用于播放视频和音乐等多媒体文件的控件。

第42.1节：添加axWindowsMediaPlayer

- 右键点击工具箱，然后点击“选择项目”。
- 选择COM组件标签，然后勾选Windows Media Player。
- axWindowsMediaPlayer将被添加到工具箱中。

选中此复选框以使用axWindowsMediaPlayer


Araç Kutusu Öğelerini Seç
.NET Framework Bileşenleri
COM Bileşenleri
Evrensel Windows Bileşenleri
WPF Bileşenleri
Gözat...
Tamam
İptal
Sıfırla

Ad	Yol	Kitaplık
☐ Shockwave Flash Object	C:\Windows\SysWOW64\Macromed\Fla...	Shockwave Flash
☐ SwInstallerCtl Class	C:\Windows\SysWow64\Adobe\Director...	Shockwave Active...
☐ SysColorCtrl class	C:\Windows\system32\cic.dll	cic 1.0 Type Library
☐ System Monitor Control	C:\Windows\System32\sysmon.ocx	System Monitor C...
☐ Tabular Data Control	C:\Windows\SysWOW64\tdc.ocx	
☐ TaskSymbol Class	C:\Windows\system32\mmcndmgr.dll	NodeMgr 1.0 Typ...
☐ VideoRenderCtl Class	C:\Windows\SysWOW64\qdvd.dll	
☐ Windows Mail Mime Editor	C:\Windows\SysWOW64\inetcomm.dll	
☑ Windows Media Player	C:\Windows\system32\wmp.dll	Windows Media P...
☐ Windows Store Remote Desktop Client...	C:\Windows\system32\mstscax.dll	Microsoft Termin...
☐ WorkspaceBrokerAx Class	C:\Windows\system32\wkspbrokerAx.dll	WorkspaceBroker...

Windows Media Player
Dil: Dilden Bağımsız
Sürüm: 1.0

然后你可以使用 axWindowsMediaPlayer :)

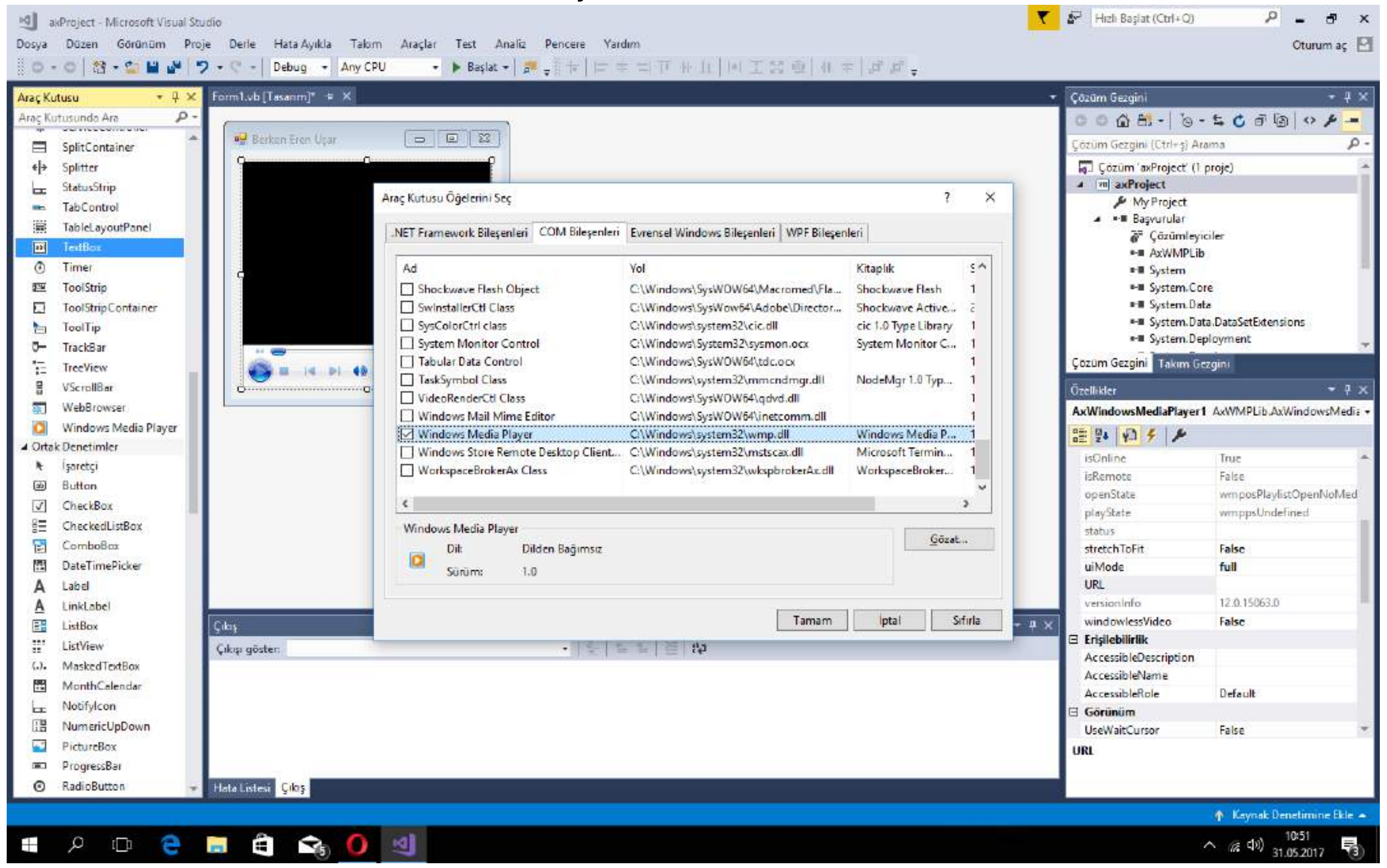
Chapter 42: Using axWindowsMediaPlayer in VB.Net

axWindowsMediaPlayer is the control for the playing multimedia files like videos and music.

Section 42.1: Adding the axWindowsMediaPlayer

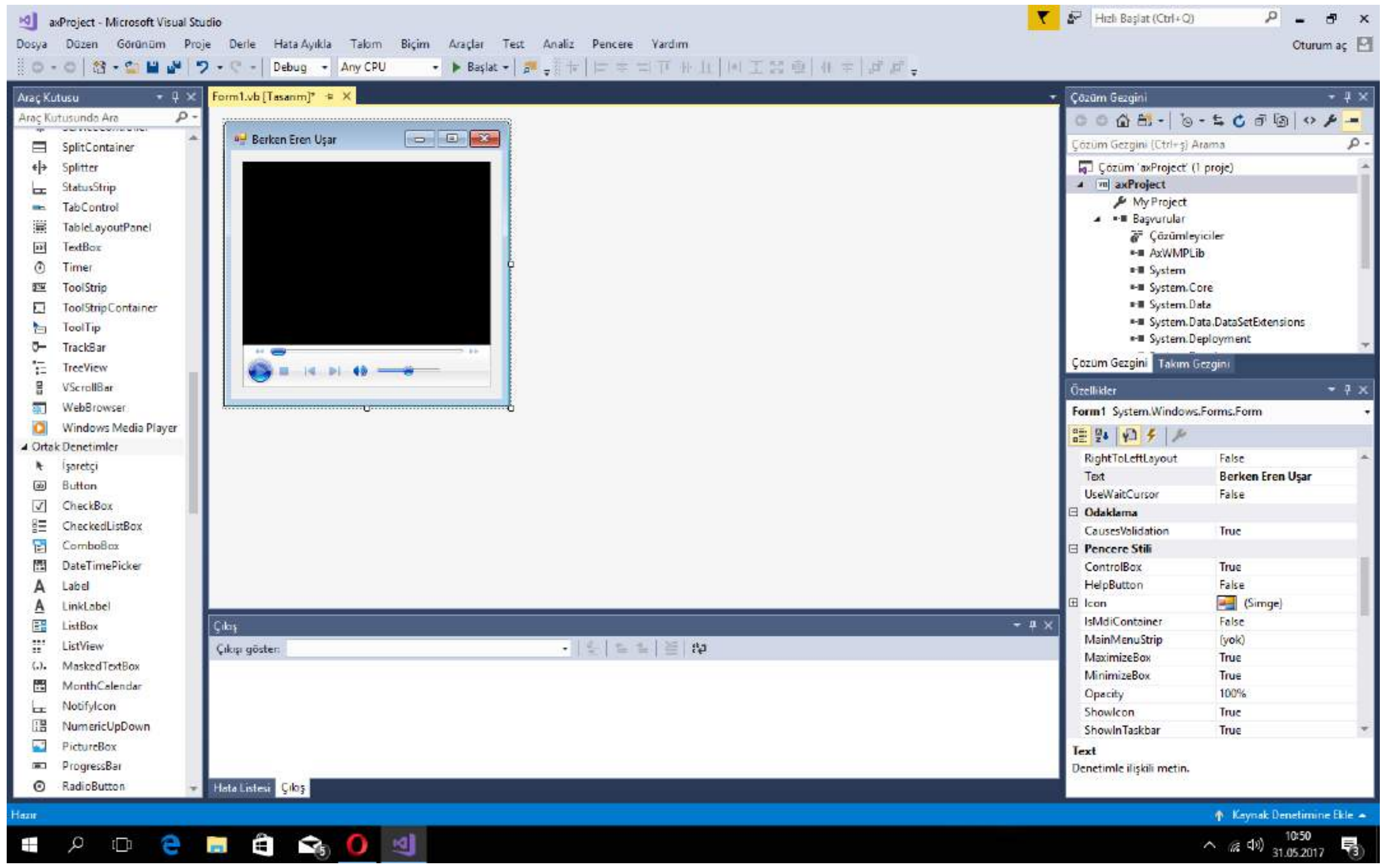
- Right-click on the Toolbox, then click "Choose Items".
- Select the COM Components tab, and then check Windows Media Player.
- axWindowsMediaPlayer will be added to Toolbox.

Select this checkbox to use axWindowsMediaPlayer


Ad	Yol	Kitaplık
☐ Shockwave Flash Object	C:\Windows\SysWOW64\Macromed\Fla...	Shockwave Flash
☐ SwInstallerCtl Class	C:\Windows\SysWow64\Adobe\Director...	Shockwave Active...
☐ SysColorCtrl class	C:\Windows\system32\cic.dll	cic 1.0 Type Library
☐ System Monitor Control	C:\Windows\System32\sysmon.ocx	System Monitor C...
☐ Tabular Data Control	C:\Windows\SysWOW64\tdc.ocx	
☐ TaskSymbol Class	C:\Windows\system32\mmcndmgr.dll	NodeMgr 1.0 Typ...
☐ VideoRenderCtl Class	C:\Windows\SysWOW64\qdvd.dll	
☐ Windows Mail Mime Editor	C:\Windows\SysWOW64\inetcomm.dll	
☑ Windows Media Player	C:\Windows\system32\wmp.dll	Windows Media P...
☐ Windows Store Remote Desktop Client...	C:\Windows\system32\mstscax.dll	Microsoft Termin...
☐ WorkspaceBrokerAx Class	C:\Windows\system32\wkspbrokerAx.dll	WorkspaceBroker...

Windows Media Player
Dil: Dilden Bağımsız
Sürüm: 1.0

Then you can use axWindowsMediaPlayer :)


Özellikler

Form1 System.Windows.Forms.Form

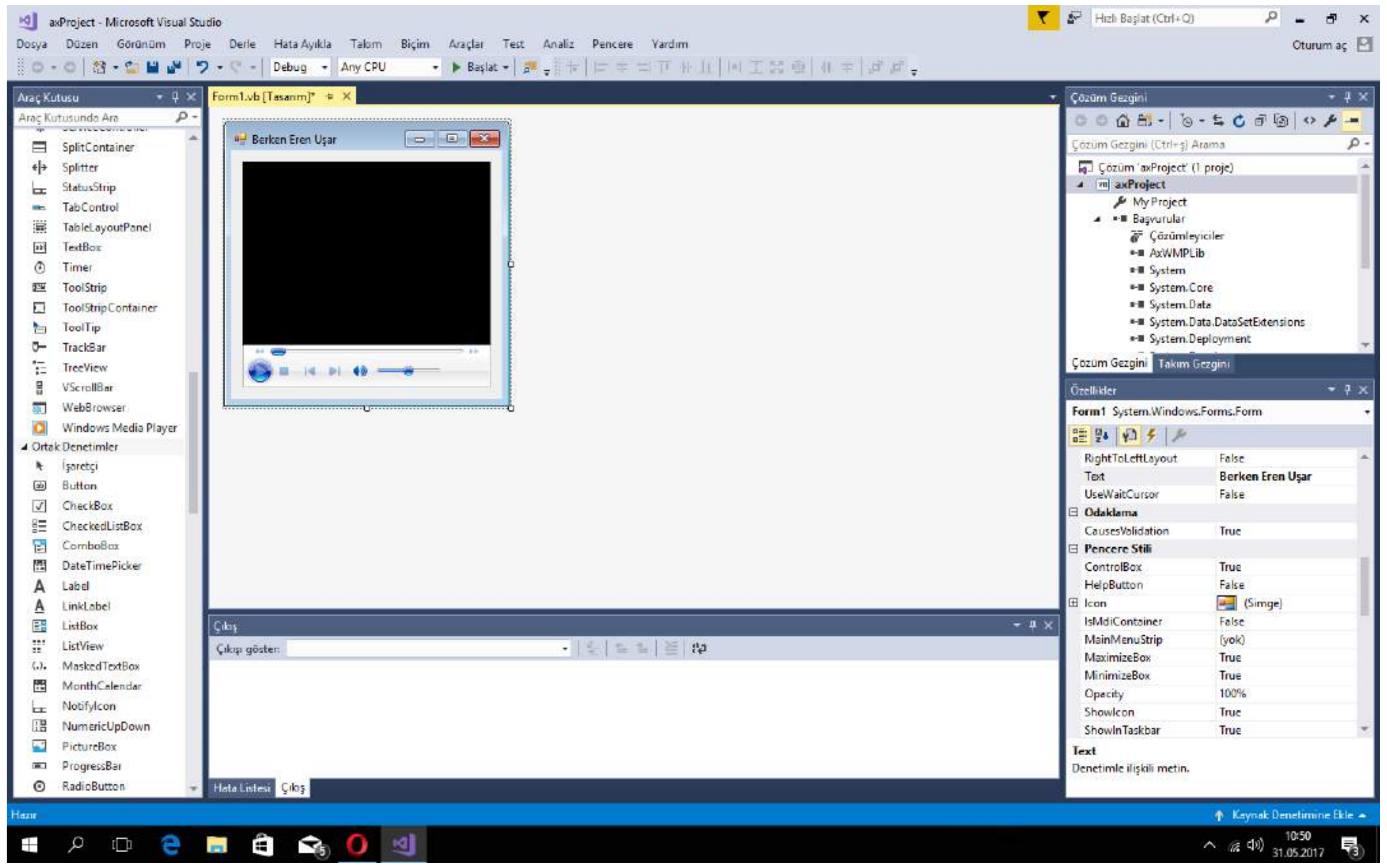
RightToLeftLayout	False
Text	Berken Eren Uşar
UseWaitCursor	False
Odaklama	
CausesValidation	True
Pencere Stili	
ControlBox	True
HelpButton	False
Icon	(Simge)
IsMdiContainer	False
MainMenuStrip	(yok)
MaximizeBox	True
MinimizeBox	True
Opacity	100%
ShowIcon	True
ShowInTaskbar	True

Text
Denetimle ilişkili metin.

第42.2节：播放多媒体文件

```
AxWindowsMediaPlayer1.URL = "C:\My Files\Movies\Avatar.mp4"
AxWindowsMediaPlayer1.Ctlcontrols.play()
```

这段代码将在 axWindowsMediaPlayer 中播放《阿凡达》。


Özellikler

Form1 System.Windows.Forms.Form

RightToLeftLayout False
Text Berken Eren Uşar
UseWaitCursor False
Odaklama
CausesValidation True
Pencere Stili
ControlBox True
HelpButton False
Icon (Simge)
IsMdiContainer False
MainMenuStrip (yok)
MaximizeBox True
MinimizeBox True
Opacity 100%
ShowIcon True
ShowInTaskbar True
Text
Denetimle ilişkili metin.

Section 42.2: Play a Multimedia File

```
AxWindowsMediaPlayer1.URL = "C:\My Files\Movies\Avatar.mp4"
AxWindowsMediaPlayer1.Ctlcontrols.play()
```

This code will play Avatar in the axWindowsMediaPlayer.

第43章：WPF XAML 数据绑定

此示例展示了如何在 MVVM 模式和 WPF 中创建 ViewModel 和 View，以及如何将两者绑定，使得任一方更改时另一方也会更新。

第43.1节：将 ViewModel 中的字符串绑定到 View 中的文本框

SampleViewModel.vb

```
'导入与 WPF 相关的类以简化操作
Imports System.Collections.ObjectModel
Imports System.ComponentModel

公共类 SampleViewModel
    继承 DependencyObject
    作为 ViewModel 的类必须继承自 DependencyObject

    一个简单的字符串属性
    公共属性 SampleString 作为 String
        获取
            返回 CType(GetValue(SampleStringProperty), String)
        结束获取

        设置(ByVal 值 作为 String)
设置值(SampleStringProperty, 值)
        结束设置
    结束 属性

    使上述字符串数据绑定实际生效的 DependencyProperty

    公共共享只读 SampleStringProperty 作为 DependencyProperty = _
                        DependencyProperty.Register("SampleString", _
                            GetType(String), GetType(SampleViewModel), _
                            新建 PropertyMetadata(Nothing))

结束类
```

可以通过使用 wpfdp 代码片段（输入 wpfdp 后按两次 TAB 键）轻松添加 DependencyProperty，然而，该代码片段类型不安全，且在 Option Strict On 下无法编译。

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>

</Window >
```

Chapter 43: WPF XAML Data Binding

This example shows how to create a ViewModel and a View within the MVVM pattern and WPF, and how to bind the two together, so that each is updated whenever the other is changed.

Section 43.1: Binding a String in the ViewModel to a TextBox in the View

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
```

A DependencyProperty can be easily added by using the `wpfdp` code snippet (type `wpfdp`, then press the `TAB` key twice), however, the code snippet is not type safe, and will not compile under `Option Strict On`.

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

```
类 SampleWindow

    Private WithEvents myViewModel 作为新的 SampleViewModel()

    Private Sub Window_Loaded(sender 作为 Object, e 作为 RoutedEventArgs)
        Me.DataContext = myViewModel
    结束子程序
结束类
```

请注意，这是一种非常初级的实现MVVM和数据绑定的方法。更稳健的做法是使用像Unity这样的平台将ViewModel"注入"到View中。

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
```

Note that this is a very rudimentary way to implement MVVM and databinding. A more robust practice would be to use a platform like Unity to "inject" the ViewModel into the View.

第44章：动态读取压缩文本文件

第44.1节：逐行读取.gz文本文件

此类打开一个.gz文件（压缩日志文件的常见格式），并在每次调用.NextLine()时返回一行

无需临时解压的内存使用，非常适合大文件。

```
Imports System.IO

类 logread_gz

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

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

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

  Function 下一行() 作为 String
    '如果到达文件末尾将返回 Nothing
    Return Me.line_ptr.ReadLine()
  End Function

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

结束类
```

注意：为了可读性，没有故障保护机制。

Chapter 44: Reading compressed textfile on-the-fly

Section 44.1: Reading .gz textfile line after line

This class open a .gz file (usual format of compressed log files) and will return a line at each call of `.NextLine()`

There is no memory usage for temporary decompression, very useful for large file.

```
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
```

Note : there is no failsafe, for readbility purpose.

第45章：线程

第45.1节：使用

Control.Invoke()进行线程安全调用

使用Control.Invoke()方法，您可以将方法或函数的执行从后台线程转移到控件创建的线程上，通常是UI（用户界面）线程。通过这样做，您的代码将被排队在控件的线程上运行，从而消除了并发的可能性。

还应检查Control.InvokeRequired属性，以确定是否需要调用Invoke，
或者代码是否已经在与控件相同的线程上运行。

Invoke()方法的第一个参数是一个委托。委托保存对另一个方法的引用、参数列表和
返回类型。

在Visual Basic 2010（10.0）或更高版本中，可以使用lambda表达式来动态创建委托方法：

```
如果 LogTextBox.InvokeRequired = True 则
    LogTextBox.Invoke(Sub() LogTextBox.AppendText("检查通过"))
否则
LogTextBox.AppendText("检查通过")
结束如果
```

而在Visual Basic 2008（9.0）或更低版本中，您必须自行声明委托：

```
委托子程序 AddLogText(ByVal Text As String)

如果 LogTextBox.InvokeRequired = True 那么
    LogTextBox.Invoke(New AddLogText(AddressOf UpdateLog), "检查通过")
否则
UpdateLog("检查通过")
结束如果

子程序 UpdateLog(ByVal Text As String)
    LogTextBox.AppendText(Text)
结束子程序
```

第45.2节：使用 Async/Await 执行线程安全调用

如果我们尝试从不同线程更改UI线程上的对象，将会得到跨线程操作异常：

```
私有子程序 Button_Click(sender As Object, e As EventArgs) Handles MyButton.Click' 跨线程操作异常
    ，因为赋值在不同于UI线程的线程上执行：

Task.Run(Sub() MyButton.Text = Thread.CurrentThread.ManagedThreadId)
End Sub
```

在 VB 14.0 和 .NET 4.5 之前，解决方案是调用UI线程上对象的赋值操作：

```
私有子程序 Button_Click(sender As Object, e As EventArgs) Handles MyButton.Click
    ' 这将在UI线程上运行代码：
MyButton.Invoke(Sub() MyButton.Text = Thread.CurrentThread.ManagedThreadId)
End Sub
```

Chapter 45: Threading

Section 45.1: Performing thread-safe calls using Control.Invoke()

Using the `Control.Invoke()` method you may move the execution of a method or function from a background thread to the thread that the control was created on, which is usually the UI (User Interface) thread. By doing so your code will be queued to run on the control's thread instead, which removes the possibility of concurrency.

The `Control.InvokeRequired` property should also be checked in order to determine whether you need to invoke, or if the code is already running on the same thread as the control.

The `Invoke()` method takes a delegate as its first parameter. A delegate holds the reference, parameter list and return type to another method.

In Visual Basic 2010 (10.0) or higher, *lambda expressions* can be used to create a delegate method on the fly:

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

Whereas in Visual Basic 2008 (9.0) or lower, you have to declare the delegate on your own:

```
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
```

Section 45.2: Performing thread-safe calls using Async/Await

If we try to change an object on the UI thread from a different thread we will get a cross-thread operation exception:

```
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
```

Before **VB 14.0** and **.NET 4.5** the solution was invoking the assignment on and object living on the UI thread:

```
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
```

使用VB 14.0，我们可以在不同的线程上运行一个Task，然后在执行完成后恢复上下文，接着使用Async/Await进行赋值：

```
Private Async Sub Button_Click(sender As Object, e As EventArgs) Handles MyButton.Click
    ' 这将在不同的线程上运行代码，然后恢复上下文
    ' 这样赋值操作就在UI线程上进行：
MyButton.Text = Await Task.Run(Function() Thread.CurrentThread.ManagedThreadId)
End Sub
```

With **VB 14.0**, we can run a Task on a different thread and then have the context restored once the execution is complete and then perform the assignment with Async/Await:

```
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
```

第46章：多线程

第46.1节：使用Thread类的多线程

此示例使用了Thread类，但多线程应用程序也可以使用BackgroundWorker来实现。AddNumber、SubstractNumber和DivideNumber函数将由不同的线程执行：

编辑：现在UI线程等待子线程完成并显示结果。

```
模块 模块1
    '声明线程并为其分配子程序
    Dim AddThread As New Threading.Thread(AddressOf AddNumber)
    Dim SubstractThread As New Threading.Thread(AddressOf SubstractNumber)
    Dim DivideThread As New Threading.Thread(AddressOf DivideNumber)

    '声明用于保存结果的变量
    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()

        '现在开始线程
AddThread.Start()
       SubstractThread.Start()
       DivideThread.Start()

        '等待并在控制台显示结果
Console.WriteLine("等待线程完成..." )
       Console.WriteLine("")

        While bFinishAddition = False Or bFinishDivision = False Or bFinishSubstration = False
          Threading.Thread.Sleep(50)      'UI线程休眠
            If bFinishAddition And Not bShownAdditionResult Then
              Console.WriteLine("加法结果 : " & addResult)
              bShownAdditionResult = True
            结束如果

            If bFinishSubstration And Not bShownSubstractionResult Then
              Console.WriteLine("减法结果 : " & SubStractResult)
              bShownSubstractionResult = True
            结束如果

            If bFinishDivision  And Not bShownDivisionResult Then
              Console.WriteLine("除法结果 : " & DivisionResult)
              bShownDivisionResult = True
            结束如果

        End While

Console.WriteLine("")
        Console.WriteLine("所有线程已完成。")
```

Chapter 46: Multithreading

Section 46.1: Multithreading using Thread Class

This example uses the `Thread` Class, but multithreaded applications can also be made using `BackgroundWorker`. The `AddNumber`, `SubstractNumber`, and `DivideNumber` functions will be executed by separate threads:

Edit: Now the UI thread waits for the child threads to finish and shows the result.

```
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 作为 整数 = 0 到 100
addResult = addResult + (n1 + n2)
            Threading.Thread.Sleep(50)      '休眠加法线程
        Next
bFinishAddition = True
    End Sub

    Private Sub SubstractNumber()
        Dim n1 作为 整数 = 22
        Dim n2 作为 整数 = 11

        For i 作为 整数 = 0 到 80
SubStractResult = SubStractResult - (n1 - n2)
            Threading.Thread.Sleep(50)
        Next
bFinishSubstration = True
    End Sub

    Private Sub DivideNumber()
        Dim n1 作为 整数 = 22
        Dim n2 作为 整数 = 11
        For i 作为 整数 = 0 到 60
DivisionResult = DivisionResult + (n1 / n2)
            Threading.Thread.Sleep(50)
        Next
bFinishDivision = True
    结束子程序

End Module
```

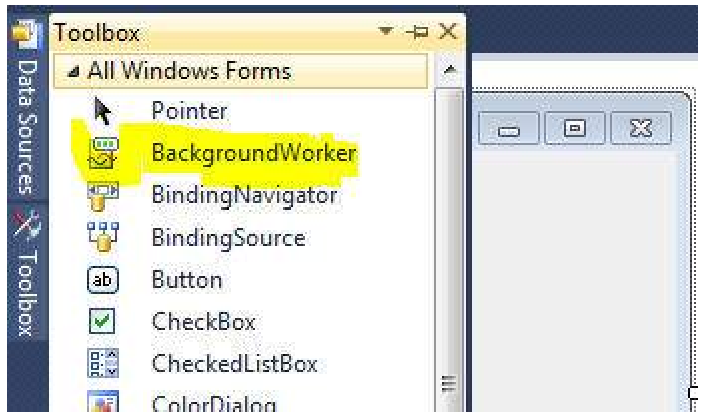
```
        Console.ReadKey()
    End Sub


    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
```

第47章：BackgroundWorker

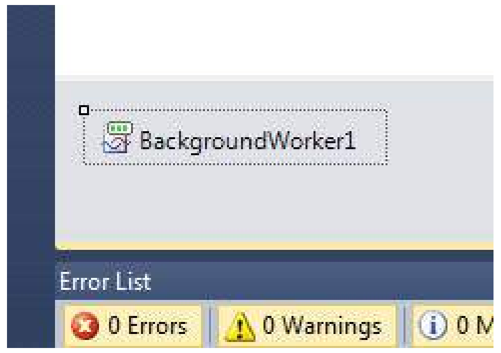
第47.1节：使用BackgroundWorker

使用后台工作者执行任务。

双击工具箱中的BackgroundWorker控件



添加后，BackgroundWorker的显示效果如图所示。



双击添加的控件，进入`BackgroundWorker1_DoWork`事件，添加在调用BackgroundWorker时要执行的代码。示例如下：

```
私有子程序 BackgroundWorker1_DoWork(ByVal sender 作为 System.Object, ByVal e 作为
System.ComponentModel.DoWorkEventArgs) 处理 BackgroundWorker1.DoWork

    “在这里执行耗时的后台任务”

End Sub
```

调用 BackgroundWorker 来执行任务可以在任何事件中进行，比如`Button_Click`、`Textbox_TextChanged`等，方法如下：

```
BackgroundWorker1.RunWorkerAsync()
```

修改RunWorkerCompleted事件以捕获 BackgroundWorker 任务完成事件，方法如下：

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

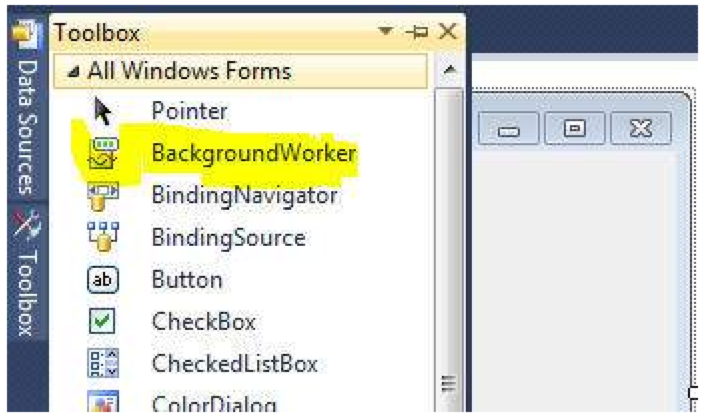
当工作线程完成分配给它的任务时，这将显示一个消息框，内容为Done。

Chapter 47: BackgroundWorker

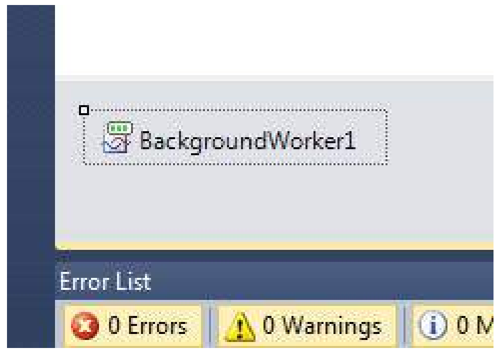
Section 47.1: Using BackgroundWorker

Executing a task with the background worker.

Double Click on the `BackgroundWorker` control from the Toolbox



This is how the BackgroundWorker appears after adding it.



Double click on the added control to get the `BackgroundWorker1_DoWork` event and add the code to be executed when the BackgroundWorker is called. Something like this:

```
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
```

Calling the BackgroundWorker to perform the task can be done at any event like `Button_Click`, `Textbox_TextChanged`, etc. as follows:

```
BackgroundWorker1.RunWorkerAsync()
```

Modify the `RunWorkerCompleted` event to capture the task finished event of the BackgroundWorker as follows:

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

This will display a message box saying `Done` when the worker finishes the task assigned to it.

第47.2节：在 BackgroundWorker中访问GUI组件

你不能从 BackgroundWorker 访问任何 GUI 组件。例如，如果你尝试执行如下操作

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

你将收到运行时错误，提示"跨线程操作无效：控件 'TextBox1' 被从创建它的线程以外的线程访问。"

这是因为 BackgroundWorker 在另一个线程上并行运行你的代码，而 GUI 组件不是线程安全的。你必须使用Invoke方法将代码设置为在主线程上运行，传入一个委托：

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

或者你可以使用 BackgroundWorker 的 ReportProgress 方法：

```
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
```

Section 47.2: Accessing GUI components in BackgroundWorker

You cannot access any GUI components from the BackgroudWorker. For example if you try to do something like this

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

you will receive a runtime error saying that "Cross-thread operation not valid: Control 'TextBox1' accessed from a thread other than the thread it was created on."

This is because the BackgroundWorker runs your code on another thread in parallel with the main thread, and the GUI components are not thread-safe. You have to set your code to be run on the main thread using the `Invoke` method, giving it a delegate:

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

Or you can use the ReportProgress method of the BackgroundWorker:

```
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
```

第48章：使用 BackgroundWorker

第48.1节：BackgroundWorker 的基本实现类

使用 BackgroundWorker 需要导入 System.ComponentModel

```
Imports System.ComponentModel
```

然后声明一个私有变量

```
Private bgWorker As New BackgroundWorker
```

您需要为后台工作者的 DoWork 和 RunWorkerCompleted 事件创建两个方法并分配给它们。

```
Private Sub MyWorker_DoWork(ByVal sender As System.Object, ByVal e As
System.ComponentModel.DoWorkEventArgs)
  '在此添加工作线程执行的代码

End Sub
```

以下子程序将在工作线程完成任务时执行

```
Private Sub MyWorker_RunWorkerCompleted(ByVal sender As Object, ByVal e As
System.ComponentModel.RunWorkerCompletedEventArgs)
    '在此添加工作线程完成任务后执行的代码

End Sub
```

然后在你的代码中添加以下代码以启动后台工作线程

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

当你调用 RunWorkerAsync() 函数时，MyWorker_DoWork 将被执行。

Chapter 48: Using BackgroundWorker

Section 48.1: Basic implementation of Background worker class

You need to import System.ComponentModel for using background worker

```
Imports System.ComponentModel
```

Then Declare a private variable

```
Private bgWorker As New BackgroundWorker
```

You need to create two methods for background worker's DoWork and RunWorkerCompleted events and assign them.

```
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
```

The below sub will be executed when the worker finishes the job

```
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
```

Then within your code add the below lines to start the background worker

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

When you call RunWorkerAsync() function, MyWorker_DoWork will be executed.

第49章：基于任务的异步模式

第49.1节：Async/Await的基本用法

你可以并行启动一些耗时的过程，然后在它们完成时收集结果：

```
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
结束函数

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

    Return 60
End Function
```

两秒后，两个结果都将可用。

第49.2节：在LINQ中使用TAP

您可以通过将 AddressOf AsyncMethod 传递给 LINQ SELECT 方法来创建一个 IEnumerable 的 Task，然后使用 Task.WhenAll 启动并等待所有结果

如果您的方法具有与之前LINQ链调用匹配的参数，它们将被自动映射。

```
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
```

要映射不同的参数，您可以用 lambda 替换AddressOf Method：

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

Chapter 49: Task-based asynchronous pattern

Section 49.1: Basic usage of Async/Await

You can start some slow process in parallel and then collect the results when they are done:

```
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
```

After two seconds both the results will be available.

Section 49.2: Using TAP with LINQ

You can create an `IEnumerable` of `Task` by passing **`AddressOf`** `AsyncMethod` to the **LINQ** **`SELECT`** method and then start and wait all the results with `Task.WhenAll`

If your method has parameters matching the previous **LINQ** chain call, they will be automatically mapped.

```
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
```

To map different arguments you can replace **`AddressOf`** `Method` with a lambda:

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

第50章：调试您的应用程序

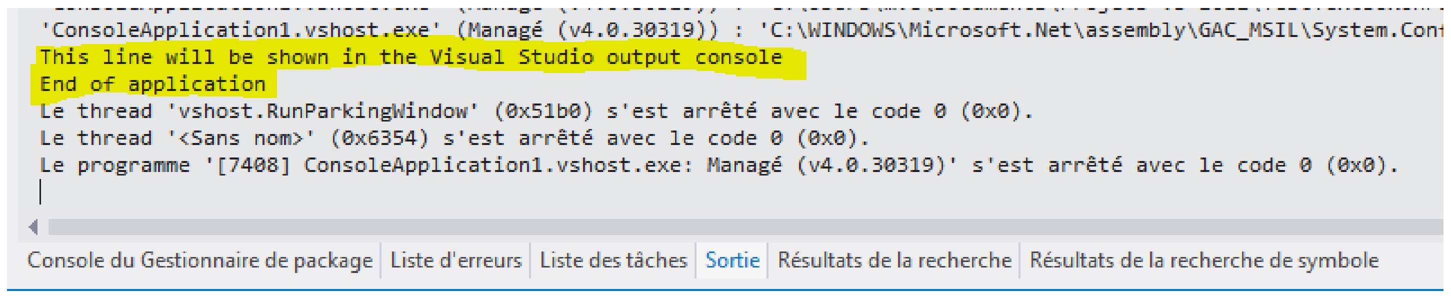
每当您的代码出现问题时，了解内部发生了什么总是一个好主意。 .Net Framework 中的类 System.Diagnostics.Debug将在这项任务中大有帮助。

Debug 类的第一个优点是，只有在以调试模式构建应用程序时才会生成代码。
当你以发布模式构建应用程序时，Debug 调用不会生成任何代码。

第50.1节：控制台中的调试

```
模块 模块1
    Sub Main()
Debug.WriteLine("这行将在 Visual Studio 输出控制台显示")

        Console.WriteLine("按任意键退出")
Console.ReadKey()

        Debug.WriteLine("应用程序结束")
    End Sub
End Module
```

将产生：



```
'ConsoleApplication1.vshost.exe' (Managé (v4.0.30319)) : 'C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System.Con
This line will be shown in the Visual Studio output console
End of application
Le thread 'vshost.RunParkingWindow' (0x51b0) s'est arrêté avec le code 0 (0x0).
Le thread '<Sans nom>' (0x6354) s'est arrêté avec le code 0 (0x0).
Le programme '[7408] ConsoleApplication1.vshost.exe: Managé (v4.0.30319)' s'est arrêté avec le code 0 (0x0).
```

Console du Gestionnaire de package
Liste d'erreurs
Liste des tâches
Sortie
Résultats de la recherche
Résultats de la recherche de symbole

第50.2节：缩进你的调试输出

```
模块 模块1

    Sub Main()
Debug.WriteLine("开始应用程序")

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

       Console.WriteLine("按任意键退出")
        Console.ReadKey()

Debug.WriteLine("应用程序结束")
    End Sub

    Sub LoopAndDoStuff(Iterations As Integer)
        Dim x As Integer = 0
Debug.WriteLine("开始循环")
        Debug.Indent()
        For i As Integer = 0 To Iterations - 1
Debug.Write("第 " & (i + 1).ToString() & " 次，共 " & Iterations.ToString() & " 次："
值 X: ")
          x += (x + 1)
```

Chapter 50: Debugging your application

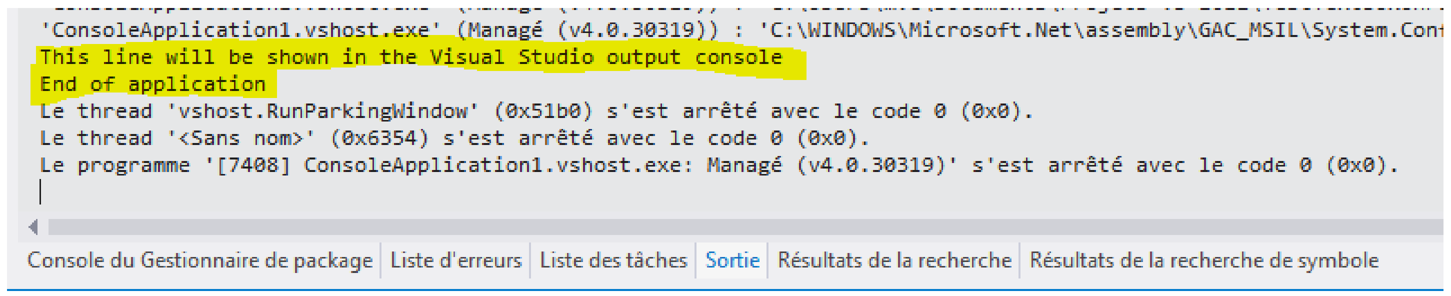
Whenever you have a problem in your code, it is always a good idea to know what is going on inside. The class System.Diagnostics.Debug in .Net Framework will help you a lot in this task.

The first advantage of the Debug class is that it produces code only if you build your application in Debug mode. When you build your application in Release mode, no code will be generated from the Debug calls.

Section 50.1: Debug in the console

```
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
```

will produce:



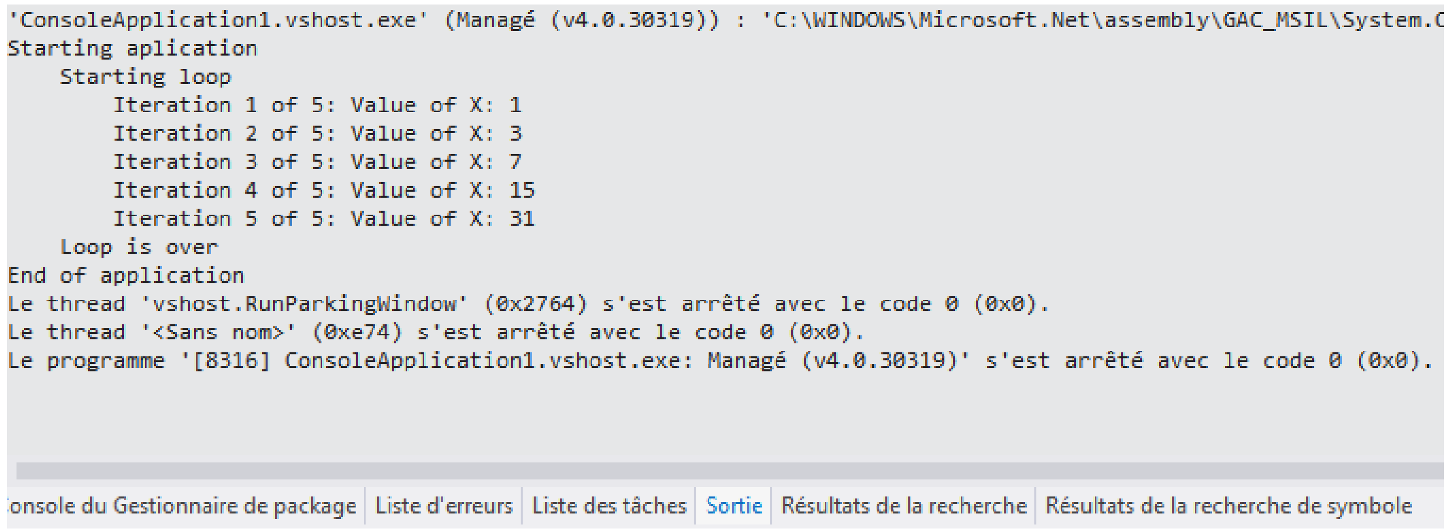
```
'ConsoleApplication1.vshost.exe' (Managé (v4.0.30319)) : 'C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System.Con
This line will be shown in the Visual Studio output console
End of application
Le thread 'vshost.RunParkingWindow' (0x51b0) s'est arrêté avec le code 0 (0x0).
Le thread '<Sans nom>' (0x6354) s'est arrêté avec le code 0 (0x0).
Le programme '[7408] ConsoleApplication1.vshost.exe: Managé (v4.0.30319)' s'est arrêté avec le code 0 (0x0).
```

Console du Gestionnaire de package
Liste d'erreurs
Liste des tâches
Sortie
Résultats de la recherche
Résultats de la recherche de symbole

Section 50.2: Indenting your debug output

```
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("循环结束")
    End Sub
End Module
```

将产生：



```
'ConsoleApplication1.vshost.exe' (Managé (v4.0.30319)) : 'C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System.C
Starting aplication
    Starting loop
        Iteration 1 of 5: Value of X: 1
        Iteration 2 of 5: Value of X: 3
        Iteration 3 of 5: Value of X: 7
        Iteration 4 of 5: Value of X: 15
        Iteration 5 of 5: Value of X: 31
    Loop is over
End of application
Le thread 'vshost.RunParkingWindow' (0x2764) s'est arrêté avec le code 0 (0x0).
Le thread '<Sans nom>' (0xe74) s'est arrêté avec le code 0 (0x0).
Le programme '[8316] ConsoleApplication1.vshost.exe: Managé (v4.0.30319)' s'est arrêté avec le code 0 (0x0).
```

onsole du Gestionnaire de package
Liste d'erreurs
Liste des tâches
Sortie
Résultats de la recherche
Résultats de la recherche de symbole

第 50.3 节：文本文件中的调试

在应用程序开始时，必须向 Debug 类的 Listeners 列表添加一个 TextWriterTraceListener。

```
模块 模块1

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

Debug.WriteLine("开始应用程序")

        Console.WriteLine("按任意键退出")
        Console.ReadKey()

Debug.WriteLine("应用程序结束")
    End Sub
End Module
```

所有生成的调试代码将输出到Visual Studio控制台和您选择的文本文件中。

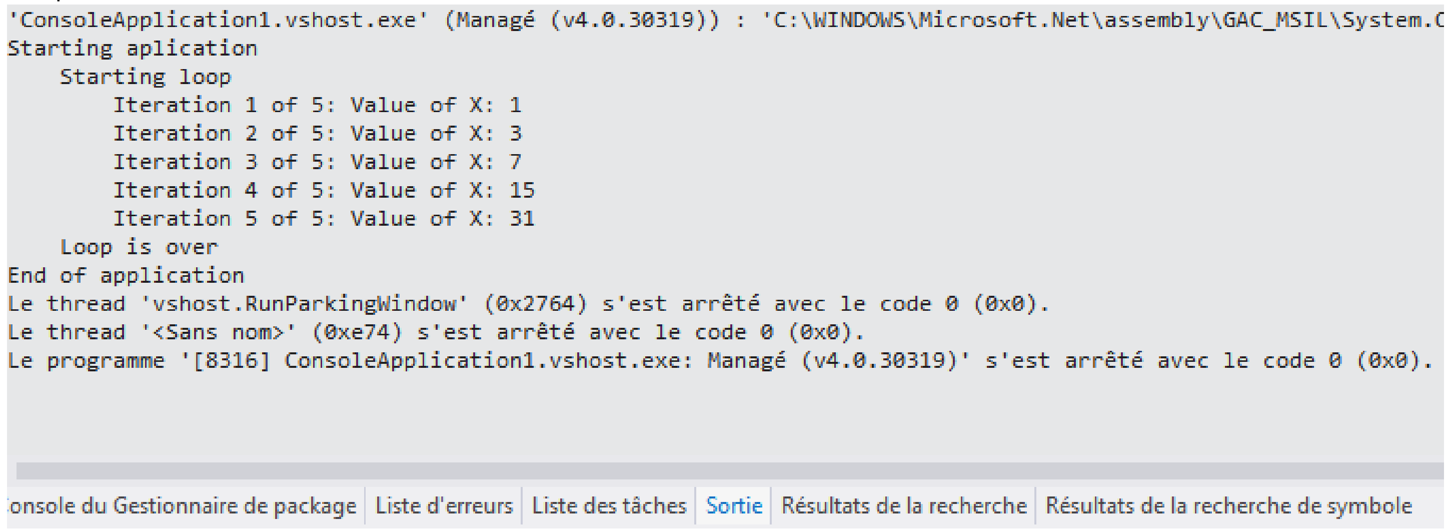
如果文件始终相同：

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

每次输出都会追加到该文件中，并且会生成一个以GUID开头然后是您的文件名的新文件。

```
        Debug.WriteLine(x.ToString())
      Next
      Debug.Unindent()
      Debug.WriteLine("Loop is over")
    End Sub
End Module
```

will produce:



```
'ConsoleApplication1.vshost.exe' (Managé (v4.0.30319)) : 'C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System.C
Starting aplication
    Starting loop
        Iteration 1 of 5: Value of X: 1
        Iteration 2 of 5: Value of X: 3
        Iteration 3 of 5: Value of X: 7
        Iteration 4 of 5: Value of X: 15
        Iteration 5 of 5: Value of X: 31
    Loop is over
End of application
Le thread 'vshost.RunParkingWindow' (0x2764) s'est arrêté avec le code 0 (0x0).
Le thread '<Sans nom>' (0xe74) s'est arrêté avec le code 0 (0x0).
Le programme '[8316] ConsoleApplication1.vshost.exe: Managé (v4.0.30319)' s'est arrêté avec le code 0 (0x0).
```

onsole du Gestionnaire de package
Liste d'erreurs
Liste des tâches
Sortie
Résultats de la recherche
Résultats de la recherche de symbole

Section 50.3: Debug in a text file

At the beginning of your application, your must add a TextWriterTraceListener to the Listeners list of the Debug class.

```
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
```

All the Debug code produced will be outputed in the Visual Studio console AND in the text file you chose.

If the file is always the same:

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

The output will be appended to the file every time AND a new file starting with a GUID then your filename will be generated.

第51章：VB.NET中的单元测试

第51.1节：税务计算的单元测试

此示例分为两个部分

- **SalaryCalculation类**：计算扣税后的净工资
- **SalaryCalculationTests类**: 用于测试计算净工资的方法

步骤1： 创建类库，命名为WagesLibrary或任何合适的名称。然后将类重命名为 **SalaryCalculation**

''' '' 薪资计算类 ''' Public Class SalaryCalculation

```
    ''' <summary>
    ''' 员工薪资
    ''' </summary>
    公共共享 薪资 作为 双精度浮点数

    ''' <summary>
    ''' 税率 (0-1)
    ''' </summary>
    公共共享 税率 作为 双精度浮点数

    ''' <summary>
    ''' 计算净薪资的函数
    ''' </summary>
    ''' <returns></returns>
    公共共享函数 计算净薪资()
        返回 薪资 - 薪资 * 税率
    结束函数
结束类
```

步骤 2 ：创建单元测试项目。添加对已创建类库的引用并粘贴以下代码

```
导入 WagesLibrary '你想测试的类库

''' <summary>
''' 用于测试薪资计算的测试类
''' </summary>
<TestClass()> 公共类 薪资计算测试类

    ''' <summary>
    ''' CalculateNetSalary 方法的测试用例
    ''' </summary>
    <TestMethod()> Public Sub CalculateNetSalaryTest()
        SalaryCalculation.Salary = 100
        SalaryCalculation.Tax = 0.1
Assert.AreEqual(90.0, SalaryCalculation.CalculateNetSalary(), 0.1)
    结束子程序
结束类
```

Assert.Equal 检查预期值与实际计算值。值 0.1 用于允许预期结果与实际结果之间的容差或变动。

步骤3：运行该方法的测试以查看结果

Chapter 51: Unit Testing in VB.NET

Section 51.1: Unit Testing for Tax Calculation

This example is divided into two pillars

- **SalaryCalculation Class** : Calculating the net salary after tax deduction
- **SalaryCalculationTests Class** : For testing the method that calculates the net salary

Step 1: Create Class Library, name it **WagesLibrary** or any appropriate name. Then rename the class to **SalaryCalculation**

''' ''' Class for Salary Calculations ''' Public Class SalaryCalculation

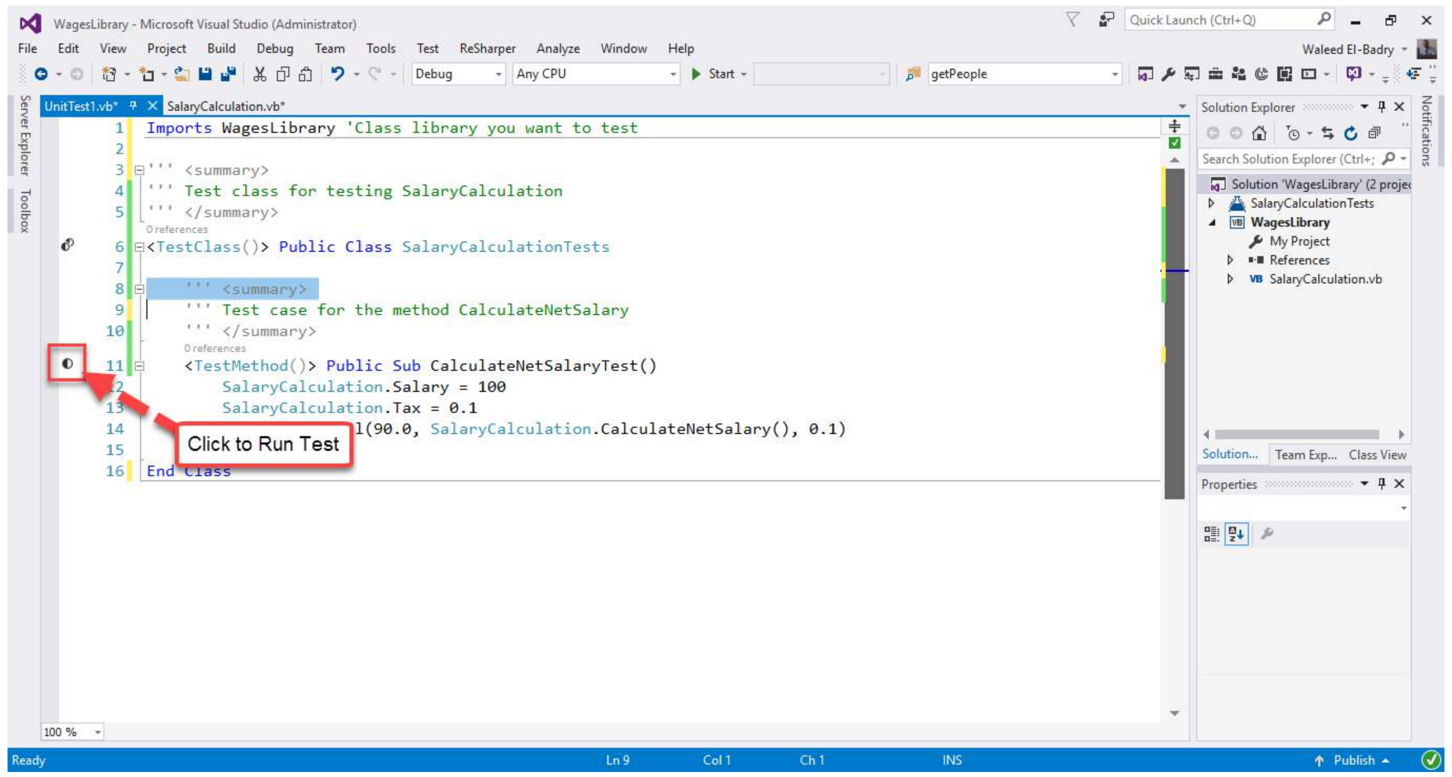
```
    ''' <summary>
    ''' Employee Salary
    ''' </summary>
    Public Shared Salary As Double

    ''' <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
```

Step 2 : Create Unit Test Project. Add reference to the created class library and paste the below code

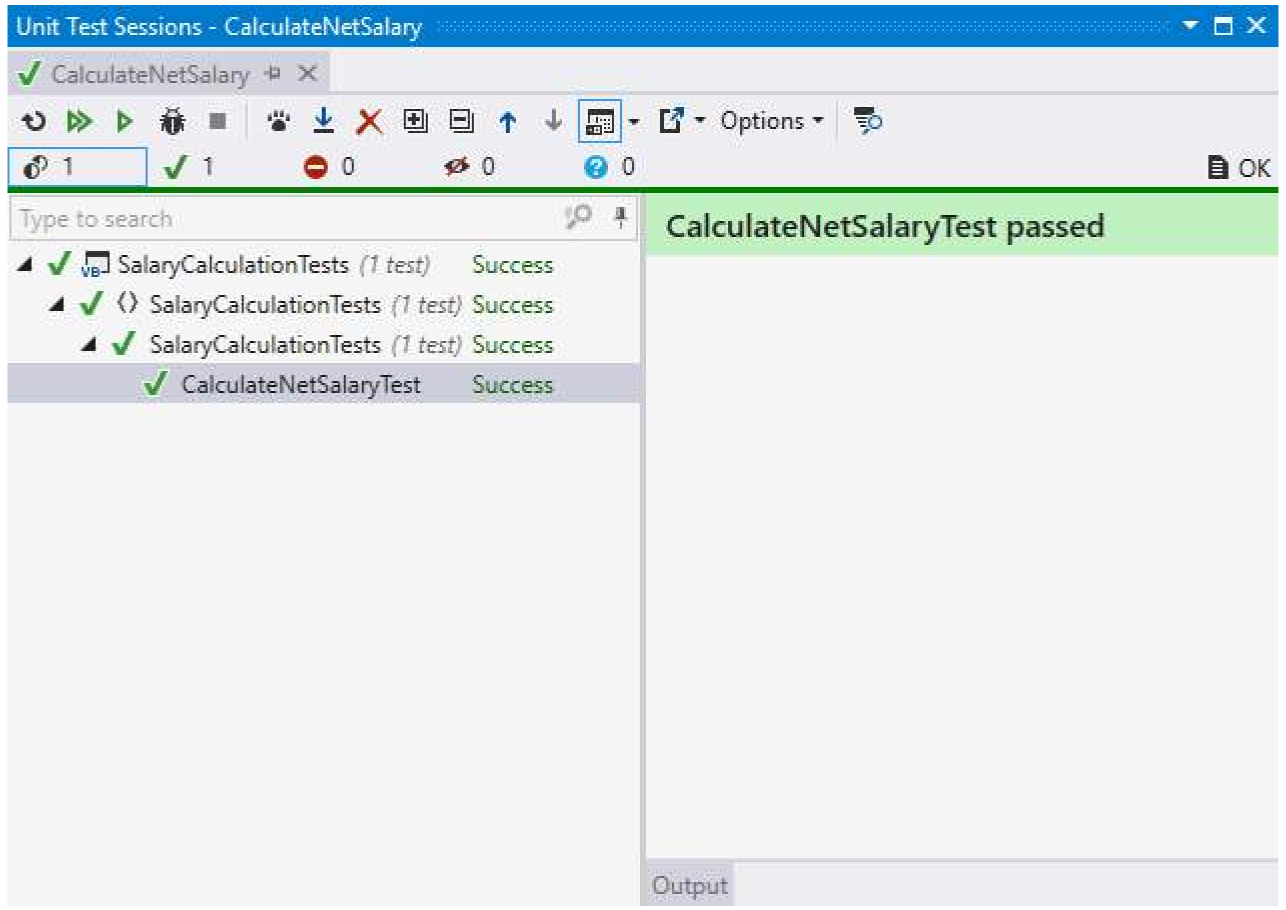
```
Imports WagesLibrary 'Class library you want to test

''' <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` checks the expected value against the actual calculated value. the value `0.1` is used to allow tolerance or variation between *expected* and *actual* result.

Step 3 : Run the test of the method to see result



```
Imports WagesLibrary 'Class library you want to test

''' <summary>
''' Test class for testing SalaryCalculation
''' </summary>
0 references
<TestClass()> Public Class SalaryCalculationTests

    ''' <summary>
    ''' Test case for the method CalculateNetSalary
    ''' </summary>
    0 references
    <TestMethod()> Public Sub CalculateNetSalaryTest()
        SalaryCalculation.Salary = 100
        SalaryCalculation.Tax = 0.1
                                   1(90.0, SalaryCalculation.CalculateNetSalary(), 0.1)

End Class
```



```
Unit Test Sessions - CalculateNetSalary
CalculateNetSalary
1   1   0   0   0   OK

SalaryCalculationTests (1 test)  Success
  SalaryCalculationTests (1 test)  Success
    SalaryCalculationTests (1 test)  Success
      CalculateNetSalaryTest  Success

CalculateNetSalaryTest passed
```

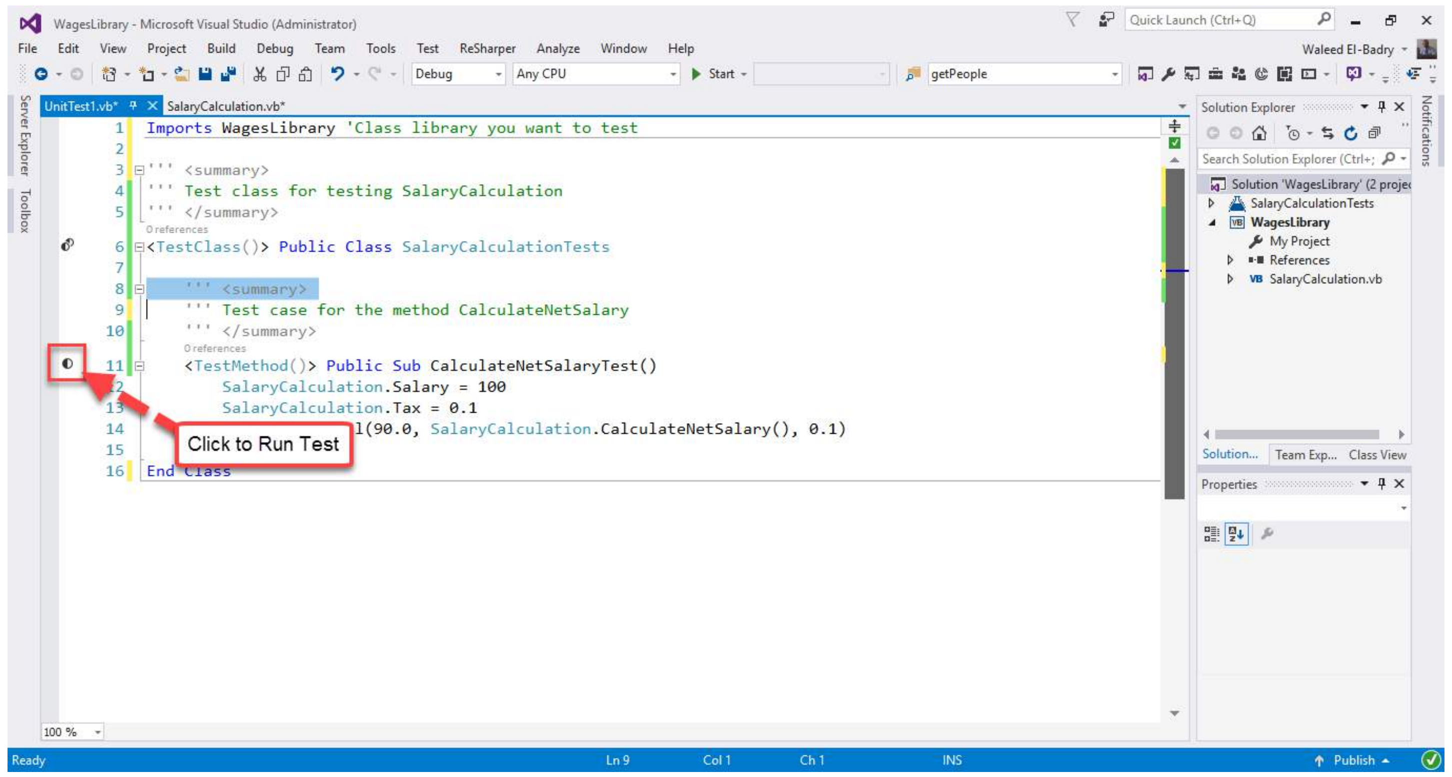
测试结果

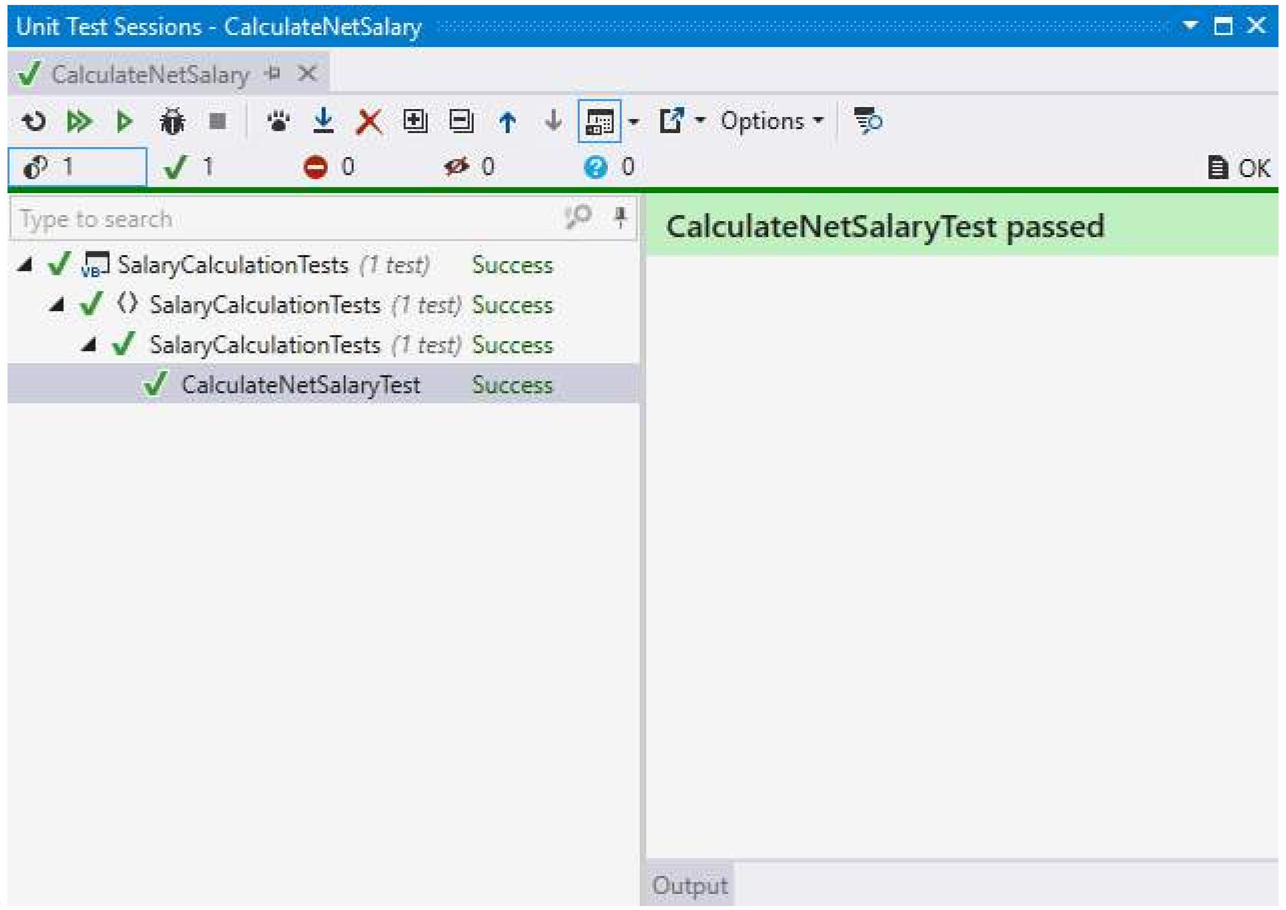
第51.2节：测试员工类的赋值和派生属性

此示例在单元测试中有更多可用的测试。

Employee.vb（类库）

```
''' <summary>
```





Test result

Section 51.2: Testing Employee Class assigned and derived Properties

This example has more tests available in unit testing.

Employee.vb (Class Library)

```
''' <summary>
''' 员工类别
''' </summary>
公共类 员工

    ''' <summary>
    ''' 员工的名字
    ''' </summary>
    公共属性 名字 作为 字符串 = ""

    ''' <summary>
    ''' 员工的姓氏
    ''' </summary>
    公共属性 姓氏 作为 字符串 = ""

    ''' <summary>
    ''' 员工的全名
    ''' </summary>
    公共只读属性 全名 作为 字符串 = ""

    ''' <summary>
    ''' 员工的年龄
    ''' </summary>
    公共属性 年龄 作为 字节

    ''' <summary>
    ''' 实例化新的员工对象
    ''' </summary>
    ''' <param name="firstName">员工名字</param>
    ''' <param name="lastName">员工姓氏</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)
    结束子程序
结束类
```

EmployeeTest.vb（测试项目）

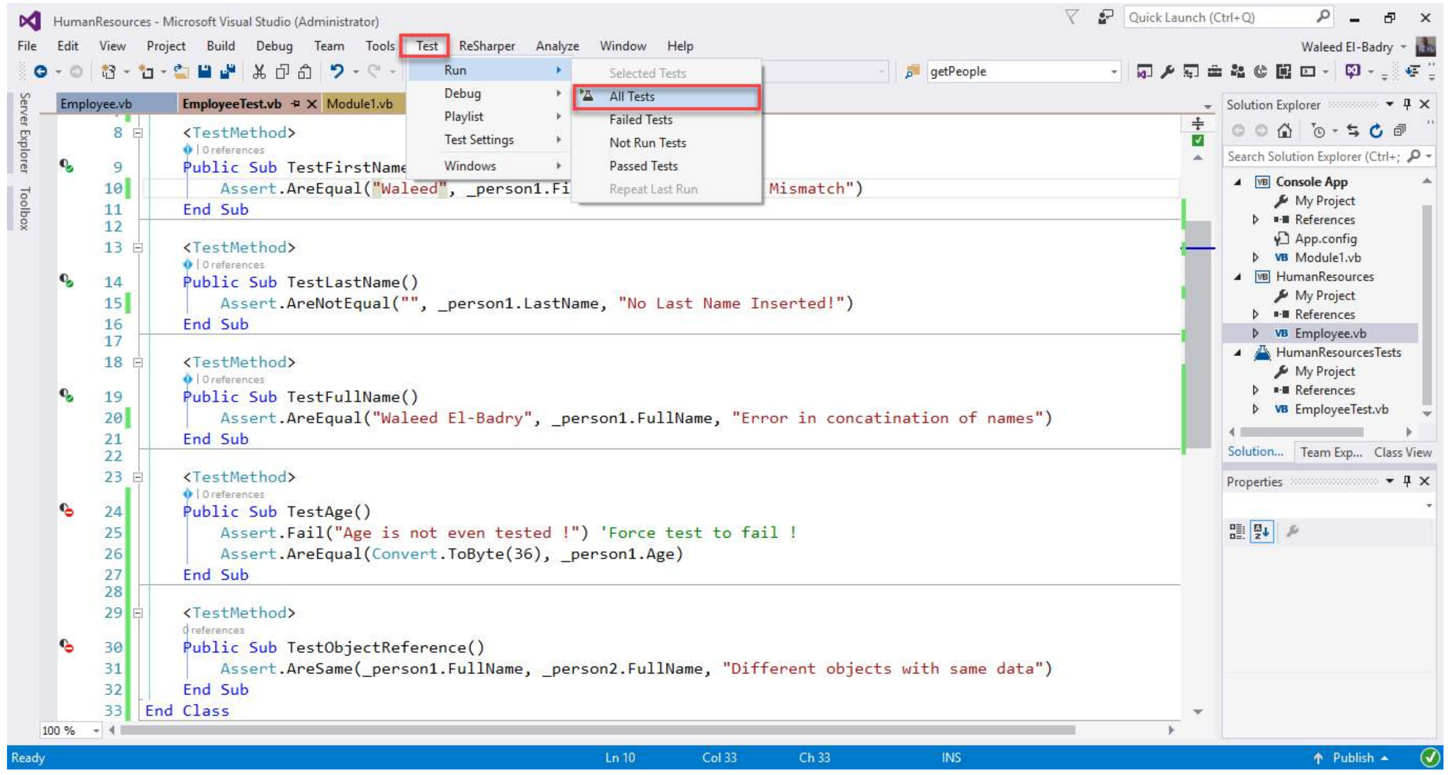
```
Imports HumanResources

<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, "名字不匹配")
    End Sub

    <TestMethod>
    Public Sub TestLastName()
Assert.AreNotEqual("", _person1.LastName, "未插入姓氏！")
    End Sub

    <TestMethod>
    Public Sub TestFullName()
Assert.AreEqual("Waleed El-Badry", _person1.FullName, "姓名拼接错误")
    End Sub
```

```
''' 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 (Test Project)

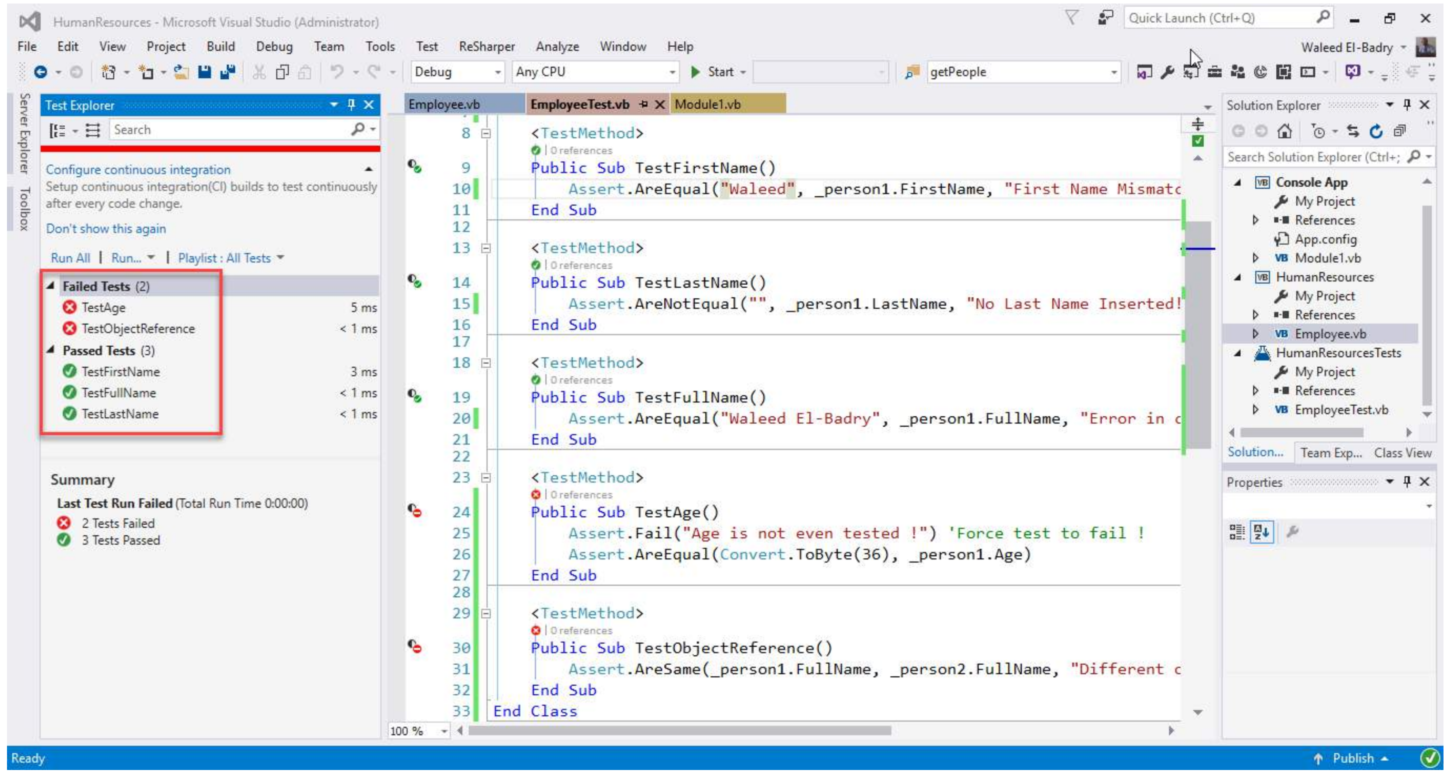
```
Imports HumanResources

<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("年龄甚至没有测试！") '强制测试失败！
Assert.AreEqual(Convert.ToByte(36), _person1.Age)
    End Sub

    <TestMethod>
    Public Sub TestObjectReference()
Assert.AreSame(_person1.FullName, _person2.FullName, "数据相同但对象不同")
    End Sub
结束类
```

运行测试后的结果



HumanResources - Microsoft Visual Studio (Administrator)

```
    <TestMethod>
    Public Sub TestFirstName
        Assert.AreEqual("Waleed", _person1.Fi                                   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
```



Test Explorer

Configure continuous integration
Setup continuous integration(CI) builds to test continuously after every code change.
Don't show this again

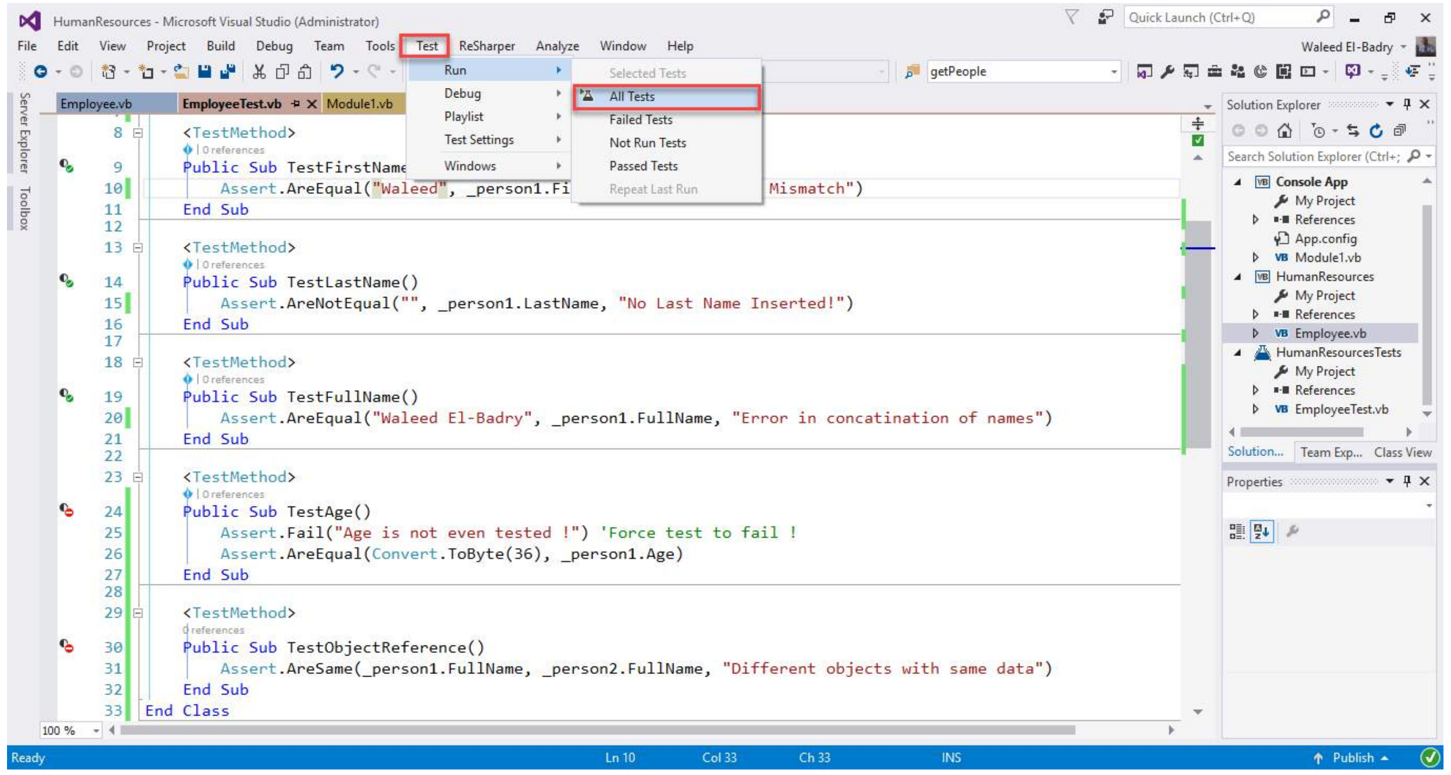
Run All | Run... | Playlist : All Tests

Failed Tests (2)	
TestAge	5 ms
TestObjectReference	< 1 ms
Passed Tests (3)	
TestFirstName	3 ms
TestFullName	< 1 ms
TestLastName	< 1 ms

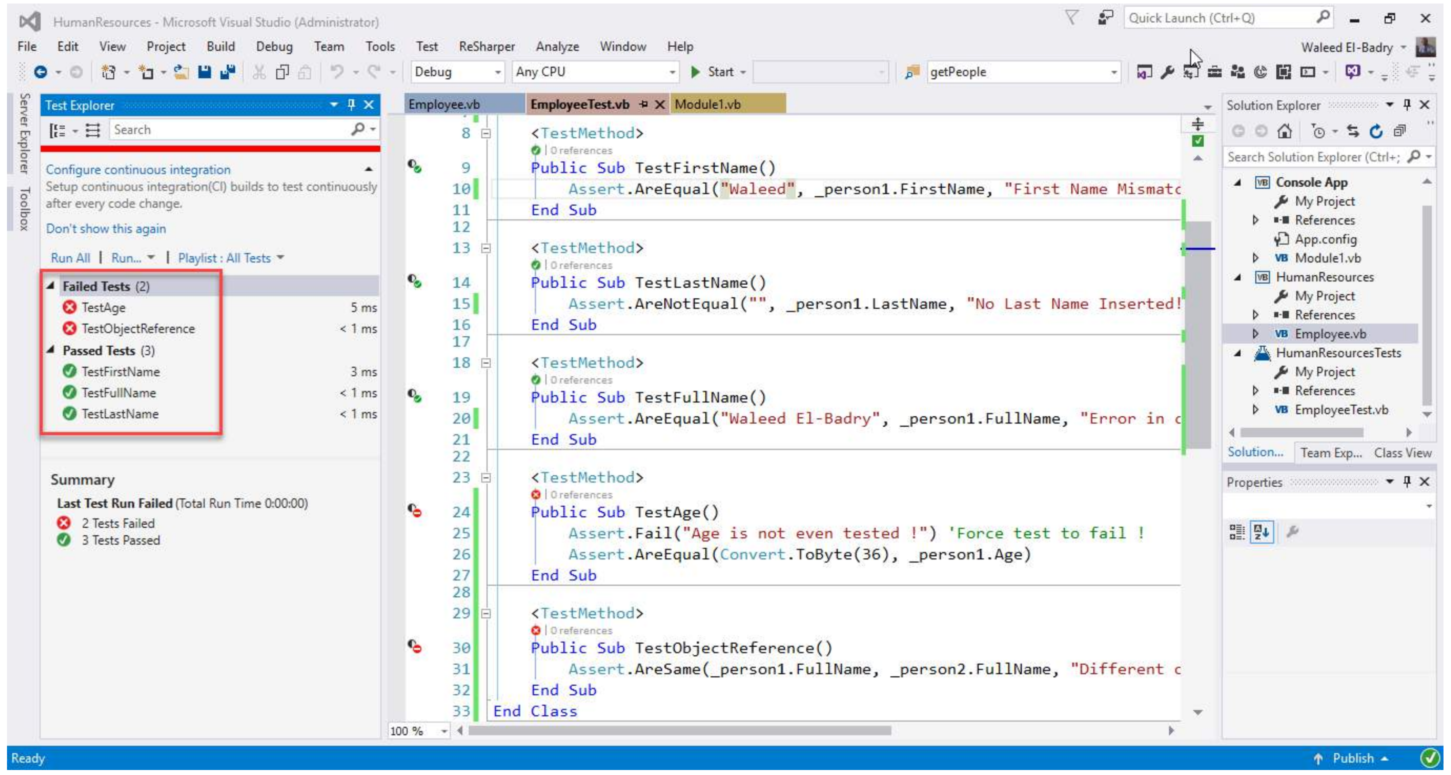
Summary
Last Test Run Failed (Total Run Time 0:00:00)
2 Tests Failed
3 Tests Passed

```
    <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
```

Result after running tests



HumanResources - Microsoft Visual Studio (Administrator)
Test > Run > All Tests



Test Explorer
Failed Tests (2): TestAge 5 ms, TestObjectReference < 1 ms
Passed Tests (3): TestFirstName 3 ms, TestFullName < 1 ms, TestLastName < 1 ms
Summary
Last Test Run Failed (Total Run Time 0:00:00)
2 Tests Failed
3 Tests Passed

鸣谢

非常感谢所有来自Stack Overflow Documentation的人员帮助提供这些内容，
更多更改可以发送至web@petercv.com以发布或更新新内容

亚当·扎克曼	第18章和第31章
亚历山德罗·马斯科洛	第26章
亚历克斯·B.	第30章
艾伦·比努亚	第5章
安德鲁·莫顿	第28章
阿克萨里达克斯	第34章
巴比伦帕	第6、10和12章
巴特·乔林	第6章
贝尔肯·乌萨尔	第20和42章
比约恩	第4章和第35章
布莱克伍德	第35章
地堡心态	第8章
卡洛斯·博劳	第39章
凯里·邦多克	第1章、第4章和第17章
切坦·桑加尼	第5章
科迪·格雷	第2章和第14章
丹·德鲁斯	第36章
达伦·戴维斯	第2章和第23章
大卫	第32章
大卫·威尔逊	第22章
辩论者	第12章
djv	第10章和第38章
Dman	第9章和第40章
Drarig29	第8章
DrDonut	第9章和第11章
ElektroStudios	第10章
Fütemire	第2章、第9章、第33章和第35章
glaubergft	第2章
Happypig375	第29章和第31章
Harjot	第1章
伊姆兰·阿里·汗	第12章
InteXX	第33章
JDC	第24章和第32章
乔纳斯·赫斯	第15章
约瑟夫·琼斯	第47章
肯德拉	第6章
keronconk	第2章
kodkod	第10章
LogicalFlaps	第2章和第30章
卢卡毛里	第19章
卢克·谢泼德	第9章、第13章和第31章
马克	第12章
马克·赫德	第3章和第8章
马丁·索尔斯	第19章
马丁·维尔扬斯	第1、3和50章
马特	第34章
马特·威尔科	第7、13、25、28、33和35章

Credits

Thank you greatly to all the people from Stack Overflow Documentation who helped provide this content,
more changes can be sent to web@petercv.com for new content to be published or updated

Adam Zuckerman	Chapters 18 and 31
Alessandro Mascolo	Chapter 26
Alex B.	Chapter 30
Allen Binuya	Chapter 5
Andrew Morton	Chapter 28
Axarydax	Chapter 34
Babbillumpa	Chapters 6, 10 and 12
Bart Jolling	Chapter 6
Berken Usar	Chapters 20 and 42
Bjørn	Chapters 4 and 35
Blackwood	Chapter 35
BunkerMentality	Chapter 8
Carlos Borau	Chapter 39
Cary Bondoc	Chapters 1, 4 and 17
Chetan Sanghani	Chapter 5
Cody Gray	Chapters 2 and 14
Dan Drews	Chapter 36
Darren Davies	Chapters 2 and 23
David	Chapter 32
David Wilson	Chapter 22
debater	Chapter 12
djv	Chapters 10 and 38
Dman	Chapters 9 and 40
Drarig29	Chapter 8
DrDonut	Chapters 9 and 11
ElektroStudios	Chapter 10
Fütemire	Chapters 2, 9, 33 and 35
glaubergft	Chapter 2
Happypig375	Chapters 29 and 31
Harjot	Chapter 1
Imran Ali Khan	Chapter 12
InteXX	Chapter 33
JDC	Chapters 24 and 32
Jonas_Hess	Chapter 15
Jones Joseph	Chapter 47
Kendra	Chapter 6
keronconk	Chapter 2
kodkod	Chapter 10
LogicalFlaps	Chapters 2 and 30
lucamauri	Chapter 19
Luke Sheppard	Chapters 9, 13 and 31
Mark	Chapter 12
Mark Hurd	Chapters 3 and 8
Martin Soles	Chapter 19
Martin Verjans	Chapters 1, 3 and 50
Matt	Chapter 34
Matt Wilko	Chapters 7, 13, 25, 28, 33 and 35

MatVAD	第2、4、8、12、14、16、31、46和48章
迈克·罗伯逊	第16章
米利伦 X	第43章
Misaz	第13、14、18、25、36和37章
Nadeem_MK	第35章
Nathan	第41章
Nathan Tuggy	第5章
Nico Agusta	第1、3、16和31章
Nitram	第35章
Proger_Cbsk	第11章和第44章
罗伯特·哥伦比亚	第4、5、8和21章
皇家土豆	第4章
瑞安·托马斯	第8章
萨姆·阿克斯	第4、8、19、25、28和33章
sansknwoledge	第29章
斯科特·米切尔	第12章
Seandk	第9章
渴望	第2、3、8、11、18、26和36章
沙扬·托克拉伊	第47章
Shog9	第10章
SilverShotBee	第12和38章
StardustGogeta	第1和19章
斯特法诺·丹东尼奥	第33、35、45和49章
史蒂文·多加特	第8和10章
TuxCopter	第8章
TyCobb	第12章
varocarbas	第4章
vbnet3d	第8、12、13、14、19、28和38章
维沙尔	第31章
视觉文森特	第45章
无效	第4、10、11、12和34章
VortixDev	第8和19章
VV5198722	第27章
wbadry	第51章
泽夫·斯皮茨	第36章
zyabin101	第8章

MatVAD	Chapters 2, 4, 8, 12, 14, 16, 31, 46 and 48
Mike Robertson	Chapter 16
Milliron X	Chapter 43
Misaz	Chapters 13, 14, 18, 25, 36 and 37
Nadeem_MK	Chapter 35
Nathan	Chapter 41
Nathan Tuggy	Chapter 5
Nico Agusta	Chapters 1, 3, 16 and 31
Nitram	Chapter 35
Proger_Cbsk	Chapters 11 and 44
Robert Columbia	Chapters 4, 5, 8 and 21
RoyalPotato	Chapter 4
Ryan Thomas	Chapter 8
Sam Axe	Chapters 4, 8, 19, 25, 28 and 33
sansknwoledge	Chapter 29
Scott Mitchell	Chapter 12
Seandk	Chapter 9
Sehnsucht	Chapters 2, 3, 8, 11, 18, 26 and 36
Shayan Toqraee	Chapter 47
Shog9	Chapter 10
SilverShotBee	Chapters 12 and 38
StardustGogeta	Chapters 1 and 19
Stefano d'Antonio	Chapters 33, 35, 45 and 49
Steven Doggart	Chapters 8 and 10
TuxCopter	Chapter 8
TyCobb	Chapter 12
varocarbas	Chapter 4
vbnet3d	Chapters 8, 12, 13, 14, 19, 28 and 38
Vishal	Chapter 31
Visual Vincent	Chapter 45
void	Chapters 4, 10, 11, 12 and 34
VortixDev	Chapters 8 and 19
VV5198722	Chapter 27
wbadry	Chapter 51
Zev Spitz	Chapter 36
zyabin101	Chapter 8

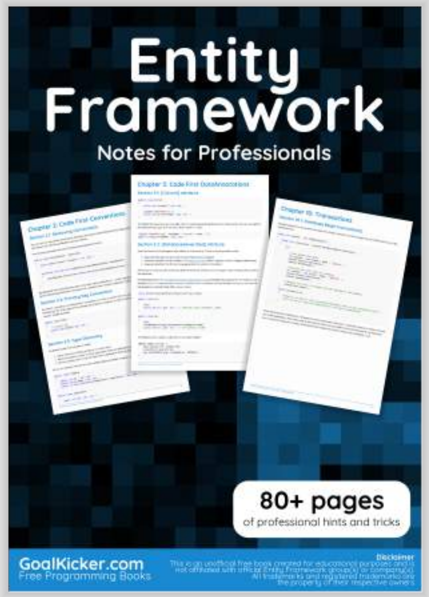
You may also like

.NET Framework
Notes for Professionals
100+ pages
of professional hints and tricks
GoalKicker.com
Free Programming Books

C
Notes for Professionals
300+ pages
of professional hints and tricks
GoalKicker.com
Free Programming Books

C#
Notes for Professionals
700+ pages
of professional hints and tricks
GoalKicker.com
Free Programming Books

C++
Notes for Professionals
600+ pages
of professional hints and tricks
GoalKicker.com
Free Programming Books


Entity Framework
Notes for Professionals
80+ pages
of professional hints and tricks
GoalKicker.com
Free Programming Books

Excel® VBA
Notes for Professionals
100+ pages
of professional hints and tricks
GoalKicker.com
Free Programming Books

Microsoft® SQL Server®
Notes for Professionals
200+ pages
of professional hints and tricks
GoalKicker.com
Free Programming Books

SQL
Notes for Professionals
100+ pages
of professional hints and tricks
GoalKicker.com
Free Programming Books

VBA
Notes for Professionals
100+ pages
of professional hints and tricks
GoalKicker.com
Free Programming Books