

# Construction Operation Simulation

Lecture #5

## Introduction to VB.NET



Amin Alvanchi, PhD

Construction Engineering and Management



[LinkedIn](#)



[Instagram](#)



[WebPage](#)

Department of Civil Engineering, Sharif University of Technology



# Outline

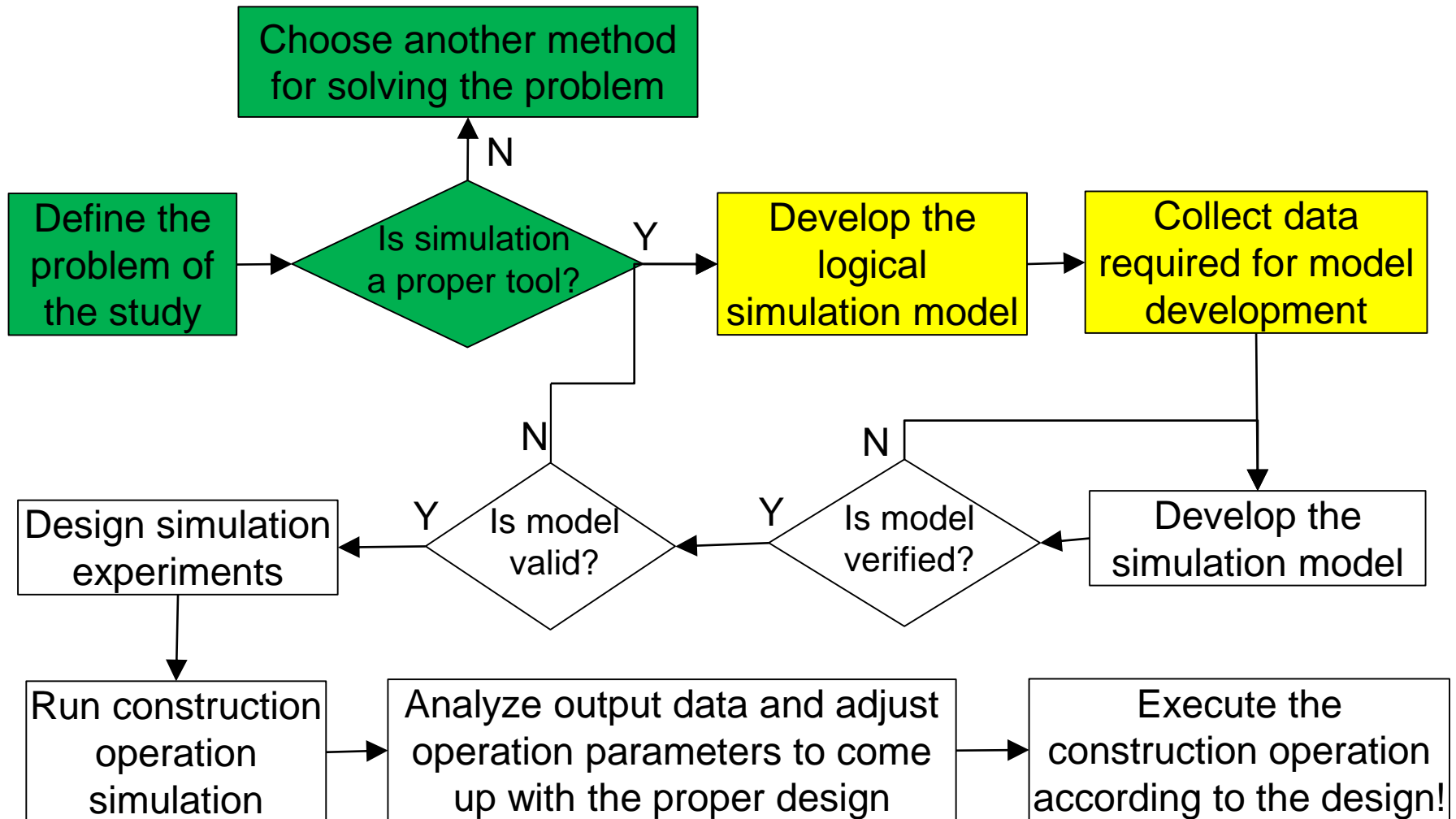
2

- Introduction
- VB.NET Controls Basics
- VB.NET Coding Environment
- VB.NET Basic Features
- Object Oriented Programming
- Collections

# Introduction

# Main steps in simulation studies

4



# Introduction

5

- Ancestor of Visual Basic .NET is BASIC (Beginner's All-purpose Symbolic Instruction Code) programming language returning back to 1964
- In 1991, Microsoft added visual components to BASIC and created Visual Basic
- After the development of .NET, VB was added with more set of controls and components and thus evolved a new language VB .NET



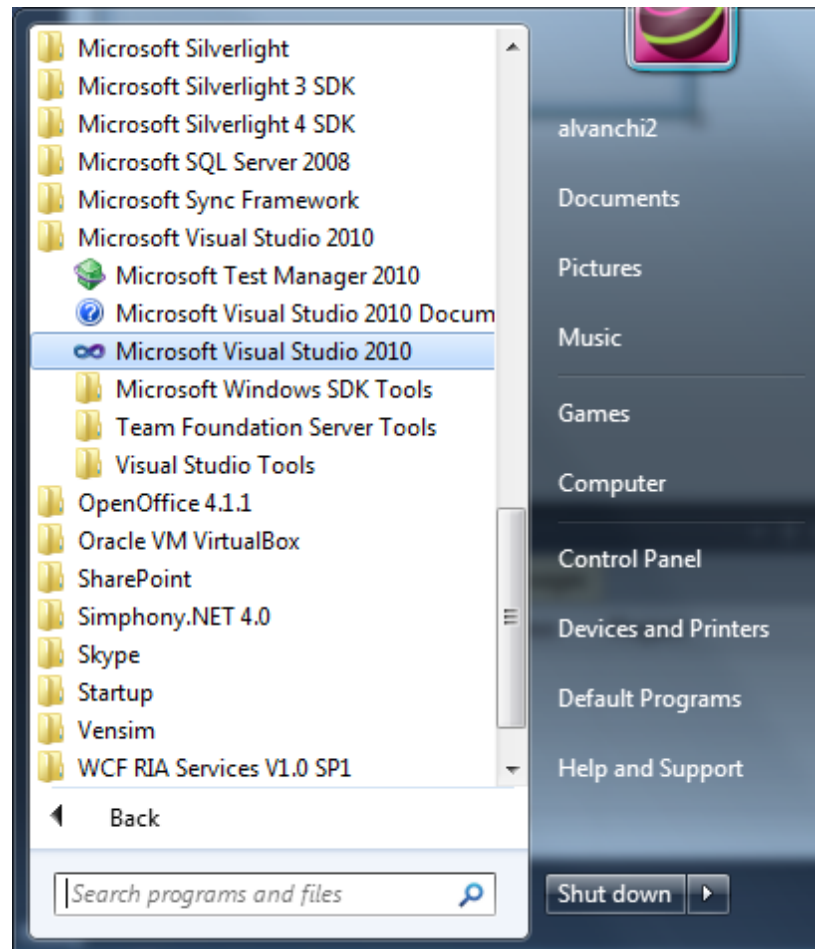
# Introduction

6

- ❑ Visual feature of the VB helps faster development of programs
- ❑ It has rich set of controls
- ❑ It is object orientated programming (OOP) language which enhances modularity, readability and maintainability

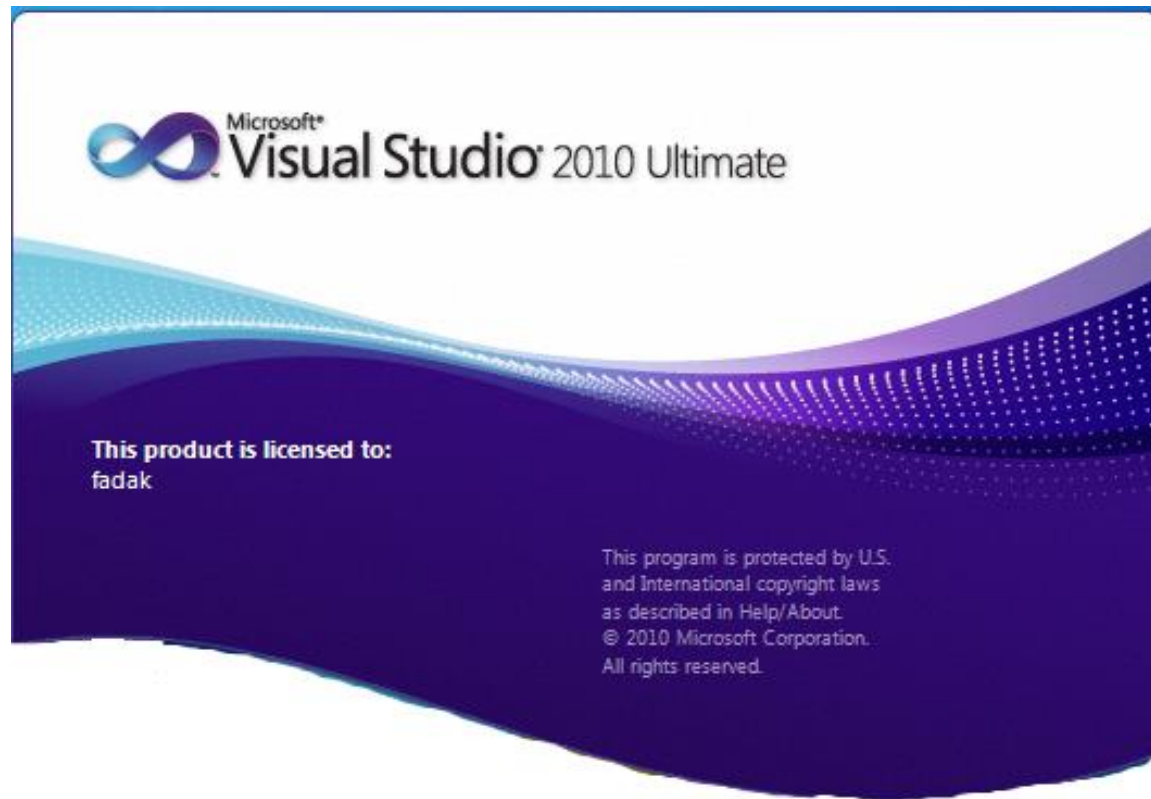
# Programming working environment

7



# Programming working environment

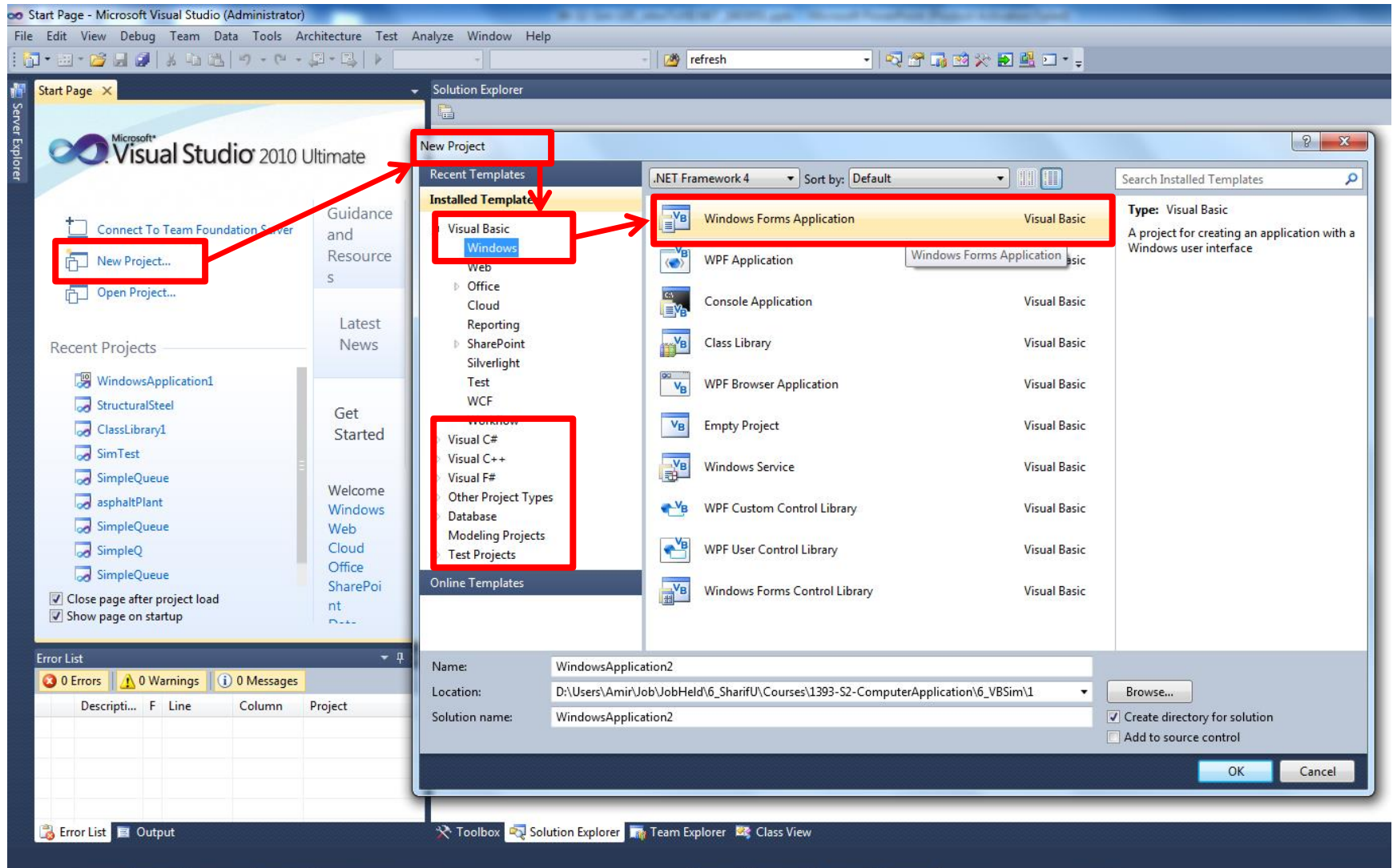
8





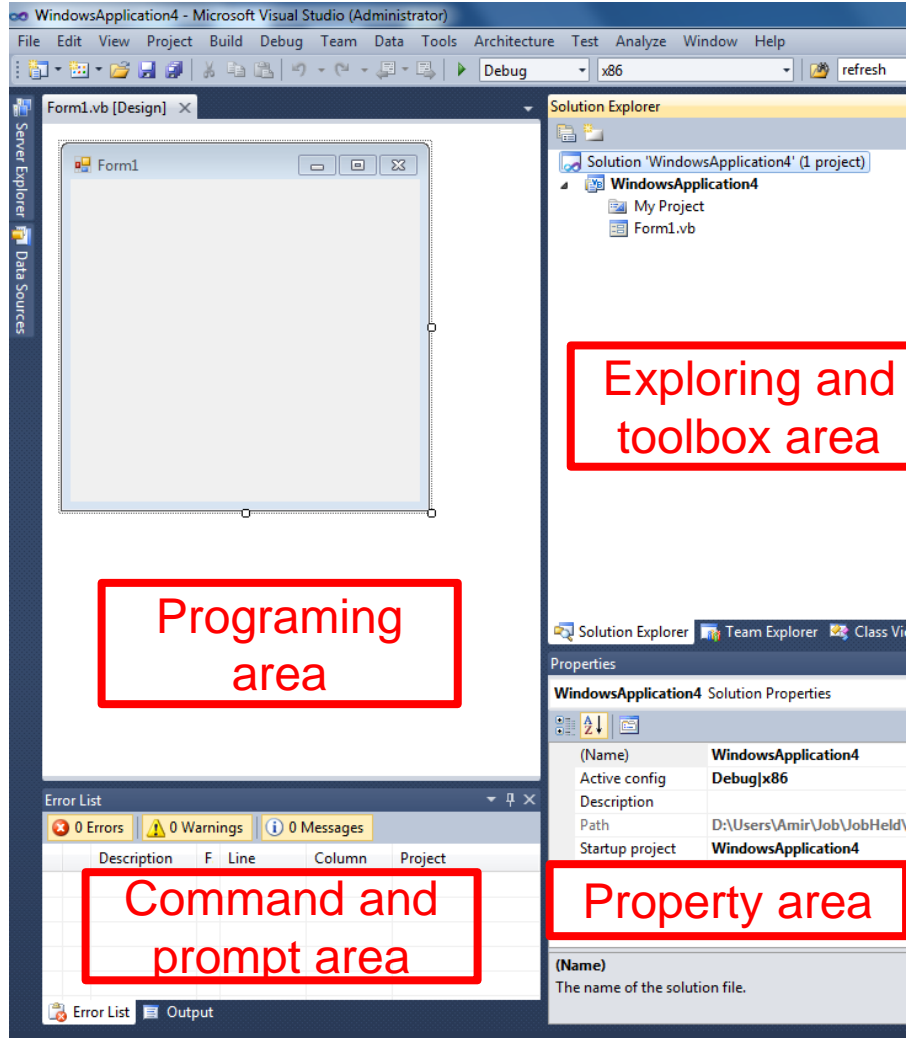
# Programming working environment

9



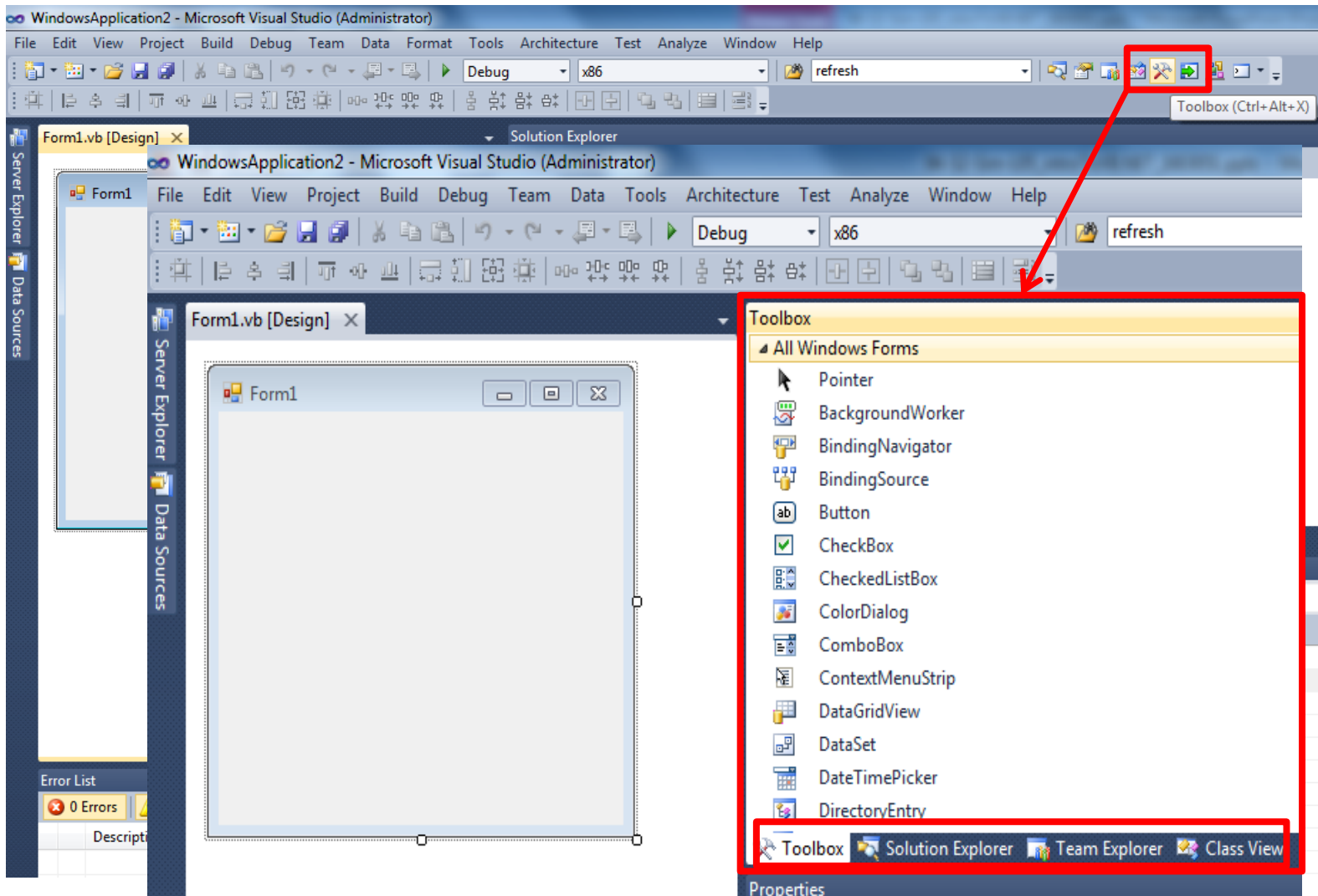
# Programming working environment

10



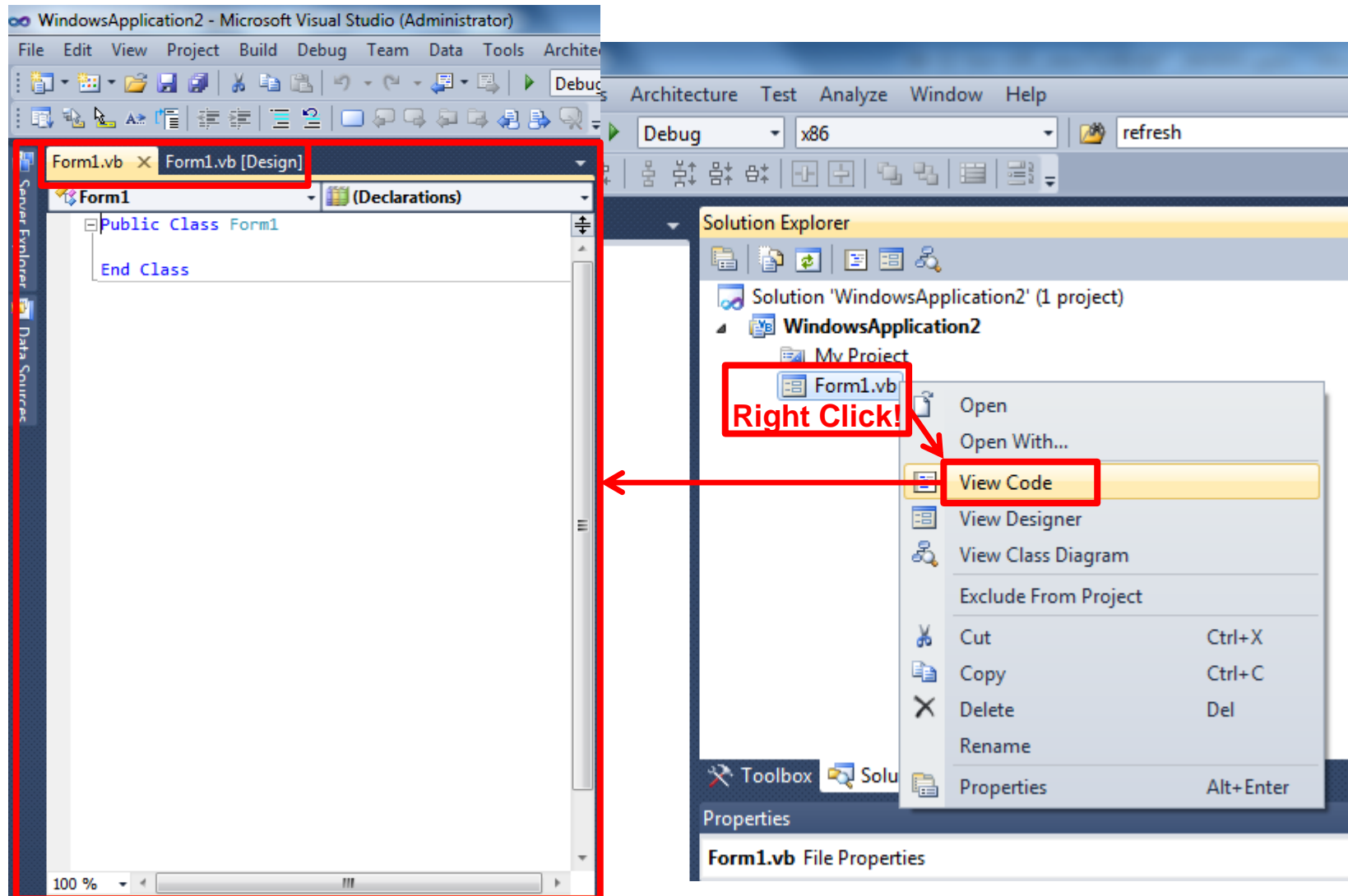
# Programming working environment

11



# Coding environment

12



# Steps in Creating a Visual Basic Program

13

1. Create the interface; that is, generate, position, and size the controls.
2. Set control properties; that is, configure the appearance of the controls.
3. Write programming codes that are executed when different events happen in the program.

# VB.NET Controls Basics

# Controls

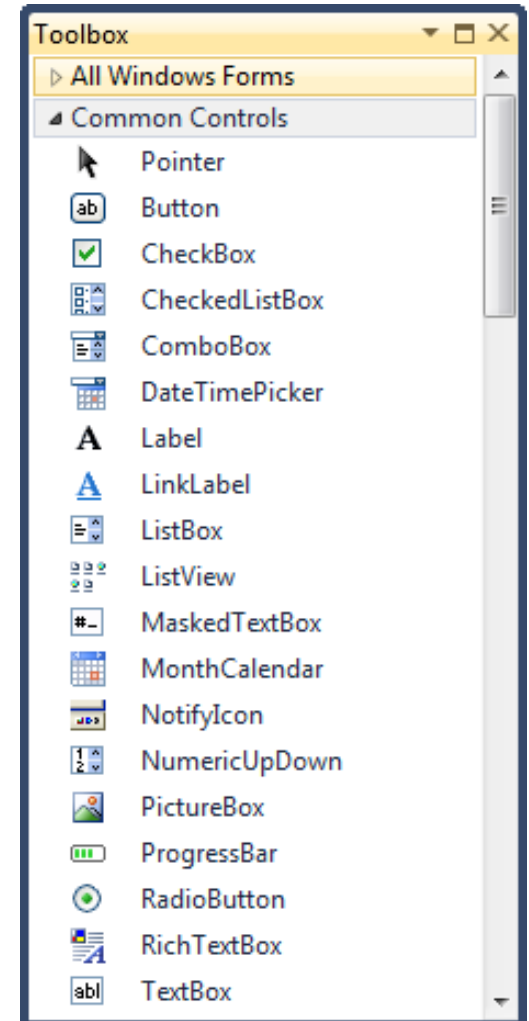
15

- Visual basic controls are set of elements (or objects) used for creating user interface in VB.NET programs
- Form is the first control we need to create in VB.NET.
- Other (predefined) controls can be selected and placed on Form(s) by using the Toolbox.

# Control Toolbox

16

- Form is the main control used in VB programs and embeds all other controls!





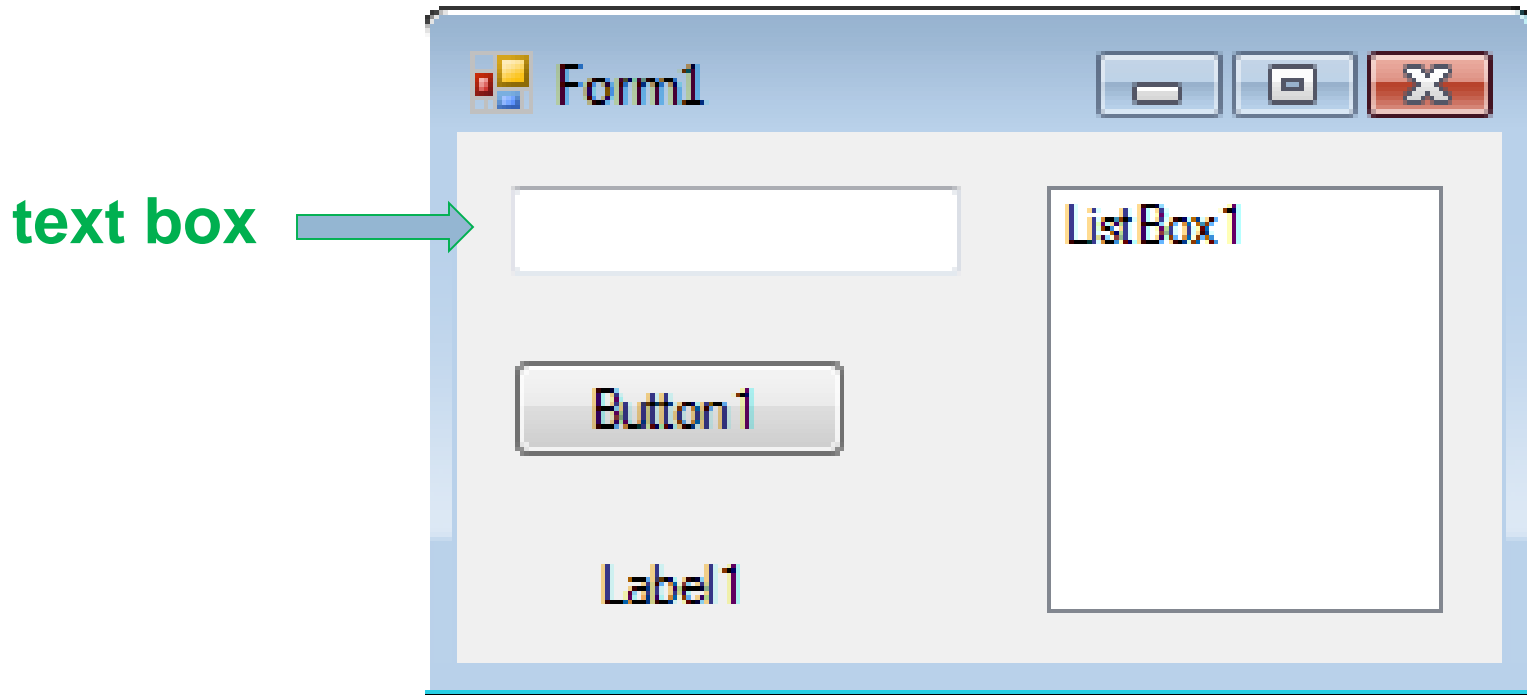
# 4 Ways to Place a Control from the Toolbox on the Form Designer

17

- Double-click
- Drag and Drop
- Click, Point, and Click
- Click, Point, and Drag

# 4 commonly used controls

18



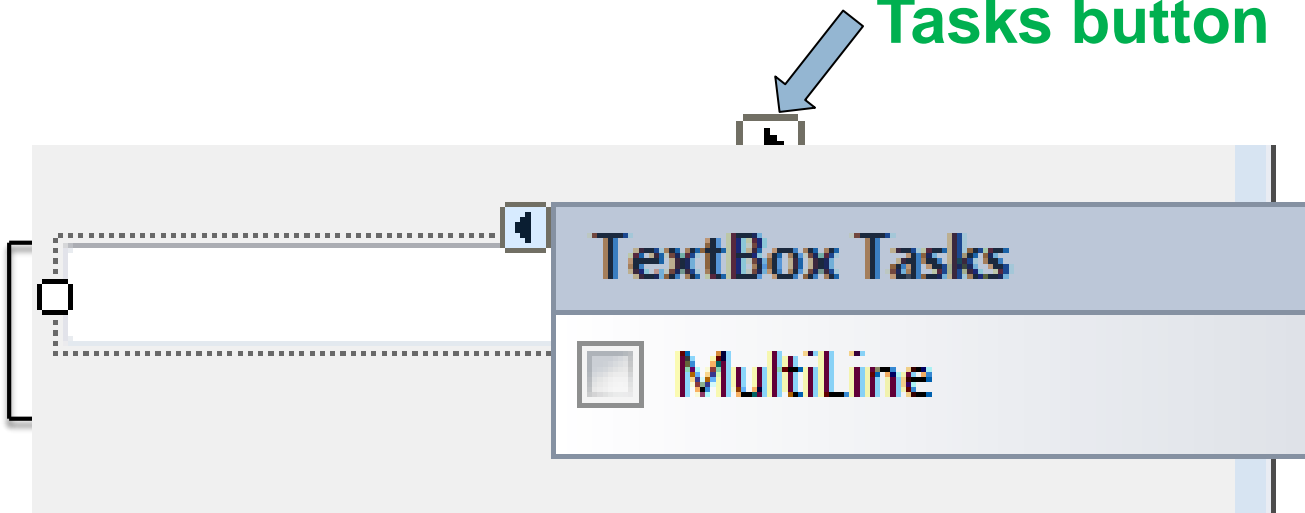
To select a control, click on it. Sizing handles will appear when a control is selected.

# Text Box Control

19

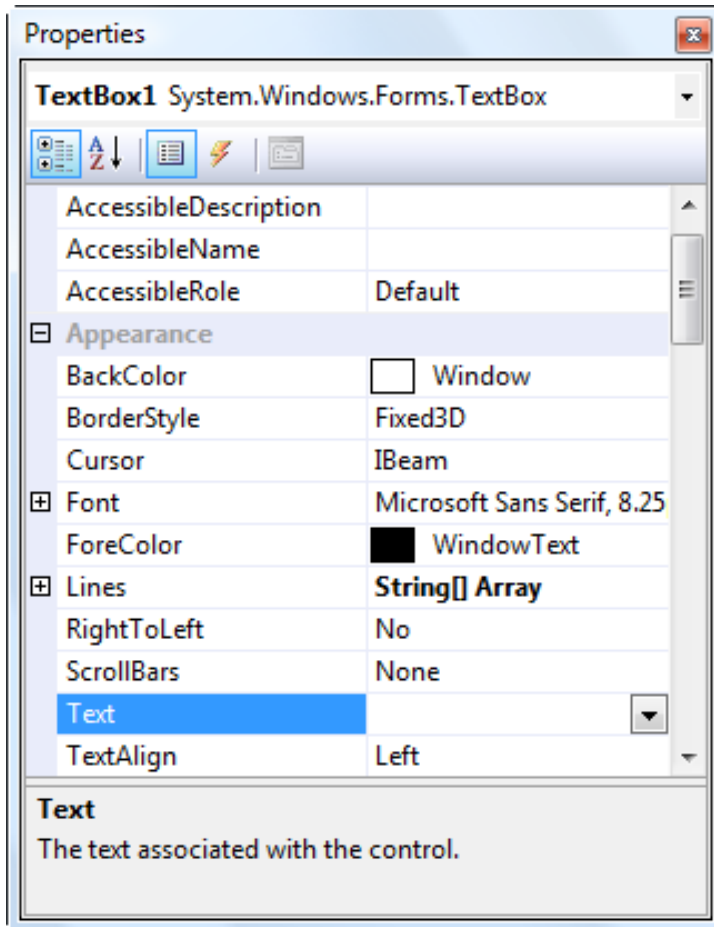
- ❑ Used for input and output
- ❑ When used for output, ReadOnly property is set to True

Tasks button

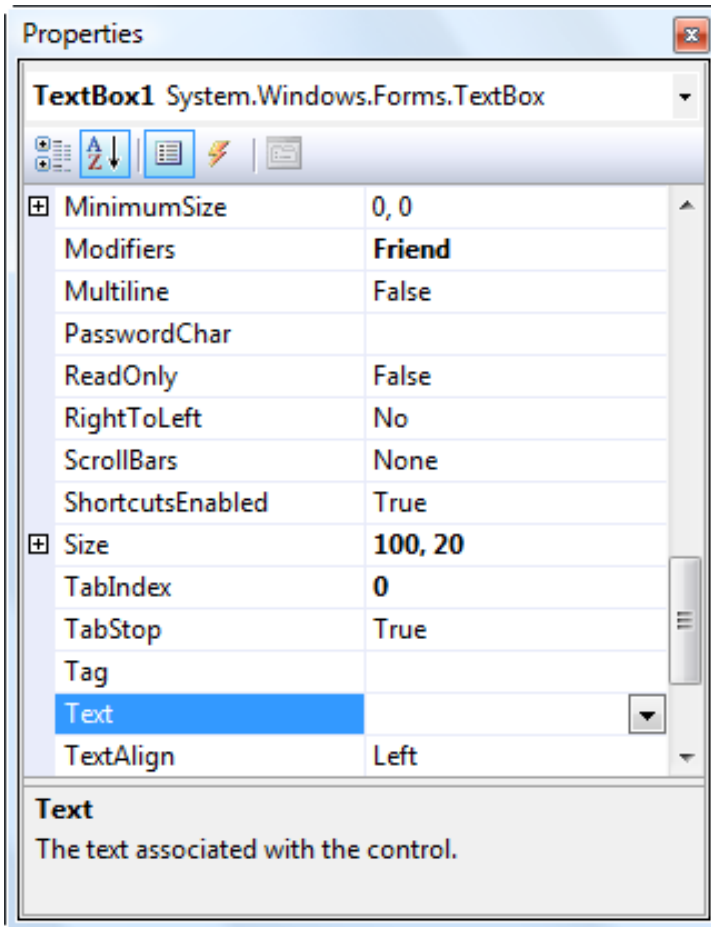


# Properties Window

20



categorized view



alphabetical view

Press F4 to display the Properties window for the selected control.

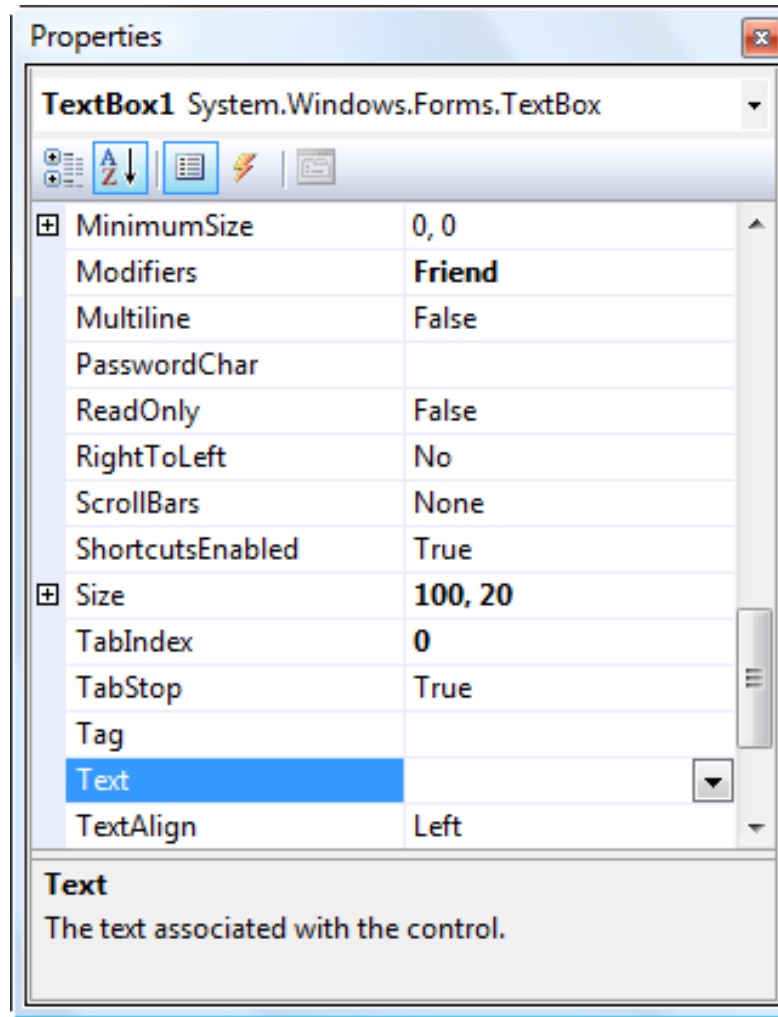
# Properties Window (continued)

21

selected  
control

properties

Description  
pane



settings

# Some Often Used Properties

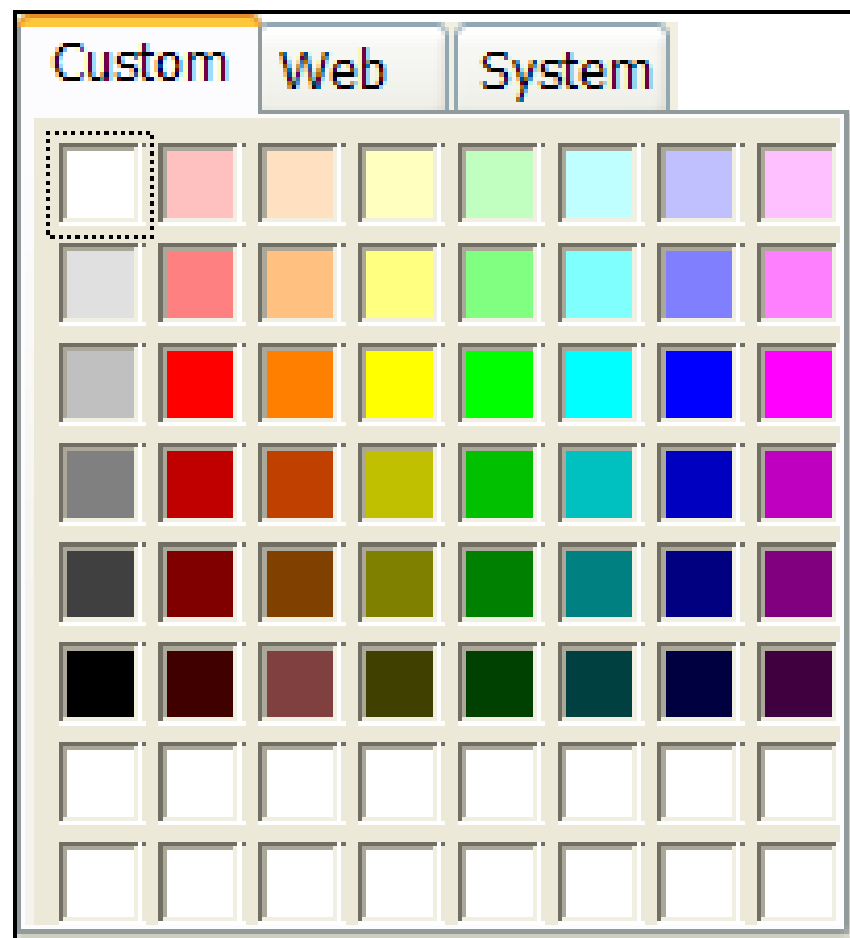
22

- Text
- Autosize
- Font.Name
- Font.Size
- ForeColor
- BackColor
- ReadOnly

# Setting the ForeColor Property

23

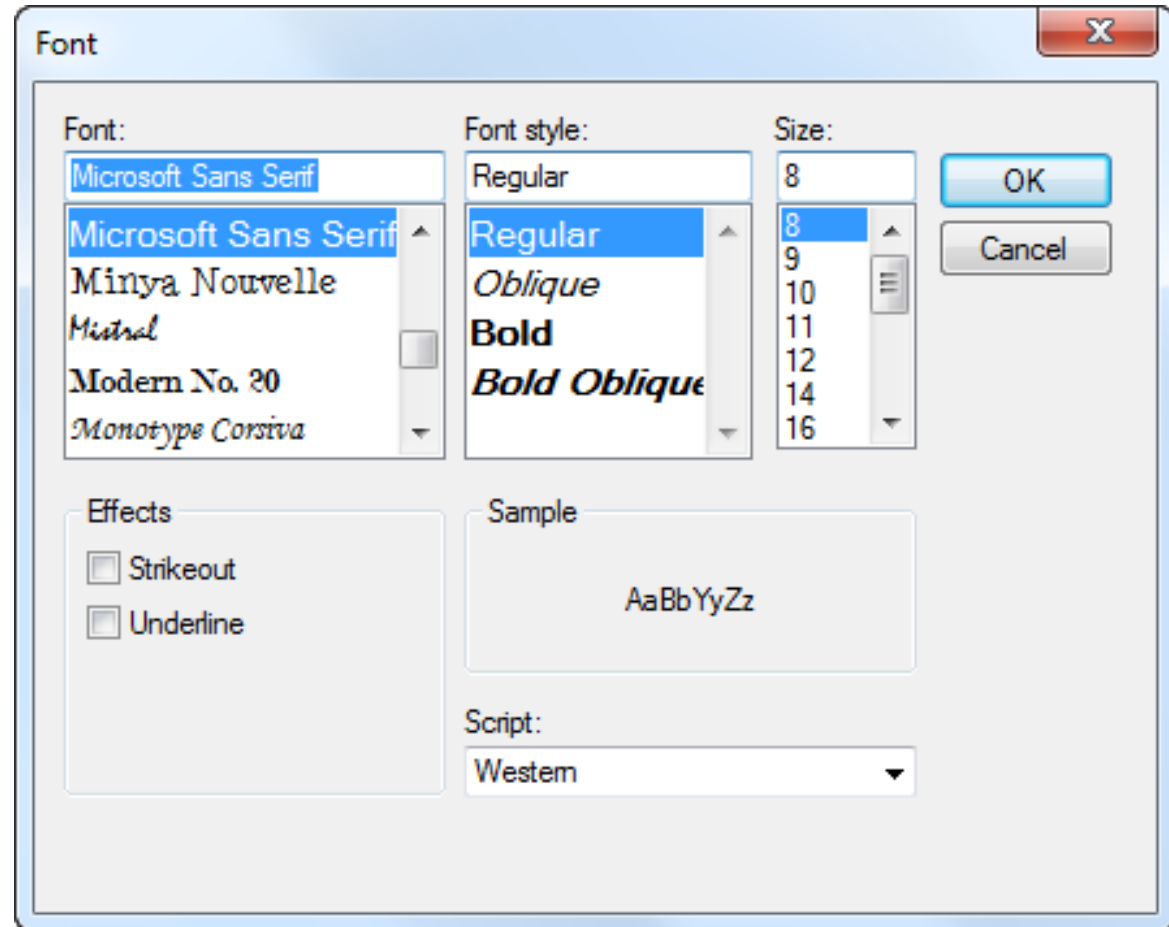
1. Click on ForeColor.
2. Click on button at right of settings box.
3. Click on Custom tab to obtain display shown.
4. Click on a color.



# Font Property

24

1. Click on Font in left column.
2. Click on ellipsis (...) at the right of the font property to show Font settings.
3. Make selections.



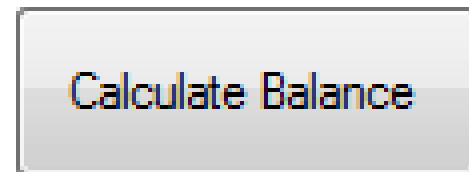
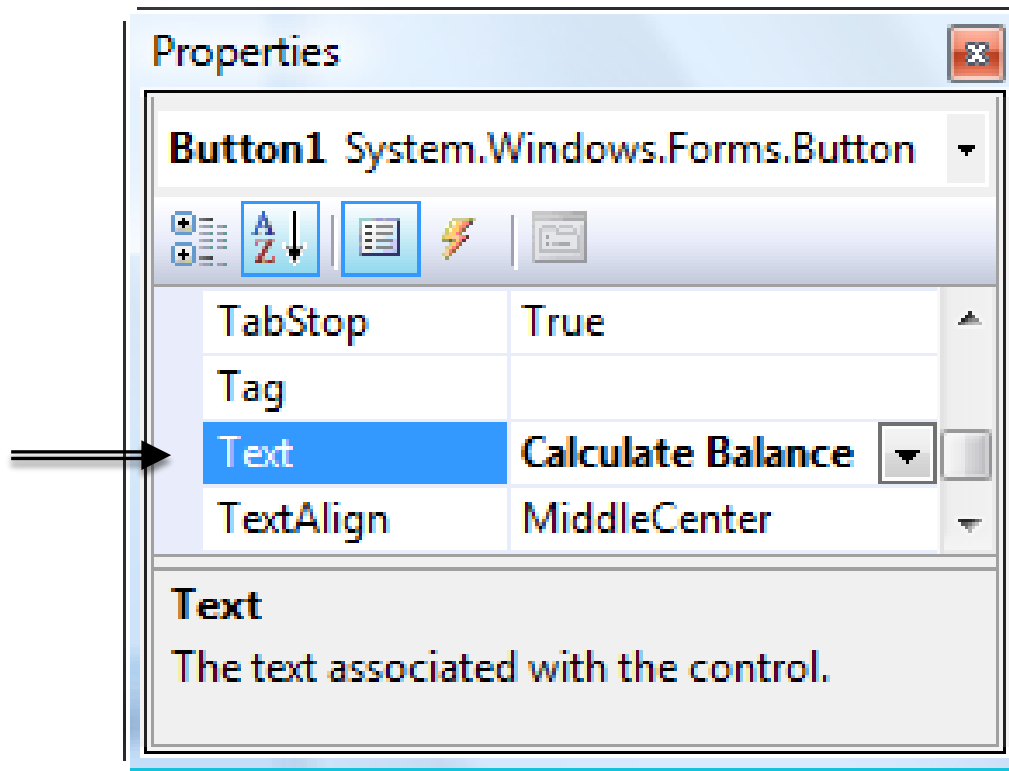


# Button Control

25

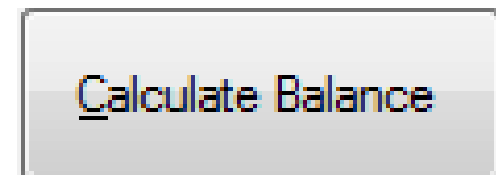
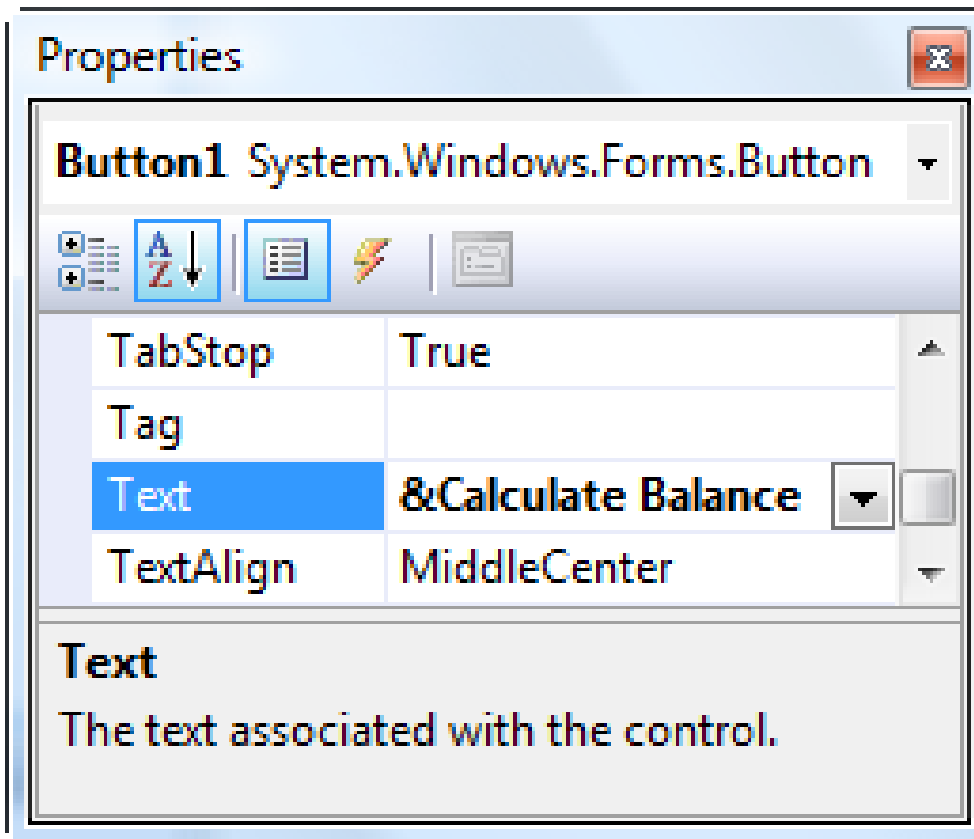
- Used to execute a defined procedure.
- The caption on the button should represent the function of executed button and its effect.

**Text  
property**



# Add an Access Key

- Write “&” before a character to make the control accessible by pushing this character!



# Label Control

27

- ❑ Used to identify the contents of a text box.
- ❑ Text property specifies caption.
- ❑ By default, label automatically resizes to accommodate caption on one line.
- ❑ When the AutoSize property is set to False, label can be resized manually. AutoSize is used primarily to obtain a multi-rowed label.

# List Box Control

28

- It is used to display several pieces of output and/ or acting based on items selected on the list.

# The Name Property

29

- Used by the programmer to refer to a control in code
- Set the Name property near top of the Properties window
- Use appropriate 3-character naming prefix
- Use descriptive names each started by a capital letter

# Control Name Prefixes

30

Control	Prefix	Example
button	btn	btnCompute
label	lbl	lblAddress
text box	txt	txtAddress
list box	lst	lstOutput

# Renaming the Form

31

- Initial name is Form1
- The Solution Explorer window lists a file named Form1.vb.
- To rename the form, change the name of this file to *frmName.vb*
- *frmName* begins with the prefix of *frm*.

# Fonts

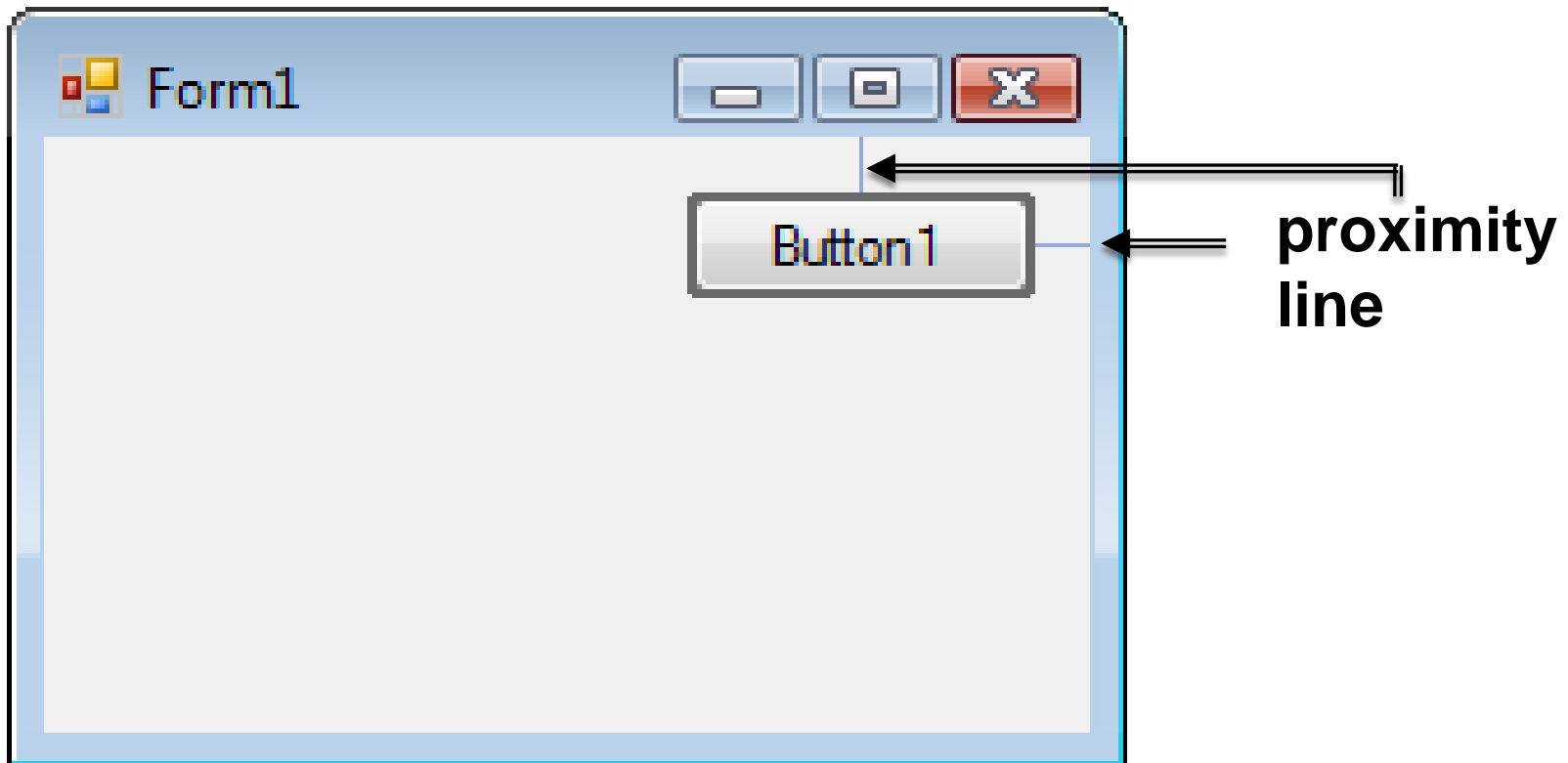
32

- Proportional width fonts, such as Microsoft Sans Serif, use less space for "I" than for "W"
- Fixed-width fonts take up the same amount of space for each character – like Courier New
- Fixed-width fonts are used for tables.



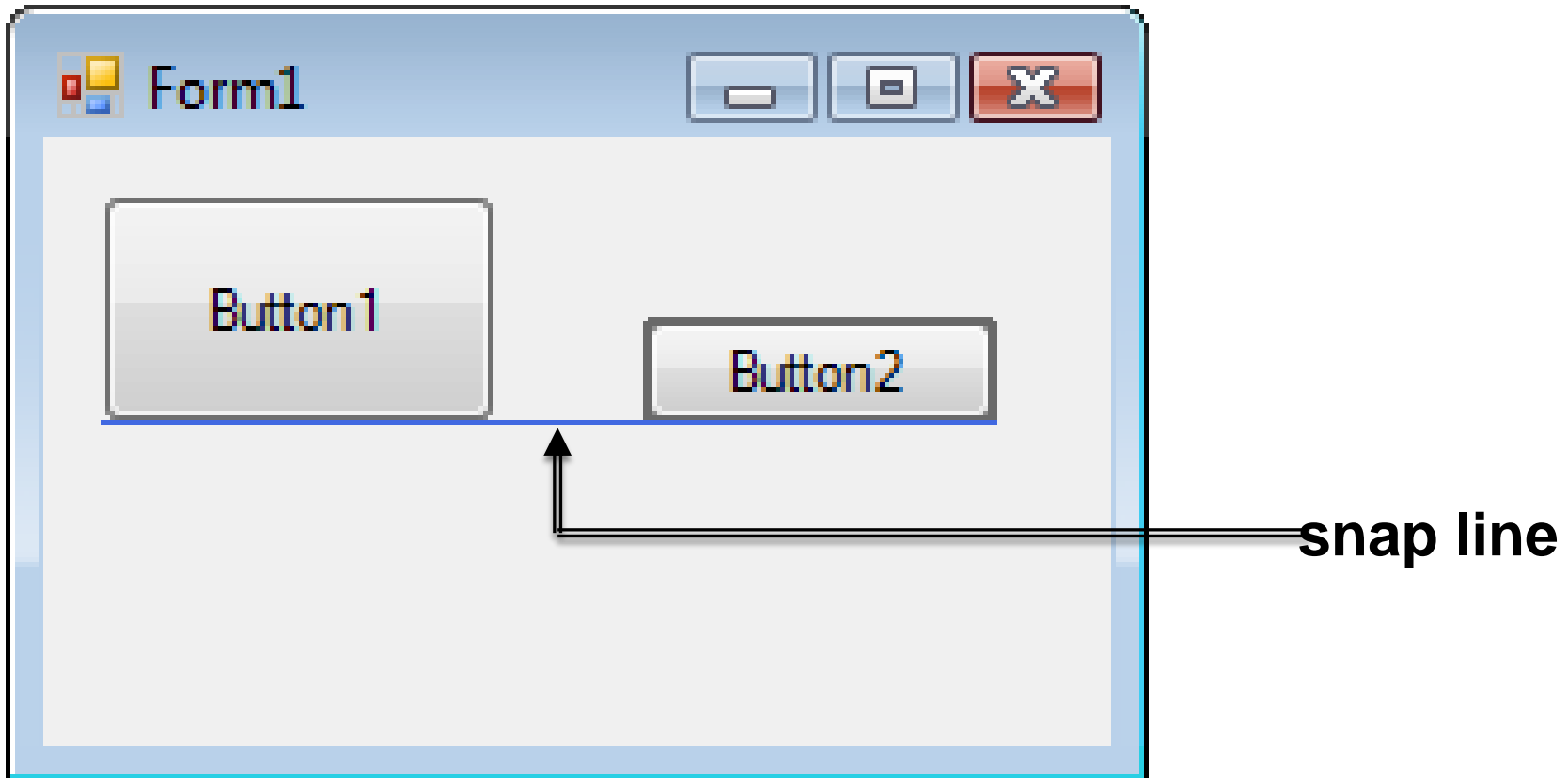
# Positioning Controls

33



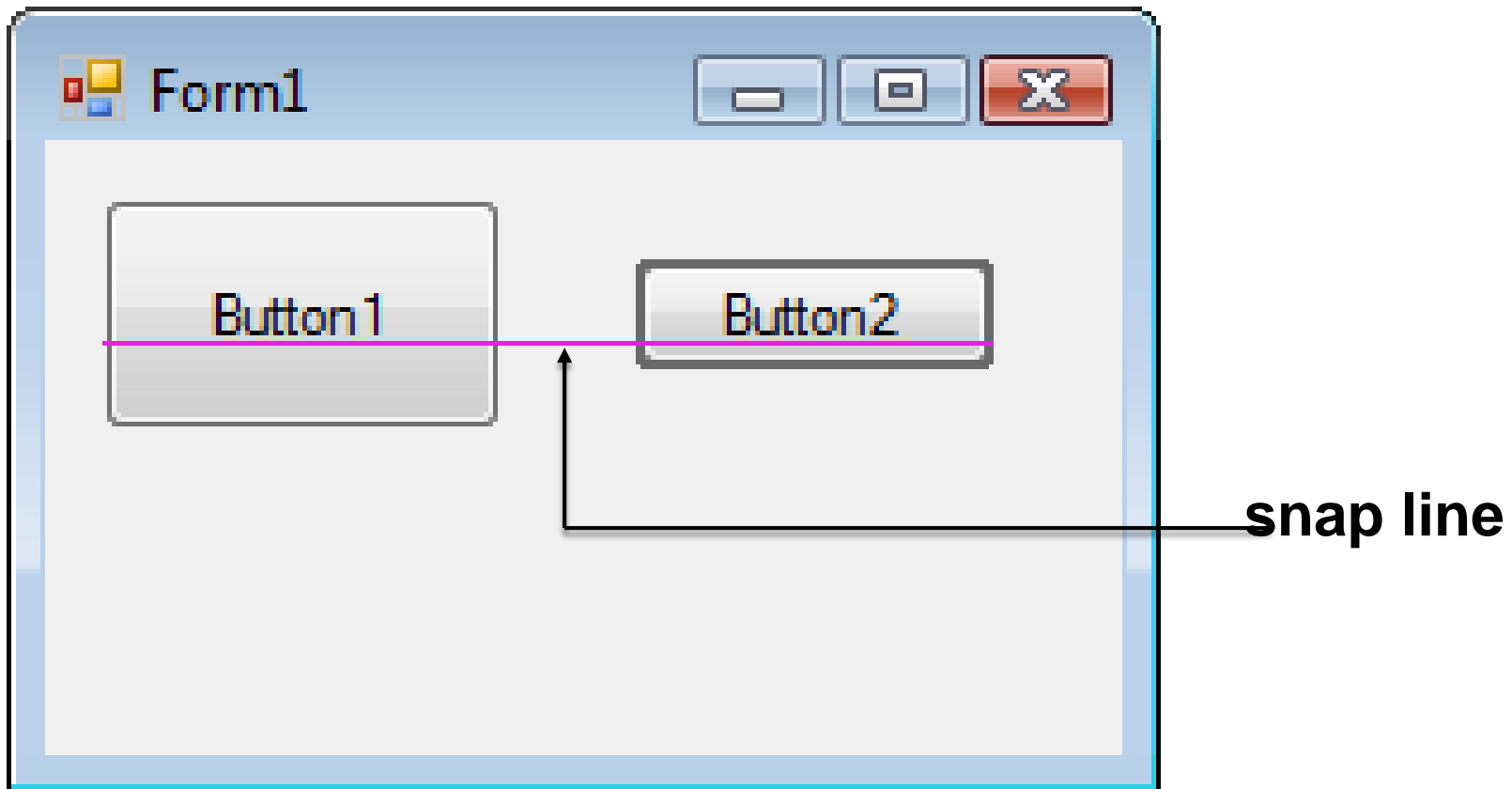
# Aligning Bottoms of Controls

34



# Aligning Middles of Controls

35

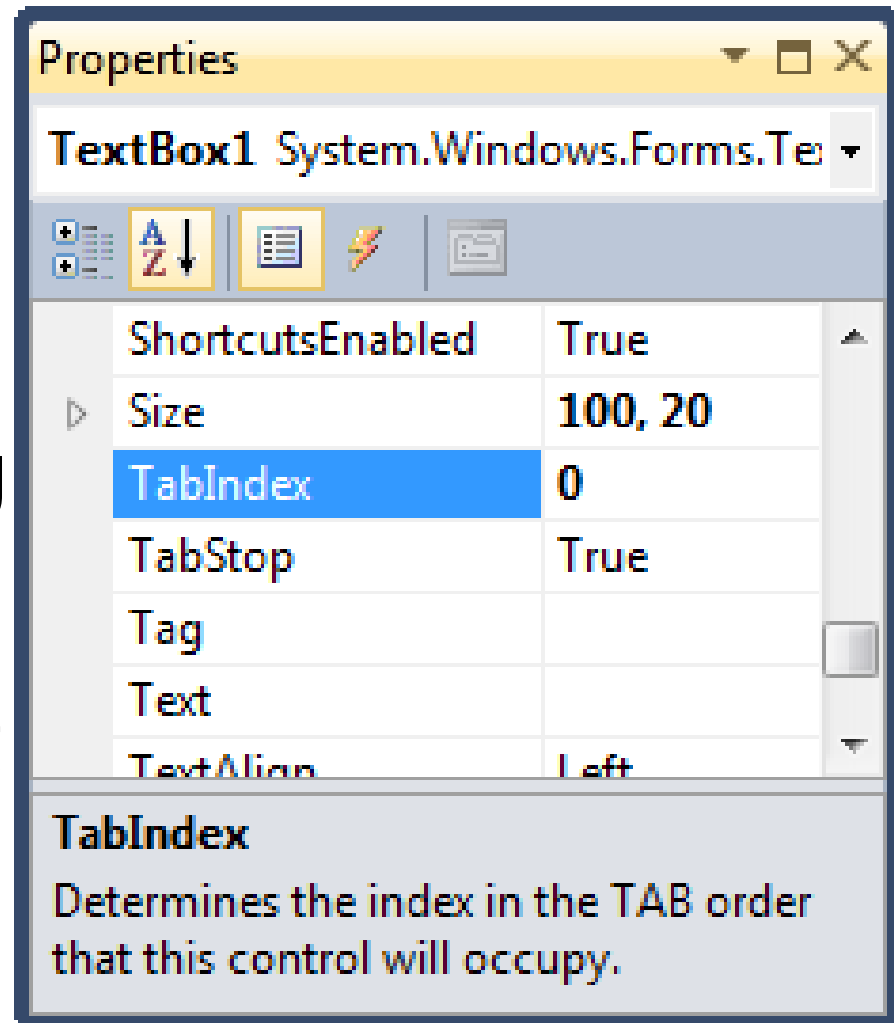


# Tab Order

36

The tab indices determine the order in which controls receive the focus during tabbing

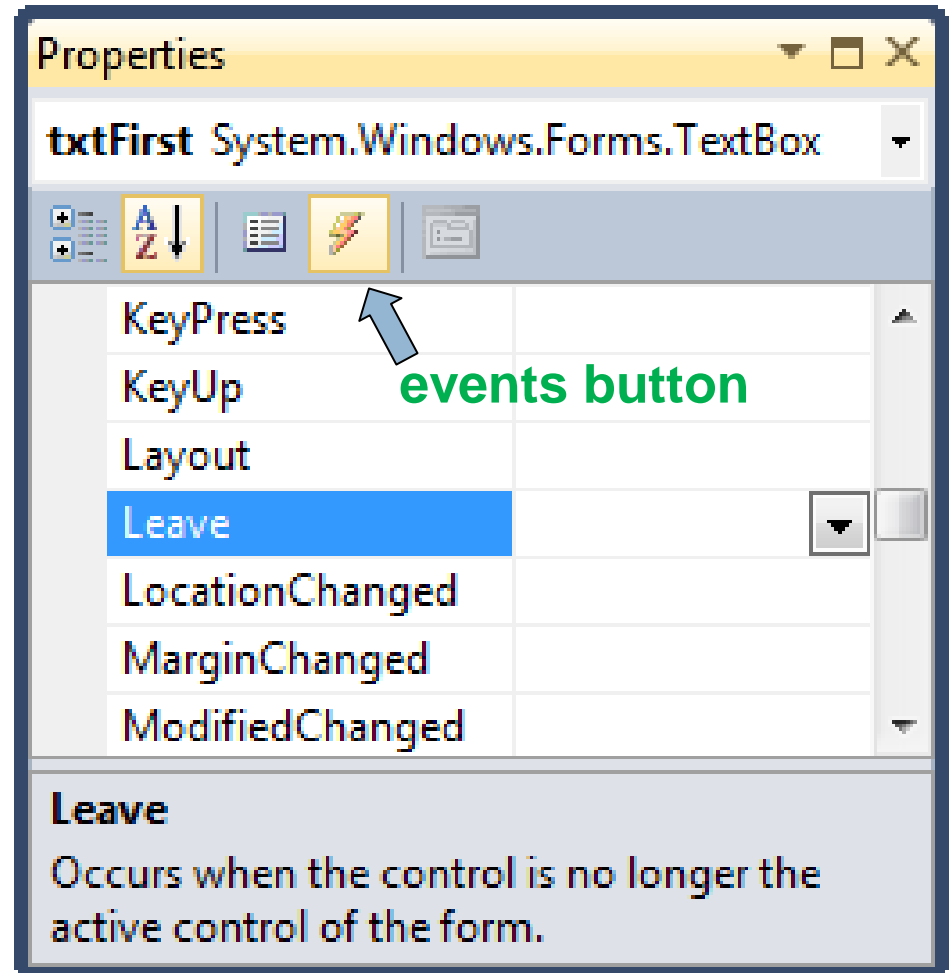
The control whose TabIndex property is set to 0 has the focus when the program begins.



# Control Events

37

- Select the control
- Click on the Events button (⚡) in the Properties window



# Control Events

38

- An **event** is a signal that informs an application that something important has occurred. Events are raised by start or finish of actions, such as the user clicking on a button
- Usually, nothing happens in a Visual Basic program until the user does something and raises an event.
- What happens is determined by statements inside the event procedure.

# Control Events-Examples

39

- ❑ Click: It is triggered by clicking a control
- ❑ DoubleClick: It is triggered by double-clicking on a control
- ❑ KeyPress: It is triggered when any key is pressed while cursor is on the control
- ❑ Load (for a Form control): It is triggered when a form is loaded when the program is run
- ❑ Enter (Focus): It is triggered when a control gets focus (either by clicking on a control or through tab cursor change or ...) and the cursor appears in the control.
- ❑ Leave (Focus): It is triggered when a control loses the focus.
- ❑ TextChanged (e.g., for text box): It is triggered after text property of a textbox control is changed.

# Other controls

40

- Form is a container for other controls
- We can place the following in a form
  - ▣ Label
  - ▣ Text Box
  - ▣ Check Box
  - ▣ Radio Button
  - ▣ Button
  - ▣ Date Picker
  - ▣ And more...



# VB.NET Coding Environment

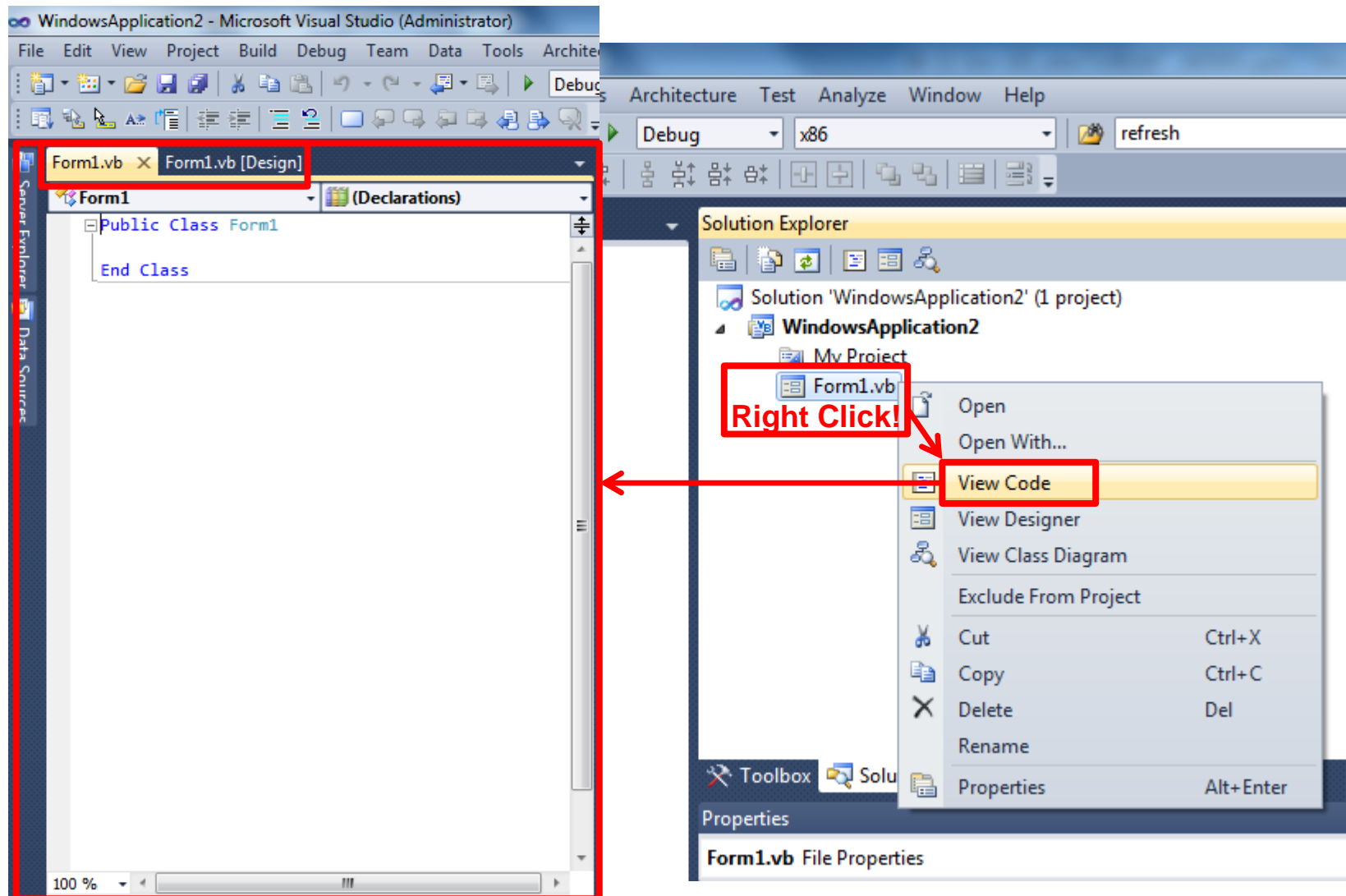
# Main Coding Environment

42

- In parallel to the form interface its coding environment is built
- Coding environment is accessible by right clicking on the form icon in the solution explorer and selecting “View Code”
- Following shape presents how you can access the main coding environment:


# Default Coding Environment

43



# Access an Event's Coding Area

44

- Basically all codes written in a Visual Basic program are stored under and called through different control ***Events***
- To access a control event:
  - ▣ Double-click on a control to access its default Event's coding environmentor
  - ▣ Select a control, click on the Events button () in the Properties window, and double-click on an event

# Access an Event's Coding Area

45

- Following codes are created through accessing different Events coding area:

```
Public Class Form1
```

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e  
        As System.EventArgs) Handles Button1.Click
```

```
    End Sub
```

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

```
    End Sub
```

```
    Private Sub TextBox1_Enter(ByVal sender As System.Object, ByVal  
        e As System.EventArgs) Handles TextBox1.Enter
```

```
    End Sub
```

```
End Class
```

# Access Control Properties Through Codes

46

## General Form:

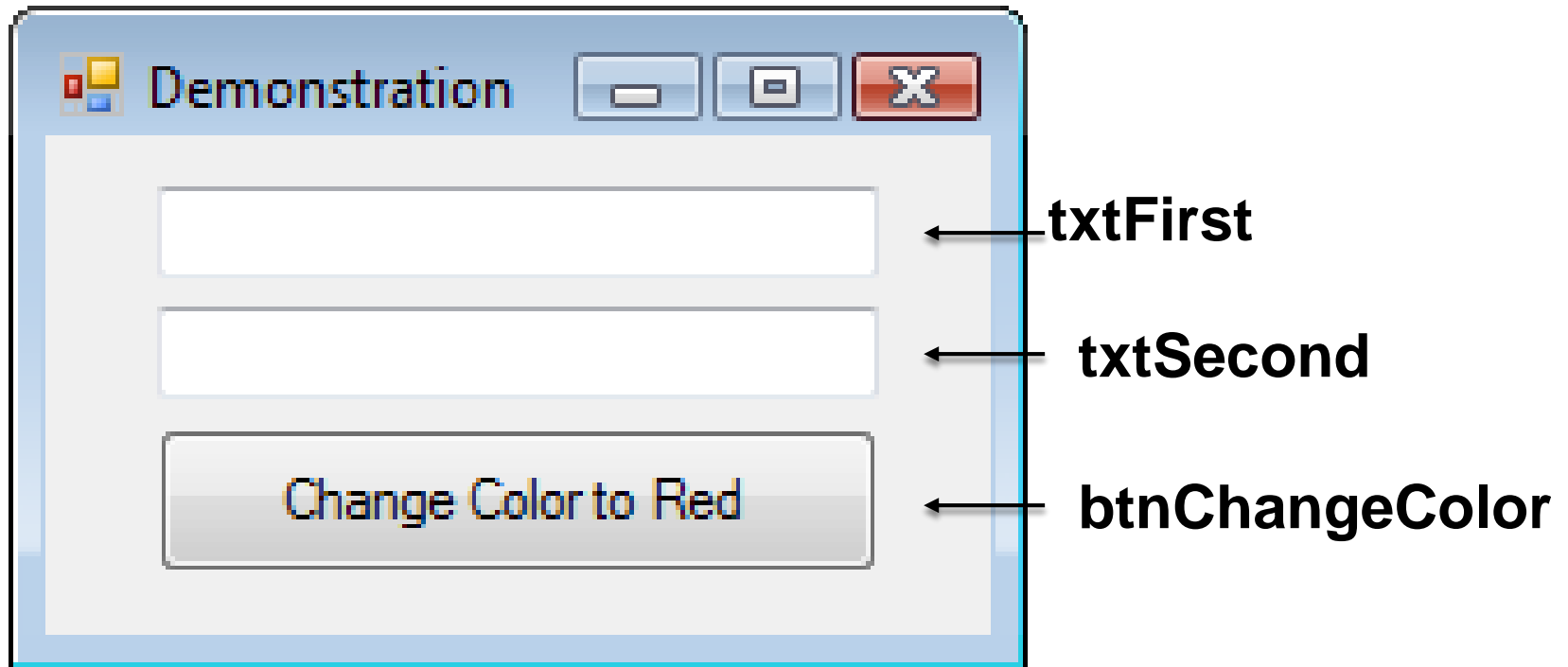
*controlName.property = setting*

## Examples:

- ❑ `textBox.ForeColor = Color.Red`
- ❑ `textBox.Visible = True`
- ❑ `textBox.Text = "Hello World"`

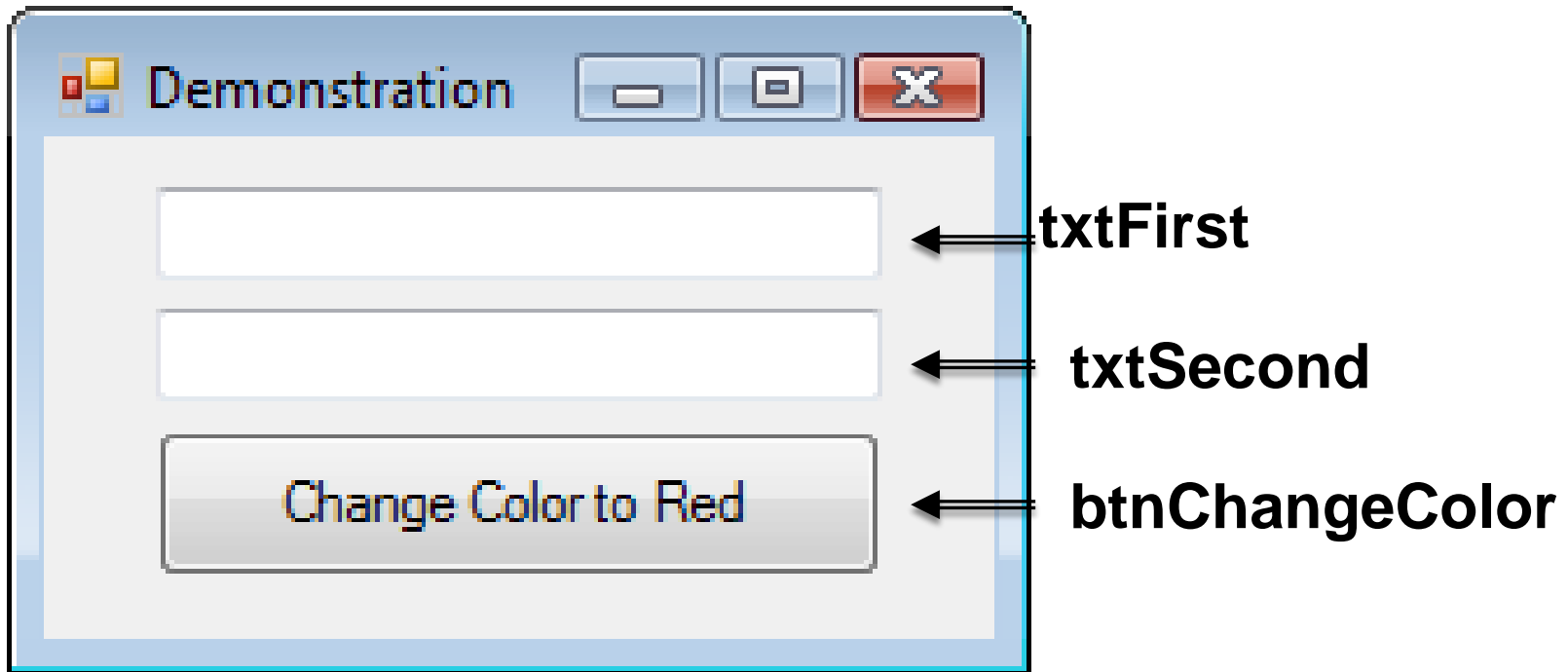
# Sample Form

47



# Sample Form

48



Double-click on txtFirst to create the outline for the Code Editor



# Code for Walkthrough

49

```
Public Class frmDemo
```

```
    Private Sub txtFirst_TextChanged(...) Handles txtFirst.TextChanged
```

```
        txtFirst.ForeColor = Color.Blue
```

```
    End Sub
```

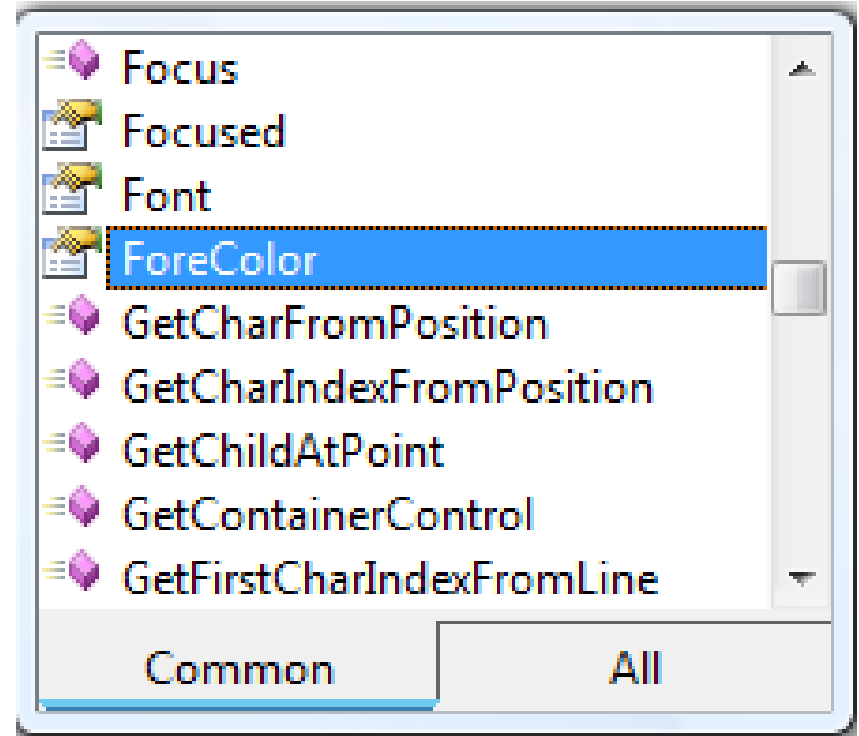
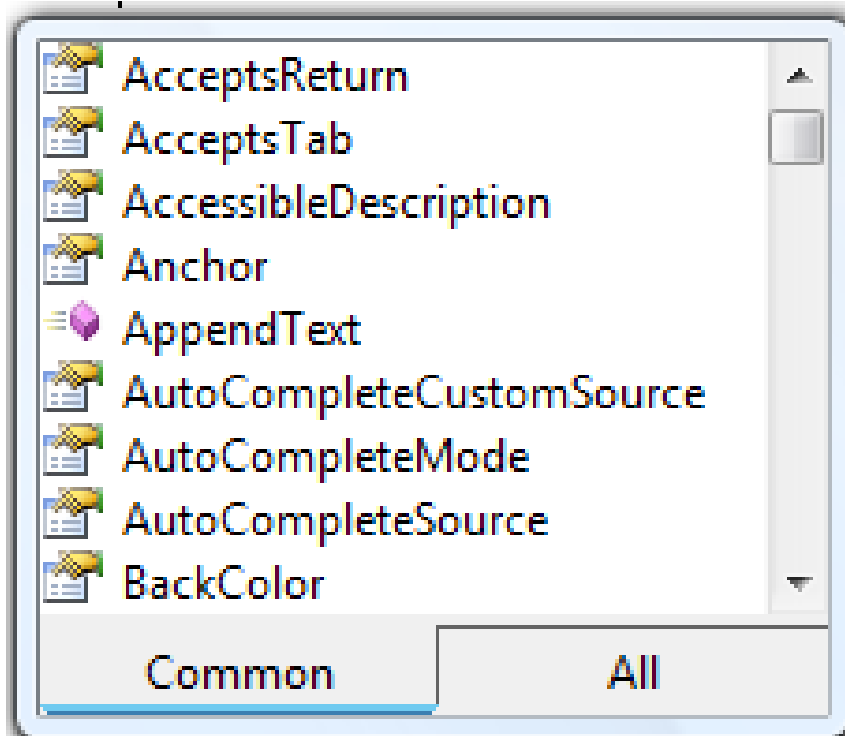
```
End Class
```

# IntelliSense

50

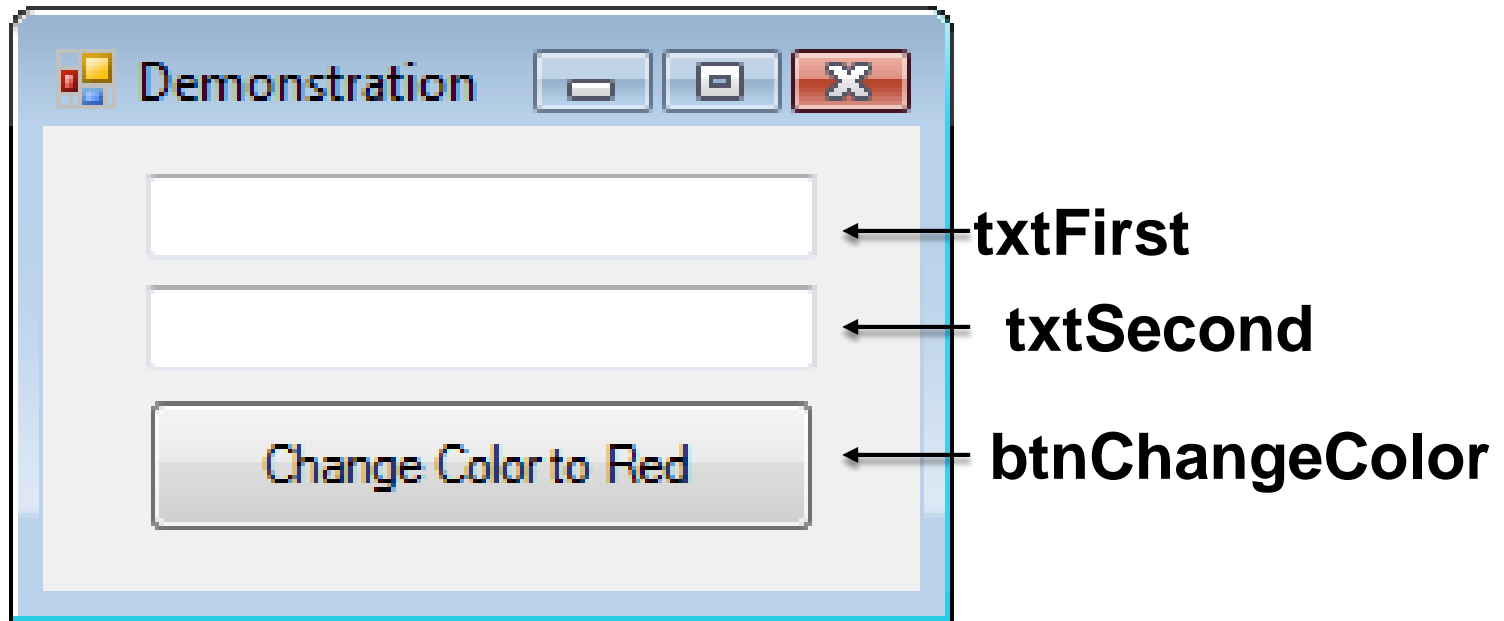
Automatically pops up to help the programmer.

`txtFirst.`



# Sample Form

51



Double-click on `btnChangeColor` to return to Code Editor and add the outline of an event procedure

# Code for Walkthrough

52

```
Public Class frmDemo
```

```
    Private Sub txtFirst_TextChanged (...) Handles txtFirst.TextChanged
```

```
        txtFirst.ForeColor = Color.Blue
```

```
    End Sub
```

```
    Private Sub btnChangeColor_Click (...) Handles btnChangeColor.Click
```




```
        txtFirst.ForeColor = Color.Red
```

```
    End Sub
```

```
End Class
```

# Sample Form

53

- Run the program by clicking on run button () on the standard toolbar, or select Run or Start Debugging option on the Debug Menu or Push F5 to go to the run mode and play with the form, and see how form reacts!
- Push the Close button () on the up-right side of the form or push stop button () on the Toolbar to return to the form design view.

# Home Assignment 5

54



*Change the sample form discussed in the class in a way that:*

- *Changes color of textFirst to red while typing*
  - *Current button changes both texts to pink*
  - *Add a new button which changes color of textSecond to brown*
  - *Add a proper code to the form by which exact text is written on textFirst after a text is written on textSecond!*
- (Due two days)***

# VB.NET Basic Features

# Data types and variables in VB .NET

56

## □ Data types and variables

- ▣ Integer, string, single, double, boolean, char;

Example:

- Dim i As Integer
- Dim s As String
- Dim j, k As Single
- Dim d As Double
- Dim reason, result, test As Boolean
- Dim c As Char



# Operators in VB .NET

57

- Operators
  - ▣ Arithmetic (+, -, \*, /, \, Mod)
  - ▣ Logical (Or, And)
  - ▣ Relational (=, <>, <, <=, >, >=)
  - ▣ Examples:
    - $d = i / j$
    - $k = l \bmod j$
    - `test = reason And result`

# If Else

58

If (Condition1) Then

Statements executed if Condition1 is true

Else If (Condition2) Then

Statements executed if Condition1 is false and  
Condition2 is true

Else

Statements executed if Condition1 and Condition2 are  
false

EndIf

# If Else

59

## Example:

```
Dim condition1, condition2, elseCondition as Boolean
```

```
If number1 = number2 Then
```

```
    condition1 = True
```

```
Else If number1 = number3 Then
```

```
    condition2 = True
```

```
Else
```

```
    elseCondition = True
```

```
End If
```

# Select Case

60

Select Case var

Case 1

stmt1 // executed if var = 1

Case 2

stmt2 // executed if var = 2

Case Else

stmt3 // executed if var is other than 1 and 2

End Select

# For Loop

61

```
For <<var>> = start To end Step <<val>>  
    Statements  
Next
```

Example:

```
Public Class frmDemo  
    Private Sub btnRunForNext_Click (...) Handles btnRunForNext.Click  
        Dim I As Integer  
        For I = 1 To 10 Step 2  
            MsgBox(I)  
        Next  
    End Sub  
End Class
```

1,3,5,7,9 are popped up in the message box consecutively!

# Do While Loop

62

1. Do While(a<>0)

    MsgBox(a)

    a = a – 1

Loop

2. Do

    MsgBox(a)

    a = a – 1

Loop While(a<>0)

# Do Until Loop

63

1. Do Until(a=0)

    MsgBox(a)

    a = a – 1

Loop

2. Do

    MsgBox(a)

    a = a – 1

Loop Until(a=0)

# Subroutines and Functions

64

- Subroutines or Procedures
  - ▣ Does not return any value
  - ▣ Can have zero or more parameters
  - ▣ Are meant to follow set of steps
- Functions
  - ▣ Always Returns some value
  - ▣ Can have zero or more parameters
  - ▣ In addition to returning value, similar to subroutines, can also follow set of steps
- In VB.NET there are varieties of predefined procedures and functions which are used through programming, however VB user can also define Procedures and Functions of their owns



# Subroutines and Functions

65

## □ Subroutines or Procedures - Declaration Syntax:

```
[accessModifiers] Sub subName[(parameter1, parameter2, ...)]
```

```
    ' Statements of the Sub procedure.
```

```
End Sub
```

- ▣ Represents optional terms
- ▣ **accessModifiers**: Specify access level, usually by using *Private* (accessible inside this coding environment) or *public* (also accessible from outside of the current coding environment) keywords which are mainly used in object oriented programming

- ▣ **Parameter**: [ByVal | ByRef] *parametername As datatype*

Use **ByVal** if you do not want to update parameter value based on updates made inside the procedure

Use **ByRef** if you want value of parameter gets updated based on updates made inside the procedure

# Subroutines and Functions

66

## □ Subroutines or Procedures - Declaration Syntax:

### **Example:**

```
Private Sub initializeSim (ByVal Truck1Shift as integer, ByVal Truck2Shift  
as integer, ByRef simTime as double, ByRef Qlength as integer, ByRef  
Truck1FirstArrival as double, ByRef Truck2FirstArrival as double)
```

```
    simTime = 0
```

```
    Qlength = 0
```

```
    Truck1FirstArrival = Truck1Shift * 8 * 60 + Rnd() * 10
```

```
    Truck2FirstArrival = Truck2Shift * 8 * 60 + Rnd() * 10
```

```
End Sub
```

# Subroutines and Functions

67

- Subroutines or Procedures – Calling Syntax:

*subName( [Value1, Value2, ...])*

## **Example:**

initializeSim (Truck1Shift, Truck2Shift, simTime, QLength,  
Truck1FirstArrival, Truck2FirstArrival)

# Subroutines and Functions

68

## □ Functions - Declaration Syntax:

[accessModifiers] Function *functionName*[(*parameter1*, *parameter2*, ...)] as  
returnValueType

' Statements of the function including setting value for *functionName*

End Sub

- Represents optional terms
- **accessModifiers**: Specify access level, usually by using *Private* (accessible inside this coding environment) or *public* (also accessible from outside of the current coding environment) keywords which are mainly used in object oriented programming
- **Parameter**: [ByVal | ByRef] *parametername* As *datatype*

Use **ByVal** if you do not want to update parameter value based on updates made inside the procedure

Use **ByRef** if you want value of parameter gets updated based on updates made inside the procedure

# Subroutines and Functions

69

## □ Functions - Declaration Syntax:

### **Example:**

Private function loadingTime (ByVal TruckCapacity as double) as double

    loadingTime = TruckCapacity \* (1+ Rnd())

End function

# Subroutines and Functions

70

## □ Functions – Calling Syntax:

*functionName*( [Value1, Value2, ...])

Or

VariableOfTheSameType = *functionName*( [Value1, Value2, ...])

### **Example:**

Dim truckLoadingTime as double

Dim truckCapacity as double = 10

TruckLoadingTime = loadingTime(truckCapacity)

# Arrays

71

- An array is a set of logically related values, such as the name of students in a class or time of events in FEL.
- By using an array, you can refer to these related values by the same name, and use a number that's called an index or subscript to tell them apart. The individual values are called the elements or items of the array.
- Index number begins from index 0 through the highest index value.

# Arrays

72

- Array – declaration:

Dim arrayName(dimension1 HighestValue[,  
otherDimensions HighestValues]) as arrayElementType

## **Example:**

Dim simClassStudents(8) as string

Dim numberOfEntities as integer = 10

Dim numberOfEventTypes as integer = 5

Dim FEL(numberOfEntities, 3) as double

‘Set values of 2<sup>nd</sup> dimension to 1 for event time, 2 for event type and 3 for entity number



# Arrays

73

- Array – call:

```
arrayName(dimension1 HighestValue[,  
otherDimentionsHighestValues]) = Value
```

Or

```
variableOfTheSameType =  
arrayName(dimension1 HighestValue[,  
otherDimentionsHighestValues])
```

# Arrays

74

- Array – call:

## **Example:**

`simClassStudents(1) = "Ali"`

`simClassStudents(1) = "Mohammad"`

`FEL(3 , 1) = loadingTime(truckCapacity) + currentSimTime`

`FEL(3 , 2) = 1` 'represent loading event

`FEL(3 , 3) = 3` 'represents entity number

# Object Oriented Programming (OOP)

# Programming progression

76

## □ Programming has progressed through:

- ▣ machine code

- ▣ assembly language

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
```

```
GCD:          TST      B
              BEQ
              MOV
              MOV      R5, B
              CALL     GCD

SIMPLE:  RETURN
```

- ▣ machine-independent programming languages  
(Fortran, Basic, etc.)

- ▣ procedures & functions

- ▣ objects

# Object Oriented Programming

77

- Concepts
  - ▣ Classes
  - ▣ Objects
  - ▣ Abstraction
  - ▣ Encapsulation
  - ▣ Inheritance

# Classes and Objects

78

- A *class* is a model or prototype or instruction for creating objects
- A class documents structure and specifications of an object
- We can create or *instantiate* multiple separate objects based on one class. In this perspective we can say objects are *instances* of classes
- Relation between a class and an object is similar to a drawing and a building
  - ▣ We build a house based on drawings
  - ▣ A building is a physical representation of a what is specified in drawings
  - ▣ We can use drawing to build several, and separate, buildings

# Classes and Objects

79

- We already have used classes and objects in our programming lesson.

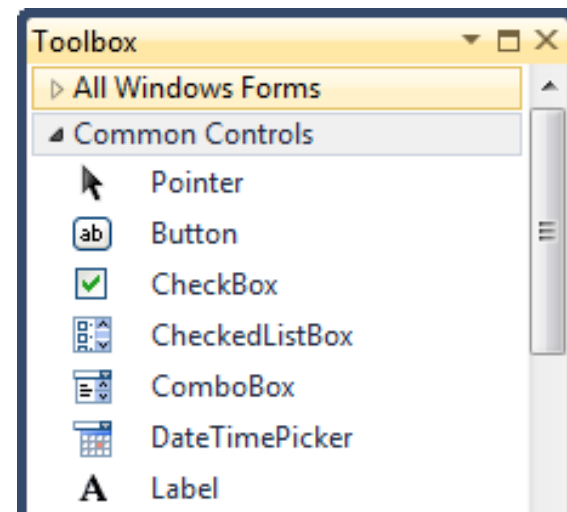
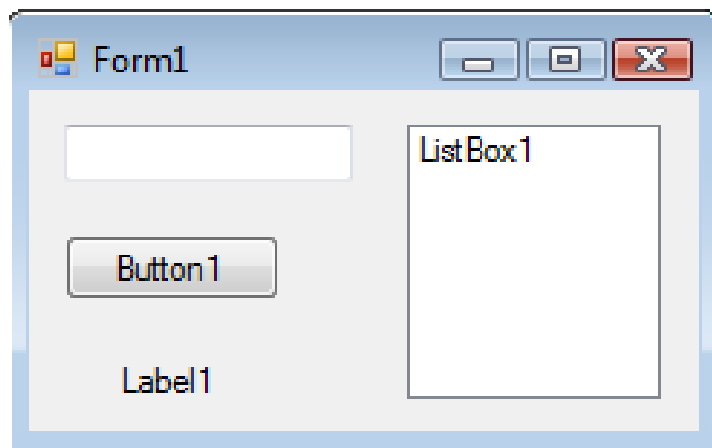


Guess which programming capabilities we discussed are using classes and objects?

# Classes and Objects

80


- Controls listed on the control Toolbox are *classes* and controls placed on forms, which are instances of those classes, are objects!
- For every control there is only one item on the Toolbox
- We can instantiate multiple controls on a form!
- VB.NET users can define classes and use objects of their owns!





# Components of an Object

81

 Based on what we have learned from controls, guess what are the main components of objects?

# Components of an Object

82

- An object consists of three main components:
  - ▣ Attribute or Property: Specifies the information associated with an object. Color of a textbox object, capacity of a truck object, and bucket size of a loader object are examples for attribute.
  - ▣ Method: Are actions performed by an object. Methods come in two forms of functions and procedures. For example loading is a method for a loader object, travelling is a method of a truck object, compacting is method for a roller.

# Components of an Object

83

- An object consists of three main components: (cont'd)
  - ▣ Event: An event is a signal that informs an application that something important has occurred. An event is usually triggered by start or finish of an action and is usually handled through a supporting method underneath. For example gaining focus is an event for a textbox, site arrival is an event for truck object and completion of asphalt loading is an event for the asphalt plant.


# Start Working with Objects

84

- For working with objects:
  - ▣ Identify classes required
  - ▣ Define (declare) classes (including definition of all components) required
  - ▣ Instantiate objects from the classes at the beginning of the main program body
  - ▣ Work with objects by calling their defined components when required in the main program!

# Identify classes required

85

- Independent entities in the program are usually coded as classes and instantiated as objects.
-  In a simulation model what are candidates to be coded as classes and objects?

# Identify classes required

86

- For example in a DES model Entities, Resources and simulation model Engine are candidates for the program classes.

# Declare Classes

87

- In general there are a variety of parameters/identifiers to be used for declaring a class, in this course we only introduce basic parameters/identifiers:

```
[ <attributelist> ] [ accessmodifier ] [ Shadows ] [ MustInherit |  
NotInheritable ] [ Partial ] _  
Class name [ ( Of typelist ) ]  
    [ Inherits classname ]  
    [ Implements interfacenames ]  
    [ statements ]
```

End Class

# Declare Classes

88

- Main form class is the default class created in a VB.NET project, e.g.:

```
Public Class Form1
    {default programming environment}
End Class
```

- New classes can be declared right after the default class within the same file, e.g.:

```
Public Class Form1
    {default programming environment}
End Class
```

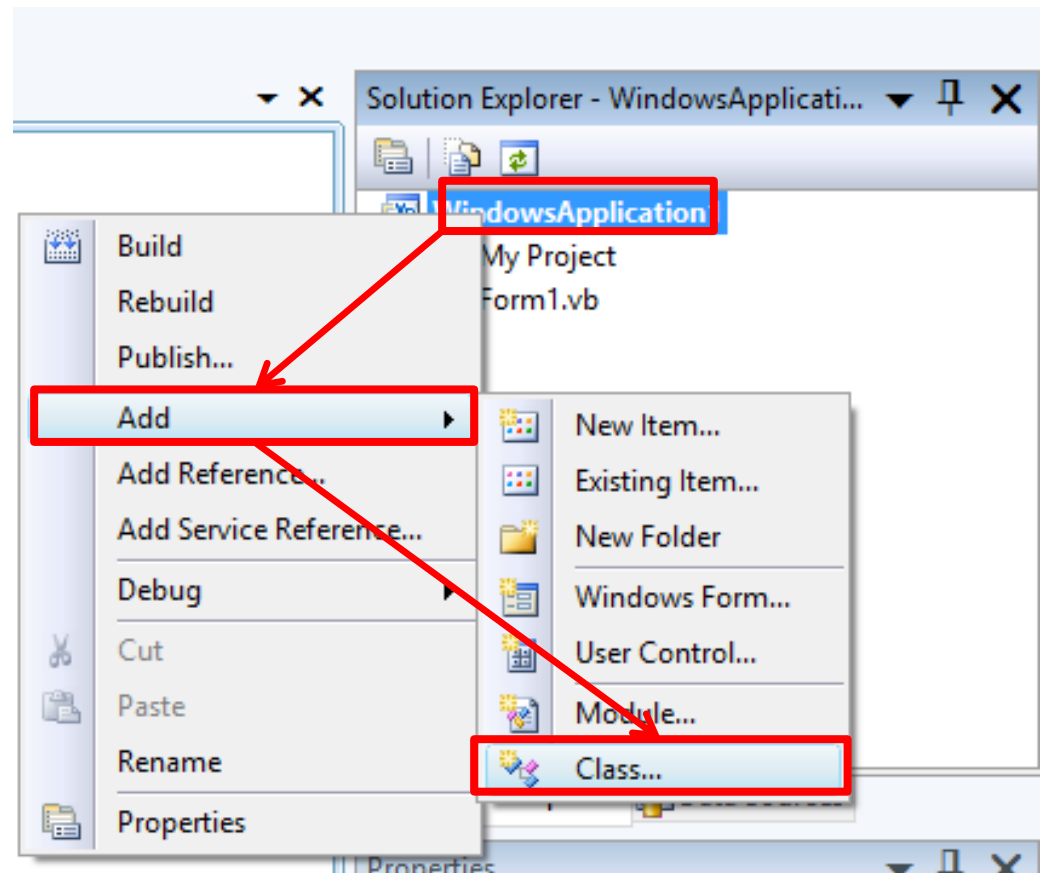
```
Public Class Engine
    {programming environment}
End Class
```



# Declare Classes

89

- New classes (usually supporting classes which are used as auxiliary parts of the program) can be declared in separate files from the default class file.

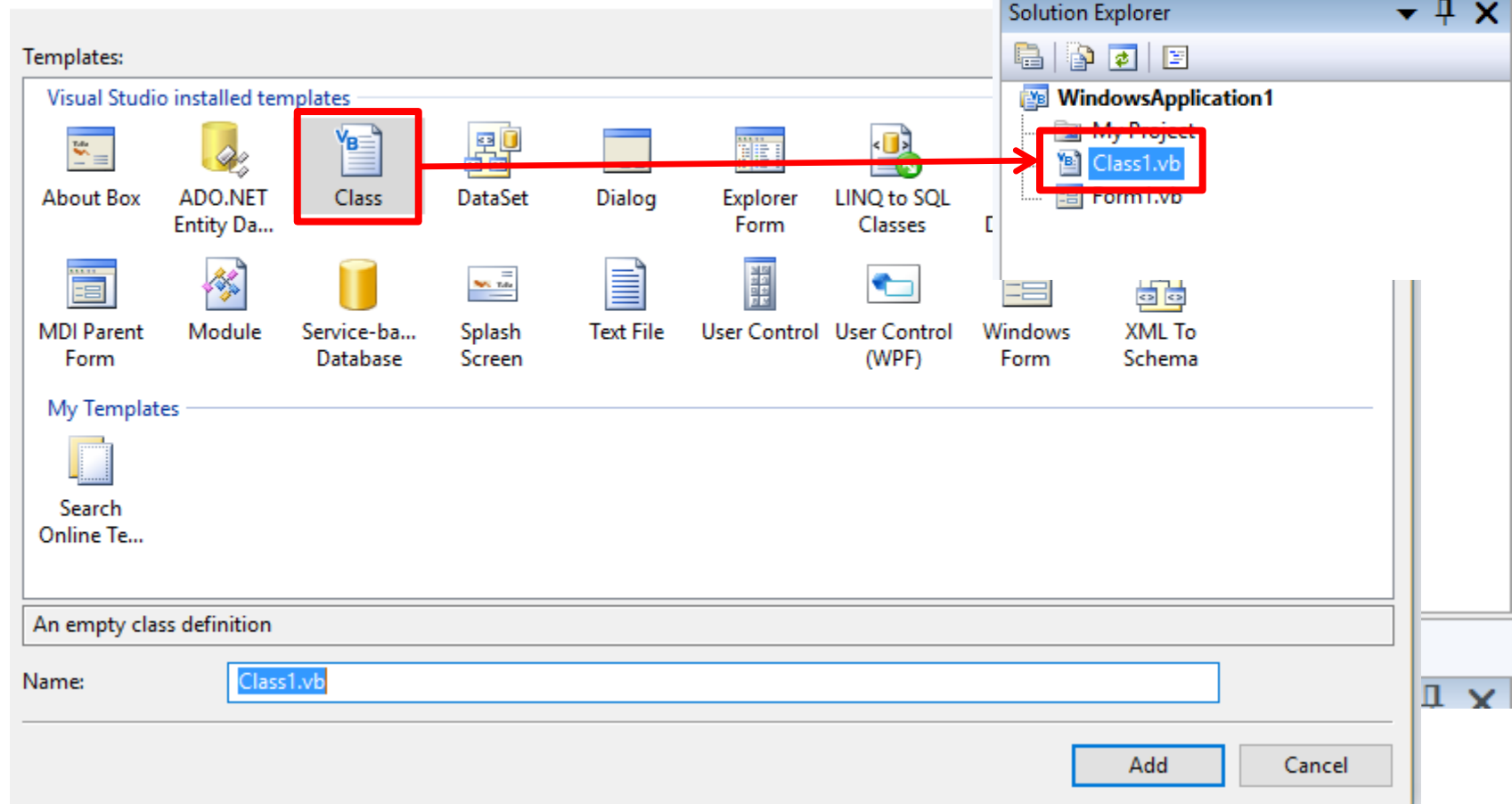


# Declare Classes

90

## □ New classes, usually

Add New Item - WindowsApplication1



# Declaring Properties

91

- Properties are usually declared at the beginning of a class.
- Using *Auto-implemented properties* is the simplest way for declaring a class property:

```
Public PropertyName As DataType
```

- *Auto-implemented properties* are used when you just simply want to get (read) values and set values for a property with no control on the reading and setting value process! Examples:

```
Public Name As String  
Public currentTime As Double = 0
```

# Declaring Properties

92

- Extensive form of properties declaration is as following:

```
Private ClassVariable As DataType  
[Public] Property PropertyName( ) As DataType  
    Get  
        Return ClassVariable [PropertyName = ClassVariable]  
    End Get  
  
    Set (ByVal Value As DataType)  
        [statements, such As validation]  
        ClassVariable = Value  
    End Set  
End Property
```

- This form of property declaration lets you to put verification and validation statements when a property is read (get) or set

# Declaring Properties

93

## □ Example:

```
Public Class Engine
    Private _currentTime As Double = 0
    Property currentTime() As Double
        Get
            if _currentTime < 0 then
                MsgBox("Check your time, why is it minus?")
            End if
            Return _currentTime
        End Get
        Set(ByVal value As Double)
            if value < 0 then
                MsgBox("Becareful! Your time should not be minus?")
            End if
            _currentTime = value
        End Set
    End Property
End Class
```

# Declaring Properties

94

- ReadOnly (can not be set out of the class), WriteOnly (can not be read out of the class) examples:

```
Public Class Engine
    Dim _currentTime as double = 0
    Dim _secreteName as String
    Public ReadOnly Property currentTime() As Double
        Get
            return _currentTime
        End Get
    End Property

    Public WriteOnly Property secreteName() As String
        Set(ByVal value As String)
            _secreteName = value
        End Set
    End Property
End Class
```

# Using Properties

95

## □ Example:

```
Public Class Form1
```

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles Button1.Click
```

```
    Dim myEngine as New Engine()
```

```
        'myEngine Object with the type of Engine class is defined
```

```
    ..... ' Statements involving myEngine object
```

```
    txtFirst.Text = myEngine.currentTime.ToString
```

```
        'Shows current time on the txtFirst textbox
```

```
End Sub
```

```
End Class
```

# Declaring Methods

96

- ❑ Methods include functions and procedures
- ❑ Declaring functions and procedures is similar to what we discussed in prior parts
- ❑ Methods are usually declared at the end of a class.
- ❑ Use “Public” keyword at the beginning of a method declaration if you are interested to access the method from outside of the class!



# Declaring Methods

97

## □ Example:

```
Public Class Loader
```

```
    Public bucketSize As Double
```

```
    Public Function loadingTime (ByVal TruckCapacity as double) as  
double
```

```
        loadingTime = TruckCapacity / bucketSize * (1+ Rnd())
```

```
    End Function
```

```
End Class
```

# Using Methods

98

## □ Example:

```
Public Class Form1
```

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles Button1.Click
```

```
Dim myLoader as New Loader()
```

```
    'myLoader Object with the type of Loader class is defined
```

```
myLoader.bucketSize = 0.1 'm3
```

```
Dim myTruckCapacity as double
```

```
..... ' Statements involving myLoader object and myTruckCapacity
```

```
txtSecond.Text = myLoader.loadingTime(myTruckCapacity).ToString
```

```
    'Shows current time on the txtFirst textbox
```

```
End Sub
```

```
End Class
```

# Declaring Events

99

- An Event's declaration consists of three parts:

1. Declaring Events: Introducing event's name and its parameters using "Event" keyword. Example:

```
Public Event TruckArrival(ByVal TruckNumber As Integer)
```

2. Raising (Triggering) Events: An event is like a message announcing that something important has occurred. The act of broadcasting the message is called raising the event.

Usually we need to set raise of an event of an object when a threshold is reached within the object. Example:

```
RaiseEvent TruckArrival (TruckNumber)
```

# Declaring Events

100

- An Event's declaration consists of three parts (cont'd):

3. Event Handlers: Event handlers are procedures that are called when a corresponding event is raised. You can use any valid subroutine with a matching signature as an event handler. You cannot use a function as an event handler, however, because it cannot return a value to the event source. Example:

```
Private Sub engine_TruckArrival (TruckNumber) Handles Me.
```

```
TruckArrival
```

```
    'Statements for handling truck arrival
```

```
End Sub
```

# Declaring Events-Example

101

```
Public Class Truck
    Public Event loadingCompletion(byVal msg as string)
    Public Event dumpingCompletion(byVal msg as string)
    Dim _loaded As Boolean
    Public Property loaded() As Boolean
        Get
            Return _loaded
        End Get
        Set(ByVal value As Boolean)
            _loaded = value
            If value = True Then
                RaiseEvent loadingCompletion("Write Loading Completion Event Handling Statements")
            Else
                RaiseEvent loadingCompletion("Write Dumping Completion Event Handling Statements")
            End If
        End Set
    End Property
    Private Sub Truck_loadingCompletion(byVal msg as string) Handles Me.loadingCompletion
        MsgBox(msg)
    End Sub
    Private Sub Truck_dumpingCompletion(byVal msg as string) Handles Me.dumpingCompletion
        MsgBox(msg)
    End Sub
End Class
```

# Constructor

102

- Used for initializing private members of a class
- Name of the constructor should always be **New()**
- Can have none or several parameters
- Does not return any value. So they are sub routines or Procedures!
- Need not to be invoked explicitly. They are invoked automatically when an object is created
- A class can have more than one constructor
- Every constructor should differ from the other by means of number of parameters or data types of parameters

# Constructor

103

## □ Example:

```
Public Class Form1
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
        System.EventArgs) Handles Button1.Click
        Dim myTruck as New Truck(True)
        'Other statemenets
    End Sub
End Class

Public Class Truck
    Public loaded() As Boolean
    Sub New(ByVal truckLoaded As Boolean)
        loaded = truckLoaded
    End Sub
End Class
```

# Inheritance

104

- Classes can inherit their components (properties, methods and events) from other classes (called parent classes)
- Inheritance is a powerful programming capability and significantly saves repeated codes
- Designing a proper inheritance structure is an important aspect of object oriented programming, however, for the sake of brevity we do not discuss it more!



# Inheritance

105

## □ Example:

```
Public Class form1
    Dim myTruck As New Truck
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
        Handles Button1.Click
        myTruck.maxLoadingCapacity = 10 ' tonne
        myTruck.maxSpeed = 80 'km/h
        myTruck.weight = 6 'tonne
        MsgBox("My truck capacity is: " & myTruck.maxLoadingCapacity.ToString)
    End Sub
    'Other properties and methods ...
End Class

Public Class MobileMachines
    Public maxSpeed As double 'in km/h
    Public weight as double 'in tonne
    'Other properties and methods ...
End Class

Public Class Truck
    inherits MobileMachines
    Public maxLoadingCapacity As double 'in tonne
    'Other properties and methods ...
End Class
```

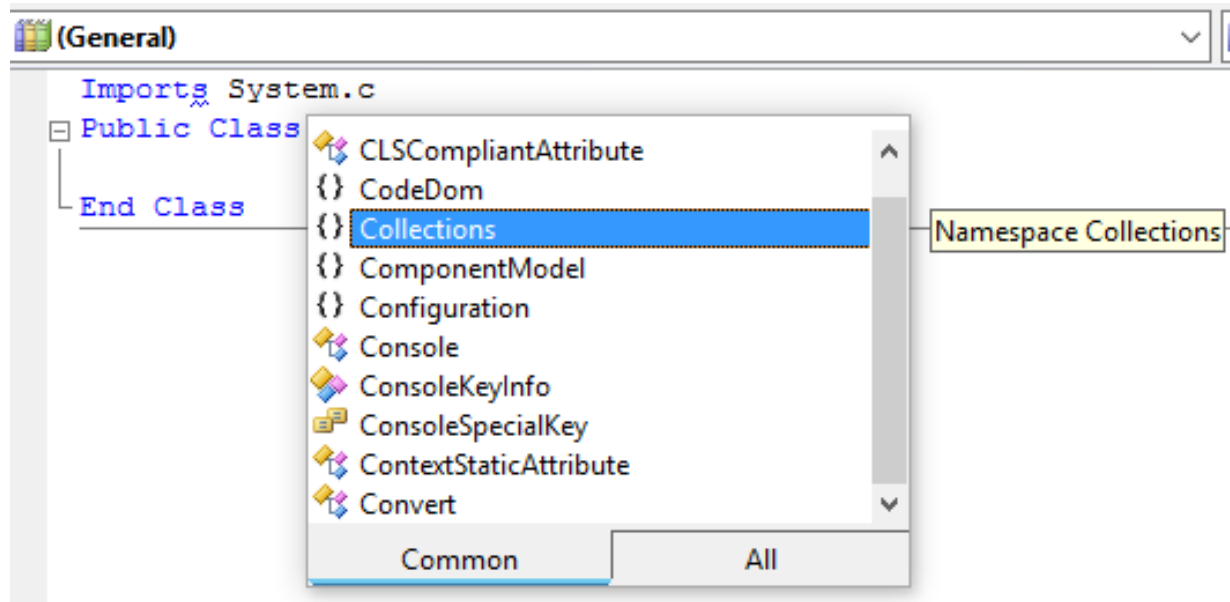
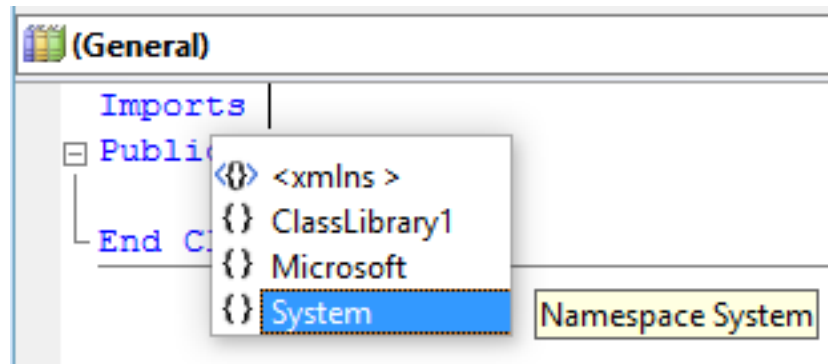
# Namespace

106

- In addition to the embedded classes available to the users, VB.NET provides *sets of prepared classes* or *class libraries*, so called Namespaces, which facilitate different types of programming, e.g., database, graphic, program diagnostic, working with other windows applications, etc.
- Importing all prepared Namespaces in a program makes the program heavy and slow, so VB.NET lets programmers select among them, based on the capabilities they require.
- Use “Imports” keyword at the beginning, before class declaration for adding required Namespaces to your program.
- By typing “Imports”, list of selectable high level Namespaces occurs, you may narrow down your specific Namespace to lower levels

# Namespace

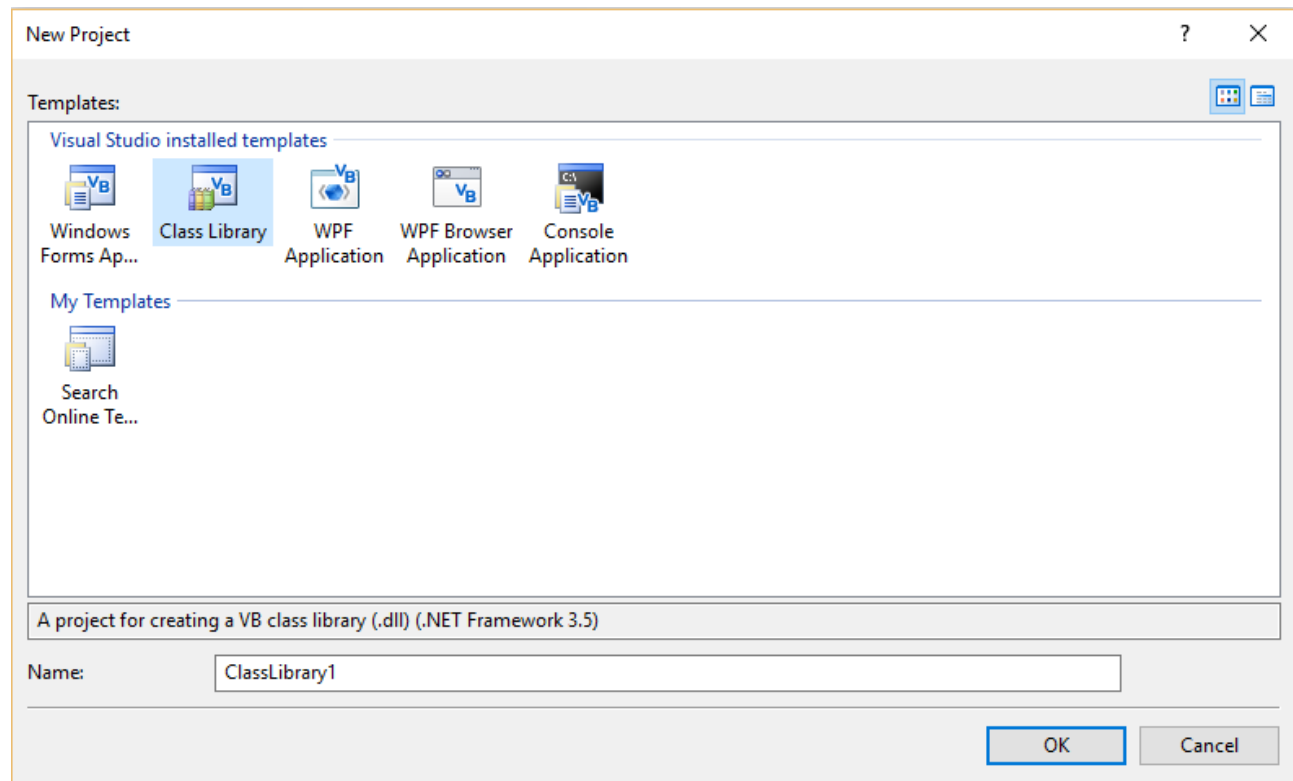
107



# Namespace

108

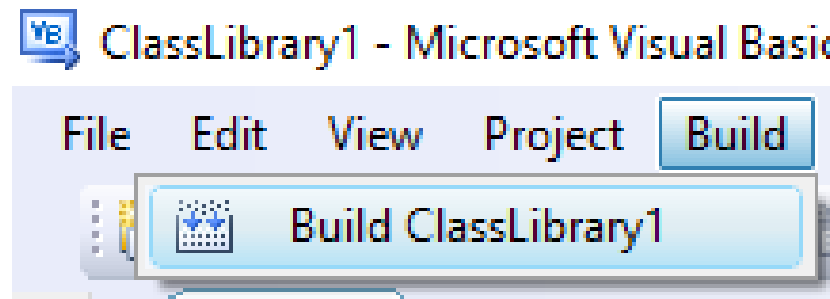
- ❑ In VB.NET programmers can specify their own specific Namespaces which can be used in their different programs.
- ❑ At the beginning of the program new projects should be set as “Class Library”.



# Namespace

109

- After completing classes required, classes should be built:

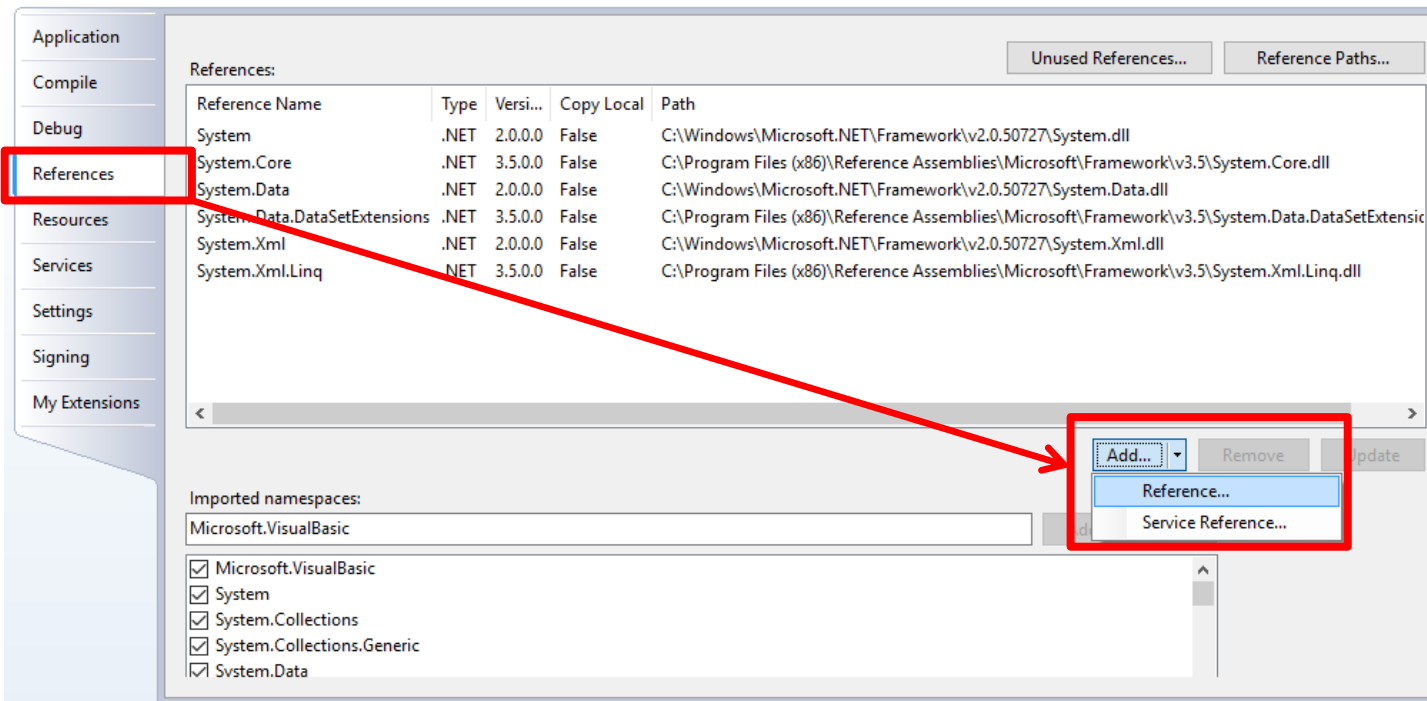


- A dll file is create at “\bin\Release” within the project’s main folder which can be used as Namespace in other projects
- A dll or dynamic-link library file are portable set of classes which can be used in different programming languages, regardless of their original programming languages.

# Namespace

110

- To be able to import a specified namespace(s), first its related dll file should be added as References to the project. References are inside the project's properties. A project's property is accessible through right-click on the project's name in the solution explorer.



# Collections

# Definition

112

- Collections include several types of VB.NET objects used for grouping and managing related objects.
- For example, every Form has a collection of controls. (You can access this collection through the form's Controls property.) This collection is an object that represents all the controls on that form.
- Collection allows you to retrieve an objects inside the collection by its index, and to loop through the elements of the collection using:  
For Each...Next Statement
- To be able to work with collections you first need to import their related namespace from: System.Collections



# Arrays and Collections

113

## Array

- Size
- Item(Index)

## Collection

- Size
- Item(Index)
- Item(Key)
- Add(Item)
- Remove(Item)
- Contains(Item)

# Different Kinds of Collections

114

- Collection
- Array List
- Sorted List
- Hash Table
- Stack
- Queue

We only discuss the basic form of collections here, in fact, basic form of collection is the parent of other kinds! So, many properties and methods are the same between different kinds of collections.

# Collections-Example

115

```
Imports System.Collections
```

```
Public Class loaderQueue
```

```
    Public Q As New Collection
```

```
    Public Function QLength() As Integer
```

```
        QLength = Q.Count
```

```
    End Function
```

```
    Public Sub addNewTruck(ByVal TruckName As String, ByVal TruckId As Integer)
```

```
        Q.Add(TruckName, TruckId.ToString)
```

```
    End Sub
```

```
    Public Sub removeTruck(ByVal TruckId As Integer)
```

```
        Q.Remove(TruckId.ToString)
```

```
    End Sub
```

```
End Class
```

# Home Assignment 6

116



Write a class called `simulationCourse`. The class has following elements:

Properties: `Location`, `CourseWeeklyHours`, `CourseSemester` `StudentName`,  
`StudentMark`, `Lecturer`, `CourseReference`,

`CourseStatus[ToCome/InProgress/Completed]`

Methods: `CourseAverageMark`, `CourseMaximumMark`

Events: `SemesterStart` [`CourseStatus` is changed to `InProgress`], `SemesterFinish`  
[`CourseStatus` is changed to `Finished`]

Instantiate the class in a button's event-handler. Use your current simulation course info to set the values of the created object (for the student info, information of at least 5 students should be added, for the other elements complete info should be entered), link `SemesterStart` and `SemesterFinish` events to two different buttons! Report course average, maximum and minimum marks using separate buttons!

**(Due two days)**

# Reference

117

- <https://msdn.microsoft.com>



**Thank you!**