Introduction to Visual Basic and Visual C++

Lesson 3

Introduction

Overview of Basic and .NET Framework

A Brief History for Basic…

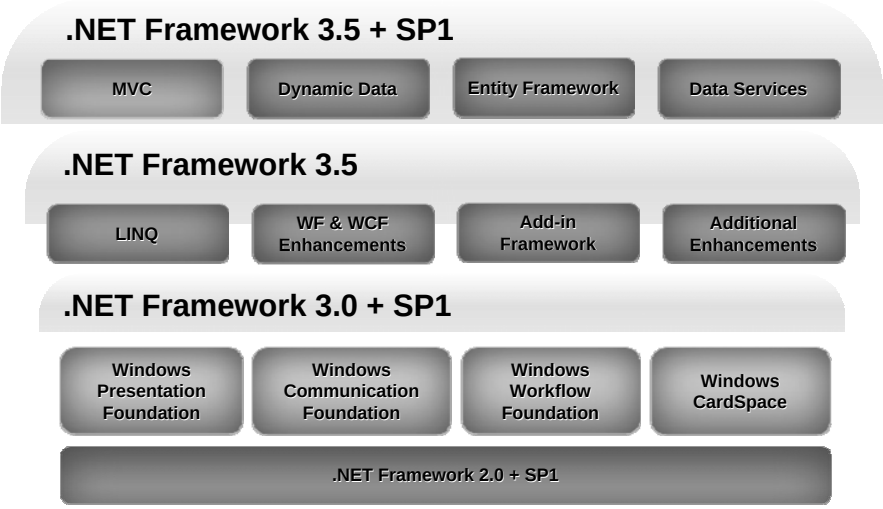
- Visual basic evolved from BASIC (**B**eginners' **A**ll-purpose **S**ymbolic **I**nstruction **C**ode).
- The BASIC language was created by Professors John Kemeny and Thomas Kurtz of Dartmouth College in the mid 1960s.
- It is a carefully constructed English-Like language basically used by the programmers to write simple computer programs.
- It served the purpose of educating laymen like we all the basic concepts of programming.
- From then on many versions of BASIC were developed to accommodate different computer platforms.

What is Visual Studio 2008?

- Best tool set for Windows Server 2008, Windows Vista and Microsoft Office 2007
  - .NET Framework 3.5 design surfaces
  - Office 2007 support including ClickOnce
  - MFC support for Vista common controls
- Improvements for Web Developers
  - HTML / CSS designer enhancements
  - Integrated AJAX and JavaScript support
- Language advances
  - .NET Framework multi-targeting support
  - Improved Data & Language integration in VB / C#

Feature of Visual Studio 2008

What is .NET Framework 3.5?



What can Visual Studio create for you?

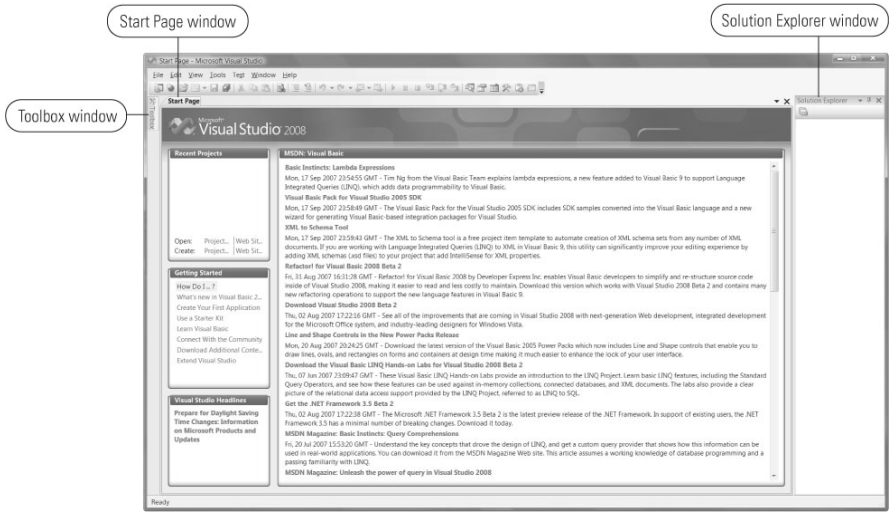
Visual Studio.NET

A Quick Look to the Visual Studio 2008 Environment

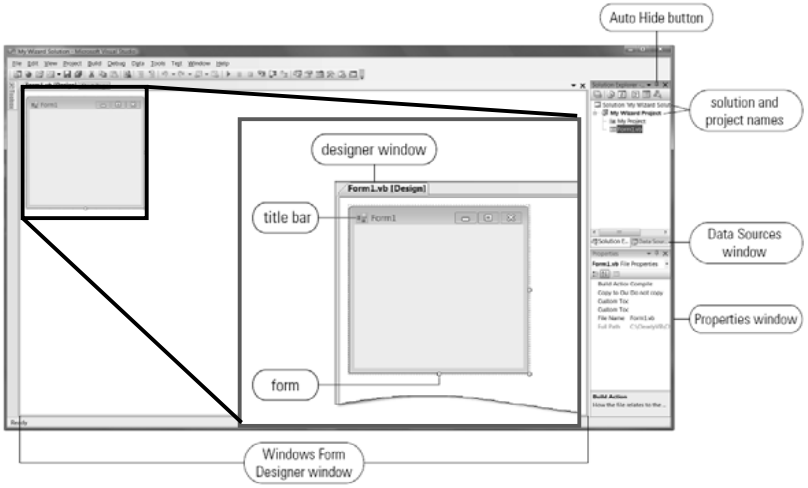
Microsoft Visual Studio .NET

- The Microsoft Visual Studio Development Environment is also called **Integrated Development Environment** (IDE):
  - A form designer
  - A code editor
  - A compiler
  - A debugger
  - An object browser
  - A Help facility

Welcome Screen

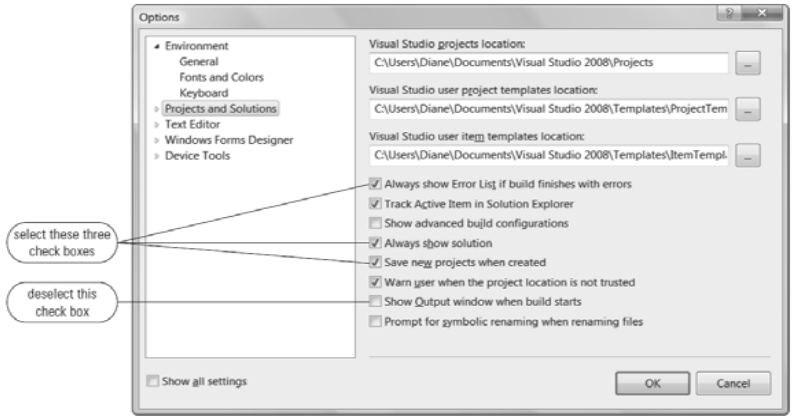


Development Environment



General Project Setup

- You can call up the Options dialog by **Tools ➔ Options**.



Visual Basic.NET

Create VB.NET Project

Development Process

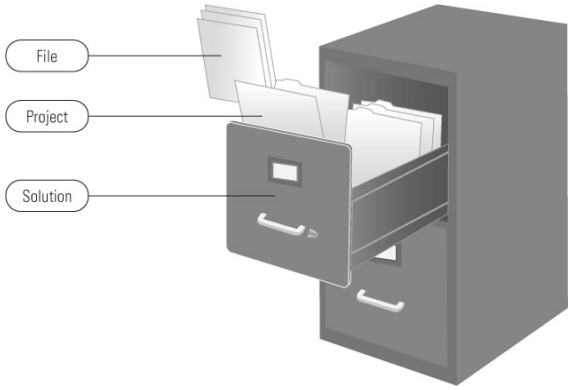
- Planning (Design)
  - Design the GUI (graphical user interface)
  - List the objects and properties needed
  - Plan the event procedures (what the code does)
- Programming (Implementation)
  - Define the GUI using objects (Form, TextBox, Label, etc.)
  - Set the properties (Color, Font, etc)
  - Write code to implement procedures

Program Development

- To create a VB.NET program you will utilize the Visual Basic .NET development environment, and you will
  - Create a window, called **Form**
  - Select elements, which are classes, from a toolbox and place them within the window, called **Controls**
  - Write code for each object that you place on the window that defines how the object responds to various events, called **Object-oriented Programming (OOP)**.

Solutions, Projects and Files

- **Solution**: can contain several projects
- **Project**: container that stores files associated with project



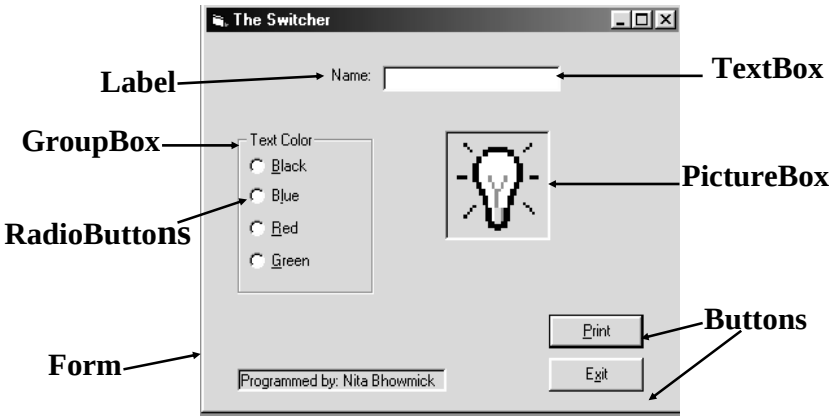
VB.NET Application Files

Extension	Description
.sln	A solution file that holds information about the project. This is the only file that is opened
.suo	A solution user options file that stores information about the selected options
.vb	A VB file that holds the definition of a form
.resx	A resource file for the form
.vbproj	A project file that describes the project and lists the files are included
.user	A project user option file that holds project option settings

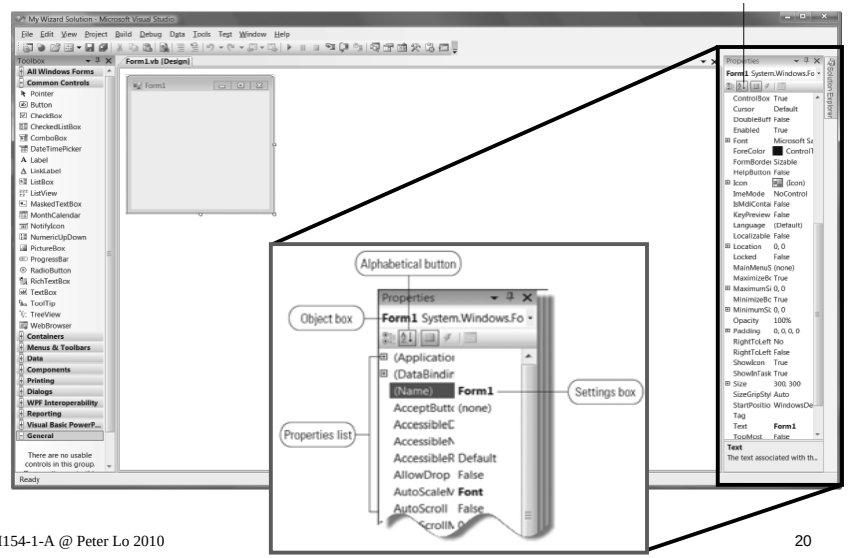
VB.NET Object Oriented Programming

- Using the VB.NET OOP Technology to work with objects and develop an event-driven program.
- Each object consists of:
  - **Classes**: Forms, Labels, Buttons, etc
  - **Objects**: A particular Form, Label, Button, etc.
  - **Properties** (attributes of an object): The Name of a form, the Text in a Label, etc.
  - **Methods** (the actions that an object performs in response to GUI events): Close, Show, Clear, etc.
  - **Event**: when a user takes an action

A Sample Graphical User Interface (GUI)

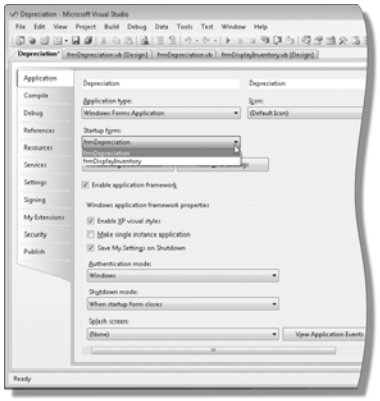


Define Properties for Object

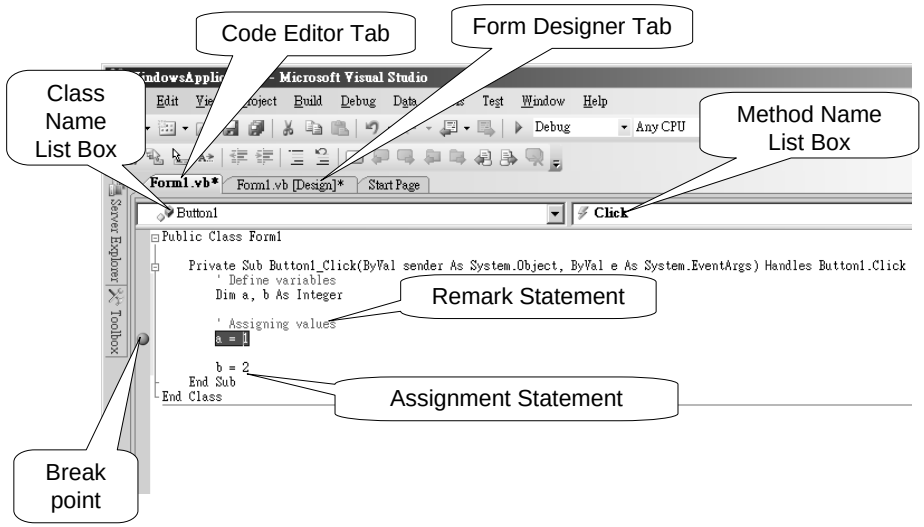


Setting Startup Object

- In the Project Property dialog box (**Project ➔ Properties**), you can select the Startup Form

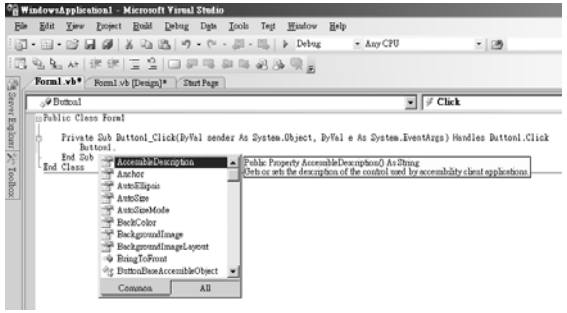


Coding Editor Windows



IntelliSense

- As you typed the code, you may have noticed that after you typed ".", a list of words appeared below the cursor.
- This feature is called **IntelliSense**. It allows you to just type the first few letters of a word until the word is selected in the list. Once selected, you can press the [Tab] key to finish the word.



Write Code

- Develop a Visual Basic code in Procedures
- A sub procedure begins with **Private Sub** and ends with **End Sub**
- Note:
  - Write code for the events you care about, the events you want to respond to with code
  - Code is written as event procedures
  - VB will ignore events for which you do not write code

Remark Statement

- Also known as Comment, used for documentation
- Non-executable
- Automatically colored Green in Editor
- Begins with an apostrophe (')
  - On a separate line from executable code
  - At the right end of a line of executable code

```
' Display the Hello World message.
```


Assignment Statement

- Assigns a value to a property or variable
- Operates from right to left
- Enclose text strings in quotation marks (" ")
- Syntax: ***variable = value***
- Examples
  - pi = 3.14
  - messageLabel.ForeColor = Color.Red
  - messageLabel.Text = "Hello World "

Continuing Long Program Lines

- Line continuation character (_):
  - Used to break up a long instruction into two or more physical lines
  - Underscore must be preceded by a space and must appear at the end of a physical line
  - Do not use it within parentheses

Space, then Underscore

```
Dim objGraphics As Graphics
objgraphics = Me.CreateGraphics
objgraphics.Clear(system.Drawing.SystemColors.Control)
objgraphics.DrawLine(system.Drawing.Pens.Blue, 0, 0, _
          Me.DisplayRectangle.Width, Me.DisplayRectangle.Height)
objgraphics.Dispose()
```


Terminate the Application

- The **Me.Close( )** instruction closes the current form at run time
  - If the current form is the main form, the application is terminated
- The **End** statement terminate the application at run time
  - It has the same effect with the **Me.Close( )** instruction in main form.

Programming Concept

Variables and Constants

Variables and Constants

- Variables
  - Memory locations that hold data that can be changed during project execution
    - Example: Number of working hours – WorkingHour
- Constant (User defined)
  - Memory locations that hold data that cannot be changed during project execution
    - Example: The value of pi
- Intrinsic Constant (System defined)
  - System defined within Visual Studio
    - Example: TextBox.ForeColor = Color.Blue

Variables and Constants

- Variables and Constants use temporary memory locations that have:
  - A Name (identifier)
  - A Data type:
    - Dim var1 As Integer
    - Dim var2 As String
    - Const PI = 3.14
  - A Scope: Subroutine, Module or Global
- Variable values can be changed as the program is executed
- Constant values cannot be changed

Declaration

- Variables and Named Constants must be declared and specify the type of data before being used in code
- In Visual Basic when you declare a Variable or Named Constant
  - An area of memory is reserved
  - A name is assigned called an Identifier
  - Follow rules and naming conventions
- Use Declaration Statements to establish Variables and Constants
  - Assign name and data type
  - Not executable unless initialized on same line
- Examples:
  - Const var1 As String = “Hello World”
  - Dim var2 As Integer

Basic Data Types in Visual Basic

Data type	Stores	Memory required
Boolean	a logical value (True, False)	2 bytes
Char	one Unicode character	2 bytes
Date	date and time information Date range: January 1, 0001 to December 31, 9999 Time range: 0:00:00 (midnight) to 23:59:59	8 bytes
Decimal	a number with a decimal place Range with no decimal place: +/-79,228,162,514,264,337,593,543,950,335 Range with a decimal place: +/-7.9228162514264337593543950335	16 bytes
Double	a number with a decimal place Range: +/- 4.94065645841247 X 10^{-324} to +/-1.79769313486231 X 10^{308}	8 bytes
Integer	integer Range: -2,147,483,648 to 2,147,483,647	4 bytes
Long	integer Range: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	8 bytes
Object	data of any type	4 bytes
Short	integer Range: -32,768 to 32,767	2 bytes
Single	a number with a decimal place Range: +-1.401298 X 10^{-45} to +/-3.402823 X 10^{38}	4 bytes
String	text; 0 to approximately 2 billion characters	

Declaring Variables

- Declared inside a procedure using a **Dim** and **As** statement
- Declared outside a procedure using **Public**, **Private** or **Dim** statements
- Always declare the variable's data type
- May declare several variables with one statement
- Use IntelliSense to assist in writing statements
- Example
  - Dim var1 As Integer
  - Public var2 As Double
  - Private var3 As String

Assigning Variables

- You assign a value to your variable with the = sign, which is sometimes called the assignment operator:
  - var1 = 42
- You can also declare a variable on one line of code, and then later assign the value on another line. This can result in an error if you try to use the variable before assigning it a value.
  - Dim var1 As Integer = 42
  - Dim var2 As String = "Hello world"

Constants - Named & Intrinsic

- Named Constants
  - User assigned name, data type and value
  - Use **CONST** keyword to declare
  - The value of named constants can not be changed during the execution.
  - Example
    - Const COURSE_NAME As String = "VB.NET"
- Intrinsic Constants
  - System defined within Visual Studio

Naming Variables and Constants

- Identifiers/Names in VB are *not case sensitive*, but required for each Variable and Constant
- Rules of Naming Conventions for an identifier:
- Must follow Visual Basic Naming Rules
  - Meaningful names
  - May consist of letters, digits, and underscores
  - Can not contain spaces, periods, or reserved words
  - Include class (data type) of variable
  - Make the first letter lowercase and then capitalize each word of the name – always use mixed case for variables and **UPPERCASE** for constants

Data Type	Prefix
String	str
Integer	int
Decimal	dec
Double	dbl
Char	chr
Boolean	bln
Byte	byt
Date	dtm
Long	lng
Short	shr
Single	sng

What is a String?

- A string is any series of text characters, such as letters, numbers, special characters, and spaces. Strings can be human-readable phrases or sentences, such as “Hello World" or an apparently unintelligible combination, such as "@#fTWRE^3 35Gert".
- String variables are created just as other variables: by first declaring the variable and assigning it a value, as shown below.

```
Dim aString As String = "This is a string"
```


Assigning String

- When assigning actual text (also called a string literal) to a String variable, the text must be enclosed in quotation marks (""). You can also use the = character to assign one String variable to another String variable, as shown in this example.
  - **Dim aString As String = "This is a string"**
  - **...**
  - **Dim bString As String = ""**
  - **bString = aString**
- The previous code sets the value of bString to the same value as aString (This is a string).

Concatenate String

- You can use the ampersand **&** character to sequentially combine two or more strings into a new string, as shown below.
  - **Dim aString As String = "Hello"**
  - **Dim bString As String = "World"**
  - **Dim cString As String = ""**
  - **cString = aString & bString**
- The previous example declares three String variables and respectively assigns “Hello" and “World" to the first two, and then assigns the combined values of the first two to the third variable.
- What do you think the value of cString is? You might be surprised to learn that the value is Across the “HelloWorld” because there is no space at the end of aString or at the beginning of bString. The two strings are simply joined together.
- If you want to add spaces or anything else between two strings, you must do so with a string literal, such as " ".

Scope and Lifetime of Variables

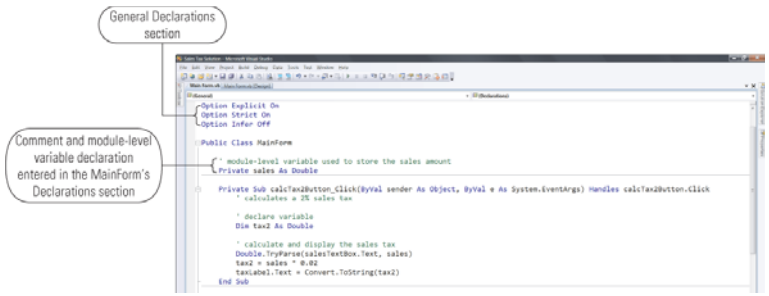
- **Scope**: indicates where the variable can be used
- **Lifetime:** indicates how long the variable remains in memory
- Declare the scope of a variable by choosing where to place the declaration statement
- Scope of variables are declared as
  - Namespace
  - Module-level
  - Local
  - Block-level
- Lifetime of a variable is the period of time the variable exists

Scope and Lifetime of Variables

- Namespace
  - Available to all procedures of project
  - Good programming practice excludes use of Namespace variables
- Module
  - Available to all procedures within that module (often a form)
  - Use Public or Private keywords
  - Place in declaration section of module (form)
- Local
  - Available only to the procedure it is declared in
  - Declare with Dim keyword and place at top of procedure
- Block
  - Available only in block of code where declared

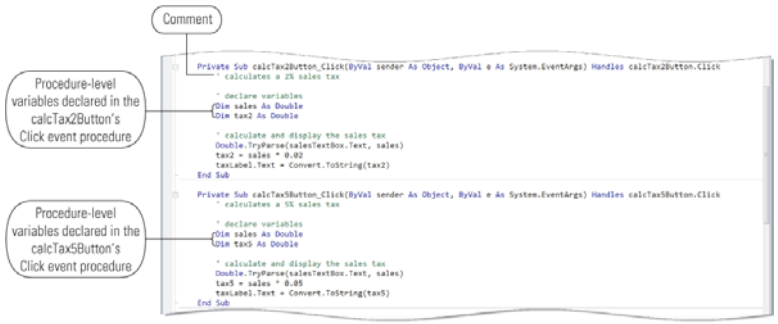
Module Scope Variables

- Declared in the form's declarations section using **Private** keyword
- Variable can be used by all procedures in the form
- Module-level variables retain their values until the application ends



Procedure Scope Variables

- Declared within a procedure using the **Dim** keyword
- Only the procedure can use the variable
- With procedure-level scope, two procedures can each use the same variable names



Block Scope Variables

- Scope is defined by where the variable is declared within a program
- Within an event handler, an If…Then…Else statement is considered a block of code
- Variables can be declared within a block of code
- The variable can be referenced only within the block of code where it is declared

```
        If intAge < 18 Then
            Dim intYears As Integer
            intYears = 18 - intAge
            Me.lblMessage.Text = "You can vote in " & intYears & " years(s)."
        Else
            Me.lblMessage.Text = "You can vote!"
        End If
```


Finding and Fixing Errors

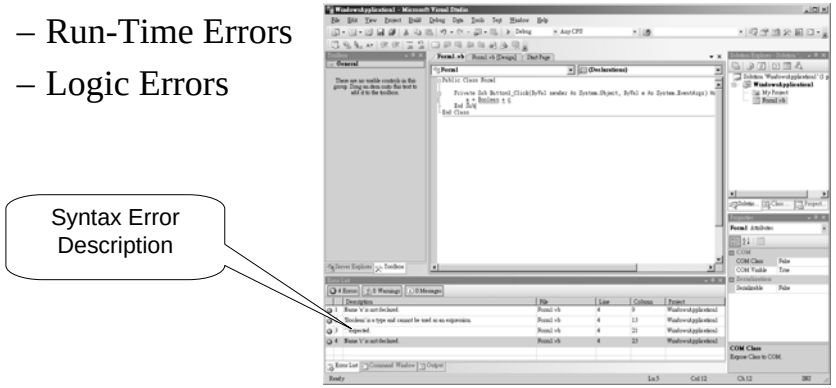
Debugging

Different Mode in VB.NET

- Three modes are defined when you are running VB.NET.
  - **Design Time** is when you are creating the form and writing the VB code. If you choose the Debug tool from Toolbar menu, the [Start], [Break], and [Stop] buttons are displayed on a Toolbar near the top of the screen.
  - **Run Time** is when you are actually running the program in VB.NET. When the program is running, the [Start] button is disabled and the Debug menu is changed to show the Break and Stop as shown on the slide.
  - **Break Time** is when the program pauses because of an error or you select the break mode, which is the button with double vertical bars.

Finding and Fixing Errors

- Programming errors come in three varieties:
  - Syntax Errors
  - Run-Time Errors
  - Logic Errors



Syntax Errors (Compile Errors)

- Statements that break VB's rules for punctuation, format or spelling
- Smart editor finds most syntax errors, compiler finds the rest
- Common sources of syntax errors are misspelling an object name, property or method, keywords, or using wrong punctuations

Run-time Errors

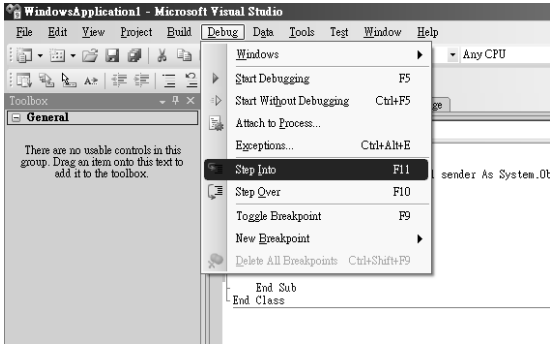
- Statements or functions that fail to execute such as impossible arithmetic operations, or unexpected termination
- Common source of these errors include division by zero, attempting arithmetic operations on non-numeric data, and taking the square root of a negative number
- Example:
  - Dim x As Double
  - x = Math.Sqr(-5)

Logic Errors

- Logic errors occur when the project runs but produces incorrect results
- Common sources of these errors include incorrect calculations, display of the wrong information, or incorrect formatting of output
- These are generally the hardest errors to fix!
  - They are errors made by the programmer or designer
  - Well commented programs enable another programmer to read the code and correct logic errors
- Example: area = 2 * pi * r

Stepping

- One of the most common debugging procedures is stepping – executing code one line at a time.
- The Debug menu provides three commands for stepping through code:
  - Step Into
  - Step Over
  - Step Out



Step Into, Step Over & Step Out

- **Step Into** executes only the call itself, then halts at the first line of code inside the function. On a nested function call, Step Into steps into the most deeply nested function.
- **Step Over** executes the entire function, then halts at the first line outside the function..
- **Step Out** resumes execution of your code until the function returns, then breaks at the return point in the calling function.

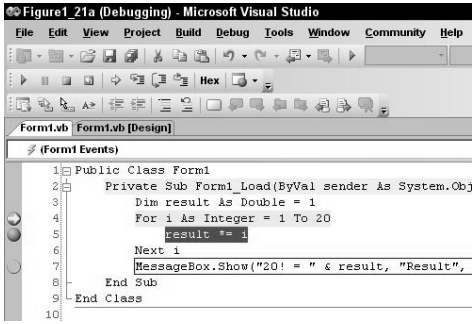
Use Step Into if you want to look inside the function call.

Use Step Over if you want to avoid stepping into functions.

Use Step Out when you are inside a function call and want to return to the calling function

Breakpoints

- A breakpoint tells the debugger that an application should break (pause execution) at a certain point or when a certain condition occurs. When a break occurs, your program and the debugger are said to be in break mode.



Different Type of Breakpoint

- The Visual Studio debugger has four types of breakpoints:
  - A function breakpoint causes the program to break when execution reaches a specified location within a specified function.
  - A file breakpoint causes the program to break when execution reaches a specified location within a specified file.
  - An address breakpoint causes the program to break when execution reaches a specified memory address.
  - A data breakpoint causes the program to break when the value of a variable changes. You can set a data breakpoint on a global variable or a local variable in the top-most scope of a function. (C++ only)