Programming with Microsoft Visual Basic.NET

Lesson 5

What have we learnt in last lesson?

- Using Toolbar in Windows Form.
- Using Tab Control to separate information into different tab page
- Storage hierarchy information into Tree view
- Development of subroutine by using Function and Procedure

Array

Declare and Assign Array

What is Array?

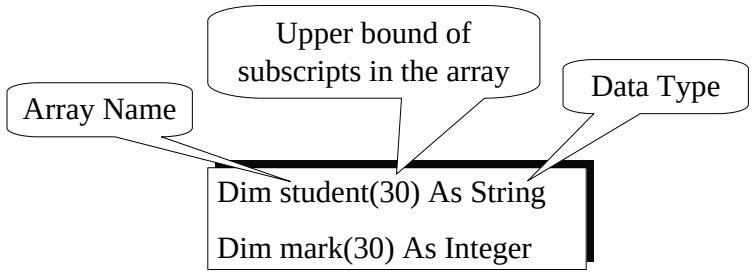
- An array is a collection of data of the same type. It is implemented in Visual Basic as an object.
- Each individual item in array that contains a value is called an element
- Arrays provide access to data by using a numeric index, or subscript, to identify each element in the array

Data Type	Default Value
Any Numeric Data Type	0
Reference Data Type	Null
Boolean Data Type	False

Why use Array?

- Suppose that you want to evaluate the exam grades for 30 students and to display the names of the students whose marks.

```
Dim student1 As String, mark1 As Integer
Dim student2 As String, mark2 As Integer
Dim student3 As String, mark3 As Integer
```



Array Terminology

- Syntax for Array Declaration:
  - **Dim** *arrayName***(***n***) As** *DataType*
  - 0 is the Lower Bound of the array
  - **n** is the Upper Bound of the array – the last available subscript in this array
- The number of elements is the *size* of the array
- Syntax for assigning values into an array
  - *arrayName***(**arrayIndex**) =** Value
- Arrays may be initialized when they are created:
  - **Dim** *arrayName***( ) As** *varType* **= {***value0***,** *value1***, _**
    *value2***, ..., *valueN***}**

Creating a Two-Dimensional Array

- A two-dimensional array holds data that is arranged in rows and columns
- A two-dimensional array can also be described as an "array of arrays".

```
Dim intVal(2, 3) As Integer
```

	column 0	column 1	column 2	column 3
row 0	intVal(0,0)	intVal(0,1)	intVal(0,2)	intVal(0,3)
row 1	intVal(1,0)	intVal(1,1)	intVal(1,2)	intVal(1,3)
row 2	intVal(2,0)	intVal(2,1)	intVal(2,2)	intVal(2,3)

ReDim Statement

- The size of an array may be changed after it is created where *arrayName* is the name of the already declared array and *m* is an Integer literal,variable, or expression.:
  - **ReDim** *arrayName***(***m***)**
- To keep any data that has already been stored in the array when resizing it, use
  - **ReDim Preserve** *arrayName***(***m***)**

Copying Arrays

- If *arrayOne*() and *arrayTwo*() have been declared with the same data type, then the statement
  - *arrayOne = arrayTwo*
- makes *arrayOne*() an exact duplicate of *arrayTwo*(). It will have the same size and contain the same information.

Using the Length Property

- The Length property of an array contains the number of elements in an array (1 less than upper bound)

```
Dim strBranchOffices(50) As String

Me.lblArraySize.Text = "The array size is " & (strBranchOffices.Length)
```

```
Dim intYear(99) As Integer
Dim intCount As Integer

' Assigns the years from 2001 to 2100 to the elements in the table
For intCount = 0 To (intYear.Length - 1)
    intYear(intCount) = 2001 + intCount
Next
```


The For Each Loop

General Format: For Each

```
For Each Control Variable Name in Array Name
    ' Lines of Code
Next
```

For Each — This type of loop iterates through an array until the array reaches the last element.

Control Variable Name — This variable will contain each individual element of the array without a subscript as the loop is processed. During the first iteration of the loop, the first element in the array is assigned to the control variable.

Array Name( ) — The name of the array that the loop cycles through. The array must be initialized first.

Next — This statement continues the loop to its next iteration.

Searching an Array

- Searching each element in an array is called a sequential search
- The BinarySearch method searches a sorted array for a value using a binary search algorithm
  - The binary search algorithm searches an array by repeatedly dividing the search interval in half

General Format: BinarySearch Procedure

```
intValue = Array.BinarySearch(arrayname, value)
```

If intValue returns a positive number or zero, a match was found at the subscript number equal to intValue.

If intValue returns a negative number, a match was not found

Sorting an Array

General Format: Sort Procedure
Array.Sort(ArrayName)
Coding Example: Dim intAges() as Integer = {16, 64, 41, 8, 19, 81, 23} Array.Sort(intAges)
After the sort executes, the values in the array are in the order 8, 16, 19, 23, 41, 64, and 81.

Passing an Array by Reference

- An array can be passed as an argument to a Sub procedure or a Function procedure

```
109     Private Sub btnCommission_Click(ByVal sender As System.Object, ByVal e As System.
        EventArgs) Handles btnCommission.Click
110
111         Dim decCommissionAmounts() As Decimal = {1345.99, 7800.16, _
112             5699.99, 3928.09, 1829.45}
113
114         ComputeDisplayTotal(decCommissionAmounts)
115
116     End Sub
117
118     Private Sub ComputeDisplayTotal(ByVal decValueOfCommission() As Decimal)
119
120         Dim decAmount As Decimal
121         Dim decTotal As Decimal = 0
122
123         For Each decAmount In decValueOfCommission
124             decTotal += decAmount
125         Next
126
127         Me.lblTotalCommission.Text = "The Total Commission is " _
128             & decTotal.ToString("C")
129     End Sub
```


Menu

Creating a Menu Bar

MenuStrip Object

- A menu bar is a strip across the top of a window that contains one or more menu names
- A menu is a group of commands, or items, presented in a list

Standard Items for a Menu

- Visual Basic 2008 contains an Action Tag that allows you to create a full standard menu bar commonly provided in Windows programs
- Action tags provide a way for you to specify a set of actions, called smart actions, for an object as you design a form
  - Drag the MenuStrip component onto the Windows Form object. Click the Action Tag on the MenuStrip object
  - Click Insert Standard Items on the MenuStrip Tasks menu
  - Click File on the menu bar to view the individual menu items and their associated icons on the File menu

File Handling

Read and Write Text File

File Handling

- To process data more efficiently, many developers use text files (or binary files) to store and access information for use within an application
- Text files have an extension that ends in .txt
- A simple text file is called a sequential file (actually it is usually called an ASCII text file)

Read Text File

- **Stream Reader** is designed for character input in a particular encoding, whereas the Stream class is designed for byte input and output.
- Use StreamReader for reading lines of information from a standard text file.
- StreamReader defaults to UTF-8 encoding unless specified otherwise, instead of defaulting to the ANSI code page for the current system.
- UTF-8 handles Unicode characters correctly and provides consistent results on localized versions of the operating system.

Handling End of File

- **ReadToEnd** works best when you need to read all the input from the current position to the end of the stream.
- If more control is needed over how many characters are read from the stream, use Read(Char[], Int32, Int32), which generally results in better performance.
- ReadToEnd assumes that the stream knows when it has reached an end. For interactive protocols, in which the server sends data only when you ask for it and does not close the connection, ReadToEnd might block indefinitely and should be avoided.
- Note than when using the Read method, it is more efficient to use a buffer that is the same size as the internal buffer of the stream. If the size of the buffer was unspecified when the stream was constructed, its default size is 4 kilobytes (4096 bytes).

Save Text File

- To add the ability to write to a file via the application, use the **Stream Writer** class.
- Stream Writer is designed for character output in a particular encoding, whereas the Stream class is designed for byte input and output.
- Use Stream Writer for writing lines of information to a standard text file.

Opening a Text File

- To open a text file, you need an object available in the **System.IO** namespace called a **StreamReader**

```
Dim objReader as IO.StreamReader

If IO.File.Exists("e:\file.txt") Then
   objReader = IO.File.OpenText("e:\file.txt")
Else
   MsgBox("Error! File not Exist.")
   Me.close()
End If
```


Reading a Text File

- To determine whether the end of the file has been reached, use the **Peek** procedure of the **StreamReader** object

```
Do While objReader.Peek <> -1
    _strInventoryItem(intCount) = objReader.ReadLine()
    _strItemId(intCount) = objReader.ReadLine()
    _decInitialPrice(intCount) = Convert.ToDecimal(objReader.ReadLine())
    _intQuantity(intCount) = Convert.ToInt32(objReader.ReadLine())
    intCount += 1
Loop
```

- Stream Reader should be close at the end

```
objReader.Close()
```

Writing to a Text File

- Writing to a text file is similar to reading a text file.
- The System.IO namespace also includes the StreamWriter class which is used to write a stream of text to a file

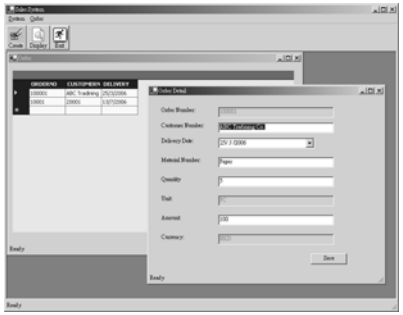
```
Private Sub btnCreateAccount_Click(ByVal sender As System.Object, ByVal e As Syst
.EventArgs) Handles btnCreateAccount.Click
    ' Initialize Variables
    Dim strCustomerName(5) As String
    Dim strPassword(5) As String
    Dim objWriter As New IO.StreamWriter("e:\NewAccounts.txt")
    Dim intCount As Integer

    For intCount = 0 To (strCustomerName.Length - 1)
        strCustomerName(intCount) = InputBox("Please enter your name:", "Login
Information")
        strPassword(intCount) = InputBox("Please enter a password:", "Password
Information")
        If IO.File.Exists("e:\NewAccounts.txt") Then
            ' Write the file line by line until the file is completed
            objWriter.WriteLine(strCustomerName(intCount))
            objWriter.WriteLine(strPassword(intCount))
        Else
            MessageBox.Show("The file is not available. Restart the program when
the file is available", "Error")
            Me.Close()
        End If
    Next

    ' The file is closed
    objWriter.Close()

End Sub
```


Multiple Forms

Multiple-Document Interface (MDI)

Multiple-Document Interface (MDI)

- Multiple-document interface (MDI) applications allow you to display multiple documents at the same time, with each document displayed in its own window.
- MDI applications often have a Window menu item with submenus for switching between windows or documents.

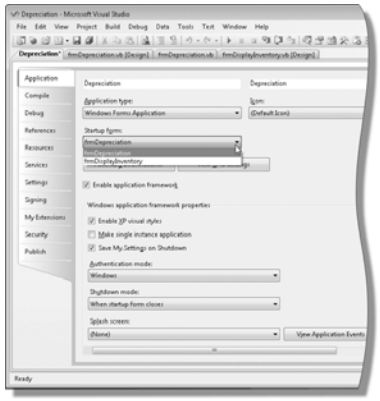


Creating an Instance of a Windows Form Object

- To display a second or subsequent form, the initial step in displaying the form is to create an instance of the Windows Form object
- When creating multiple Windows Forms objects, Visual Basic allows you to generate two types of forms:
  - A **Modal Form** retains the input focus while open **ShowDialog( )**
  - A **Modeless Form** allows you to switch the input focus to another window **Show( )**

Startup Objects

- Every application begins executing a project by displaying the object designated as the Startup object



Application Class

- If you have an application with multiple forms (windows), you can exit the "application" and close all the open forms (windows) by using the Exit method of the Application Class.
  - **Application.Exit( )**

The FormClosing Event

- Form Closing event:
  - Occurs when a form is about to be closed by the program code or by the user
  - Allows you to trap the closing action and take any necessary actions such as saving data
  - Can be used to cancel the close action
  - Set e.Cancel = True to cancel the closing action
- Form may be closed by Me.Close() statement or by user clicking the Close button on title bar

Example of Form Closing Event

USE THE FORMCLOSING EVENT PROCEDURE

Example 1—writes information to a sequential access file

```
Private Sub MainForm_FormClosing(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.FormClosingEventArgs) _
    Handles Me.FormClosing
    My.Computer.FileSystem.WriteAllText("date.txt", _
        dateLabel.Text, True)
End Sub
```

Example 2—verifies that the user wants to exit the application

```
Private Sub MainForm_FormClosing(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.FormClosingEventArgs) _
    Handles Me.FormClosing
    Dim button As DialogResult
    button = MessageBox.Show("Do you want to exit?", "Payroll", _
        MessageBoxButtons.YesNo, _
        MessageBoxIcon.Exclamation, _
        MessageBoxDefaultButton.Button2)
    If button = Windows.Forms.DialogResult.No Then
        ' stop the form from closing
        e.Cancel = True
    End If
End Sub
```

Message box

Interactive Dialog Box

The MessageBox.Show Method

- MessageBox.Show method:
  - Displays a message box with text, one or more buttons, and an icon
- When a message box is displayed, the program waits until the user selects a button
- MessageBox.Show returns an integer value indicating which button the user selected
- DialogResult values include:
  - Windows.Forms.DialogResult.Yes
  - Windows.Forms.DialogResult.No

How to use MessageBox.Show

USE THE MESSAGEBOX.SHOW METHOD

Syntax

MessageBox.Show(*text***,** *caption***,** *buttons***,** *icon***[,** *defaultButton***])**

Argument	Meaning
text	text to display in the message box; use sentence capitalization
caption	text (usually the application's name) to display in the title bar of the message box; use book title capitalization
buttons	buttons to display in the message box; can be one of the following constants: MessageBoxButtons.AbortRetryIgnore MessageBoxButtons.OK (default setting) MessageBoxButtons.OKCancel MessageBoxButtons.RetryCancel MessageBoxButtons.YesNo MessageBoxButtons.YesNoCancel
icon	icon to display in the message box; typically, one of the following constants: MessageBoxIcon.Exclamation MessageBoxIcon.Information MessageBoxIcon.Question MessageBoxIcon.Stop
defaultButton	button automatically selected when the user presses Enter; can be one of the following constants: MessageBoxDefaultButton.Button1 (default setting) MessageBoxDefaultButton.Button2 MessageBoxDefaultButton.Button3

Example 1

```
MessageBox.Show("Record deleted.", "Payroll", _
    MessageBoxButtons.OK, MessageBoxIcon.Information)
```

displays an informational message box that contains the message "Record deleted."

Example 2

```
MessageBox.Show("Delete this record?", "Payroll", _
    MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation, _
    MessageBoxDefaultButton.Button2)
```

displays a warning message box that contains the message "Delete this record?"

Style for Message Dialog

USE THE VALUE RETURNED BY THE MESSAGEBOX.SHOW METHOD

Number	DialogResult Value	Meaning
1	Windows.Forms.DialogResult.OK	user chose the OK button
2	Windows.Forms.DialogResult.Cancel	user chose the Cancel button
3	Windows.Forms.DialogResult.Abort	user chose the Abort button
4	Windows.Forms.DialogResult.Retry	user chose the Retry button
5	Windows.Forms.DialogResult.Ignore	user chose the Ignore button
6	Windows.Forms.DialogResult.Yes	user chose the Yes button
7	Windows.Forms.DialogResult.No	user chose the No button

Example 1

```
Dim button As DialogResult
button = MessageBox.Show("Delete this record?", _
    "Payroll", MessageBoxButtons.YesNo, _
    MessageBoxIcon.Exclamation, _
    MessageBoxDefaultButton.Button2)
If button = Windows.Forms.DialogResult.Yes Then
    instructions to delete the record
End If
```

Example 2

```
If MessageBox.Show("Play another game?", _
    "Math Monster", MessageBoxButtons.YesNo, _
    MessageBoxIcon.Exclamation) = _
    Windows.Forms.DialogResult.Yes Then
    instructions to start another game
Else ' No button
    instructions to close the game application
End If
```

Drawing

Bitmap, Graphic & Pen

Graphic Object

- The **Graphics** object provides methods for drawing a variety of lines and shapes.
- Simple or complex shapes can be rendered in solid or transparent colors, or using user-defined gradient or image textures.
- Lines, open curves, and outline shapes are created using a Pen object.
- To fill in an area, such as a rectangle or a closed curve, a Brush object is required.

Pen and Brush Object

- You use pen and brush objects to render graphics, text, and images with GDI+.
- A pen is an instance of the **Pen** class, and is used to draw lines and outlined shapes.
- A brush is an instance of any class that derives from the **MustInherit** (abstract) **Brush** class, and can be used to fill shapes or paint text.
- Color objects are instances of classes that represent a particular color, and can be used by pens and brushes to indicate the color of rendered graphics.
- A Pen object draws a line of specified width and style.
- Use the DashStyle property to draw several varieties of dashed lines.
- The line drawn by a Pen object can be filled in a variety of fill styles, including solid colors and textures.
- The fill style depends on brush or texture that is used as the fill object.

Exception Handling

Error Capture and Handling

Exception Handling

- The Try-Catch set of statements detects exceptions and takes corrective action

General Format: Try-Catch block

```
Try
        ' Try Block of Code – Executable statement(s) that may generate an exception.
Catch (filter for possible exceptions)
        ' Catch Block of Code for handling the exception
[Optional: Additional Catch blocks]
[Optional Finally]
        ' Optional statements that will always execute before finishing the Try block
End Try
```


Exception Type

Exception Type	Condition when Exception Occurs	Code Example
ArgumentNullException	A variable that has no value is passed to a procedure	`Dim strTerm As String` `Me.lstDisplay.Items.Add(strTerm)`
DivideByZeroException	A value is divided by zero	`intResult = intNum / 0`
FormatException	A variable is converted to another type that is not possible	`strTerm = "Code"` `intValue = Convert.ToInt32(strTerm)`
NullReferenceException	A procedure is called when the result is not possible	`Dim strTerm as String` `intValue = strTerm.Length`
OverflowException	A value exceeds its assigned data type	`Dim intCost as Integer` `intCost = 58 ^ 4000000000`
SystemException	Generic	`Catches all other exceptions`

Exception Handling Example

```
Dim decNumerator As Decimal
Dim decDenominator As Decimal
Dim decDivision As Decimal

decNumerator = Convert.ToDecimal(Me.txtNum.Text)
decDenominator = Convert.ToDecimal(Me.txtDen.Text)

Try
    decDivision = decNumerator / decDenominator
Catch Exception As DivideByZeroException
    MessageBox.Show("Attempt to divide by zero")
End Try
```