UNIT 4

Error

- Three categories:
  - Syntax errors
  - Run - time errors
  - Logic errors.

Syntax Error

- *Syntax errors* are those that appear while you write code.
- Visual Basic checks your code as you type it in the Code Editor window and alerts you if you make a mistake, such as misspelling a word or using a language element improperly.
- Syntax errors are the most common type of errors.
- You can fix them easily in the coding environment as soon as they occur.

Run-Time Errors

- *Run-time errors* are those that appear only after you compile and run your code.
- These involve code that may appear to be correct in that it has no syntax errors, but that will not execute.
- For example, you might correctly write line of code to convert textbox value to integer, but textbox value is blank then error will occur.
  - a = Convert.ToInt32(TextBox1.Text)

Logical Errors

- *Logic errors* are those that appear once the application is in use.
- They are most often unwanted or unexpected results in response to user actions.
- Logic errors are generally the hardest type to fix, since it is not always clear where they originate.

Exception

- **Exceptions** : It is a runtime errors that occur when a program is running and causes the program to abort without execution.
- Such kind of situations can be handled using Exception Handling.
- **Exception Handling :**
  - By placing specific lines of code in the application we can handle most of the errors that we may encounter and we can enable the application to continue running.

- VB .NET supports two ways to handle exceptions,
  - Unstructured exception Handling
    - using the ***On Error goto*** statement
  - Structured exception handling
    - using ***Try….Catch…..Finally***
-

Unstructured Exception Handling

- The **On Error GoTo** statement enables exception handling and specifies the location of the exception-handling code within a procedure.
- How the **On Error GoTo** statement works:

On Error GoTo [*lable* | 0 | -1] | Resume Next

Code

Statement	Meaning
On Error GoTo -1	Resets **Err** object to Nothing, disabling error handling in the routine
On Error GoTo 0	Resets last exception-handler location to **Nothing**, disabling the exception.
On Error GoTo <labelname>	Sets the specified label as the location of the exception handler
On Error Resume Next	Specifies that when an exception occurs, execution skips over the statement that caused the problem and goes to the statement immediately following. Execution continues from that point.

Structured Exception Handling

- **On Error Goto** method of exception handling sets the internal exception handler in Visual Basic.
- It certainly doesn't add any structure to your code,
- If your code extends over procedures and blocks, it can be hard to figure out what exception handler is working when.

- Structured exception handling is based on a particular statement, the **Try…Catch…Finally** statement, which is divided into a
  - **Try** block,
  - optional **Catch** blocks,
  - and an optional **Finally** block.

Syntax

```
Try
    //code where exception occurs
Catch e as Exception
     // handle exception
Finally
     // final Statement
End Try
```

- The **Try** block contains code where exceptions can occur.
- The **Catch** block contains code to handle the exceptions that occur.
- If an exception occurs in the **Try** block, the code *throws* the exception.
- It can be caught and handled by the appropriate **Catch** statement.

- After the rest of the statement finishes, execution is always passed to the **Finally** block, if there is one.
- Code

Exception Object

- The **Exception object** provides information about any encountered exception.
- properties of the **Exception** object:
  - **HelpLink :** can hold an URL that points the user to further information about the exception.
  - **InnerException** : It returns an exception object representing an exception that was already in the process of being handled when the current exception was thrown.

- **Message** : It holds a string, which is the text message that informs the user of the nature of the error and the best way or ways to address it.
- **StackTrace :** It holds a stack trace, which you can use to determine where in the code the error occurred.

Standard Exception

Exception Type	Description
Exception	Base type for all exceptions
IndexOutOfRangeException	Thrown when you try to access an array index improperly
NullReferenceException	Thrown when you try to access a null reference
InvalidOperationException	Thrown when a class is in an invalid state
ArgumentException	Thrown when you pass an invalid argument to a method
ArithmeticException	Thrown for general arithmetic errors

Collection

- Visual Basic .NET Collections are data structures that holds data in different ways for flexible operations . The important datastructres in the Collections are
  - ArrayList
  - HashTable
  - Stack
  - Queue

ArrayList

- **ArrayList** is one of the most flexible data structure from VB.NET Collections.
- ArrayList contains a simple list of values and very easily we can add , insert , delete , view etc.. to do with ArrayList.
- It is very flexible because we can add without any size information , that is it grow dynamically and also shrink .
- Syntax :
  - Dim arralist_name as new Arraylist()
  - E.g. Dim ItemList As New ArrayList()

Important functions in ArrayList

- Add : Add an Item in an ArrayList
- Insert : Insert an Item in a specified position in an ArrayList
- Remove : Remove an Item from ArrayList
- RemoveAt: remove an item from a specified position
- Sort : Sort Items in an ArrayList

Add item in ArrayList

- Syntax : ArrayList.add(Item)
  - Item : The Item to be add the ArrayList

  Example
  - Dim ItemList As New ArrayList()
  - ItemList.Add("Item4")

Insert item in ArrayList

- Syntax : ArrayListName.insert(index,item)
  - Index : The position of the item in an ArrayList
  - Item : The Item to be add the ArrayList
- Example
- ItemList.Insert(3, "item6")

Remove item from ArrayList

- Syntax : ArrayList.Remove(item)
  - Item : The Item to be add the ArrayList
- Example
- ItemList.Remove("item2")
- Syntax : ArrayList.RemoveAt(index)
  - index : the position of an item to remove from an ArrayList
- ItemList.RemoveAt(2)

Sort ArrayList

- Syntax : ArrayList.Sort()
- Example
- Itemlist.Sort()

Difference

VALUE TYPE	REFERENCE TYPE
Value type they are stored on stack	Reference type they are stored on heap
When passed as value type new copy is created and passed so changes to variable does not get reflected back	When passed as Reference type then reference of that variable is passed so changes to variable does get reflected back
Value type store real data	Reference type store reference to the data.
Value types are faster in access	Reference types are slower in access.
Value type consists of primitive data types, structures, enumerations.	Reference type consists of class, array, interface, delegates
Value types derive from System.ValueType	Reference types derive from System.Object
Value types can not contain the value null.	Reference types can contain the value null.