

UNIT-I

Introduction of DOT NET

- Dot NET is developed by Microsoft Company and announced about it in July 2000 at Orlando, Florida.
- Dot NET comes with different versions of Microsoft Visual Studio as 2003,2005,2008,2010,2012,2019 etc.
- Dot NET is basically invented for next generation platform for Windows and Internet software development.
- Dot NET is not a language but is a software technology to build computer applications.
- Dot NET is a service provider that requires to develop any type of application.
- Dot NET support multiple language integrity. It means we can use more than 30 programming languages under Dot NET such as VB.NET, C#.NET, J#.NET, F#.NET etc.
- Applications developed under Dot NET are partial platform independent. It means Dot NET application can execute on any architecture of hardware machine but necessarily required Microsoft Operating System XP or later versions on which Dot NET framework is installed.
- Dot NET applications executes faster because of base class libraries which are available in Dot NET framework.
- Using dot net we can developed following four type of applications.
 1. Console Application
 2. Windows Application
 3. Internet Application
 4. Mobile Applications

Dot NET Framework Features:

Dot NET framework is just like an engine of a train. Any application which are developed, debug, compiled, executes and deployed are possible only due to presence of Dot NET Framework.

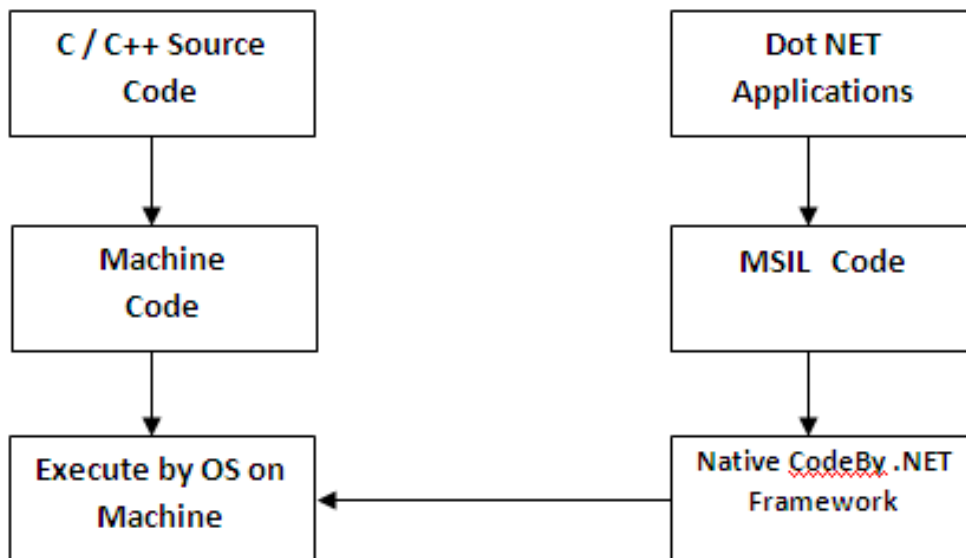
First of all Dot NET applications that developed using any Dot NET language like VB.NET compiled into MSIL code (Microsoft Intermediate language code) then such MSIL code transferred to Dot NET Framework. Dot NET Framework

perform many operations and finally converts demanded MSIL code into Native code (machine code) and handover to Operating System to executes on machine.

Dot NET framework is like a mini operating system for Dot NET applications.

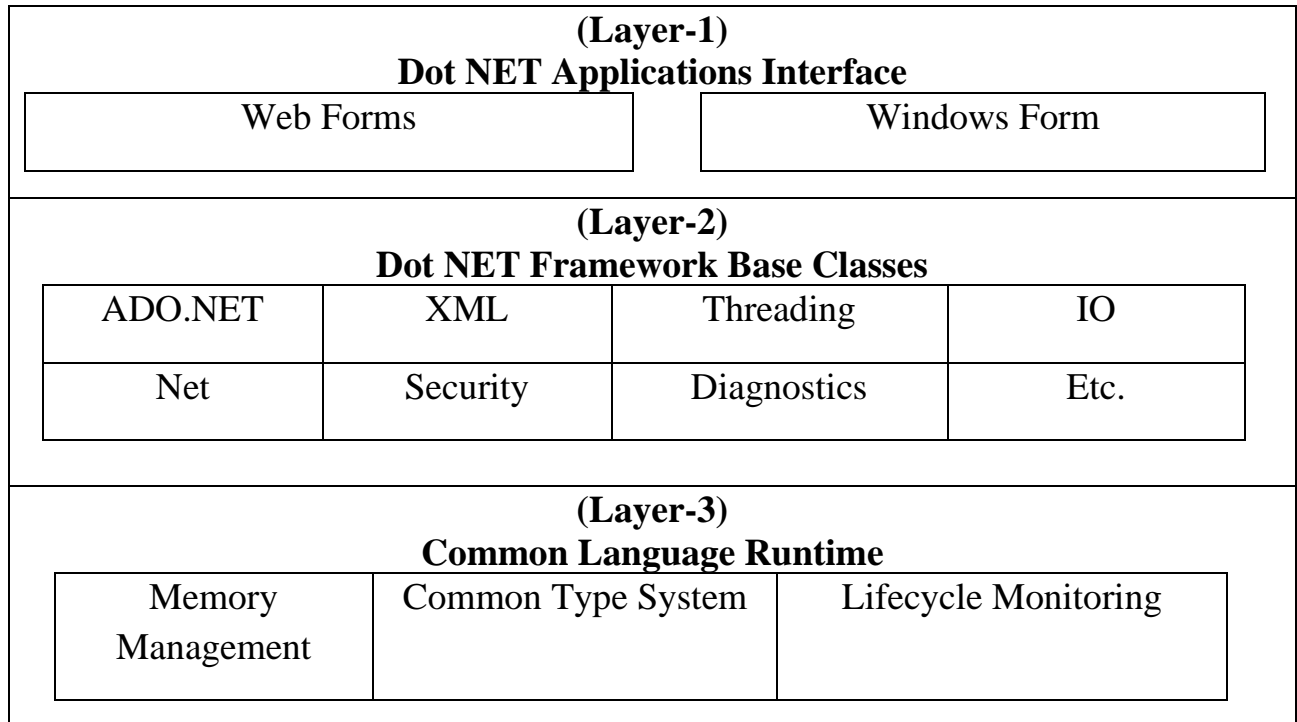
Traditional languages v/s Dot NET languages:

Applications developed under traditional programming languages like C, C++ are platform dependent because their compiler converts source code into machine code for targeted machine. On different type of machine application never executes but applications developed under Dot NET will be executes on all type of machine due to Dot NET Framework.



Dot NET Framework Architecture:

Whole architecture of Dot NET Framework is divided into three layers.



Layer-1: This layer accept MSIL code of Dot NET applications from out source that were developed using Dot NET tools they may be web forms or windows form. This layer provide interface for such application.

Layer-2: This layer is consist of thousands of base classes. Dot NET applications that uses pre define classes will be proved by base class library before execution.

Layer-3: This layer converts MSIL code into Native code (machine code) using JIT (Just in time) compiler. It also performs all the task that done by any operating system for dot net application. Due to this layer, it is possible of multiple language integrity. For all Dot Net language, this layer provides a common run time.

CLR (Common Language Runtime)

A runtime is an environment in which programs are executed. The CLR is therefore the environment in which we run our Dot NET applications that have been compiled to a common language, namely MSIL, often referred to simply as IL. CLR can be represented as.

Common Type System (Data Types, etc.)		
IL to native code compilers (JIT Compiler)	Execution Support	Security
Garbage collection, stack walk, code manager		
Class loader and memory layout		

The CLR is responsible for managing the execution of code compiled for .NET platform. Code requiring the CLR at runtime in order to execute is referred to as “managed code”. Compilers that target the .NET platform generate managed code that relies on a core set of services provided by the CLR.

Common Type System

The most important features Multiple language integrity of dot NET is possible due to Common type system of CLR. In which all commonly used data types, even base types such as *longs* and *Booleans* are actually implemented as objects. Since all languages are using the same library of types, calling one language from another does not require type conversion.

This result in the need for some readjustment, particularly for Visual Basic developers. For Example, what we called an *Integer* in VB6 is now known as a *Short* in Visual Basic.NET.

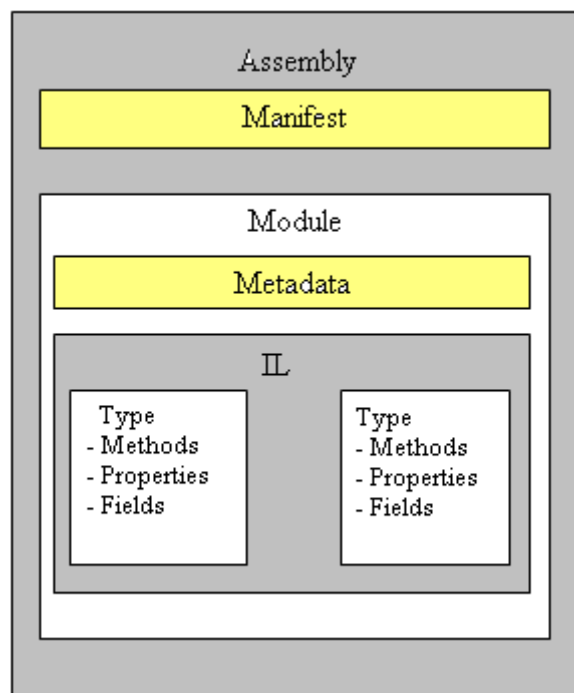
MSIL

The full form of MSIL is “Microsoft Intermediate Language”. This is 16 bits code designed by Microsoft to hide actual source code of program. This code some time also called IL code. Compilers of all Dot NET languages converts their source code into MSIL code. This codes are transfer machine to machine.

When Dot NET application executes then MSIL codes are processed by Dot NET framework and converts into Native code. When MSIL code convert into machine code at targeted machine then such machine code called Native code.

Assemblies and class libraries

An assembly is the primary unit of deployment for managed code. An assembly is composed of a manifest and one or more modules. The manifest can be stored in a separate file or in one of the modules. The manifest contains information about the identity of the assembly, a declarative security request, a list of other assemblies it depends on and a list of all exposed types and resources.



The identity information stored in the manifest includes its textual name and version number. If the assembly is public, the manifest will also contain the assembly's public key. The public key is used to guarantee uniqueness and may also be used to identify the source of the assembly.

The assembly is responsible for declaring the security it requires. Requests for permission fall into one of three categories: required, optional and denied. The identity information may be used as evidence for determining whether or not to approve the security requests.

The manifest contain a list of other assemblies it depends on and all types and resources exposed by the assembly. The manifest also contains a list of other

assemblies depends on. The CLR uses this information to locate an appropriate version of the required assemblies at runtime. The list of dependencies also includes the exact version number of each assembly at the time the assembly was created.

Module: A module is either a DLL or an EXE. It contains IL, associated metadata and may optionally contain assembly's manifest.

Type: A type is a template used to describe the encapsulation of data and associated set of behaviors. A type has properties, methods and fields.

Class Libraries:

Middle layer of Dot NET Framework consists of thousands of base classes/class libraries. Dot NET applications have a number of object declarations of predefined classes. All these classes are defined in the base class libraries of dot Net Framework. When application loaded then all required base classes will be loaded into memory from these class libraries. All classes are organized using *Namespaces*. One Namespace contains one or more namespaces and classes. They are used as

Ex:

Namespace- System.Data

Classes- DataSet, DataTable, DataColumn etc.

Introduction to Visual Studio

Visual Studio is a complete package designed by Microsoft to develop, debug, execute, build and deploy Dot NET applications. Initially Microsoft designed Visual Studio 6.0. Later it was modified, reconfigured and launched called Microsoft Visual Studio.NET. It comes with many versions such as Microsoft Visual Studio 2003, MSVS2005, MSVS2008, MSVS2010, MSVS2012 etc. We can use any one of them.

Visual Studio provides IDE (Integrated Development Environment) to develop software. It comes with compilers of many languages such as VB.NET, C#.NET, J#.NET and F#.NET. We can choose any language for development of new software.

After choosing language we can choose IDE for any type of project like web, windows, console etc.

Project basics & Type of project in .NET:

In Visual Studio every new development is considered as a project. Because one project is consist using number of files such as Form designer file, code file and other supporting files.

In Dot NET we can develop different type of projects.

1. Console based
2. Windows based
3. Web based

Console based: In such project user interfaces are text based (CUI- Character User Interface). User communicates with application using only keyboard and monitor. To execute application needs supports of operating system and Dot NET Framework on every machine.

Window based: In such project user interfaces are graphical based (GUI- Graphical User Interface). User communicates with application using mouse, keyboard, monitor and other pointer devices. To execute application needs supports of operating system and Dot NET Framework on every machine.

WEB based: In such project user interfaces are internet based. User need to work with application it is necessarily required web browser and web server(IIS). Web application executes at web server by Dot NET Framework and executed outputs are in the form of HTML, CSS and JavaScript. These codes are respondent to requested web browser(Internet Explorer). Web browser executes HTML, CSS and JavaScript code and provides GUI to the user.

IDE of VB.NET

VB.NET is one programming language of Dot NET. Using VB.NET we can write programming code for any type of applications like console, windows or web based. IDE means Integrated Development Environment in which all the tools which are available at one place necessary required to develop any

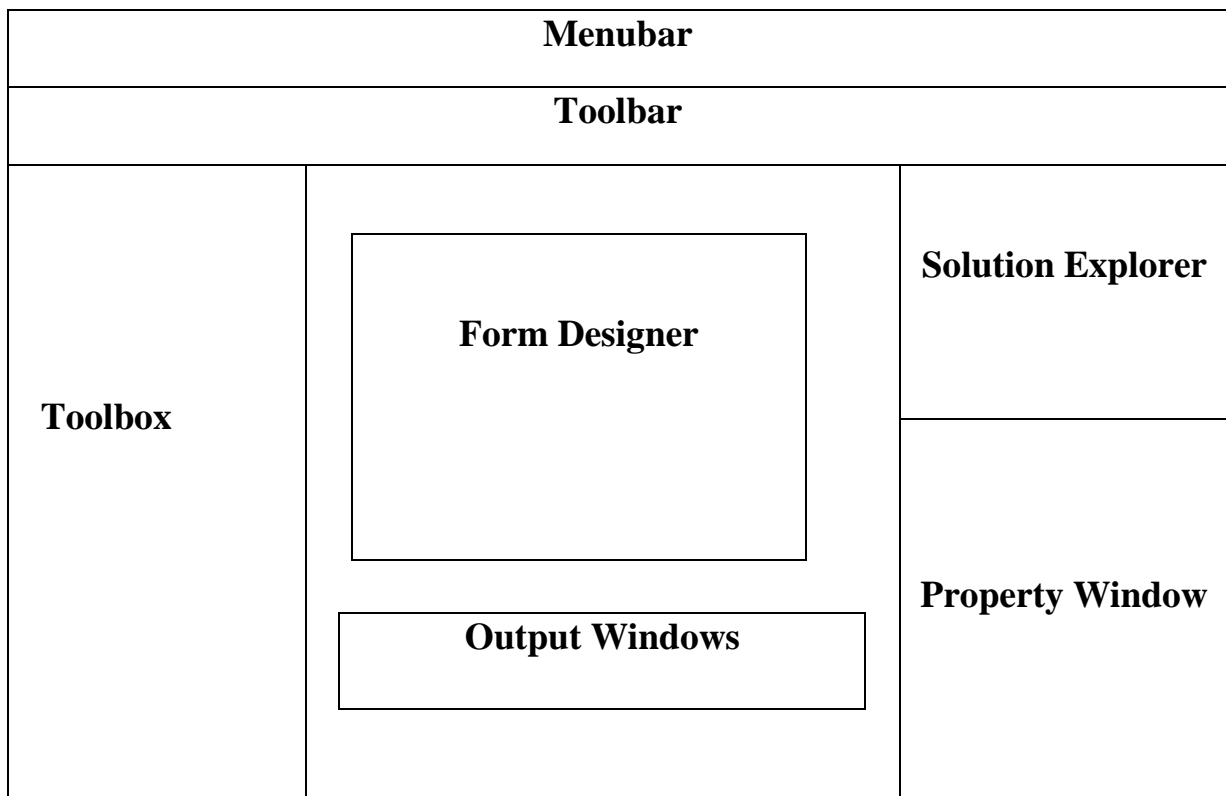
application like design forms using controls, writing code, adding new forms, switching forms, changing default properties, debug, executed, build application and deploy.

To open IDE of VB.NET we have to follow these steps.

Start→Microsoft Visual Studio 2005 or later versions→Choose VB.NET Language→Choose Windows Application→OK. We Get IDE of VB.NET that consist using following tools-

Menubar, Toolbar, Solution Explorer, Toolbox, Property windows, Form designer, Output Windows, Object browser and others.

Block Diagram of VB.NET: The default place for all tools are situated as follow-



1. Menubar

It is situated at the top of VB.NET IDE. This bar contains all the commands in menu form that requires to develop an application. For Example- File, Edit, View, Debug etc. Using menubar application can be saved, open, close, cut, copy, paste, search, execute, build, add and remove different tools.

2. *Toolbar*

This bar situated at the top of VB.NET IDE and just below the menubar. It contains all the recent usable commands in iconic form.

3. *Solution Explorer*

It is situated at the right side of VB.NET IDE. It is a small window that shows all items in file form that were added in to current working project. Using this window we can open form designer, code behind, open any form. We can set start up form and build application in executable format.

4. *Toolbox*

It is situated at the left side of VB.NET IDE. It is a small window that contain all the controls in iconic forms which are used to design any window form such as Label, Textbox, Button, Checkbox, RadioButton, Calendar control, dialog boxes and many mores.

5. *Property windows*

It is a small window that situated at right side and just below the solution explorer of VB.NET IDE. When we select any controls on form or item then it shows their properties along with default values. We can set or change properties of any selected controls using this window. For example- background, font, for color etc.

6. *Form designer*

When we add a new form in our project then we can show that form in the middle part of VB.NET IDE. Form designer of any form has two sections. Form design and code behind. Form design is a window that is design using controls of toolbox and to write code for form design we use code behind window in which we can write code in VB.NET language.

7. *Output Windows*

By default this window appears when we execute application. This window shows all the processing activities at the time of execution. We can use this window to shows any output if required.

8. *Object browser.*

By default this window is not available in VB.NET IDE. To add this Window we can use toolbar or Menubar(View→Object browser). This window shows all the base classes which are available in Dot NET Framework.

Working process project in VB.NET

VB.NET

1. Open VB.NET. (start→MS Visual Studio 2005 or any→Choose VB language→choose console / window application→select folder using browser→give the name of project→OK). We get default one form (for window application) or one module code window (for console application).
2. Design form using controls resides in toolbox and write VB code in code behind. For console application write VB code in module.
3. We can add one or more form / modules from solution explorer.
4. To execute any form / module select project name in solution explorer then right click and select properties. When we open properties then we get start up object. Choose any required form / module.
5. To execute form / module press F5 key or click arrow button of toolbar.

UNIT-II

The VB.NET language:

Dot NET supports a number of programming languages, VB.NET is one of them.

The main features of VB.NET languages are-

- VB.NET is Object Oriented Programming language.
- Code of VB.NET is not case sensitive.
- Every line of code is terminated by new line.
- Syntax of VB.NET are very simple to use.
- Compiler of VB.NET converts source code into MSIL code.
- VB.NET programs are executed by Dot NET Framework.

Variables

A place holder of data is called variable. It is represented by a name.

Ex: area, circum, x, y

Declaring Variables

By default in VB.NET it is necessary to declare variable name before using that one. For this *Dim* keyword is used as a prefix with variable name.

Ex:

Dim area

Dim x

Data Types of variable:

We can assign any type of data to variable.

Ex:

Dim area

area=3.44

'number

VB.NET

```
area="very large"      'string
```

When variables are declared with data type then we can not assign another type of data.

Variable with data type is declared as

```
Dim var1 As DataType
```

Ex: *Dim area As Single*

```
area=3.44              'No Error
```

```
area="very large"     'Error because it is string
```

VB.NET provides a list of data types. Some of them are following-

Type	Storage Size	Example	Default Value
<i>Byte</i>	<i>1 Byte (unsigned)</i>	<i>Dim count as Byte</i>	<i>0</i>
<i>Short</i>	<i>2 Bytes</i>	<i>Dim count as Short</i>	<i>0</i>
<i>Integer</i>	<i>4 Bytes</i>	<i>Dim count as Integer</i>	<i>0</i>
<i>Long</i>	<i>8 Bytes</i>	<i>Dim count as Long</i>	<i>0</i>
<i>Decimal</i>	<i>16Bytes</i>	<i>Dim count as Decimal</i>	<i>0</i>
<i>Single</i>	<i>4 Bytes</i>	<i>Dim rates as Single</i>	<i>0</i>
<i>Double</i>	<i>8 Bytes</i>	<i>Dim rates as Double</i>	<i>0</i>
<i>Char</i>	<i>2 Byte</i>	<i>Dim ch as char</i>	<i>Binary 0</i>
<i>String</i>	<i>-</i>	<i>Dim name as String</i>	<i>Nothing</i>
<i>Date</i>	<i>8 Byte</i>	<i>Dim adm_date as Date</i>	<i>--</i>
<i>Boolean</i>	<i>2 Bytes (True/False)</i>	<i>Dim flag as Boolean</i>	<i>False</i>
<i>Object</i>	<i>4 Bytes</i>	<i>Dim tx as Object</i>	<i>Nothing</i>

Type conversions

We can change type of variable in any expression using following type conversions.

<i>CByte(Expression)</i>	Convert any value in expression to Byte
<i>CShort(Expression)</i>	Convert any value in expression to Short
<i>CInt(Expression)</i>	Convert any value in expression to Integer
<i>CLng(Expression)</i>	Convert any value in expression to Long
<i>CDec(Expression)</i>	Convert any value in expression to Decimal
<i>CSng(Expression)</i>	Convert any value in expression to Single
<i>Cdbl(Expression)</i>	Convert any value in expression to Double
<i>CChar(Expression)</i>	Convert any value in expression to char
<i>CStr(Expression)</i>	Convert any value in expression to String
<i>Cdate(Expression)</i>	Convert any value in expression to date
<i>CBool(Expression)</i>	Convert any value in expression to Boolean
<i>CObj(Expression)</i>	Convert any value in expression to Object

Example:

```
Dim i as Integer, Dim j as Integer
```

```
Dim dev as Double
```

```
Dev=Cdbl(i) / j
```

User Defined Types

We can define own data types on using creating Structure.

Ex: *Public Structure Employee*

```
Public name as String
```

```
Public id as Integer
```

```
End Structure
```

This creates a new datatype.

Declaration : `dim emp1 as Employee`

Access : `emp1.name="Lokesh Rathore"`

Forcing variables declaration

We can force variable declaring before using them in code editor by adding following syntax at the top of code editor.

Option Explicit On

'area=3.44 It gives error because variable area is not declared

Dim area as single

area=3.44 ' It is correct

By default in VB.NET it is necessary to declare variable before using them. But if we want to use variable without declaration then we have to disable forcing variable declaration using following syntax

Option Explicit Off

Write this syntax at the top of code editor, after that compiler does not force to declare variable before using them.

Ex: `area=3.44`

If following syntax is also written at the top of code editor then compiler will force to declare variable along with data types.

Option Strict On

Option Explicit On

Dim area as single

area=3.44

Dim cirum 'error because declared without data type

By default *Option Strict Off*

Scope and Lifetime of variable

Scope tells which methods or module or class can use the value stored in a variable where as *Lifetime* of Variable tells how long a variable exist. In VB.NET variables are declared inside the methods and out side the methods within module or class.

Inside the method: Scope of the variable is only within that method where it is declared and life time of variable is until method is loaded into memory.

Outside the method: Scope of the variable is within all methods of module / class where it is declared and life time of variable is until module or class is loaded into memory.

Ex:

Module module1

Dim a as Integer

Sub Main()

Dim b as Integer

a=10

b=20

Console.WriteLine(a)

Console.WriteLine(b)

Show()

End Sub

Sub Show()

Console.WriteLine(a)

' Console.WriteLine(b) not accessible

End Sub

End Module

In this example variable *a* is declared outside all the methods therefore it is accessible to all method of current module where as variable *b* is declared inside the method *main* therefore it is accessible to only inside *main* method.

Constants

A value that never changed during the program execution called constant. All the values are called literal constant. If we want to make variable constant then use *const* keyword.

Syntax: *const Const_name As Type=Value*

Ex: *const PI As Single=3.14*

We can use 3.14 constant using PI in any expression. If we make any changes in PI after declaration then compiler returns an error.

PI=3.44 'error

Variables and constant are uses in program as following

```

Option Strict On           ' enable type checking
Option Explicit On         ' enable variable declaration
Module Module1
  Sub main()
    Dim r As Single
    Dim a, c As Double
    Const pi As Single = 22 / 7
    Console.WriteLine("Input radius of circle:")
    r = CSng(Console.ReadLine())
    a = pi * r * r
    c = 2 * pi * r
    Console.WriteLine("Area={0},Circumference={1}", a, c)
    Console.Read()
  End Sub
End Module

```

Output:

```

Input radius of circle:
2.5
Area=19.6428567171097,Circumference=15.7142853736877

```

Operators of VB.NET

Arithmetic	Relational	Logical
+ (addition)	< (Less than)	AND
- (subtraction)	<=(Less than or equal))	OR
* (Multipcation)	>(Greater than)	NOT
/ (Division)	>=(Greater than or equal)	
\ (Division)	= (equality)	
MOD (Remainder)	<>(Not Equal)	

Arrays

Collection of same data types called array. Variable can store only one data at a time but using array we can store any number of same type of data. Data in array are stores in continuous order and all represented by the same name. Data of the array are access using index vale (0,1,2,).

Types of array

In VB.NET array can be declared as following types

1. One Dimensional array
2. Two Dimensional array
3. Multi Dimensional array
4. Dynamic array

One Dimensional array

When a number of data are stores in one row then it is called one dimensional array. In VB.NET it is declared as following

Dim arr-name(size) As DataType

Ex: *Dim marks(5) As Integer*

We can assign 6 integer values to *marks* array from index number 0 to5.

Accessing array data as following-

Lower Bound = 0 and Upper Bound = Size

marks(0)=44

marks(1)=54

marks(2)=64

marks(3)=34

marks(4)=24

marks(5)=74

We can use loop from 0 to Size to access array data.

Program for one dimensional array

Module Module1

Sub main()

Dim arr(5) As Integer

Dim i As Integer

Console.WriteLine("Input 5 Numbers:")

arr(0) = 0 'for sum of numbers

For i = 1 To 5

arr(i) = CInt(Console.ReadLine())

arr(0) += arr(i)

Next

Console.WriteLine("First is sum of all")

For i = 0 To 5

Console.WriteLine(arr(i))

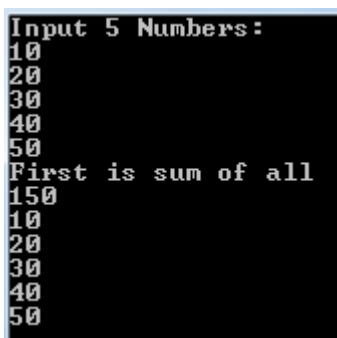
Next

Console.Read()

End Sub

End Module

Output:



```
Input 5 Numbers:
10
20
30
40
50
First is sum of all
150
10
20
30
40
50
```

Two Dimensional array

When a number of data are stores in tabular or matrix form (Row x Column) then it is called two dimensional array. In VB.NET it is declared as following

```
Dim arr(Rows, Columns) As DataType
```

Ex: *Dim table(2,2) As Integer*

```
table(1,2) = 10           'access second row, third column data
```

Program for two dimensional array

```
Module Module1
```

```
Sub Main()
```

```
Dim m(2, 3) As Integer           '3x4
```

```
Dim i, j As Integer
```

```
'Read matrix
```

```
For i = 0 To 2
```

```
Console.WriteLine("Input 4 data row wise")
```

```
For j = 0 To 3
```

```
m(i, j) = CInt(Console.ReadLine())
```

```
Next
```

```
Next
```

```
'print input matrix
```

```
For i = 0 To 2
```

```
Console.WriteLine("")
```

```
For j = 0 To 3
```

```
Console.Write(m(i, j) & " ")
```

```
Next
```

```
Next
```

```
Console.Read()
```

```
End Sub
```

```
End Module
```

Output:

```

Input 4 data row wise
10
20
30
40
Input 4 data row wise
11
22
33
44
Input 4 data row wise
55
66
77
88

10 20 30 40
11 22 33 44
55 66 77 88

```

Multi Dimensional array

When a number of data are stores in three or more dimensions then it is called multi dimensional array. In VB.NET it is declared as following

Ex: *Dim cube(3,3,3) As Integer*

Dynamic array

When array is declared with size then we can never resize array during runtime. But we can create array at run time as our need of size using *Redim* and *preserved* keyword. *Redim* keyword is used set size at runtime and *preserved* keyword used to resized array size after first use. They are declared as following.

Syntax *Dim darr() As DataType*

Redim darr(size)

Redim Preserved darr(expanded-size)

Program for dynamic array

Module Module1

Sub main()

Dim darr() As Integer

Console.WriteLine("Input maximum limit of data:")

Dim n As Integer = CInt(Console.ReadLine())

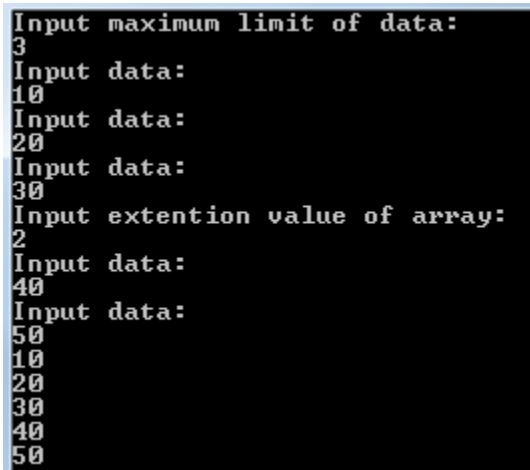
ReDim darr(n)

```
Dim i As Integer
For i = 1 To n
    Console.WriteLine("Input data:")
    darr(i) = CInt(Console.ReadLine())
Next

' To extend dynamic array with old data
Console.WriteLine("Input extention value of array:")
Dim m As Integer = CInt(Console.ReadLine())
ReDim Preserve darr(n + m)
For i = n + 1 To n + m
    Console.WriteLine("Input data:")
    darr(i) = CInt(Console.ReadLine())
Next
'print all data of dynamic array
For i = 1 To n + m
    Console.WriteLine(darr(i))
Next
Console.Read()
```

End Sub
End Module

Output:



```
Input maximum limit of data:
3
Input data:
10
Input data:
20
Input data:
30
Input extention value of array:
2
Input data:
40
Input data:
50
10
20
30
40
50
```

Control array & Collection

Control array means group of same type controls and having only one event handler function.

Ex: When we need 12 command buttons in our program then in place of writing 12 subroutines to handle click event we simply make control array and use only one event handler.

In Visual Basic 6.0 to make control array give same name for all controls but this is not possible in VB.NET because concept of control array does not exist that means to handle control array, there is no inbuilt functionality. So programmer own self create something very like control array in code. The main feature of control array is that all the controls in it share the same event handler, and so we can use the *AddHandler* method to assign the same event handler to multiple controls.

Collection: Collections means an object that contain a group of more then one objects and all belongs to only one class, such group of object is called collection.

Procedures (Methods)

A block of code that can be called by another block then such block called procedures.

In VB.NET Procedures are following types.

1. Subroutines
2. Functions
3. Properties

Subroutines

A block of codes that never return any value after to the calling procedure called subroutine.

In VB.NET subroutine is defined using *sub* keyword as-

```
Sub Subroutine-name()
```

```
Codes
```

```
End Sub
```

Example:

```
Sub Main()  
    show      'calling subroutine  
  
End Sub  
  
' Subroutine  
  
Sub show()  
    Console.WriteLine("Subroutine Called")  
  
End Sub
```

Output: Subroutine Called

Here *Main()* is a subroutine that calling another subroutine *Show()*

Function & Returning values from function

A block of codes that definitely returns only one value to the calling procedure called function. *Return* keyword is used to return value.

In VB.NET function is defined using *Function* keyword and value is return using *Return* keyword as-

```
Function Function-name() As Function-Type  
    Codes  
  
End Function
```

Example:

```
Sub Main()  
    Dim x as Integer  
    x=Fun()      'calling Function  
    Console.WriteLine(x)  
  
End Sub
```

```
' Function  
Function Fun() As Integer  
    Dim a as Integer  
    a=10  
    Return a  
End Function
```

Output: 10

Here *Main()* is a subroutine that calling Function *Fun()* and getting returning value.

Passing Variables & Number of arguments

We can pass any number of values to the subroutine and function directly or using variables. Passing values are called arguments. Arguments are passed to the subroutine or function by using following two techniques.

1. Call by values
2. Call by reference
 1. Call by value (*ByVal*): When passing arguments are received by subroutine / function using *ByVal* keyword then there will be no change in actual argument if we made in formal argument inside subroutine / functions.
 2. Call by reference (*ByRef*): When passing arguments are received by subroutine / function using *ByRef* keyword then there will be also change in actual argument if we made in formal argument inside subroutine / functions.

For Example-

Module Module11

```
Sub LG(ByVal a As Integer, ByRef b As Single)
    a = 10
    b = 22.5
End Sub
```

```
Sub main()
    Dim x As Integer
    Dim y As Single
    x = 20
    y = 11.5
    Console.WriteLine("Before calling procedure: x=" & x & " y=" & y)

    LG(x, y)      'Calling Subroutine by passing variables

    Console.WriteLine("After calling procedure: x=" & x & " y=" & y)

    Console.Read()
End Sub
End Module
```

Output:

```
Before calling procedure: x=20 y=11.5
After calling procedure: x=20 y=22.5
```

In this example, x and y are two variables of $main()$ subroutine. They are passed to another subroutine $LG()$. It specifies two formal arguments, a with *Byval* and b with *Byref*. Subroutine $LG()$ changed value of a and b . We observe that value of x changed but value of y does not change.

Optional arguments

When subroutine / function is defined with a fixed number of arguments with some data type then it is necessary that we have pass equal number of arguments and same data type.

For example, if any subroutine / function is defined with two arguments of integer type then we must be passed exactly two arguments of integers.

Some times we want to pass less number of arguments to the subroutine / function then it is possible using *Optional* keyword. We define arguments with *optional* keyword from right most argument towards left.

Example:

Module Module1

```
Function UseOpt(ByVal a As Integer, Optional ByVal b As Integer = 0,
               Optional ByVal c As Integer = 0) As
               Integer
Return a + b + c
```

End Function

Sub main()

```
Dim r As Integer
r = UseOpt(10, 20, 30) 'Pass 3 values
Console.WriteLine(r)

r = UseOpt(10, 20) 'Pass 2 values
Console.WriteLine(r)

r = UseOpt(10) 'Pass 1 values
Console.WriteLine(r)
Console.Read()
```

End Sub
End Module

Output:



In this example, we define *UseOpt()* function with three arguments. Two arguments are define with *optional* keyword with default values. When we call this function with less number of arguments then function uses their default value.

Control flow statement:

By default In VB.NET, flow of control moves from top to bottom line. There are following statements that can transfer flow of control anywhere in the program.

1. Conditional control statement
2. Looping control statement

Conditional control statement:

If we have more than one set of codes and we want to execute any one set of codes by given condition then we can use following conditional control statements.

1. *If – Else - End If*
2. *Select – Case – End Select*

If-Else-End If statement

Condition is given with *If* statement. When it is true then code of *If* block will be execute otherwise code of *Else* block. We can skip *Else* Block.

Common Syntax for *IF-Else-End If*

```
If condition Then  
[statements]  
Else If condition Then  
[statements]  
-  
-  
Else  
[statements]  
End If
```

This syntax can be used in following manner.

1. If we want to execute only one set of statement conditionally

Example: *If n < 0 Then n = -n*

2. If we want to execute one or more statements conditionally

Example:

```
If n < 0 Then  
    n = -n  
End If
```

3. If there are two set of statements and want to execute any one of them conditionally.

Example:

```
If n Mod 2 = 0 Then  
    Console.WriteLine("{0} is Even number", n)  
Else  
    Console.WriteLine("{0} is Odd number", n)  
End If
```

4. If there are a many set of statements and want to execute any one of them.

Example:

```
If p >= 60 Then  
    div = "First"  
ElseIf p >= 45 Then  
    div = "Second"  
ElseIf p >= 33 Then  
    div = "Third"  
Else  
    div = "Fail"  
End If
```

5. Nested If- Else

Examples:

```
If a > b Then
    If a > c Then
        Console.WriteLine("large:"&a)
    Else
        Console.WriteLine(("large:"&c)
    End If
Else
    If b > c Then
        Console.WriteLine(("large:"&b)
    Else
        Console.WriteLine(("large:"&c)
    End If
End If
```

Select....Case Statement

The Select Case statement executes one of several groups of statements depending on the value of an expression.

```
Select case Variable/Expression
    Case 1
        Statement
    Case 2
        Statement
    Case 3 to 8
        Statement
    Case Is>8
        Statement
    Case Else
        Statement
End Select
```

Example:

```
Select Case n
    Case 1
        Console.WriteLine(a + b)
    Case 2
        Console.WriteLine(a - b)
    Case 3
        Console.WriteLine(a * b)
```

```
Case 4
    Console.WriteLine(a / b)
Case 5
    Console.WriteLine(a \ b)
Case 6
    Console.WriteLine(a Mod b)
Case Else
    Console.WriteLine("Invalid selection")
End Select
```

Looping control statement

If we want to execute statements many times then use following looping control statements.

1. *For* loop
2. *While* loop

For Loop

For loop is popular and simplest loop. It executes loop statements for given number of loop iterations.

Syntax:

```
For index=start to end step n
    [statements]
Exit For 'to comes out loop before end
Next index
```

Here *step n* is used to increase / decrease by *n number*

Example1:

```
For i=1 to 5
    Console.Writeline(i)
Next
```

Output: 1 2 3 4 5

Example2:

```
For i=5 to 1 step -1
```

```
        Console.WriteLine(i)
    Next

Output: 5  4  3  2  1
```

While loop

While loop executes its statements until given conditions remains true.

Syntax:

```
        While condition
            [statements]
        End While
```

Example:

```
        While n<10
            Console.WriteLine(n)
            n=n+1
        End While
```

Msgbox & Inputbox

We can display and read any data using dialog box. Such dialog box appear at run time when *MsgBox* and *InputBox* statement executes.

MsgBox: This is used to display any message along with variable's value on dialogbox.

Syntax:

```
        MsgBox("Message")
```

InputBox: This is used to read any value in string form from keyboard that can be assign to variable using dialog box.

Syntax:

```
        Dim var as String=InputBox("Input any text")
```

Example:

Module Module1

Sub main()

Dim a As Integer

Dim val As String

val = InputBox("Input Integer Value")

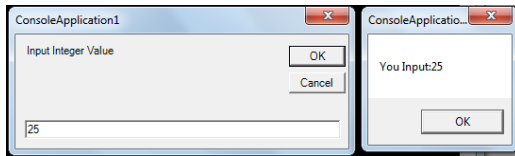
a = CInt(val)

MsgBox("You Input:" & a)

End Sub

End Module

Output:



UNIT-IV

Object Oriented Programming:

VB.NET is an object oriented programming language because this language exist all the features of OOPs. There are following-

1. Class & Object
2. Data abstraction & Data Encapsulation
3. Inheritance
4. Polymorphism

Classes & Objects

Class is the descriptions of object whereas object is instance of class. Class can be consider as user defined data type that consist using data members and methods whereas object is the variable of the class that can used data and methods of the class.

Syntax for class:

Class class-name

<data member>

<subroutines>

<functions>

<properties>

<event handlers>

End Class

Syntax for Object-

Dim obj-name As New class-name()

Or

Dim obj-name As class-name

Obj-name=New class-name()

Syntax for accessing members of class- (use . operator)

Obj-name.member

Example of class and object-

```
Class A
  Private x As Integer
  Public Sub SR(ByVal a As Integer)
    x = a
  End Sub

  Public Function Fun() As Integer
    Return x
  End Function
End Class

Sub main()
  Dim ob As A = New A()
  ob.SR(2)
  Dim var As Integer = ob.Fun()
  Console.WriteLine(var)
End Sub
```

Output: 2

In this program *A* is class have *x* is data member, *SR()* is subroutine and *Fun()* is function. *ob* is object and this object access class member using . (dot) operator.

Constructor & Destructor:

Constructor is a method of the class that automatically called when a new object is created into memory. In VB.NET constructor is defined using *New()* Keyword. We can pass arguments to constructor while object defined with class. It can be defined more than one times in a class but each one must be different on the basis of passing arguments to the construct (constructor overloading).

In contrast ***destructor*** is also a method of the class that also automatically when an existing object of the class is removed from memory. In VB.NET destructor

is defined using *finalized()* method. We can not passed arguments to the destructor and we can defined only one destructor in one class.

Example:

```

Class A
    Private a As Integer
    Private b As Integer
    Public Sub New()
        a = 10
        b = 20
    End Sub
    Public Sub New(ByVal x As Integer)
        a = x
        b = 20
    End Sub
    Public Sub New(ByVal x As Integer, ByVal y As Integer)
        a = x
        b = y
    End Sub
    Protected Overrides Sub finalize()
        Console.WriteLine("Object removed")
        Beep()
        Console.Read()
    End Sub

    Public Sub show()
        Console.WriteLine(a & "," & b)
    End Sub
End Class
Sub main()
    Dim ob1 As A = New A()
    Dim ob2 As A = New A(55)
    Dim ob3 As A = New A(11, 22)

    ob1.show()
    ob2.show()
    ob3.show()

    Console.Read()
End Sub

```

Output:

10, 20

55,20

11,22

Object removed

Object removed

Object removed

In this example, A is class that defined three constructors and one destructor. When object of class creates with no argument then constructor of no arguments will be called.

Inheritance:

An ability to acquire traits of base class into derived class called inheritance. In VB.NET inheritance is possible by *Inherits* keyword to access base class members in derived class.

Syntax:

*Class derived-class**Inherits base-class**End Class*

Example:

*Class B**Private x As Integer**Protected y As Integer**Public z As Integer**Public Sub SR1(ByVal a As Integer)**x = a**End Sub**Public Function Fun1() As Integer**Return x**End Function**End Class**Class D**Inherits B 'Creates Inheritance*

```

    Public Sub SR2(ByVal a As Integer)
        y = a
    End Sub

```

```

    Public Function Fun2() As Integer
        Return y
    End Function

```

```
End Class
```

```

Sub main()
    Dim od As D = New D()
    od.SR1(10)
    od.SR2(20)
    od.z=30

    Console.WriteLine( od.fun1() )
    Console.WriteLine( od.fun2() )
    Console.WriteLine( od.z )

```

```

    End Sub
End Module

```

Output:

```

10
20
30

```

In this example, *B* is base class that define three data members *x*, *y* and *z*. we can not directly access *Private x* in derived. We can access *protected y* into derived class and can be accessed in to any where in the program even out of base, derived class.

Constructor in case of inheritance –

When Base class & Derived class both have constructors, in that case first of all constructor of base class will be called after then constructor of derived class when object of derived class creates into memory.

Example:

Class B

```
Public Sub New()  
    Console.WriteLine("Base Class Constructor")  
End Sub  
End Class  
  
Class D  
    Inherits B  
    Public Sub New()  
        Console.WriteLine("Derived Class Constructor ")  
    End Sub  
End Class  
  
Sub main()  
    Dim od As D = New D()  
End Sub  
Output:  
Base Class Constructor  
Derived Class Constructor
```

Access Specifiers:

Access specifier means those keywords of VB.NET that define with members of the class and specify the nature of members that there will be accessible within same class where they are define or also accessible to outside that class.

There are following three important access specifiers defined VB.NET.

1. *Private*
2. *Protected*
3. *Public*

They are used with data and methods as following

```
Class B  
    Private x As Integer  
    Protected y As Integer  
    Public z As Integer  
    Public Sub SRI(ByVal a As Integer)  
        x = a  
    End Sub  
  
    Public Function Fun1() As Integer
```

Return x
End Function

End Class

1. *Private*: Members are accessible only inside the class where they are defined using *private* keyword. In this example, *Private x* is only accessible in to *Class B*.
2. *Protected*: Members are accessible to inside the class where they are defined using *protected* keyword as well as only inside the derived class of it. In this example, *Protected y* is accessible in to *Class B* and its derived class too.
3. *Public*: Members are accessible to inside the class where they are defined using *public* keyword, as well as inside to all its derived and any where in the program. In this example, *Public z* is also accessible outside the *Class B*.

Interfaces

Interface is one type of class in which methods are only declared but not defined with code. Methods of interface are defined with code inside the class that will implement the interface. It is one type of interface where interface consider as base and implemented class will considered derived.

A class that will implement interface, it is compulsory to define method declared in interface.

In VB.NET, interface is defined using *Interface* keyword and implement on class using *Implement* keyword.

Example:

```
Interface I  
    Sub show()    'method declared  
End Interface
```

```
Class A  
    Implements I
```

```

    Public Sub show() Implements I.show
        Console.WriteLine("Interface Method defined in class")
    End Sub
End Class

```

```

Sub main()
    Dim ob As New A
    ob.show()
End Sub

```

Output:

Interface Method defined in class

In this program *I* is an interface and it is implement in class *A* using *Implements* keyword. When we define method in class then it is compulsory to link with interface name as *Implements I.show*.

Polymorphism:

When there are more than one methods having the same name and it is possible to call all methods then it is called as polymorphism.

Polymorphism is classified into following two categories.

1. Compile time polymorphism
2. Runtime polymorphism

Compile time polymorphism (Method overloading): In such polymorphism, all methods having same name but they are differentiate by arguments. In VB.NET, if more than one subroutines or functions are defined in class or module and each one are different on the basis of arguments. When we call then a particular subroutine or function is identified by matching passing arguments and arguments specified in subroutine or function.

Example:

```

Module module1
    Sub show()
        Console.WriteLine("hello")
    End Sub
End Class
Sub show(ByVal a As Integer)

```



```
        Console.WriteLine("a")
    End Sub

    Sub main()
        Show()
        Show(10)
    End Sub
End Module
```

Output:

hello

10

In this example, subroutine show() is defined two times. One with one argument and another without argument. They will be link at compile time with calling subroutines. This is also called method overloading.

Runtime polymorphism: When more than one methods having same name as well as all methods are defined with similar arguments then linking of method calling will take place at run time. It is called runtime polymorphism.

In VB.NET, to make runtime polymorphism *Overridable* & *Overrides* keywords are used. *Overridable* keyword is used method of base class & *Overrides* keyword is used with method of derived class.

When object of base class call method then obviously method of base class will be called, but if object of derived class hold reference of object class then method of derived class will be called.

Example:

```
Public Class Base
    Public Overridable Sub show()
        Console.WriteLine("Base show Method")
    End Sub
End Class
```

```
Public Class Derived
    Inherits Base
    Public Overrides Sub show()
```

```

        Console.WriteLine("Derived show method")
    End Sub
End Class

Sub poly(ByVal ob As Base) ' gets base or derived object
    ob.show() 'call show method of base or derived
End Sub

Sub main()
    Dim ob As New Base
    Dim od As New Derived
    poly(ob) 'pass base object to base variable
    poly(od) 'pass derived object to base variable
    Console.Read()
End Sub

```

Output:

```

Base show Method
Derived show method

```

In this example, *show()* method is defined in base class using *Overridable* and also defined in derived class using *Overrides*. We also define a method *poly()* with object reference of base class and this method called *show()* method of base class or derived class.

When *poly()* method is called by passing object of base class then *show()* of base class will be called. If we pass object of derived class then *show()* method of derived class will be called. In this way, runtime polymorphism are managed.

MyClass v/s MyBase Keyword

MyClass: When method of base class is defined using *Overridable* keyword and same name of method is also defined in derived class using *Overrides* keyword. In this situation if object of derived class will be called method of derived class, but if want to call method of base class then we have to use *MyClass* keyword.

For Example:

```
Public Class Base
```

```

Public Overridable Sub show()
    Console.WriteLine("Base show Method")
End Sub

```

```

Public Sub callshow()
    show()
End Sub
Public Sub callshow1()
    MyClass.show()
End Sub

```

```
End Class
```

```

Public Class Derived
    Inherits Base
    Public Overrides Sub show()
        Console.WriteLine("Derived show method")
    End Sub

```

```
End Class
```

```

Sub main()
    Dim od As New Derived
    od.callshow() 'derived show
    od.callshow1() 'base show

```

```
End Sub
```

Output:

```

Derived show method
Base show Method

```

In this example, we have to defined *show()* methods in both class. We have also to defined a method *callshow()* in base class that will call *show()* method. When object of derived class call *callshow()* method then by default this method call *show()* method of derived class. But we want to call method of base class. In this situation we have to use **MyClass** keyword to call *show()* of base class i.e. *MyClass.show()* .

MyBase:

When base class and derived class both have constructors with arguments and we have to pass arguments to base class constructor from derived class constructor then we will use *MyBase* keyword in derived class constructor.

Example:

Class Base

Private a As Integer

Public Sub New(ByVal x As Integer)

a = x

Console.WriteLine("Base value{0}", a)

End Sub

End Class

Class Derived

Inherits Base

Private b As Integer

Public Sub New(ByVal x As Integer, ByVal y As Integer)

MyBase.New(x) 'passing argument to base class constructor

b = y

Console.WriteLine("Derived value{0}", b)

End Sub

End Class

Sub main()

Dim od As Derived = New Derived(10, 20)

End Sub

Output:

Base value10

Derived value20

In this example, we have to pass arguments to base class constructor from derived using *MyBase*.

Exception Handling: Using Try, Catch, Finally, Throw Keywords.

Exceptions are just runtime errors. Exceptions occur when a program is running (as opposed to syntax errors, which will prevent VB .NET from running your program at all). You can trap such exceptions and recover from them using Try, Catch, Finally, Throw Keywords rather than letting them bring your program to an inglorious end.

Using Try, Catch, Finally Keywords

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. 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—actually an object based on the Visual Basic Exception class—so 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.

Example of Exception Handling used Try, Catch and Finally keywords

```

Sub main()
    Try
        Dim a, b, c As Integer
        Console.WriteLine("Input two number:")
        a = CInt(Console.ReadLine())
        b = CInt(Console.ReadLine())
        c = a / b
        Console.WriteLine(c)
    Catch ex As Exception
        Console.WriteLine("{0}:Error", ex.Message)
    Finally
        Console.WriteLine("Execution code completed")
    End Try
    Console.Read()
End Sub

```

Output:

```

Input two number:
10
0
Arithmetic operation resulted in an overflow.:Error
Execution code completed

```

Throw Keywords

You can throw an exception using the Throw keyword to catch even exception made or not.

Example of Throw keyword

```

Sub main()
    Try
        Throw Err.GetException
    Catch ex As Exception
        Console.WriteLine("{0}:Error", ex.Message)
    End Try
    Console.Read()

```

End Sub

Output:

```
Object reference not set to an instance of an object.:Error
```

Graphics Handling: Using Graphics & Pen classes for drawing colors and figures.

The graphics handling in Visual Basic.NET is based on GDI+ (GDI stands for Graphics Device Interface). A graphics device interface such as GDI+ allows you to display graphics on a screen-or a printer-without having to handle the details of a specific display device.

In Visual Basic 2D graphics, the drawing origin (0, 0) is at the upper left of the drawing surface; the positive X axis extends to the right and the positive Y axis downward.

Graphics class

Graphics class gives us a drawing surface that we can work with instead. The actual methods we'll be using here, such as DrawRectangle, FillRectangle, DrawEllipse and FillEllipse, are all methods of the Graphics class.

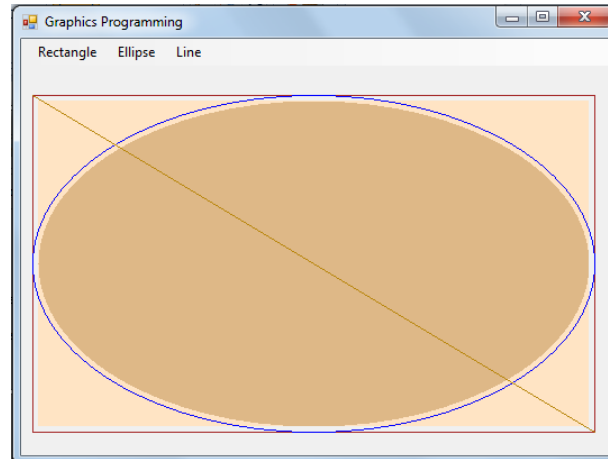
Pen class

GDI+ also supports a Pen class that specifies just how you draw figures. In particular, you can customize pens, specifying line color, line width, and line style. When you draw a figure-such as an ellipse-you must create and provide a Pen object that GDI+ will use to draw that ellipse.

Brush class

Similarly, GDI+ supports a Brush class that you can use to fill the figures you've drawn in, as when you want to fill a rectangle or polygon with color.

Example:



```

Private Sub EmptyRectangleToolStripMenuItem_Click()
    Dim g As Graphics = Me.CreateGraphics()
    Dim r As New Rectangle(10, 50, 500, 300)
    g.DrawRectangle(Pens.Brown, r)
End Sub

Private Sub FilledRectangleToolStripMenuItem_Click()
    Dim g As Graphics = Me.CreateGraphics()
    Dim f As New RectangleF(15, 55, 490, 290)
    g.FillRectangle(Brushes.Bisque, f)
End Sub

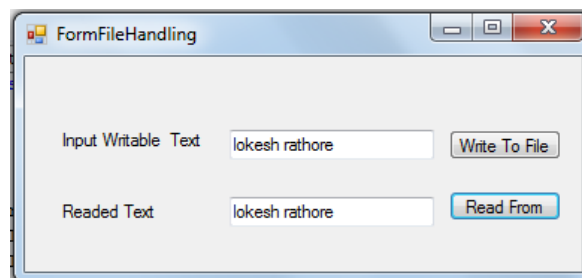
Private Sub EmptyEllipseToolStripMenuItem_Click()
    Dim g As Graphics = Me.CreateGraphics()
    Dim r As New Rectangle(10, 50, 500, 300)
    g.DrawEllipse(Pens.Blue, r)
End Sub

Private Sub FilledEllipseToolStripMenuItem_Click()
    Dim g As Graphics = Me.CreateGraphics()
    Dim f As New RectangleF(15, 55, 490, 290)
    g.FillEllipse(Brushes.BurlyWood, f)
End Sub

Private Sub LineToolStripMenuItem_Click()
    Dim g As Graphics = Me.CreateGraphics()
    g.DrawLine(Pens.DarkGoldenrod, 10, 50, 510, 350)
End Sub

```

File Handling: Opening or creating file, Writing & Reading text



```
Imports System.io
```

VB.NET

```
Public Class Form01FileHandling
Private Sub ButtonWrite_Click()
    Dim fw As FileStream = New FileStream("c:\xyz.txt", FileMode.Create,
                                        FileAccess.Write)

    Dim w As New StreamWriter(fw)
    w.WriteLine(TextBox1.Text)
    w.Flush()
    w.Close()
End Sub
Private Sub ButtonRead_Click()
    Dim fr As FileStream = New FileStream("c:\xyz.txt", FileMode.Open,
                                        FileAccess.Read
                                        )

    Dim r As New StreamReader(fr)
    TextBox2.Text = r.ReadToEnd()
    r.Close()
End Sub
End Class
```


UNIT-3

Window form controls-

1. **Textbox:** Using this control we can input simple text or password characters.
Important Properties: Text, ReadOnly, PasswordChar, Multiline, MaxLength, TabIndex
Important Event: Text_Changed
Accessing data in code: ***TextBox1.Text***
2. **Rich Textbox:** This control is used to input text in one or more lines and paragraphs.
Important Properties: Text, ScrollBar, Maxlength, RaedOnly, TabIndex
Important Event: Text_Changed
Accessing data in code: ***RichTextBox1.Text***
3. **Label:** This control is used to display ant text on the form.
Important Properties: Text, Autosize, BorderStyle, TextAlign
Accessing data in code: ***Label1.Text***
4. **Link Label:** This control provides a facility to set hyperlink on given label so that we can open any web page on internet by click on this control with control key.
Important Properties: Text, Autosize, BorderS
Accessing data in code: ***RichTextBox1.Text***
5. **Button:** If want to execute code to perform any operation then code are written inside the click event of button contol.
6. **Checkbox:** If we have many options and can select one or more of them then in that case checkboxes are used.
7. **Radio Button:** If we have many options but select any one of them then in that case radio buttons are used.
8. **Panel:** This control provides a frame without any label. This is used to place controls inside them. One form can be sub divided into many parts using this control.
9. **GroupBox:** This control provides a frame with a label name and it can be used to place a set of vb controls like radio buttons and checkbox of same group so that we can choose any one options among many radio button controls.

- 10. Picture Box / Image:** Using this control we can display any picture / image on form and set its width and height.
- 11. Listbox:** This control can contain a number of items in list format along with scrolls. Through this we can select one or more items.
- 12. Combo box:** This control can contain a list of items but only one will be shown on front. If arrow button is press of this control then a complete list will be appear along with scroll bars. We can select any one item from combo box.
- 13. Check list box:** It contain a list of items along with check boxes so that we can select or unselect items from list for further operations.
- 14. Scrollbar:** If any contents is not fit into given frame then using scroll bar we can view whole part of contents.
- 15. Timer:** Using this control we can execute a set of code after given time interval that set using timer control.
- 16. Menu:** A list of commands can be shown on the top of form. It creates a menu and sub menu list. We can write code behind the click event of every menu commands.
- 17. Context menu:** A number of commands can be display on form when user **right click** of mouse button. We can also write code behind of every command's click event.
- 18. List View:** Using this control we can display items in list format.
- 19. Tree view:** Using this control we can display a number of items in tree like format.
- 20. Toolbar:** It is situated at the top of the form. It is used to shows a number of operation list in iconic form that has to be most recent usable in application.
- 21. Status bar:** It is situated at the bottom of form when added from toolbox. It can be used to show current status of the running form.

Dialog boxes

Dialog box is a small window that contains all necessary options to perform specific task. User need not to design them. In VB.NET there are following dialog boxes can be used to perform various operations such as browsing file system to select files, choosing colors and printing.

1. Open Dialog box
2. Save dialog box

3. Color dialog box
4. Font dialog box
5. Print dialog box

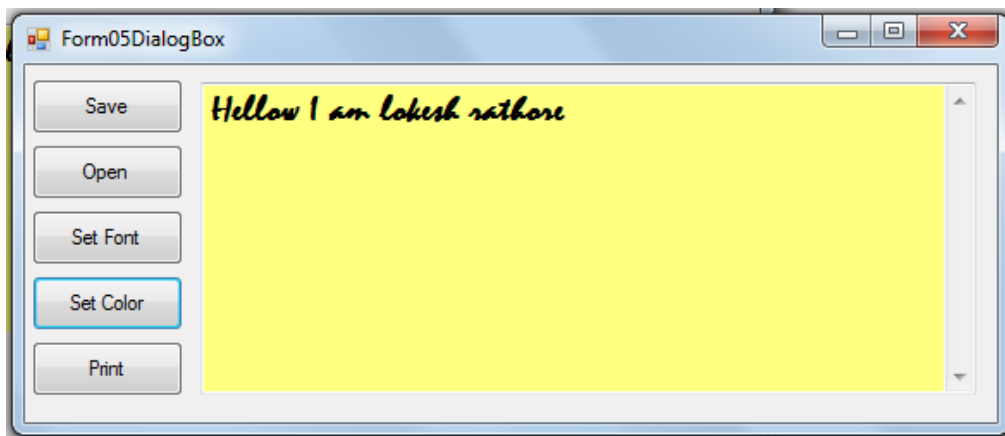
All dialog boxes have following common method to show dialog box at run time.

ShowDialog(): If it is true then dialog box will appear on request any command.

When any dialog box place on Window form then it will placed on bottom tray and apply on window form using code.

Working with dialog box:

Step1: Design a form as following to save, open file, set text font, color and print.

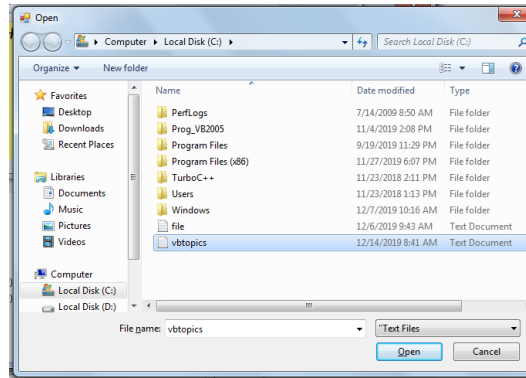


Step2: Using toolbox place all dialog boxes on this window form.

Step3: Double click on each button and write required dialog box code to open at run time.

1. **OpenDialogBox:** Using this dialog box we can select any required file using browser. After selecting file this dialog box returns path of the file with filename when we click on **Open** button of dialog box.

It look like this-



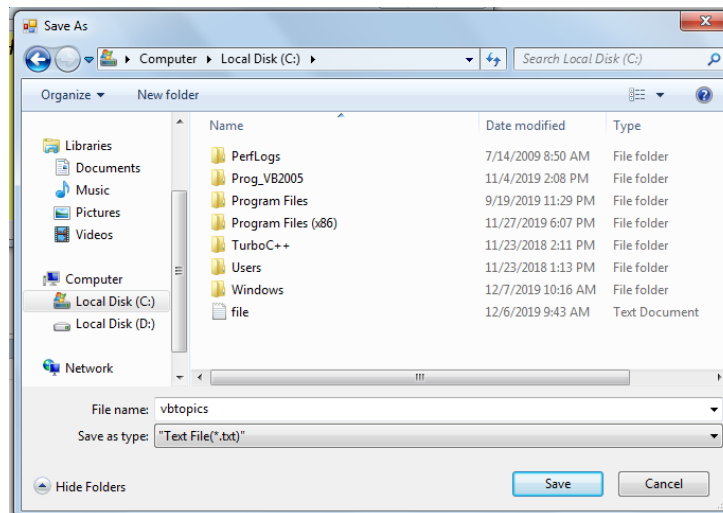
This dialog box appear when we write following code on button click.

```
If OpenFileDialog1.ShowDialog() <> Windows.Forms.DialogResult.Cancel Then
    Dim fr As FileStream = New FileStream(OpenFileDialog1.FileName,
    FileMode.Open, FileAccess.Read)
    Dim r As New StreamReader(fr)
    TextBox1.Text = r.ReadToEnd()
    r.Close()
End If
```

These codes shows open dialog box and read content of file into textbox from browsing location choose from this dialog box.

2. SaveDialogBox: Using this dialog box we can select a location where we want to save file with the help of browser. When we click on Save button of dialog box then it returns path of file with new file name.

It look like-



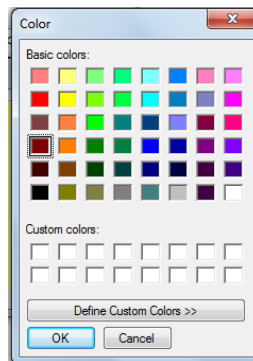
This dialog box appear when we write following code on save button click.

```
If SaveFileDialog1.ShowDialog() <> Windows.Forms.DialogResult.Cancel Then
    Dim fw As FileStream = New FileStream(SaveFileDialog1.FileName,
    FileMode.Append, FileAccess.Write)
```

```
Dim w As New StreamWriter(fw)
w.WriteLine(TextBox1.Text)
w.Flush()
w.Close()
End If
```

This code open save dialog box and write textbox content into a file where location return by this dialog box.

3. **ColorDialogBox:** This dialog box provides a color palette so that we can choose any color. When OK button press then it gives color value. It look like-

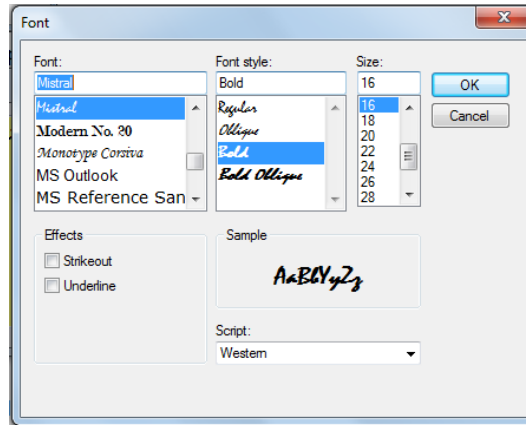


This dialog box appears when we write following code on color button click.

```
If ColorDialog1.ShowDialog() <> Windows.Forms.DialogResult.Cancel Then
    TextBox1.BackColor = ColorDialog1.Color
End If
```

This code open color dialog box and change back ground color of textbox that return by this dialog box.

4. **FontDialogBox:** This dialog box provides all font related setting like font types, font names and font setting. After setting font when we click OK button of dialogbox then it returns all font related value. It look like-

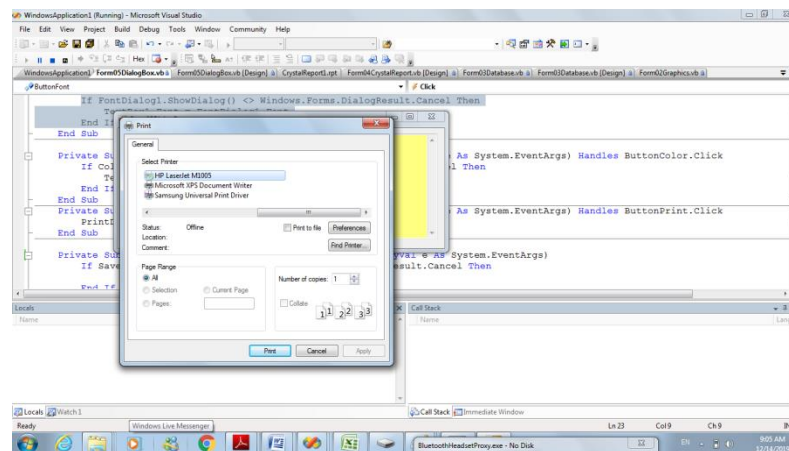


This dialog box appears when we write following code on font button click.

```
If FontDialog1.ShowDialog() <> Windows.Forms.DialogResult.Cancel Then
    TextBox1.Font = FontDialog1.Font
End If
```

This code open color dialog box and change back ground color of textbox that return by this dialog box.

5. **PrintDialogBox:** It is used to print anything of application. It look like



This dialog box appears when we write following code on print button click.

```
PrintDialog1.ShowDialog() .
```

UNIT-5

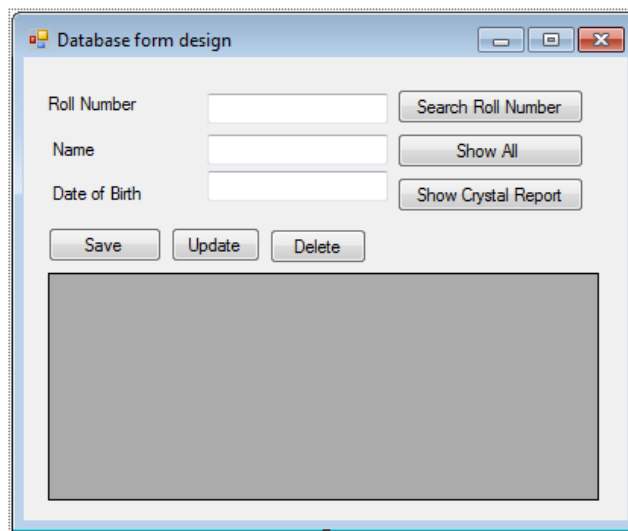
Working with database:

Step1: Open Window application project of VB.NET language.

Step2: Add Sql-Server Database file(college.dbf) using solution explorer by Add New Item.

Step3: Using Server Explore, select database file and create a table(student) and add any three fields(id-int, name-varchar,dob-datetime).

Step4: Design a window form as following.



Step5: Import following namespaces at the top of code behind of the form

```
Imports System.Data.SqlClient
Imports System.Data
```

Step6: Write following code in the click event of *save, update and delete buttons* to execute insert, update and delete commands respectively into database. (**SqlConnection, SqlCommand class**)

```
' establish connection code
Dim con As New SqlConnection("connection string")
con.Open()

'execute insert/update/delete command
Dim q As String = "insert /update /delete query"
Dim cmd As New SqlCommand(q, con)
cmd.ExecuteNonQuery()
```

Step7: Write following code in the click event of *Search button* to find any one record into database. (**SqlConnection, SqlCommand and SqlDataReader class**)

```
' establish connection code
Dim con As New SqlConnection("connection string")
con.Open()

'Access record using data reader
Dim q As String = "select command with where clause"
Dim cmd As New SqlCommand(q, con)
Dim dr As SqlDataReader = cmd.ExecuteReader
If (dr.Read) Then
    TextBoxName.Text = dr(1).ToString
    TextBoxDOB.Text = dr(2).ToString
End If
```

Step8: Write following code in the click event of *Show All button* to find any one record into database. (**SqlConnection, SqlCommand, SqlDataAdapter, Dataset class and DataGridView Control**)

```
' establish connection code
Dim con As New SqlConnection("connection string")
con.Open()

'Access all records using data adapter and dataset
Dim q As String = "select command "
Dim cmd As New SqlCommand(q, con)
Dim da As New SqlDataAdapter
Dim ds As New DataSet
da.Fill(ds)
'Show records within dataset into GridView controls
DataGridView1.DataSource = ds.Tables(0)
```

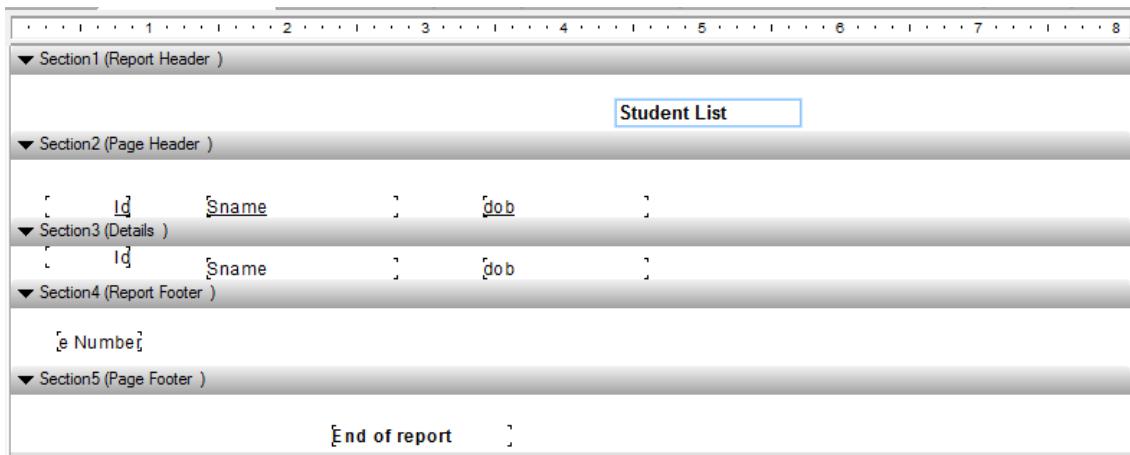
Crystal Report

It is third party software of vb.net to prepare a report of database records so that they can be printed on paper. To work with crystal report follow these steps.

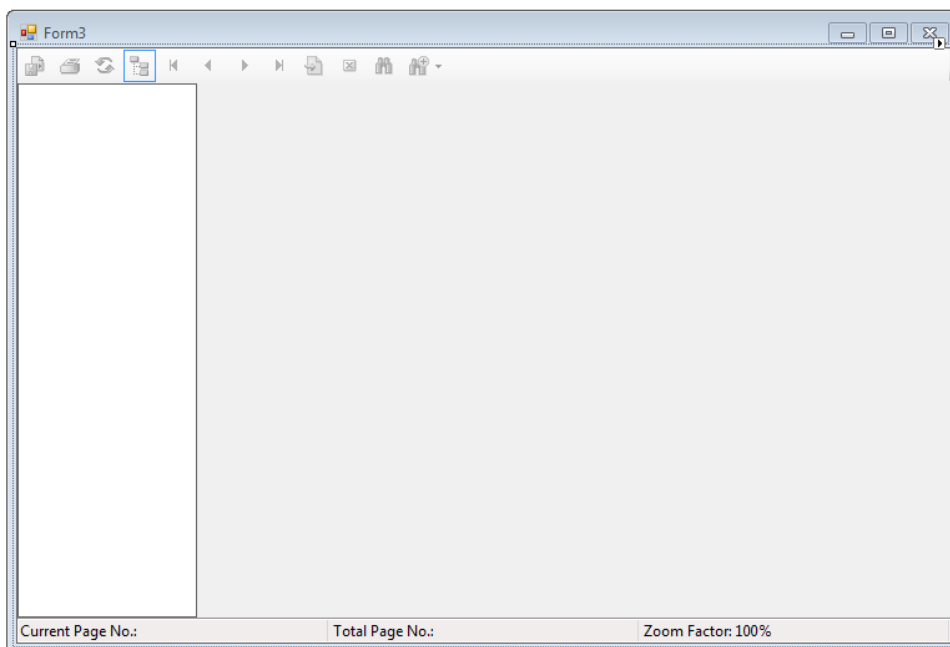
Step1: Add dataset file (Ex. Dataset1.xsd) using add new item option from solution explorer and assign required database table to dataset.

Step2: Add Crystal Report file(Ex. CrystalReport1.rpt) using add new item option from solution explorer.

Step3: Select data source(Ex. Dataset1.xsd) of crystal report and design crystal report by drag-drop fields in detail **section**.



Step4: From toolbox add CrystalReportViewer control on a new form to view report and print on paper.



Step5: Write following code in click event of a button that can generate report.

```
' establish connection code
Dim con As New SqlConnection("connection string")
con.Open()

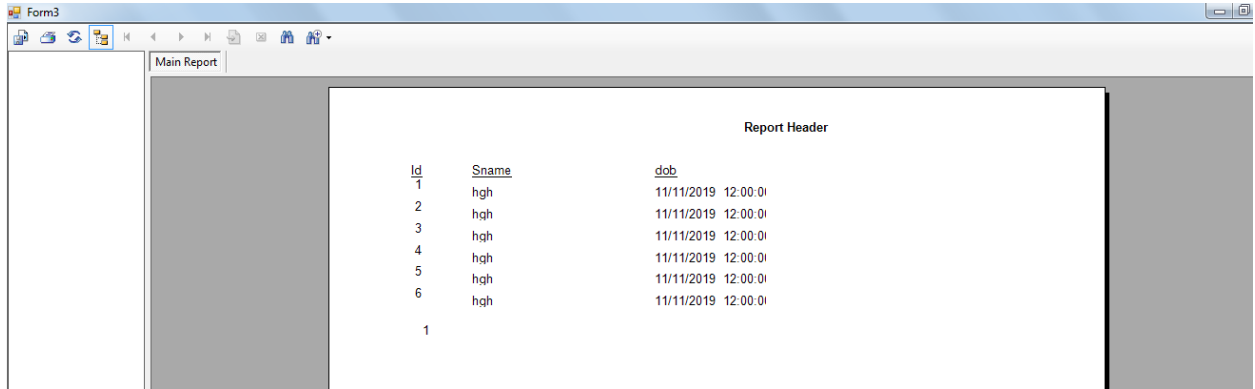
'Access all records using data adapter and dataset
Dim q As String = "select command "
Dim cmd As New SqlCommand(q, con)
Dim da As New SqlDataAdapter
Dim ds As New DataSet
da.Fill(ds)

'Load crystal report
Dim cr As New ReportDocument
cr.Load("path of CrystalReport1.rpt")
```

VB.NET

```
cr.SetDataSource(ds.Tables(0))  
'Show report viewer form  
Dim frm3 As New Form3  
frm3.CrystalReportViewer1.ReportSource = cr  
frm3.Show()
```

At run time crystal report of student record will be generated when button is clicked.



End of Unit-5