UNIT – 1
INTRODUCTION TO DOT NET

♦ Introduction:

.Net Framework is a software development platform developed by Microsoft for building and running Windows applications. The .Net framework consists of developer tools, programming languages, and libraries to build desktop and web applications. It is also used to build websites, web services, and games.

The .Net framework used to create applications, which would run on the Windows Platform. The first version of the .Net framework was released in the year 2002. It is used to create a form based, console-based, mobile and web-based application or web services that can run on Microsoft environment.

The .NET provides tools and libraries that enable developers to create Windows software much faster and easier. .NET Framework must be installed on a user's PC to run .NET applications.

.NET Framework supports more than 60 programming languages in which 11 programming languages are designed and developed by Microsoft. The remaining Non- Microsoft Languages which are supported by .NET Framework but not designed and developed by Microsoft.

11 Programming Languages which are designed and developed by Microsoft are:

1) C#.NET	2) VB.NET	3) C++.NET
4) J#.NET	5) F#.NET	6) WINDOWS POWERSHELL
7) JSCRIPT.NET	8) IRON RUBY	9) IRON PYTHON
10) C OMEGA	11) ASML (Abstract State Machine Language)	

Release History of .NET Framework and its compatibility with the different Windows version

.NET Version	CLR Version	Development tool	Windows Support
1.0	1.0	Visual Studio .NET	XP SP1
1.1	1.1	Visual Studio .NET 2003	XP SP2, SP3
2.0	2.0	Visual Studio 2005	N/A
3.0	2.0	Expression Blend	Vista
3.5	2.0	Visual Studio 2008	7, 8, 8.1, 10
4.0	4	Visual Studio 2010	N/A
4.5	4	Visual Studio 2012	8
4.5.1	4	Visual Studio 2013	8.1
4.5.2	4	N/A	N/A
4.6	4	Visual Studio 2015	10 v1507
4.6.1	4	Visual Studio 2015 Update 1	10 v1511
4.6.2	4	N/A	10 v1607
4.7	4	Visual Studio 2017	10 v1703
4.7.1	4	Visual Studio 2017	10 v1709
4.7.2	4	Visual Studio 2017	10v 1803
4.8	4	Visual Studio 2019	11
4.8.1	4	Visual Studio 2019	11
6		Visual Studio 2022	11

⧫ Overview of .Net:-

- The .NET Framework has been developed for the following objectives and requirements:
- To provide a consistent object-oriented environment to develop applications.
- To provide a code execution environment that simplifies deployment and versioning.
- To provide a code execution environment that guarantees the safety of the code that is executing.
- To provide a code execution environment that eliminates the issues faced by scripted environments with respect to performance.

The .NET Framework is made up of two major components:

1) **Common language runtime (CLR) and**
2) **Framework class library (FCL)**

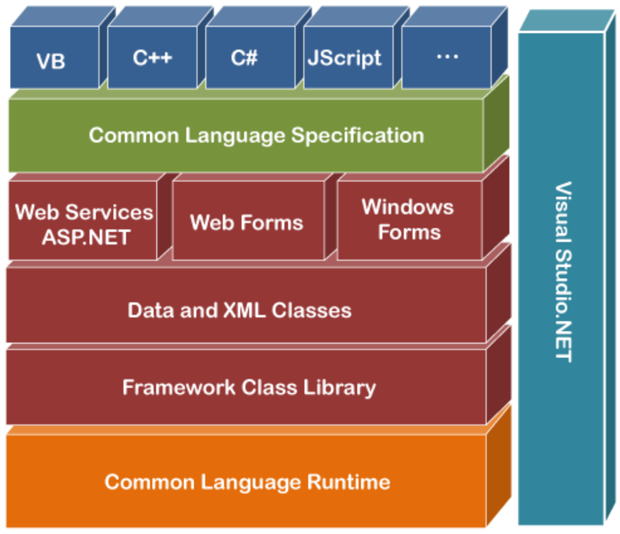
1) CLR: - The CLR is the foundation of the .NET Framework and provides various services that applications can use.
2) FCL: - The FCL is a collection of over 7000+ types that cater to all the services, anddata structures that applications will ever need.

⧫ .Net Architecture:-

.Net Architecture is a software architecture used for building applications that run on Microsoft's .NET platform. It consists of class libraries and reusable components.

Microsoft .NET consists of four major components:

1) **Common Language Specification (CLS) – blue in the diagram below**
2) **Framework Class Library (FCL) – red**
3) **Common Language Runtime (CLR) – green**
4) **.NET Tools – yellow**



1) Common Language Specification (CLS):

The CLS is a common platform that integrates code and components from multiple .NET programming languages. In other words, a .NET application can be written in multiple programming languages with no extra work by the developer.

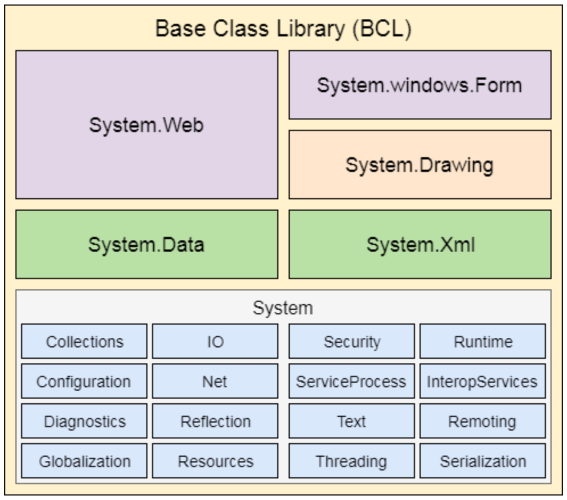
.NET includes new object-oriented programming languages such as C#, Visual Basic .NET, J# (a Java clone) and Managed C++. These languages, plus other experimental languages like F#, all compile to the Common Language Specificationand can work together in the same application.

2) Framework Class Library (FCL):

.NET Framework Class Library is the collection of classes, namespaces, interfaces and value types that are used for .NET applications. The FCL is a collection of over 7000+ classes and data types that enable .NET applications to read and write files, access databases, process XML, display a graphical user interface, draw graphics, use Web services, etc.

Microsoft has divided the FCL into hierarchical namespaces. The FCL has about 100 namespaces in all. Each namespace holds classes and other types that share a common purpose. For example, much of the window manager portion of the Windows API is encapsulated in the System.Windows.Forms namespace. In this namespace classes that represent windows, dialog boxes, menus, and other elements commonlyused in GUI applications are present.

The following diagram shows the FCL classes and their associated namespaces.



3) Common Language Runtime (CLR):

It is an important part of a .NET framework that works like a virtual component of the .NET Framework to execute the different languages program like c#, Visual Basic, etc. A CLR also helps to convert a source code into the byte code, and this byte code is known as CIL (Common Intermediate Language) or MSIL (Microsoft Intermediate Language). After converting into a byte code, a CLR uses a JIT compiler at run time that helps to convert a CIL or MSIL code into the machine or native code. The CLR is the execution engine for .NET applications and serves as the interface between .NET applications and the operating system. The .NET Framework provides a runtime environment called the Common Language Runtime or CLR (similar to the Java Virtual Machine or JVM in Java), which handles the execution of code.

The CLR provides many services such as:

1) Loads and executes code
2) Converts intermediate language to native machine code.
3) MSIL Code
4) Native code
5) Separates processes and memory
6) Manages memory and objects
7) Enforces code and access security
8) Handles exceptions
9) Provides debugging etc.

❖ Some Important Features of .Net:-

The following are major features of .NET

1) Cross Language Interoperability support (Interop):

Language interoperability is the ability of a code to interact with another code that is written by using a different programming language. Language interoperability can help maximize code reuse and improve the overall efficiency of the development process.

The .NET Framework provides many great features including the Interoperability. Microsoft has a migration utility to automatically migrate existing Java source code into C#.

2) Common language runtime (CLR):

It is an important part of a .NET framework that works like a virtual component of the .NET Framework to executes the different languages program like c#, Visual Basic, etc. A CLR also helps to convert a source code into the byte code, and this byte code is known as CIL (Common Intermediate Language) or MSIL (Microsoft Intermediate Language). After converting into a byte code, a CLR uses a JIT compiler at run time that helps to convert a CIL or MSIL code into the machine or native code.

3) Base class library (BCL):

The .NET Framework includes a set of standard class libraries which is called as Base Class Library. A class library is a collection of methods and functions. .NET provides 7000+ classes and data types. The BCL is common for all type of application in .NET framework. BCL divides into two parts:

1) **User defined class library:** It is the collection of small parts of deployment an application's part.

It contains either the DLL (Dynamic Link Library) or exe (Executable) file.

2) **Predefined class library:**

Namespace - It is the collection of predefined class and method that present in .Net. In other languages such as, C we used header files, in java we used package similarly we used "using system" in .NET, where using is a keyword and system is a namespace.

4) Common type system (CTS):

The CTS defines how types are declared, used, and managed in the runtime. It ensures that objects written in different languages can interact with each other. For example, a class defined in C# can inherit from a class defined in Visual Basic, and a method written in F# can accept a parameter of a type defined in C#.

5) Assemblies:

In .NET, an assembly is a compiled code library containing all the code and resources needed for applications to run. An assembly is either a .DLL or .EXE that forms a part of an application. It contains MSIL code that is executed by CLR. The following are other important points related to an assembly:

The four parts of an assembly are:

1) **Assembly Manifest** - Contains name, version, culture, and information about referenced assemblies.
2) **Type metadata - C**ontains information about types defined in the assembly.
3) **MSIL** - MSIL code.
4) **Resources** - Files such as BMP or JPG file or any other files required by application.

6) Simplified deployment:

The simplified deployment feature in .NET refers to various mechanisms and tools provided by the .NET platform to make the deployment of applications easier, more flexible, and more reliable. Deployment refers to the process of making an application or software available for use.

7) Intellisense:

IntelliSense is a code-completion help provided by Microsoft's Visual Studio IDEs for .NET development. .NET IDE supports Intellisense feature that helps in automatic code compilations. It helps in reducing typo effects and mistakes.

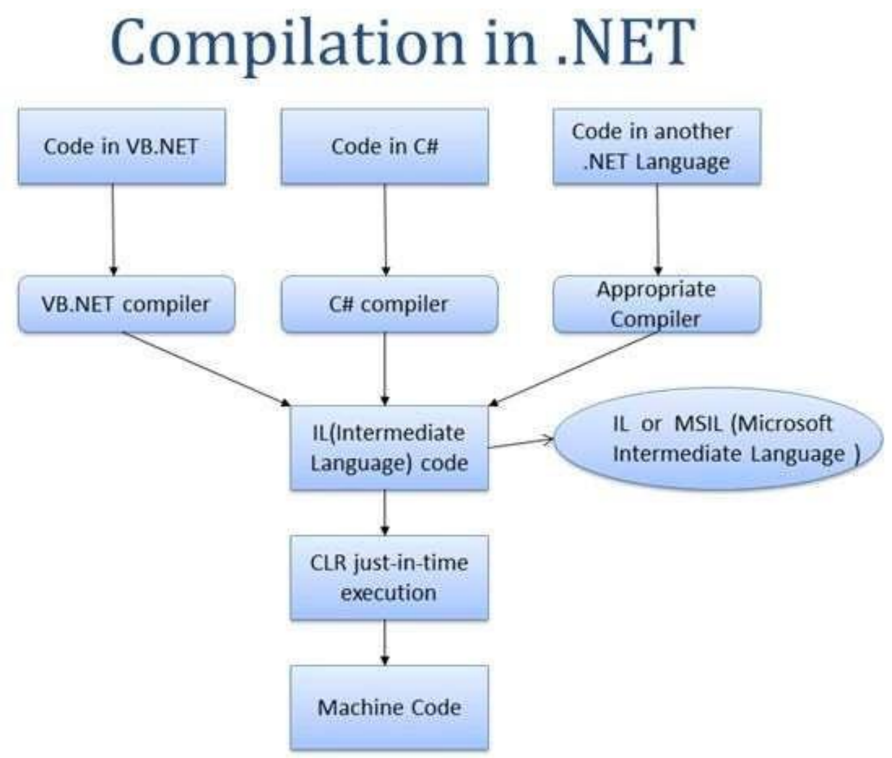
8) **Security:** Dot Net Platforms are secure and safe for organization applications. Some of the additional features are the type of safety, security, code access, and role-based authentication. These make Dot Net applications more robust and safe

❖ MSIL: Microsoft Intermediate Language (MSIL)

An intermediate language generated by compiler is called MSIL. All .Net assemblies are represented in MSIL. The main Advantage of using MSIL is it provides equal performance for multiple language programming, as code is compiled to native code. Example: Performance of application developed in C# is similar to VB.net or any other .Net compliant language that is because of MSIL.

When the code is compiled, the compiler translates your code into Microsoft intermediate language (MSIL). The common language runtime includes a JIT compiler for converting this MSIL then to native code. MSIL contains metadata that is the key to cross language interoperability. Since this metadata is standardized across all .NET languages, a program written in one language can understand the metadata and execute code, written in a different language.

MSIL includes instructions for loading, storing, initializing, and calling methods on objects, as well as instructions for arithmetic and logical operations, control flow, direct memory access, exception handling, and other operations.


Compilation in .NET
Code in VB.NET
Code in C#
Code in another .NET Language
VB.NET compiler
C# compiler
Appropriate Compiler
IL(Intermediate Language) code
IL or MSIL (Microsoft Intermediate Language)
CLR just-in-time execution
Machine Code

❖ Metadata:

In .NET, metadata refers to the information about the structure and behavior of code in .NET assemblies (compiled code libraries). This information is stored in the assembly's manifest and is used by the .NET runtime to understand and manage the code. Metadata is machine-readable information about a resource, or “data about data.” Metadata is stored in the assembly (a .dll or .exe file) alongside the compiled code.

Components of Metadata:

1. **Types**:
   - **Classes**: Define objects with methods, properties, fields, events, and other members.
   - **Structures**: Similar to classes but are value types.
   - **Interfaces**: Define a contract that classes or structures can implement.
   - **Enums**: Define a set of named constants.
   - **Delegates**: Define a type that represents references to methods.
2. **Members**:
   - **Fields**: Variables declared directly in a class or structure.
   - **Properties**: Provide a flexible mechanism to read, write, or compute the values of private fields.
   - **Methods**: Define actions or functions that can be performed.
   - **Events**: Enable a class or object to notify other classes or objects when something of interest occurs.
3. **References**:
   - **Assembly References**: Metadata includes information about other assemblies that your code depends on.
4. **Manifest**:
   - **Assembly Metadata**: Contains details about the assembly, such as its name, version, culture, and strong name information (if signed).
   - **Module Metadata**: If the assembly contains multiple modules, the manifest will include metadata about each module.

Type Metadata: For example, consider a simple class:

```
public class Person
{
   public string Name { get; set; }
   public int Age { get; set; }

   public void SayHello()
   {
      Console.WriteLine("Hello, my name is "+ Name);
   }
}
```

The metadata for this class includes:

- The class name Person.
- Properties Name and Age with their types (string and int).
- Method SayHello and its return type (void).

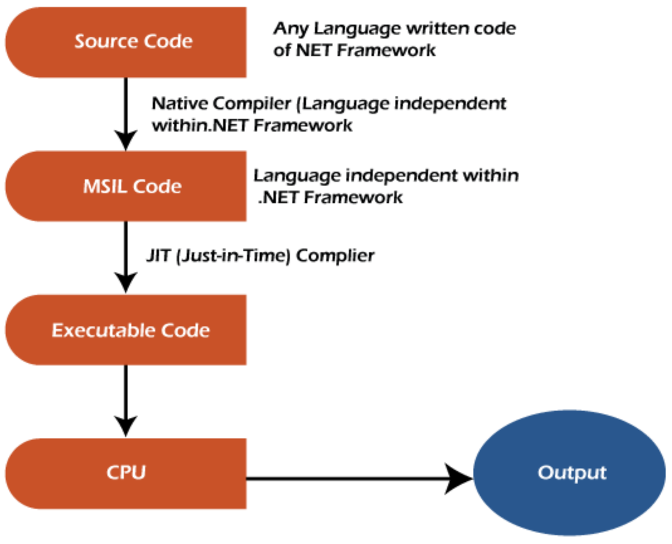
❖ Managed Code & Unmanaged Code:

1) Managed Code:

The code, which is developed in .NET framework, is known as managed code. This code is directly executed by CLR with help of managed code execution. Any language that is written in .NET Framework is managed code.

The managed runtime environment provides different types of services like garbage collection, type checking, and exception handling etc. to code automatically without the interference of the programmer.

In the case of .NET Framework, the compiler always compiles the managed code in the intermediate language (MSIL) and then Just-In-Time Compiler of CLR compiles the intermediate language in the native code


Source Code
Any Language written code of NET Framework
Native Compiler (Language independent within.NET Framework
MSIL Code
Language independent within .NET Framework
JIT (Just-in-Time) Complier
Executable Code
CPU
Output

Managed Code Execution Process

Advantages of using Managed Code:

1) It improves the security of the application.
2) It implement the garbage collection automatically.
3) It also provides runtime type checking/dynamic type checking.

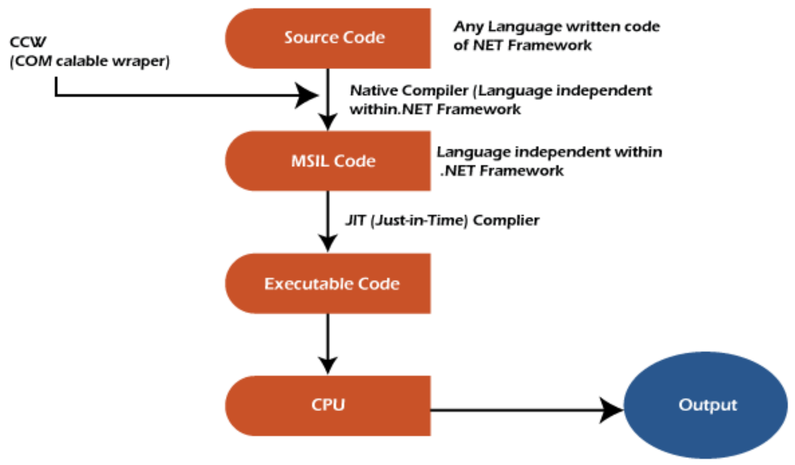
Disadvantages of using Managed Code:

1) The main disadvantage of managed language is that you are not allowed to allocate memory directly, or you cannot get the low-level access of the CPU architecture.

2) Unmanaged Code:

- The code, which is developed outside .NET, Framework is known as unmanaged code.
- Code that is directly executed by the Operating System is known as un-managed code.
- Typically applications written in VB 6.0, C++, C, etc. are all examples of unmanaged code.

- Unmanaged code can be unmanaged source code and unmanaged compile code.
- Unmanaged code is executed with help of wrapper classes.
- Unmanaged Code is typically used when developers need to interact with low-level system resources or hardware devices.
- For example, if you need to write a driver for a specific piece of hardware, you would likely need to use Unmanaged Code.


CCW
(COM calable wraper)
Source Code
Any Language written code of NET Framework
Native Compiler (Language independent within.NET Framework
MSIL Code
Language independent within .NET Framework
JIT (Just-in-Time) Complier
Executable Code
CPU
Output

Unmanaged Code Execution Process

Advantages of Unmanaged Code:

- **Performance**: Direct access to system resources can enhance execution speed.
- **Legacy Code Integration**: Allows reuse of existing unmanaged libraries and APIs.
- **Low-Level System Operations**: Provides access to hardware and system-level functions.
- **Interoperability**: Facilitates calling unmanaged functions from .NET via P/Invoke.

Disadvantages of Unmanaged Code:

- **Memory Management**: Requires manual memory handling, increasing risk of leaks and overflows.
- **Type Safety**: Lacks runtime type safety, leading to potential type mismatches.
- **Exception Handling**: Unmanaged exceptions are not integrated with .NET's exception handling.
- **Security Risks**: More prone to security vulnerabilities due to absence of runtime protection.

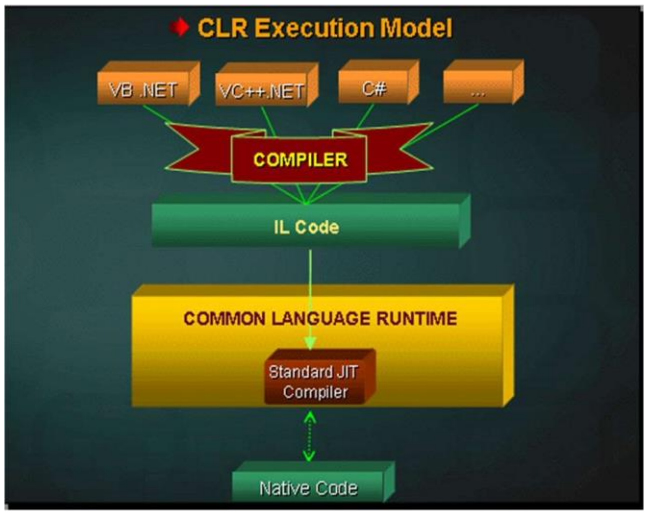
❖ Difference between Managed and Unmanaged code in .NET:

MANAGED CODE	UNMANAGED CODE
It is executed by managed runtime environment Or managed by the CLR.	It is executed directly by the operating system.
It provides security to the application written in .NET Framework.	It does not provide any security to the application.
Memory buffer overflow does not occur.	Memory buffer overflow may occur.

It provide runtime services like Garbage Collection, exception handling, etc.	It does not provide runtime services like Garbage Collection, exception handling, etc.
The source code is compiled in the intermediate language know as IL or MSIL or CIL.	The source code directly compile into native language
It does not provide low-level access to the programmer.	It provide low-level access to the programmer.

❖ Common Language Runtime:

- CLR stands for Common Language Runtime, which is a component of the Microsoft .NET framework.
- CLR is the foundation of .NET framework & it is the execution engine for .NET applications.
- The CLR acts as the interface between .NET applications and the operating system.
- It is a runtime environment that manages the execution of .NET programs and provides various services to them such as memory management, exception handling, security, and type safety.
- CLR runtime "manages" the execution of your code, code that works on the CLR is called as managed code. Any other code, you guessed it, is called unmanaged code.
- When a .NET program is compiled, it is translated into a Microsoft intermediate language called the MSIL / Common Intermediate Language (CIL). When a code starts to execute, a process knowing as Just in Time Compilation (JIT) converts the MSIL code into the machine code and executes it.
- This allows .NET programs to be platform-independent. This is shown in the following diagram:



- The CLR also provides a set of base class libraries that can be used by .NET programs. These libraries provide a wide range of functionality such as file I/O, network communication, and database access, allowing .NET programs to be developed more quickly and easily.
- Overall, the CLR is an essential component of the .NET framework that provides a runtime environment and services that make it easier to develop and run .NET applications.

- The following are some of the benefits of the CLR:
  1) Performance improvements.
  2) The ability to easily use components developed in other languages.
  3) Extensible types provided by a class library.
  4) New language features such as inheritance, interfaces, and overloading for object- oriented programming.
  5) Support for multithreading, support for structured exception handling and custom attributes etc.

❖ Common Type System (CTS):

The Common Type System (CTS) is a standard for defining and using data types in the .NET framework. CTS defines a collection of data types, which are used and managed by the run time to facilitate cross-language integration.

The .NET Framework supports many programming languages such as C#, VB.NET, J#, F#, etc. Every programming language has its own data type. One language data type cannot be understood by other languages. But, there can be situations where we need to communicate between two different programming languages. For example, we need to write code in the VB.NET language and that code may be called from C# language. .NET have a Common Type System (CTS) which ensures that data types defined in two different languages get compiled to a common data type.

CLR in .NET Framework will execute all programming language's data types. This is possible because CLR has its own data types which are common to all programming languages. At the time of compilation, all language-specific data types are converted into CLR's data type. This data type system of CLR is common to all .NET Supported Programming languages and this is known as the Common Type System (CTS) in .NET Framework.

CTS supports two different kinds of types: 1) Value Types 2) Reference Types

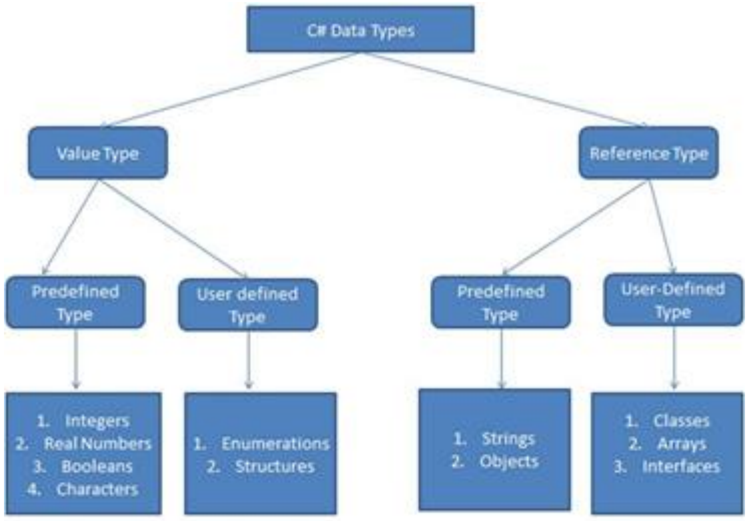
1) Value Types:

- Value types directly contain their data & instances
- Value types are also known as exact types.
- Value types are simple bit pattern like int, float, char.
- In .NET a value type derives from the system.object namespace.
- System.object namespace support an interface that provides information about the kind of data that is stored as well as the data values.

2) Reference Types:

- Reference Types are derived from the system.object namespace.
- Reference Types are self typing which means they describes their own interface memory items.
- Reference Types stores a reference to values memory address and are allocated on to heap.

- Reference type can be self-describing type, pointer type or interface type.



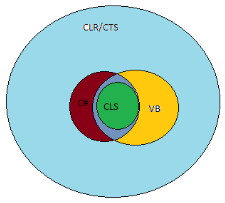
❖ Common Language Specifications:

Common Language Specification (CLS) is a set of rules that defines a minimum set of language features and guidelines are needed by many common applications. These features include data types, operators, methods, events, and exception handling. The purpose of the CLS is to enable developers to create components and libraries that can be used by any .NET-compliant language.

The .NET Framework is a software development platform that provides a common runtime environment, a set of libraries, and a framework for building and deploying applications. It supports multiple programming languages, including C#, Visual Basic .NET, F#, and others. However, each programming language has its own syntax, semantics, and features, which can make it difficult to use code written in one language from another language. The CLS addresses this issue by defining a common subset of features that all .NET-compliant languages must support.

The CLS includes guidelines for defining data types, operators, methods, events, and exception handling. For example, it specifies that all .NET-compliant languages must support the Int32 data type, the + operator for adding integers, and the try-catch statement for handling exceptions. The benefit of CLS is developers can ensure that their code can be used by other .NET languages, regardless of the language in which it was originally written.

In addition to the CLS, the .NET Framework also includes a Common Type System (CTS), which defines a common set of data types that all .NET-compliant languages must support. The CTS ensures that objects created in one language can be used by another language, and that data can be exchanged between different languages.



Together, the CLS and CTS provide a foundation for interoperability between different .NET-compliant languages. This, in turn, promotes code reuse and reduces the amount of time and effort required to develop new applications. Developers can write code in the language of their choice, while still being able to take advantage of the features and benefits of the .NET Framework.

❖ FCL:

FCL stands for the Framework Class Library, which is a collection of reusable classes, interfaces, and value types that are used to develop applications on the .NET Framework. It is also sometimes referred to as the Base Class Library (BCL).

The FCL is a fundamental part of the .NET Framework, providing a set of core functionality that all .NET applications can use. It includes classes for working with collections, streams, files, input/output, network communications, threading, security, and more. In addition, it includes support for developing graphical user interfaces (GUIs) through Windows Forms and WPF, as well as web applications through ASP.NET.

The FCL is organized into namespaces, which group related classes and types together. For example, the System.IO namespace includes classes for working with files and directories, while the System.Net namespace includes classes for working with network communications.

Developers can use the FCL to build their applications by referencing the appropriate namespaces and classes in their code. Because the FCL is part of the .NET Framework, it is available to any .NET-compliant language, including C#, Visual Basic .NET, F#, and others. This makes it easier for developers to create applications that can be easily maintained, extended, and reused.

Overall, the FCL provides a rich set of functionality that enables developers to build a wide variety of applications on the .NET Framework.

❖ Types, Object & Namespaces:

1) Types:

In .NET, a type is a classification that defines the properties, methods, and behaviors of data. It specifies how data is stored and processed. Types can be broadly categorized into two main categories:

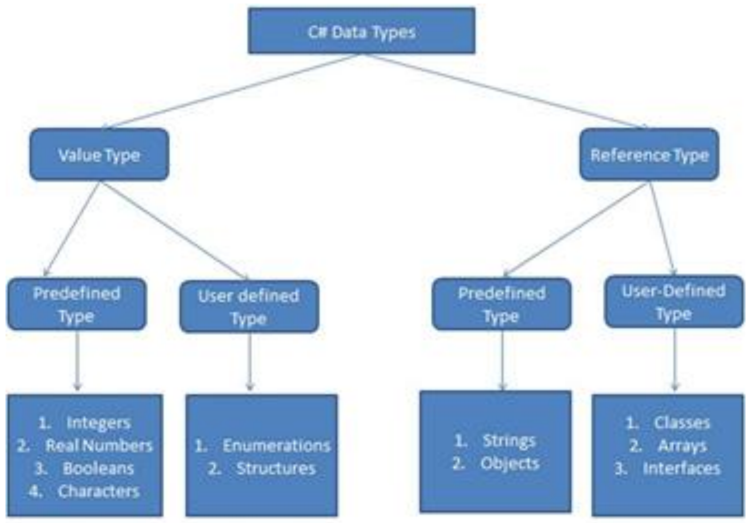
1) Value Types 2) Reference Types

1) Value Types:

- Value types directly contain their data & instances
- Value types are also known as exact types.
- Value types are simple bit pattern like int, float, char.
- In .NET a value type derives from the system.object namespace.
- System.object namespace support an interface that provides information about the kind of data that is stored as well as the data values.

2) Reference Types:

- Reference Types are derived from the system.object namespace.
- Reference Types are self-typing which means they describes their own interface memory items.
- Reference Types stores a reference to values memory address and are allocated on to heap.
- Reference type can be self-describing.



2) Object: Objects are instances of types (classes or structures). They encapsulate data and behavior together.

- **Creating Objects**: You create an object using the `new` keyword.

  Example:

  Person person = new Person();

- **Using Objects: Access their methods and properties.**

  Example:

  Person person = new Person();

  person.Name = "John";

  person.Age = 30;

3) .Net Namespaces:

Namespaces are the way to organize .NET classes into a logical grouping according to their functionality. The .NET Framework Class Library (FCL) is a large collection of thousands of Classes. These Classes are organized in a hierarchical tree. The System Namespaces is the root for types in the .NET Framework. We can uniquely identify any Class in the .NET Framework Class Library (FCL) by using the full Namespaces of the class. In .Net languages every program is created with a default Namespaces. Programmers can also create their own Namespaces in .Net languages.

Defining a Namespace: To define a namespace in C#, we will use the **namespace** keyword followed by the name of the namespace and curly braces containing the body of the namespace

Syntax:

```
namespace name_of_namespace
{
            // Classes
            // Interfaces
            // Structures
}
```

Accessing the Members of Namespace:

The members of a namespace are accessed by using dot(.) operator. A class in C# is fully known by its respective namespace.

Syntax: [namespace_name].[member_name]

Example:

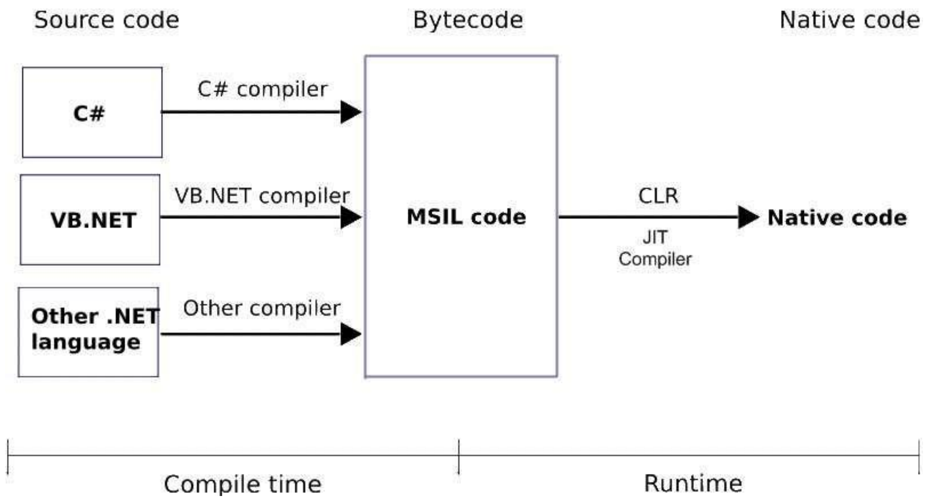
```
using System;
namespace NamespacesDemo
{
   class Program
   {
      static void Main(string[] args)
      {
         Student.std.display();
         Console.ReadKey();
      }
   }
}
namespace Student
{
   public class std
   {
      public static void display()
      {
         Console.Write("Welcome");
      }
   }
}
```

Note:

- Two classes with the same name can be created inside 2 different namespaces in a single program.
- Inside a namespace, no two classes can have the same name.

❖ JIT (Just-In-Time compiler):

Just-In-Time compiler (JIT) is a part of Common Language Runtime (CLR). A language-specific compiler converts the source code to the intermediate language called MSIL/CIL. This intermediate language is then converted into the machine code by the Just-In-Time (JIT) compiler.



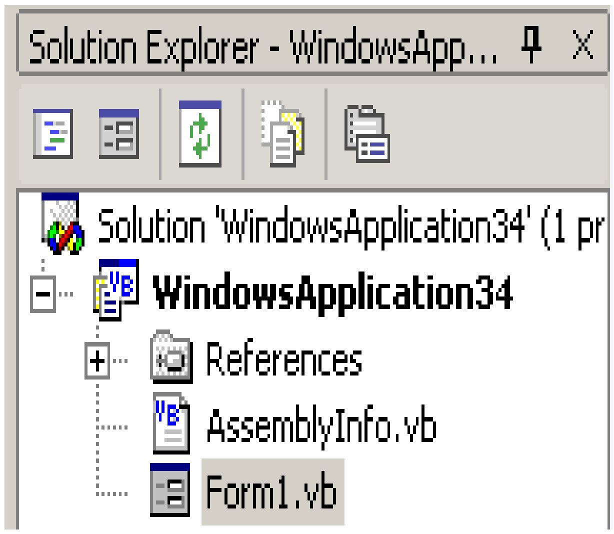
There are 3 types of JIT compilers which are as follows:

1) **Pre-JIT:** Per-JIT compiler compiles source code into native code in a single compilation cycle.
2) **Econo- JIT:** Econo-JIT compiles methods that are called at runtime. However, these compiled methods are discarded when they're not required.
3) **JIT (Normal JIT):** They're called "JIT" or "Normal JIT". Normal JIT only compiles the methods which are called at runtime. These methods are compiled the first time they're called and then they're stored in cache. When the same methods are called again, the compilation code from cache is used for execution.

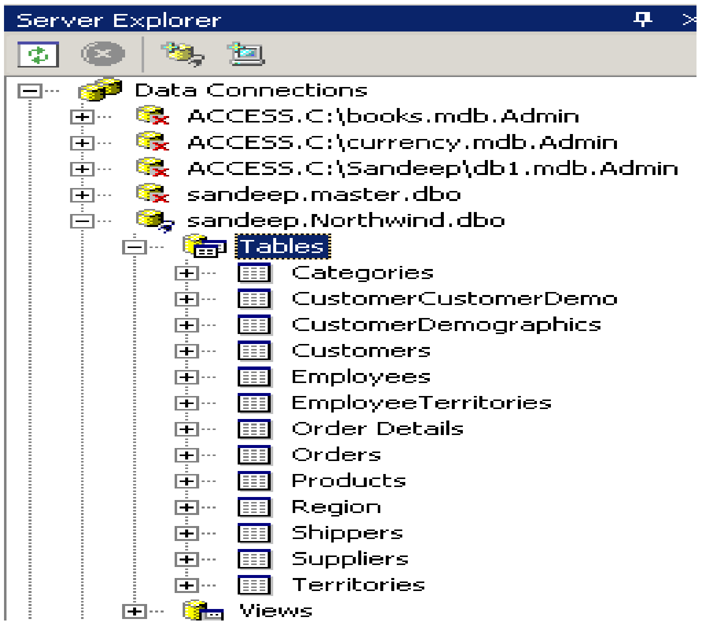
❖ Visual Studio .NET IDE:

IDE stands for Integrated Development Environment. Visual Studio .NET IDE is the Development Environment for all .NET based applications which comes with rich features. IDE offers set of tools the helps to develop an application

1) **Solution Explorer Window:** The Solution Explorer window gives an overview of the solution we are working with and lists all the files in the project.


Solution Explorer - WindowsApp...
Solution 'WindowsApplication34' (1 pr
WindowsApplication34
References
AssemblyInfo.vb
Form1.vb

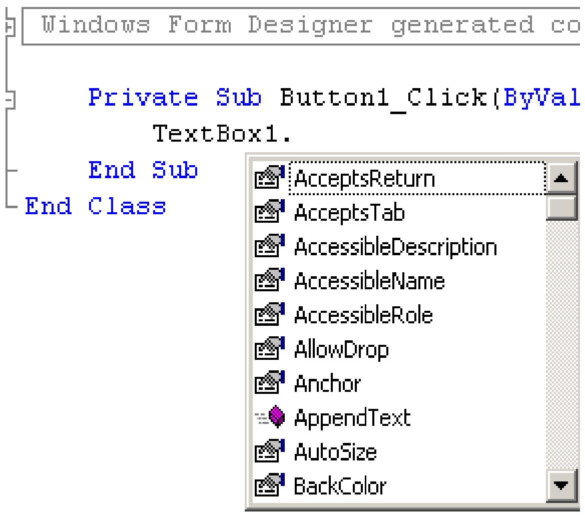
2) **Server Explorer Window:** The Server Explorer window is a great tool that provides "drag and drop" feature and helps us work with databases in an easy graphical environment.



```
Data Connections
  ACCESS.C:\books.mdb.Admin
  ACCESS.C:\currency.mdb.Admin
  ACCESS.C:\Sandeep\db1.mdb.Admin
  sandeep.master.dbo
  sandeep.Northwind.dbo
    Tables
      Categories
      CustomerCustomerDemo
      CustomerDemographics
      Customers
      Employees
      EmployeeTerritories
      Order Details
      Orders
      Products
      Region
      Shippers
      Suppliers
      Territories
    Views
```

3) **Intellisense:**

Intellisense is what that is responsible for the boxes that open as we type the code. IntelliSense provides a list of options that make language references easily accessible.

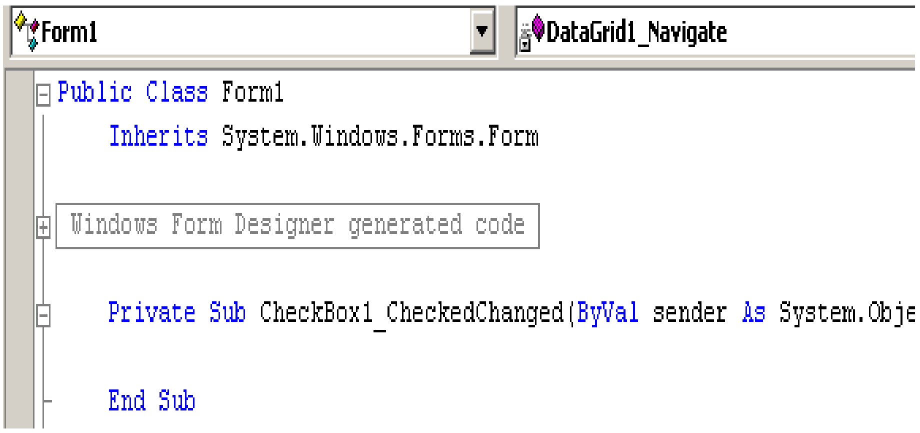


```
 Windows Form Designer generated co

    Private Sub Button1_Click(ByVal
        TextBox1.
    End Sub
End Class
```

4) **Code Designer Window:**

- Code allows us to edit and write code.
- This is the window that opens when we double-click on a form or any control.
- Notice the two drop-down list boxes at the top
- The left box allows us to select the object's code we are working with and
- The right box allows us to select the part of code that we want to work.
- Also notice the "+" and "-" boxes in the code designer of the code window in the image below.



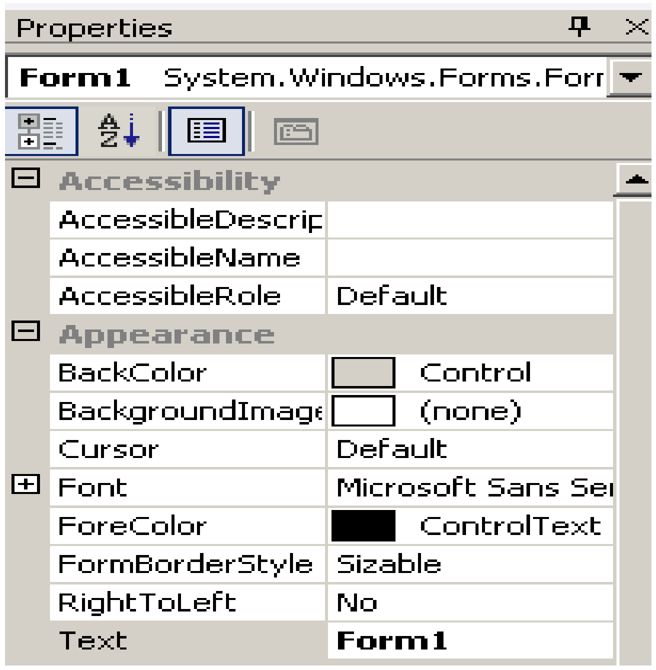
```
Form1                                    DataGrid1_Navigate
Public Class Form1
    Inherits System.Windows.Forms.Form

Windows Form Designer generated code

    Private Sub CheckBox1_CheckedChanged(ByVal sender As System.Obje

    End Sub
```

5) **Properties Window:** The properties window allows us to set properties for various objects at design time. E.g. if you want to change Font, Font Size, Backcolor, forecolor etc. you can change the properties using Properties Window.



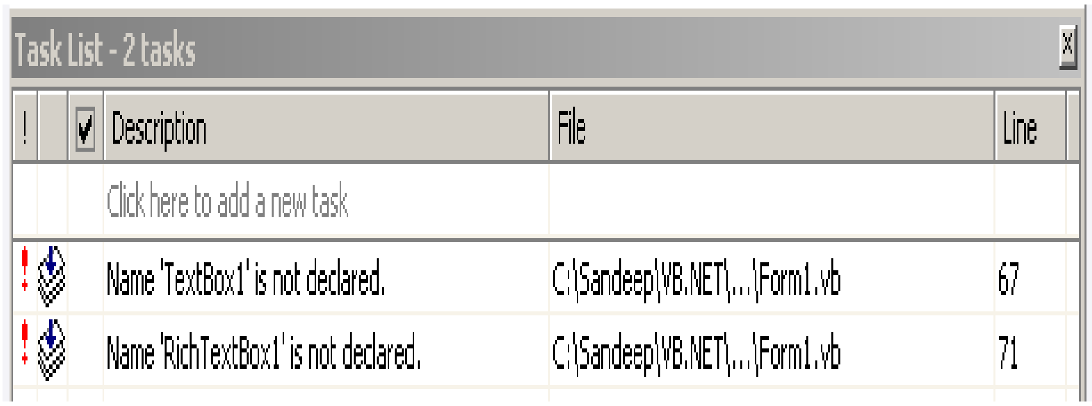
Properties

Form1 System.Windows.Forms.Forr

Accessibility	
AccessibleDescrip	
AccessibleName	
AccessibleRole	Default
Appearance	
BackColor	☐ Control
BackgroundImage	☐ (none)
Cursor	Default
Font	Microsoft Sans Sei
ForeColor	■ ControlText
FormBorderStyle	Sizable
RightToLeft	No
Text	**Form1**

6) **Task List Window:**

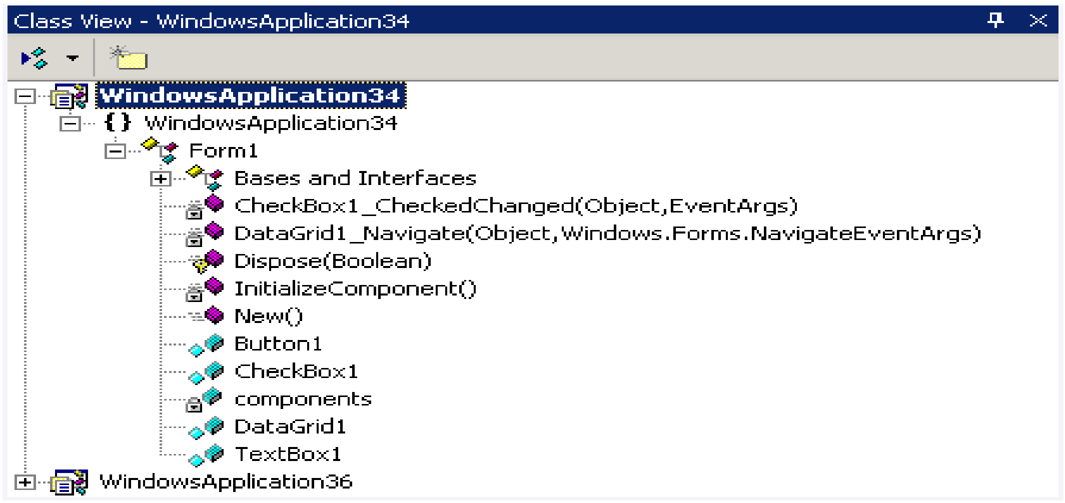
- The task list window displays all the tasks that assumes we still have to finish.



Task List - 2 tasks

!		✓	Description	File	Line
			Click here to add a new task		
!			Name 'TextBox1' is not declared.	C:\Sandeep\VB.NET\...\Form1.vb	67
!			Name 'RichTextBox1' is not declared.	C:\Sandeep\VB.NET\...\Form1.vb	71

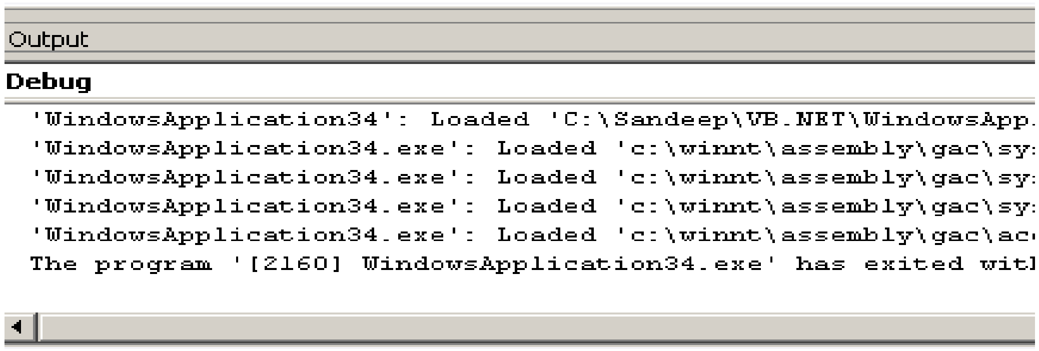
7) **Class View Window:** Represents solutions and projects in terms of the classes they contain and the members of these classes. The class view window displayed all the methods and events for the controls which were available on the form.



Class View - WindowsApplication34

```
WindowsApplication34
  { } WindowsApplication34
    Form1
      Bases and Interfaces
      CheckBox1_CheckedChanged(Object,EventArgs)
      DataGrid1_Navigate(Object,Windows.Forms.NavigateEventArgs)
      Dispose(Boolean)
      InitializeComponent()
      New()
      Button1
      CheckBox1
      components
      DataGrid1
      TextBox1
WindowsApplication36
```

8) **Output Window:** The output window displays the results of building and running applications.



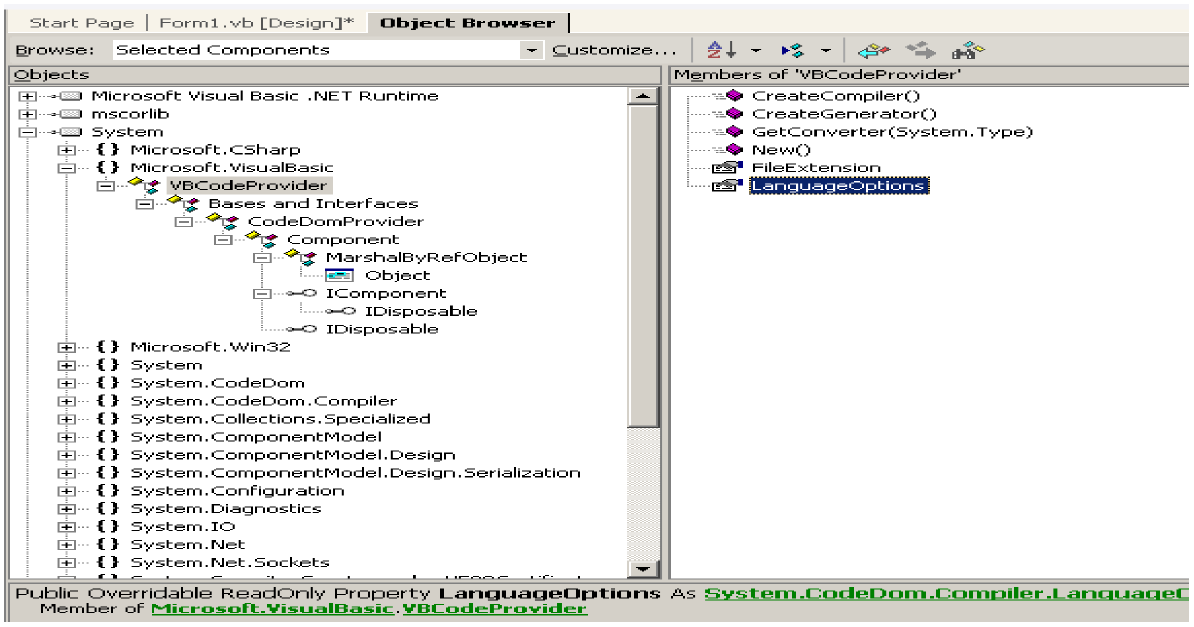
Output

Debug

```
'WindowsApplication34': Loaded 'C:\Sandeep\VB.NET\WindowsApp
'WindowsApplication34.exe': Loaded 'c:\winnt\assembly\gac\sy
'WindowsApplication34.exe': Loaded 'c:\winnt\assembly\gac\sy
'WindowsApplication34.exe': Loaded 'c:\winnt\assembly\gac\sy
'WindowsApplication34.exe': Loaded 'c:\winnt\assembly\gac\ac
The program '[2160] WindowsApplication34.exe' has exited wit
```

9) **Object Explorer Window:**

- The object explorer window allows us to view all the members of an object.



Start Page | Form1.vb [Design]* | Object Browser

Browse: Selected Components | Customize...

Objects

```
Microsoft Visual Basic .NET Runtime
mscorlib
System
  { } Microsoft.CSharp
  { } Microsoft.VisualBasic
    VBCodeProvider
      Bases and Interfaces
        CodeDomProvider
          Component
            MarshalByRefObject
              Object
            IComponent
              IDisposable
            IDisposable
  { } Microsoft.Win32
  { } System
  { } System.CodeDom
  { } System.CodeDom.Compiler
  { } System.Collections.Specialized
  { } System.ComponentModel
  { } System.ComponentModel.Design
  { } System.ComponentModel.Design.Serialization
  { } System.Configuration
  { } System.Diagnostics
  { } System.IO
  { } System.Net
  { } System.Net.Sockets
```

Members of 'VBCodeProvider'

```
CreateCompiler()
CreateGenerator()
GetConverter(System.Type)
New()
FileExtension
LanguageOptions
```

Public Overridable ReadOnly Property **LanguageOptions** As System.CodeDom.Compiler.LanguageC
Member of Microsoft.VisualBasic.VBCodeProvider

****

UNIT – II

INTRODUCTION TO C#.NET

❖ Introduction:

- C# (pronounced "C sharp") is a simple, modern, object-oriented, and type-safe programming language.
- It will immediately be familiar to C and C++ programmers. C# combines the high productivity of Rapid Application Development (RAD) languages and the raw power of C++.
- C# is an elegant and type-safe object-oriented language that enables developers to build a variety of secure and robust applications that run on the .NET Framework.
- You can use C# to create traditional Windows client applications, XML Web services, distributed components, client-server applications, database applications, and much, much more.
- As an object-oriented language, C# supports the concepts of encapsulation, inheritance, and polymorphism.
- All variables and methods, including the Main method, the application's entry point, areencapsulated within class definitions.

Note:

- ➢ **C# Syntax is highly expressive yet with 90 keywords.**
- ➢ **C# is Case Sensitive Language**

❖ A Simple "Hello World!" Program in C#:

To create and run a console application

1) Start Visual Studio.
2) On the menu bar, choose File - > New - > Project.
3) Expand Installed, expand Templates, expand Visual C#, and then choose Console Application.
4) In the Name box, specify a name for your project, and then choose the OK button.
5) If Program.cs isn't open in the Code Editor, open the shortcut menu for Program.cs in Solution Explorer, and then choose View Code.
6) Replace the contents of Program.cs with the following code.

// A Hello World! program in C#

```
using System;
namespace HelloWorld
{
        class Hello
        {
                static void Main()
                {
                        Console.WriteLine("Hello World!");
                        Console.WriteLine("Press any key to exit.");
                        Console.ReadKey();
                }
        }
}
```

1) Using System:

- The first few lines of your program all start with the keyword "using". A keyword is simply areserved word that is a built-in part of the C# programming language.
- The System namespace includes the console class which is necessary for input output with theconsole. It is important to note that each line of code terminates a semicolons ‘;’

2) namespace HelloWorld:

- The programmers may choose to execute a namespace with these name or company name

3) { }

- In C#, the curly braces are used to start and end sections of related code called a "code block". Every starting curly brace ('{') will have a matching ending curly brace ('}')

4) static void Main(string[] args)

- The main is the entry point method in C# when a console application runs it looks for the method entitled Main(string[] args)
- After the main method is found program will begin to execute at the first line after the main entry point.

5) Console Output: **The following code output a line of text to the console.**

Console.Write Line("Hello World!");

This code needs to be inserted just after the entry point of the program.

```
static void Main()
{
        Console.Write Line("Hello World!");
}
```

The code can be compiled & run by pressing f5 or debug -> start debugging.

The code will output Hello World! On screen.

```
static void Main()
{
        Console.Write Line("Hello World!");
        Console.Write Line("Press any key to exit.");
        Console.ReadKey(); // wait for user input
}
```

❖ Entry point method in C#:

- The Main method is the entry point of a C# console application or windows application.
- Libraries and services do not require a Main method as an entry point.
- When the application is started, the Main method is the first method that is invoked.

- There can only be one entry point in a C# program.
- The Main method is the entry point of an .exe program; it is where the program control starts and ends.
- Main is declared inside a class or struct. Main must be static and it should not be public. (In the earlier example, it receives the default access of private.) The enclosing class or struct is not required to be static.
- Main can either have a void or int return type.
- The Main method can be declared with or without a string [] parameter that contains command- line arguments.

Different ways of declaring main method in c#:

There are four ways in which you can declare main method.

```
1)  static void Main()
{
        // main block
}
2)  static int Main()
{
        // main block
}
3)  static void Main(string args[])
{
        // main block
}
4)  static int Main(string args[])
{
        // main block
}
```

❖ Command Line Arguments:

The main method can be declared with or without string parameter but that parameter contains command line arguments which can be used to provide to program some values at the movement of executions.

Example:

```
using System;
public class CmdLine
{
        static void Main(String args[])
        {
                Console.WriteLine("Number of Command Line Parameters:"+args.Length);
                for (int i = 0; i < args.Length; i++)
                {
                        Console.WriteLine("Arguments:"+args[i]);
                }
                Console.ReadKey();
        }
}
```

Run the program using some arguments CmdLine A B C
The Output will be

```
Number of Command Line Parameter = 3
Arguments: A
Arguments: B
Arguments: C
```

❖ Difference between an EXE and a DLL:

EXE	DLL
The term EXE is a shortened version of the word executable as it identifies the file as a program	On the other hand, DLL stands for Dynamic Link Library, which commonly contains functions and procedures that can be used by other programs.
An EXE file contains the entry point or the part of the code where the operating system is supported tobegin the execution of the application	DLL files do not have entry point and cannot be executed their own
An EXE cannot be reused in program	The most major advantage of DLL files is in its reusability. A DLL file can be used in other applications.
Launching an EXE would mean creating a process for it to run on and a memory space.	DLL is not launched by itself and is called by another application, it does not have its own memory space and process.
An EXE file can be run independently	A DLL cannot run independently is used by other applications.
EXE cannot be shared with other applications.	DLL can be shared with other applications.
The purpose of an EXE is to launch a separate application of its own.	The purpose of a DLL is to have a collection of methods/classes which can be re-used from some other application.
EXE execution is slow	DLL execution is very Fast
EXE cannot be versioned	DLL can be versioned

❖ C# Data Types:

A data type specifies the type of data that a variable can store such as integer, floating, characters etc. There are 3 types of data types in C# language. Value types, Pointer types and Reference types.

Data types in C# is mainly divided into three categories

- Value Data Types
- Reference Data Types
- Pointer Data Type

1) **Value Types:** In C#, the Value Data Types will directly store the variable value in memory. Following are different Value Data Types in C# programming language:

   Signed & Unsigned Integral Types: There are 8 integral types which provide support for 8-bit, 16-bit, 32-bit, and 64-bit values in signed or unsigned form.

Type	Description	Range	Suffix
byte	1 Byte 8-bit unsigned integer	0 to 255	

sbyte	1 Byte 8-bit signed integer	-128 to 127	
short	2 byte 16-bit signed integer	-32,768 to 32,767	
ushort	2 Byte 16-bit unsigned integer	0 to 65,535	
int	4 Byte 32-bit signed integer	-2,147,483,648 to 2,147,483,647	
uint	4 Byte 32-bit unsigned integer	0 to 4,294,967,295	u
long	8 Byte 64-bit signed integer	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	l
ulong	8 Byte 64-bit unsigned integer	0 to 18,446,744,073,709,551,615	ul
float	4 Byte 32-bit Single-precision floating point type	-3.402823e38 to 3.402823e38	f
double	8 Byte 64-bit double-precision floating point type	-1.79769313486232e308 to 1.79769313486232e308	d
decimal	16 Byte 128-bit decimal type for financial and monetary calculations	(+ or -)1.0 x 10e-28 to 7.9 x 10e28	m
char	1 byte 16-bit single Unicode character	Any valid character, e.g. a,*, \x0058 (hex), or\u0058 (Unicode)	
bool	8-bit logical true/false value	True or False	
object	Base type of all other types.		
string	Stores a sequence of characters, surrounded by double quotes		
DateTime	Represents date and time	0:00:00am 1/1/01 to 11:59:59pm 12/31/9999	

2) **Reference Data Types:** The Reference Data Types will contain a memory address of variable value because the reference types won't store the variable value directly in memory. The actual data for reference types is stored on the heap memory. The built-in reference types are **string, object.**

i) **String:** It represents a sequence of characters and its type name is **System.String**. So, string and String are equivalent.

Example:

```
string s1 = "hello"; // creating through string keyword
String s2 = "welcome"; // creating through String class
```

ii) **Object:** In C#, all types, predefined and user-defined, reference types and value types, inherit directly or indirectly from Object. Object is the base class for all types in C#, so any type can be assigned to an object variable.

When a variable of a value type is converted to object, it's called boxing. When a variable of type object is converted to a value type, it's called unboxing. Its type name is System.Object.

3) **Pointer Data Type:** The Pointer Data Types will contain a memory address of the variable value. To get the pointer details we have a two symbols **ampersand (&) and asterisk (*)**.

i) **ampersand (&):** It is Known as Address Operator. It is used to determine the address of a variable.

ii) **asterisk (*):** It also known as Indirection Operator. It is used to access the value of an address.

Example:

int* p1, p;   // Valid syntax

int *p1, *p;   // Invalid

❖ Parameter Passing Mechanism:

In C#, arguments can be passed to parameters either by value or by reference. This is the default mechanism for passing parameters to a method. Parameters are means of passing values to a method.

There are four different ways of passing parameters to a method in C# which are as:

1) Pass By Value
2) Pass By Ref (reference)
3) Out (reference)
4) Params (parameter arrays)

1) Pass By Value:

By default, parameters are passed by value. When you pass a parameter by value, a copy of the parameter's value is passed to the method. Changes to the parameter within the method do not affect the original variable.

Example:

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Square(int x) // The parameter x is passed by value.
        {
            x *= x;
        }

        static void Main(string[] args)
        {
            int n = 5;
            Console.WriteLine("The value before calling the method: "+ n);
            Square(n);   // Passing the variable by value.
            Console.WriteLine("The value after calling the method: "+ n);
            Console.ReadKey();
        }
    }
}
```

Output:

The value before calling the method: 5

The value after calling the method: 5

2) Pass By Reference:

Pass by reference in C# allows a method to modify the original variable passed to it. When a parameter is passed by reference using the ref keyword, the method receives a reference to the original variable rather than a copy of its value. This means any changes made to the parameter inside the method will directly affect the original variable.

Example:

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Square(ref int x) // The parameter x is passed by Reference.
        {
            x *= x;
        }

        static void Main(string[] args)
        {
            int n = 5;
            Console.WriteLine("The value before calling the method: "+ n);
            Square(ref n);   // Passing the variable by value.
            Console.WriteLine("The value after calling the method: "+ n);
            Console.ReadKey();
        }
    }
}
```

3) Out Parameter:

There is a big difference between ref and out parameters at the language level. Before you can pass a ref parameter, you must assign it to a value. On the other hand, you don't need to assign an out parameter before passing it to a method.

Example:

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void GetValues(out int x, out int y)
        {
            x = 10;
            y = 20;
        }
        static void Main(string[] args)
        {
            int a, b;
            GetValues(out a, out b);
            Console.WriteLine(" "+a+" "+b); // Output: 10, 20
            Console.ReadKey();
        }
    }
}
```

4) Parameter Arrays:

Allows a method to accept a variable number of arguments. The params keyword must be the last parameter in the method.

Example:

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        static int AddNumbers(params int[] numbers)
        {
            int sum = 0;
            foreach (int number in numbers)
            {
                sum += number;
            }
            return sum;
        }

        static void Main(string[] args)
        {
            int sum1 = AddNumbers(1, 2, 3, 4, 5);
            int sum2 = AddNumbers(10, 20);
            int sum3 = AddNumbers(100);

            // Printing the results
            Console.WriteLine("Sum1: "+sum1); // Output: Sum1: 15
            Console.WriteLine("Sum2: "+sum2); // Output: Sum2: 30
            Console.WriteLine("Sum3: "+sum3); // Output: Sum3: 100
            Console.ReadKey();
        }
    }
}
```

❖ Type Casting:

Type casting in C# is the process of converting a variable from one type to another. According to our needs, we can change the type of data. In C#, we can perform a different kinds of conversions.

1) Implicit Conversion
2) Explicit Conversion
3) Type Conversion Methods

1) Implicit Conversion:

Implicit Casting in C# (also known as automatic conversion) is a type conversion that happens automatically when you assign a value of one data type to a variable of another compatible data type. Implicit casting is safe and does not require any special syntax because the conversion is guaranteed to succeed without data loss.

Converting a smaller type to a larger type size:

char -> int -> long -> float -> double

Example:

```
int num = 123;
double largeNum = num; // Implicit casting from int to double
Console.WriteLine(largeNum); // Output: 123
```

2) **Explicit Conversion: Manual Conversion**

Explicit casting is called as manual conversion you must explicitly specify the type you want to convert to, using the cast operator (type). Explicit conversion will be done with the cast operator (). Converting a larger type to a smaller size type.

double -> float -> long -> int -> char

Example:

```
double largeNum = 123.45;
int num = (int)largeNum; // Explicit casting from double to int
Console.WriteLine(num); // Output: 123 (fractional part is lost)
```

3) **Type Conversion Methods:**

It is also possible to convert data types explicitly by using different built-in methods.

C# provides built-in methods for Type-Conversions as follows

Method	Description
ToBoolean	It will converts a type to Boolean value
ToChar	It will converts a type to a character value
ToByte	It will converts a value to Byte Value
ToDecimal	It will converts a value to Decimal point value
ToDouble	It will converts a type to double data type
ToInt16	It will converts a type to 16-bit integer
ToInt32	It will converts a type to 32 bit integer
ToInt64	It will converts a type to 64 bit integer
ToString	It will converts a given type to string
ToUInt16	It will converts a type to unsigned 16 bit integer
ToUInt32	It will converts a type to unsigned 32 bit integer
ToUInt64	It will converts a type to unsigned 64 bit integer

Example:

string str = "123";

int num = Convert.ToInt32(str); // Converts string to int

Console.WriteLine(num); // Output: 123

❖ C# boxing and unboxing:

C# Type System contains three Types; they are Value Types, Reference Types and Pointer Types. C# allows us to convert a Value Type to a Reference Type, and back again to Value Types. The operation of converting a value type to a reference type is called Boxing and the operation of converting reference type to value type is called Unboxing.

1) **Boxing:**

Example:

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int num = 1;
            Object Obj = num; //Boxing
            Console.Write("Value is :" + Obj);
            Console.ReadKey();
        }
    }
}
```

The first line we created a Value Type num and assigned a value to num. The second line, we created an instance of Object Obj and assign the value of num to Obj. This process of converting a value of a Value Type into a value of a corresponding Reference Type is called Boxing.

2) **UnBoxing:**

Example:

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            object obj = 123; // Boxing
            int num = (int)obj; // Unboxing: obj is converted back to int
            Console.WriteLine(num); // Output: 123
            Console.ReadKey();
        }
    }
}
```

The first two line shows how to Box a Value Type. The next line (int num = (int) Obj) shows extracts the Value Type from the Object. That is converting a value of a Reference Type into a value of a Value Type. This operation is called UnBoxing.

❖ Partial Class Definition:

- A partial class in C# allows a class to be split into multiple files, with each file containing a portion of the class definition
- Each source file contains a section of the class definition, and all parts are combined when the application is compiled.
- This feature is particularly useful in large projects where multiple developers might be working on the same class.
- A partial class is created by using a **partial** keyword.
- This keyword is also useful to split the functionality of methods, interfaces, or structure into multiple files.

Important Points:

1) All the partial definitions must proceed with the key word "Partial".
2) Method signatures (return type, name of the method, and parameters) must be unique for the aggregated typed (which was defined partially).
3) The partial types must have the same accessibility.
4) If any part is sealed, the entire class is sealed.
5) If any part is abstract, the entire class is abstract.
6) Inheritance at any partial type applies to the entire class.

Example:

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Calculator calculator = new Calculator();

            int n1 = calculator.Add(10, 5);
            int n2 = calculator.Subtract(10, 5);
            int n3 = calculator.Multiply(10, 5);
            int n4 = calculator.Divide(10, 5);

            Console.WriteLine("Sum: "+n1);            // Output: Sum: 15
            Console.WriteLine("Difference: "+n2);  // Output: Difference: 5
            Console.WriteLine("Product: "+n3);    // Output: Product: 50
            Console.WriteLine("Quotient: "+n4);  // Output: Quotient: 2
            Console.ReadKey();
        }
    }
    public partial class Calculator
    {
        public int Add(int a, int b)
        {
            return a + b;
        }

        public int Subtract(int a, int b)
        {
            return a - b;
        }
    }
    public partial class Calculator
    {
        public int Multiply(int a, int b)
        {
            return a * b;
        }

        public int Divide(int a, int b)
        {
            return a / b;
        }
    }
}
```

❖ Control Statements in C#:

Control statements in C# are used to control the flow of execution in a program. Every programming language, basically, has three types of control statements.

1) Conditional Statements (Decision)
2) Iterative Statements (Loop)
3) Jumps Statements.

1) Conditional Statements (Decision):

Conditional statements allow your program to make decisions based on certain conditions. Conditional statement results in either True or False. Following are the different types of Conditional Statements:

i) Simple if Statement   ii) if …else   iii) else-if Ladder

iv) Nested if else Statement   v) switch Statement

i) Simple if statement:

The simple if statement executes a block of code only if a specified condition is true.

Syntax:

```
if (condition)
{
    // Code to execute if condition is true
}
```

Example:

```
int a=10;
if (a > 5)
{
    Console.WriteLine("a is greater than 5");
}
```

ii) If…else Statement:

The `if-else` statement executes one block of code if the condition is true, and another block if the condition is false.

Syntax:

```
if (condition)
{
        // Code to execute if condition is true
}
else
{
        // Code to execute if condition is false
}
```

Example:

```
int a = 10;
if (a > 15)
{
   Console.WriteLine("a is greater than 15");
}
else
{
   Console.WriteLine("a is less than or equal to 15");
}
```

iii) else-if Ladder:

The `else-if` ladder is used to test multiple conditions. If the first condition is false, the next condition is checked, and so on.

Syntax:

```
if (condition1)
{
   // Code to execute if condition1 is true
}
else if (condition2)
{
   // Code to execute if condition2 is true
}
else
{
   // Code to execute if all conditions are false
}
```

Example:

```
int a = 10;
if (a > 15)
{
     Console.WriteLine("a is greater than 15");
}
else if (a > 5)
{
    Console.WriteLine("a is greater than 5 but less than or equal to 15");
}
else
{
    Console.WriteLine("a is 5 or less");
}
```

iv) Nested if…else Statement:

An if...else statement can exist within another if...else statement. Such statements are called nested if...else statement. Nested if statements are generally used when we have to test one condition followed by another.

Syntax:

```
if (condition1)
{
   // Code to execute if condition1 is true

   if (condition2)
   {
      // Code to execute if both condition1 and condition2 are true
   }
   else
   {
      // Code to execute if condition1 is true but condition2 is false
   }
}
else
{
   // Code to execute if condition1 is false
}
```

Example:

```
int number = 10;
if (number > 0)
{
   Console.WriteLine("The number is positive.");

   if (number % 2 == 0)
   {
      Console.WriteLine("The number is even.");
   }
   else
   {
      Console.WriteLine("The number is odd.");
   }
}
else
{
   Console.WriteLine("The number is not positive.");
}
```

v) switch Statement:

In C#, Switch statement is a multiway branch statement. The switch statement tests the value of a given variable (or expression) against a list of case values and when a match is found, a block of statements associated with that case executed.

Syntax:

```
switch(expression)
{
     case value1:
     //code to be executed;
      break;
     case value2:
     //code to be executed;
     break;
     ......
     default:
     //code to be executed if all cases are not matched;
     break;
}
```

Example:

```
int number = 2;
switch (number)
{
   case 1:
      Console.WriteLine("Number is One");
      break;
   case 2:
      Console.WriteLine("Number is Two");
      break;
   case 3:
      Console.WriteLine("Number is Three");
      break;
   default:
      Console.WriteLine("Invalid Number");
      break;
}
```

2) Iterative Statements (Loops/Repetitive):

Iterative statements, also known as loops or repetitive control structures, are used in programming to repeat a block of code multiple times. The execution of these statements continues until a specified condition becomes true.

Types of Iterative Statements in C#:

i) for loop	ii) while loop
iii) do..while loop	iv) for each loop

i) for loop:

A for loop in C# is a control structure that allows you to repeat a block of code a specific number of times. The basic structure of a for loop in C# includes three parts:

- **Initialization:** Sets up a loop counter or variable that will be used to control the loop.
- **Condition:** Tests the loop counter against a condition; if true, the loop continues; if false, the loop stops.
- **Increment/Decrement:** Updates the loop counter after each iteration.

Syntax:

```
for (initialization; condition; increment/decrement)
{
     // Code to be executed repeatedly
}
```

Example:

```
using System;
class Program
{
   static void Main()
   {
      // A simple for loop that counts from 0 to 4
      for (int i = 0; i < 5; i++)
      {
         Console.WriteLine("i is " + i);
      }
   }
}
```

ii) while loop:

The while is an entry-controlled loop statement. The test condition is evaluated and if the condition is true, then the body of the loop is executed. This process of repeated execution of the body continues until the test condition finally becomes false and the control is transferred out of the loop. The body of the loop may have one or more statements.

Syntax:

```
while (condition)
{
     // Code to execute repeatedly
}
```

Example:

```
int i = 0;
while (i < 5)
{
     Console.WriteLine(i); i++;
}
```

iii) The do Statement:-

The do statement program proceeds to evaluate the body of the loop first. At the end of the loop, the test condition in the while statement is evaluated. If the condition is true, the program continues to evaluate the body of the loop once again. This process continues as long as the condition is true. When the condition becomes false, the loop will be terminated. The do....while construct provides an exit-controlled loop and therefore the body of the loop is always executed at least once.

Syntax:

```
do
{
    // Code to execute repeatedly
} while (condition);
```

Example:

```
int i = 0;
do
{
    Console.WriteLine(i);
    i++;
}while (i < 5);
```

iv) foreach loop:

The foreach loop is used to iterate over elements in a collection or array. It is particularly useful when you want to access each element in a collection without needing to know the index.

Syntax:

```
foreach (type variable in collection)
{
    // Code to execute repeatedly
}
```

Example:

```
string[] fruits = { "Apple", "Banana", "Cherry" };
foreach (string fruit in fruits)
{
    Console.WriteLine(fruit);
}
```

3) Jump Statements:

Jumping statements are control statements that transfer execution control from one point to another point in the program. These statements can be used to exit loops, skip iterations, return from methods, or jump to specific code locations.

Types of Jump Statements.

i) Break Statement   ii) Continue statement.   iii) goto Statements

i) Break Statement:

Break statement is used to terminate a loop. When a break statement is encounters in the loop then loopis terminated and control is transferred to the statement which appears immediately after the loop.

Syntax: break;

Example:

```
for (int i = 0; i < 10; i++)
{
        if (i == 5)
        break; // Exit the loop when i equals 5
        Console.WriteLine(i);
}
```

ii) Continue Statement:

When continue statement encounters in the loop then remaining statement in the loop are ignored andcontrol is transfer back to condition of loop.

Syntax: continue;

Example:

```
for(i=1;i<=5;i++)
{
        if(i==3)
        {
                continue;
        }
        Console.WriteLine(i);
}
```

iii) Goto Statement:

The C# goto statement is also known jump statement. It is used to transfer control to the other part of the program. It unconditionally jumps to the specified label.

Syntax:

goto label;

label:

// Code to execute when the label is reached

Example:

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            // label
            repeat:
            Console.WriteLine("Enter a number less than 10");
```

```
                int num = Convert.ToInt32(Console.ReadLine());

                if (num >= 10)
                {
                    // transfers control to repeat
                    goto repeat;
                }
                Console.WriteLine(num + " is less than 10");
                Console.ReadKey();
            }
        }
    }
```

****

UNIT-III

ASP.NET

❖ ASP.NET Introduction:

- ASP stands for Active Server Pages. ASP.NET is a web application framework developed by Microsoft that enables developers to build web applications.
- ASP.NET is a server side scripting technology that enables to be executed by an Internet server.
- ASP.NET is a Microsoft Technology
- ASP.NET is a program that runs inside IIS
- IIS (Internet Information Services) is Microsoft's Internet server
- IIS comes as a free component with Windows servers
- ASP.NET is a managed framework that facilitates building server-side applications.
- .NET is language independent, which means you can use any .NET supported language to make .NETapplications.
- The most common languages for writing ASP.NET applications are C# and VB.NET.
- VB.NET is directly based VB (Visual Basic), C# was introduced together with the .NET framework,and is therefore a somewhat new language.

❖ Server Controls in ASP.NET:

ASP.NET server controls are components used in ASP.NET web applications to generate dynamic content on web pages. Server controls are tags that are understood by the server. These controls are basically the original HTML controls but enhanced to enable server side processing.

Types of Server Controls:

There are three types of server controls:

1) HTML Server Controls - Traditional HTML tags
2) Web Server Controls - New ASP.NET tags
3) Validation Server Controls - For input validation

1) HTML Server Controls:

HTML server controls are HTML tags understood by the server. These are standard HTML elements that are converted to server controls by adding the runat="server" attribute. This attribute indicates that the element should be treated as a server control. The id attribute is added to identify the server control.

Note: All HTML server controls must be within a <form> tag with the runat="server" attribute. The runat="server" attribute indicates that the form should be processed on the server.

Example:

```
<body>
<form name="frmWelcome" runat="server">
Welcome to ASP.NET
<input type="text" runat="server" id="txtName" />
</form>
</body>
```

2) Web Server Controls:

Web server controls are special ASP.NET tags understood by the server. Like HTML server controls, Web server controls are also created on the server and they require a runat="server" attribute to work. Mostly all Web Server controls inherit from a common base class, namely the WebControl class defined in the System.Web.UI.WebControls namespace.

The syntax for creating a Web server control is:

<asp:control_name id="id" runat="server" />

Example:

```
<html>
<body>
    <form runat="server">
    <asp:Button id="button1" Text="Click me!" runat="server" OnClick="submit"/>
    </form>
</body>
</html>
```

Some server-side controls used in ASP.NET:

1) Label:

The Label control is used to display static or dynamic text on a web page. It does not accept user input.

Important Properties:

- **Text**: The text content of the label.
- **ForeColor**: The color of the text.
- **BackColor**: The background color of the label.

Important Events:

- **None**: The Label control does not have any event handlers, as it's typically used for displaying information only.

Example:

```
<asp:Label ID="lblMessage" runat="server" Text="Welcome!" ForeColor="Blue" />
```

2) TextBox:

The TextBox control allows users to input text. It can be used for entering single-line or multi-line text input.

Important Properties:

- **Text**: Gets or sets the text within the TextBox.
- **TextMode**: Specifies the behavior of the TextBox (e.g., SingleLine, MultiLine, Password).
- **MaxLength**: Sets the maximum number of characters allowed in the TextBox.
- **ReadOnly**: Makes the TextBox read-only if set to true.
- **Enabled**: Enables or disables the TextBox.

Important Events:

- **TextChanged**: Triggered when the content of the TextBox changes and the control loses focus or a postback occurs.

Example:

```
<asp:TextBox ID="txtInput" runat="server" TextMode="SingleLine"
MaxLength="100" />
```

3) Button:

The Button control is used to perform an action when clicked by the user. It typically triggers a postback to the server, where server-side logic can be executed.

Important Properties:

- Text: The text displayed on the button.
- Enabled: Specifies whether the button is enabled (true) or disabled (false).

Important Events:

- Click: Fired when the button is clicked.

Example:

```
<asp:Button ID="btnSubmit" runat="server" Text="Submit" OnClick="btnSubmit_Click"
/>
```

4) **CheckBox:**

The CheckBox control allows users to make binary choices (checked or unchecked).

Important Properties:

- Checked: Specifies whether the checkbox is checked (true) or not (false).
- Text: The label displayed next to the checkbox.

Important Events:

- CheckedChanged: Triggered when the user changes the state of the checkbox.

Example:

<asp:CheckBox ID="chkAgree" runat="server" Text="I agree to the terms and conditions" />

5) **RadioButton**

The RadioButton control allows the user to select a single option from a group of choices.

Important Properties:

- Checked: Determines whether the radio button is selected (true).
- GroupName: Specifies a group to which the radio button belongs. Only one button in the group can be selected at a time.
- Text: The label displayed next to the radio button.

Important Events:

- CheckedChanged: Triggered when the user selects or deselects the radio button.

Example:

<asp:RadioButton ID="rbMale" runat="server" Text="Male" GroupName="Gender" />
<asp:RadioButton ID="rbFemale" runat="server" Text="Female" GroupName="Gender" />

6) **DropDownList:**

The DropDownList control allows the user to select a single item from a list of options.

Important Properties:

Items: Represents the collection of ListItem objects in the DropDownList.

- SelectedIndex: Gets or sets the index of the selected item.
- SelectedItem: Gets the currently selected item.
- SelectedValue: Gets or sets the value of the selected item.

Important Events:

- SelectedIndexChanged: Triggered when the selected item in the DropDownList changes.

Example:

```
<asp:DropDownList ID="ddlOptions" runat="server">
   <asp:ListItem Text="Option 1" Value="1" />
   <asp:ListItem Text="Option 2" Value="2" />
</asp:DropDownList>
```

7) **GridView:**

The GridView control displays tabular data in rows and columns, with support for features like sorting, paging, and editing.

Important Properties:

- AutoGenerateColumns: Determines whether columns are automatically created based on the data source.
- Columns: Specifies the columns to display (if AutoGenerateColumns is false).
- DataSource: Specifies the source of data for the GridView (e.g., a DataTable, List, etc.).
- AllowPaging: Enables paging if set to true.
- AllowSorting: Enables sorting if set to true.

Important Events:

- RowCommand: Triggered when a button in a GridView row is clicked.
- SelectedIndexChanged: Triggered when the selected row index changes.
- RowDataBound: Occurs when a row is being bound to data, useful for customizing row content.

Example:

```
<asp:GridView ID="gvUsers" runat="server" AutoGenerateColumns="False"
AllowSorting="True">
   <Columns>
      <asp:BoundField DataField="UserID" HeaderText="User ID" />
      <asp:BoundField DataField="UserName" HeaderText="User Name" />
      <asp:BoundField DataField="Email" HeaderText="Email" />
   </Columns>
</asp:GridView>
```

8) **HyperLink:**

The HyperLink control represents a link to another web page or resource.

Important Properties:

- NavigateUrl: Specifies the URL to which the link should navigate.
- Text: The text displayed for the hyperlink.

Important Events:

- None: The HyperLink control does not raise postback events.

Example:

```
<asp:HyperLink ID="hlGoogle" runat="server" NavigateUrl="https://www.google.com"
Text="Go to Google" />
```

9) LinkButton:

The LinkButton control in ASP.NET is similar to a hyperlink, but it functions like a button in that it triggers a postback to the server. When clicked, it behaves as a button that performs server-side processing, but it appears as a standard HTML hyperlink.

Important Properties:

- Text: The text displayed for the LinkButton.
- CommandName: A string that represents the command to execute. Often used for handling multiple buttons with a common event handler.
- CommandArgument: Additional information about the command. It's often used to pass extra data when multiple buttons share the same event handler.
- PostBackUrl: Specifies the URL to post back to when the button is clicked.

Important Events:

- Click: Triggered when the LinkButton is clicked.
- Command: Triggered when the LinkButton is clicked. Useful for distinguishing between different buttons in a GridView or Repeater.

Example:

```
<asp:LinkButton ID="lnkSubmit" runat="server" Text="Click Me"
OnClick="lnkSubmit_Click" />
```

10) ImageButton:

The ImageButton control in ASP.NET is a button that displays an image instead of the standard button. When clicked, it behaves like a regular button (causing a postback to the server), but visually it appears as an image.

Important Properties:

- ImageUrl: The URL of the image displayed on the button.

- AlternateText: The text displayed when the image is unavailable or for accessibility (screen readers).
- PostBackUrl: Specifies the URL to post to when the ImageButton is clicked (useful for cross-page postback).
- CommandName: Used to specify a command for command events.
- CommandArgument: Provides additional data about the command (similar to LinkButton).

  Important Events:
- Click: Triggered when the ImageButton is clicked.
- Command: Used when the button is part of a group of buttons, and specific commands are needed.
- OnClientClick: Allows you to specify client-side JavaScript that runs when the button is clicked (before the postback).
- Special Event for ImageButton:
- OnClick: The ImageButton has an additional feature: it provides the X and Y coordinates of the click on the image. These are the pixel coordinates on the image where the user clicked.

Example:

```
<asp:ImageButton ID="imgBtnSubmit" runat="server"
ImageUrl="~/Images/submit_button.png" OnClick="imgBtnSubmit_Click" />
<asp:Label ID="lblMessage" runat="server" />
```

3) Validation Server Controls:

Validation server controls are used to validate user-input. If the user-input does not pass validation, it will display an error message to the user. It provides a way to check the accuracy and validity of user input before the web application processes it. ASP.NET provides the following validation controls:

1) RequiredFieldValidator 2) RangeValidator

3) CompareValidator 4) RegularExpressionValidator

5) CustomValidator.

Example:

```
<html>
<body>
<form runat="server">
```

```
      Enter a number from 1 to 100:
      <asp:TextBox id="tbox1" runat="server" /><br /><br />
      <asp:Button Text="Submit" runat="server" />
      <asp:RangeValidator ControlToValidate="tbox1" MinimumValue="1"
        maximumValue="100" Type="Integer" Text="The value must be from 1 to 100!"
        runat="server" />
   </form>
   </body>
   </html>
```

❖ Validation Controls:-

ASP.NET provides a powerful set of validation controls that ensure the data entered by users is valid or not before submitting it to the server. ASP.NET provides controls that automatically check user input and require no extra code for validation. Validation server controls are used to validate user-input. If the user-input does not pass validation, it will display an error message to the user. ASP.NET provides the following validation controls:

ASP.NET provides the following validation controls:

1) RequiredFieldValidator	2) RangeValidator
3) CompareValidator	4) RegularExpressionValidator
5) CustomValidator	

1) RequiredFieldValidator:

The RequiredFieldValidator control ensures that the required field is not empty. RequiredFieldValidator control is used to make an input control as a mandatory field orthe input field should not be empty. It will throw an error if user leaves input control empty. Some important properties of RequiredFieldValidator are:

- **ControlToValidate:** This control is used to set the field of the text box for validation.
- **ErrorMessage:** The error message displayed if the validation fails.
- **InitialValue:** Initial value is displayed by default to guide the users on how the value must be added. This property is also used by the developers for a drop-down list.

Example:

```
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
<asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server"
   ControlToValidate="TextBox1"
   ErrorMessage="This field is required.">
</asp:RequiredFieldValidator>
```

2) RangeValidator:

It is used to check whether the value of the input control is inside a particular range or not.

It confirms that user input data is within a specified range of values, if the input value is not between a minimum and maximum value, it will give error.

Some important properties of RangeValidator is:

- **Type:** It defines the type of the data. The available values are: Currency, Date, Double, Integer, and String.
- **MinimumValue:** It specifies the minimum value of the range.
- **MaximumValue:** It specifies the maximum value of the range.
- **ErrorMessage:** The error message displayed if the validation fails.

Example:

```
<asp:TextBox ID="AgeTextBox" runat="server"></asp:TextBox>
<asp:RangeValidator ID="RangeValidator1" runat="server"
   ControlToValidate="AgeTextBox"
   MinimumValue="18" MaximumValue="65"
   Type="Integer"
   ErrorMessage="Please enter a number between 18 and 65.">
</asp:RangeValidator>
```

3) CompareValidator:

The CompareValidator control compares a value in one control with a fixed value or a value in another control. It checks if the values are equal, not equal, greater than, less than, or within other comparative operations. It is also commonly used to ensure the value entered is of a specific data type. It has the following specific properties:

- **Type:** It specifies the data type.
- **ControlToValidate:** This control is used to set the field of the text box for validation.
- **ControlToCompare:** It specifies the value of the input control to compare with
- **ValueToCompare:** It specifies the constant value to compare with.
- **Operator:** It specifies the comparison operator, the available values are: Equal, NotEqual, GreaterThan, GreaterThanEqual, LessThan, LessThanEqual, and DataTypeCheck.

Example:

```
<asp:TextBox ID="Password" runat="server" TextMode="Password"></asp:TextBox>
<asp:TextBox ID="ConfirmPassword" runat="server"
TextMode="Password"></asp:TextBox>
```

```
<asp:CompareValidator ID="CompareValidator1" runat="server"
    ControlToValidate="ConfirmPassword"
    ControlToCompare="Password"
    ErrorMessage="Passwords do not match.">
</asp:CompareValidator>
```

4) RegularExpressionValidator:

The RegularExpressionValidator allows validating the input text by matching against a pattern of a regular expression. The most important property is the ValidationExpression property.

Example:

```
<asp:TextBox ID="EmailTextBox" runat="server"></asp:TextBox>
<asp:RegularExpressionValidator ID="RegexValidator1" runat="server"
    ControlToValidate="EmailTextBox"
    ValidationExpression="^\w+@[a-zA-Z_]+?\.[a-zA-Z]{2,3}$"
    ErrorMessage="Please enter a valid email address.">
</asp:RegularExpressionValidator>
```

5) CustomValidator:

The CustomValidator control allows writing application specific custom validationroutines for both the client side and the server side validation. The client side validation is accomplished through the ClientValidationFunction property. The client side validation routine should be written in a scripting language, such as JavaScript or VBScript, which the browser can understand.

The server side validation routine must be called from the control's ServerValidate event handler. The server side validation routine should be written in any .Net language, like C# or VB.Net.

Example: ASP.NET Code

```
Enter Your Mobile Number:<asp:TextBox ID="TextBox1"
runat="server"></asp:TextBox>
<asp:CustomValidator ID="CustomValidator1" runat="server" ErrorMessage="Please
Enter 10 digit Number" ControlToValidate="TextBox1"
onservervalidate="CustomValidator1_ServerValidate"></asp:CustomValidator>
```

C#.NET Code

```
protected void CustomValidator1_ServerValidate(object source,
ServerValidateEventArgs args)
{
        if (args.Value.Length == 10)
        {
            args.IsValid = true;
        }
        else
        {
            args.IsValid = false;
        }
}
```

6) ValidationSummary:

The ValidationSummary control does not perform any validation but shows a summary of all errors in the page. The summary displays the values of the ErrorMessage property of all validation controls that failed validation.

The following two properties list out the error message:

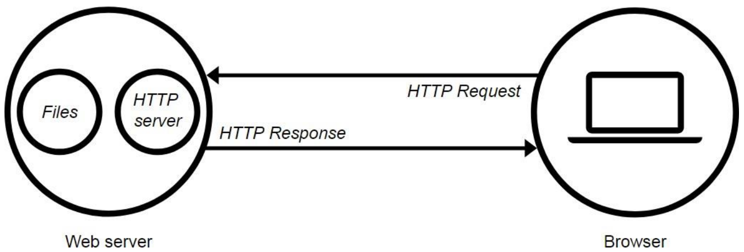
- ShowSummary : shows the error messages in specified format.
- ShowMessageBox : shows the error messages in a separate window.

Example:

<asp:ValidationSummary ID="ValidationSummary1" runat="server" DisplayMode = "BulletList" ShowSummary = "true" HeaderText="Errors:" />

❖ Web server:

- Definition: A web server is a computer that runs websites. A web server is also known as an internetserver.
- Web server can refer to either the hardware (the computer) or the software (the computer application) that helps to deliver content that can be accessed through the Internet.



- The primary function of a web server is to deliver web pages on the request to clients.
- This means delivery of HTML documents and any additional content that may be included by adocument, such as images, style sheets and scripts.
- When client sends request for a web page, the web server search for the requested page ifrequested page is found then it will send it to client with an HTTP response.

- If the requested web page is not found, web server will the send an HTTP response: Error 404 Notfound.

❖ Web browser:

- **Definition**: - A web browser, or simply "browser," is an application used to access andview websites.
- Common web browsers include Microsoft Internet Explorer, Google Chrome, Mozilla Firefox,and Apple Safari.
- A web browser is a software program that allows a user to locate, access, and display web pages.
- In common usage, a web browser is usually shortened to "browser." Browsers are used primarilyfor displaying and accessing websites on the internet,
- An information resource is identified by a Uniform Resource Identifier (URI) and may be a webpage, image, video, or other piece of content.
- The major web browsers are Internet Explorer, Firefox, Google Chrome, Safari, and Opera.

❖ Web Forms Page Life Cycle

- The **ASP.NET Web Forms life cycle** is the process that happens when a user requests a web page until it is fully displayed in the browser.
- These include initialization, instantiating controls, restoring and maintaining state, running eventhandler code, and rendering.
- In general, the life cycle for a Web Forms page is similar to that of any Web process that runs onthe server.
- It is important to understand the sequence of events that occur when a Web Forms page isprocessed.
- In order to memorize whole page life cycle process of Asp.net, keep store this word "SILVER" inyour memory, which is defined as

  S = Start

  I = Initialization

  L = Load

  V = Validate

  E = Event Handlers

  R = Render

- **Page request:** The life cycle begins when the user requests a page. Page Request is the first step of the page life cycle. When a user request is made, the server checks the request and compiles the pages. Once the compilation is done, the request is sent back to the user

- **Starting of page life cycle:** Page Start helps in creating two objects: request and response. The request holds all the information which the user sent, while the response contains all the information that is sent back to the user.
- **Page initialization -** At this stage, the controls on the page are assigned unique ID by setting the UniqueID property and the themes are applied. For a new request, postback data is loaded and the control properties are restored to the view-state values.
- **Page load:** Page Load helps to load all the control properties of an application. It also helps to provide information using view state and control state.
- **Validation:** Validate method of the validation control is called and on its successful execution, the IsValid property of the page is set to true. Validation happens when the execution of an application is successful. It returns two conditions: true and false. If execution is successful, it returns true, otherwise false. If validation controls (like RequiredFieldValidator) are used, they check for valid user inputs during this phase.
- **Postback event handling:** If the request is a postback (old request), the related event handler is invoked. If this is a postback request, events such as button clicks are handled. For example, if a user clicks a button, the corresponding event handler (like Button_Click) is called.
- **Page rendering -** At this stage, view state for the page and all controls are saved. The page calls the Render method for each control and the output of rendering is written to the OutputStream class of the Response property of page.
- **Unload:** Unload is a process that helps in deleting all unwanted information from the memory once the output is sent to the user.

❖ Navigation controls:

Navigation controls are very important for websites. Navigation controls are basically used to navigate the user through webpage. Navigation controls in ASP.NET are user interface components designed to help users navigate through different parts of a web application. These controls are typically used to provide easy access to different pages of the website. There are three controls in ASP.NET, Which are used for Navigation on the webpage.

1) TreeView control
2) Menu Control

3) SiteMapPath control

1) The TreeView Control:

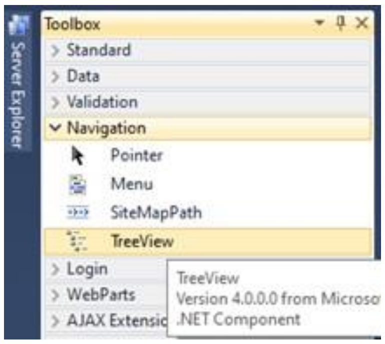
The **TreeView control** in ASP.NET is used to display hierarchical data, such as a site structure, categories, file systems, or any other information that can be organized in a tree-like format. It allows users to expand and collapse nodes to view or hide nested items. Each node in the TreeView can represent a parent or child item.

Properties of TreeView Control:

- DataSourceID: This property is used to specify the data source to be used using sitemap files data source.
- ShowLines: This property is used to specify the lines to connect the individual item in the tree.
- NodeStyle: Defines the appearance for all nodes (e.g., font, color, etc.).
- CssClass: This property is used to specify the CSS class attribute for the control.
- ExpandDepth: This property is used to specify the level at which items in the tree are expanded.
- NavigateUrl: Specifies the URL that a node will navigate to when clicked.
- HoverNodeStyle: Defines the appearance of a `TreeNode` when hovered.
- SelectedNodeStyle: Sets the appearance of the selected `TreeNode`.

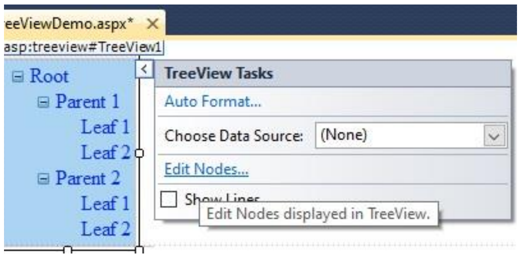
Here are the steps to create a simple TreeView in ASP.NET using the Design View:

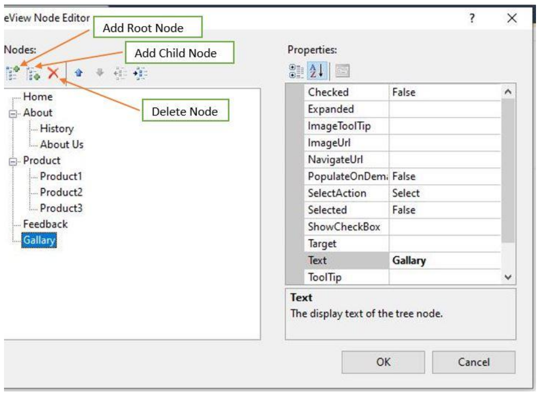
1. Open Design View: In Visual Studio, open your .aspx file and switch to the Design View.
2. Drag and Drop TreeView: From the Toolbox (under the "Navigation" section), drag the TreeView control and drop it onto the page.



3. Add TreeView Nodes:
   - Right-click the TreeView in the design view and select Edit Nodes.
   - In the TreeView Tasks menu, click Edit Nodes.
   - In the dialog that appears, click Add Root Node to add the parent node.

- Select the parent node, and click Add Child Node to add child nodes.
- Enter the text for each node.



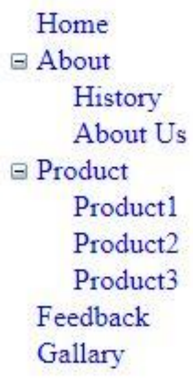


Checked	False
Expanded	
ImageToolTip	
ImageUrl	
NavigateUrl	
PopulateOnDem	False
SelectAction	Select
Selected	False
ShowCheckBox	
Target	
Text	Gallary
ToolTip	

Text
The display text of the tree node.

4. Save and Run: Save your changes and run the application to test the TreeView structure.

```
<asp:TreeView ID="TreeView1" runat="server">
    <Nodes>
        <asp:TreeNode Text="Home" Value="Home"></asp:TreeNode>
        <asp:TreeNode Text="About" Value="About">
            <asp:TreeNode Text="History" Value="History"></asp:TreeNode>
            <asp:TreeNode Text="About Us" Value="About Us"></asp:TreeNode>
        </asp:TreeNode>
        <asp:TreeNode Text="Product" Value="Product">
            <asp:TreeNode Text="Product1" Value="Product1"></asp:TreeNode>
            <asp:TreeNode Text="Product2" Value="Product2"></asp:TreeNode>
            <asp:TreeNode Text="Product3" Value="Product3"></asp:TreeNode>
        </asp:TreeNode>
        <asp:TreeNode Text="Feedback" Value="Feedback"></asp:TreeNode>
        <asp:TreeNode Text="Gallary" Value="Gallary"></asp:TreeNode>
    </Nodes>
</asp:TreeView>
```



```
 Home
⊟ About
     History
     About Us
⊟ Product
     Product1
     Product2
     Product3
 Feedback
 Gallary
```

2) The Menu Control:-

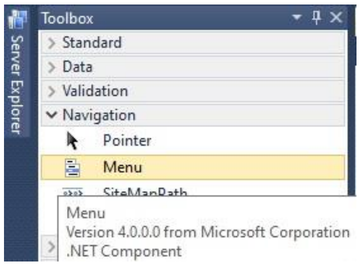
Menu is a navigation control in ASP.NET which is used to display menu in web page. This can be used to display the site data structure vertically and horizontally. It displays two types of menu static menu and dynamic menu. Static menu is always displayed in menu Control, by default only menu items at the root levels are displayed. Dynamic menu is displayed only when the use moves the mouse pointer over the parent menu that contains a dynamic sub menu.

Properties of Menu Control:

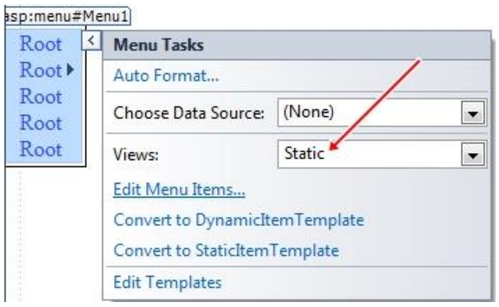
- DataSourceID: This property is used to specify the data source to be used using sitemap file as data source.
- CssClass: This property is used to specify the CSS class attribute for the control.
- ImgeUrl: This property is used to specify the image that appear next to the menu item.
- Orientation: This property is used to specify the alignment of menu control. It can be horizontal or vertical.
- Tooltip: This property is used to specify the tooltip of the menu item when you mouse over.
- Text: This property is used to specify the text to display in the menu.
- NavigateUrl: This property is used to specify the target location to send the user when menu item is clicked.
- Target: This property is used to specify the target page location. It can be in new window or same window. **_self**: Opens the link in the same frame. **_blank**: Opens the link in a new window or tab.
- Value: This property is used to specify the unique id to use in server side events.

Here are the steps to create a simple Menu control in ASP.NET using Design View:

1. Open Design View: In Visual Studio, open your .aspx file and switch to Design View.
2. Drag and Drop Menu Control: From the Toolbox (under "Navigation"), drag the Menu control and drop it onto the page.

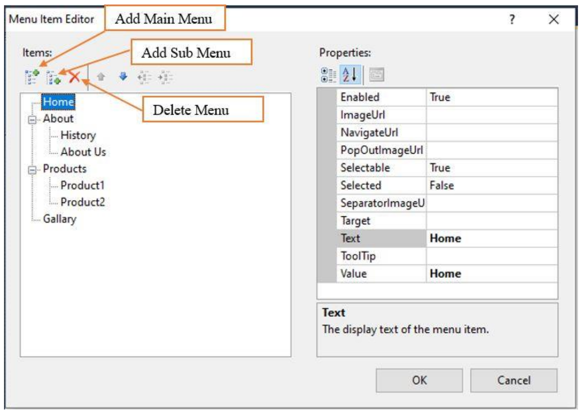


3. Add Menu Items:
   - Right-click the Menu control in Design View and select Edit Menu Items.
   - In the MenuItem Collection Editor, click Add to create a root-level menu item.
   - Set properties such as Text (e.g., "Home", "Products", etc.), Value, and NavigateUrl for the root item.
   - To add sub-items (child menu items), select a root menu item and click Add again to create child items under it.
   - Repeat to add additional menu items or sub-items.



Choose Data Source: (None)

Views: Static

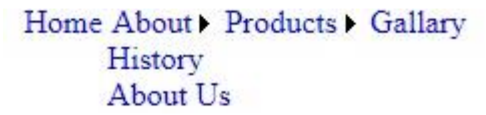


Enabled	True
ImageUrl	
NavigateUrl	
PopOutImageUrl	
Selectable	True
Selected	False
SeparatorImageU	
Target	
Text	Home
ToolTip	
Value	Home

Text
The display text of the menu item.

4. Customize Menu Appearance: Use the Properties window to adjust settings such as Orientation (Horizontal/Vertical).
5. Run and Test: Save your changes and run the application to see the Menu and ensure it displays and functions correctly.

```
<asp:Menu ID="Menu1" runat="server" Orientation="Horizontal">
    <Items>
        <asp:MenuItem Text="Home" Value="Home"></asp:MenuItem>
        <asp:MenuItem Text="About" Value="About">
            <asp:MenuItem Text="History" Value="History"></asp:MenuItem>
            <asp:MenuItem Text="About Us" Value="About Us"></asp:MenuItem>
        </asp:MenuItem>
        <asp:MenuItem Text="Products" Value="Products">
            <asp:MenuItem Text="Product1" Value="Product1"></asp:MenuItem>
            <asp:MenuItem Text="Product2" Value="Product2"></asp:MenuItem>
        </asp:MenuItem>
        <asp:MenuItem Text="Gallary" Value="Gallary"></asp:MenuItem>
    </Items>
</asp:Menu>
```



3) The SiteMapPath Control:

Site maps are XML files which are mainly used to describe the logical structure of the web application. It defines the layout of all pages in web application and how they relate to each other. Whenever you want you can add or remove pages to your site map there by managing navigation of website efficiently. Site map files are defined with .sitemap extension. <sitemap> element is the root node of the sitemap file.

It has three attributes:

- Title: It provides textual description of the link.
- URL: It provides the location of the valid physical file.
- Description: It is used for tooltip of the link.

Properties of SiteMapPath Control:

- PathSeparator: This property is to get or set the out separator text.
- NodeStyle: This property is used to set the style of all nodes that will be displayed.
- RootNodeStyle: This property is used to set the style on the absolute root node.
- PathDirection: This property is used to set the direction of the links generated in the output.
- CurrentNodeStyle: This property is used to set the style on node that represent the current page.
- ShowToolTips: This property is used to set the tooltip for the control. Default value is true.
- PathSeparatorStyle: This property is used to set the style of path separator.

Here are the steps to create a SiteMapPath **control in ASP.NET using Design View:**

1) **Create a Web.sitemap File**:
   - In Solution Explorer, right-click the project, choose **Add > New Item**.
   - Select **Site Map** and name it `Web.sitemap`.
   - Add site navigation nodes in the `Web.sitemap` file:

```
<siteMap>
 <siteMapNode url="~/Default.aspx" title="Home" description="Home Page">
  <siteMapNode url="~/Products.aspx" title="Products" />
  <siteMapNode url="~/Contact.aspx" title="Contact" />
 </siteMapNode>
</siteMap>
```

2) **Open Design View**: Open your `.aspx` file in Visual Studio and switch to **Design View**.
3) **Drag and Drop SiteMapPath Control**:
   - From the **Toolbox** (under "Navigation"), drag the `SiteMapPath` control to the design surface.
   - The control will automatically pick up the navigation structure from `Web.sitemap`.
4) **Customize Appearance (Optional)**: Use the **Properties** window to adjust the appearance and behavior of the `SiteMapPath` (e.g., separators, node style, etc.).
5) **Run and Test**: Save the page and run the application to see the breadcrumb navigation provided by the `SiteMapPath` control.

❖ Response.Redirect Method:

- One of the common ways to redirect a user to another page using server-side code is using the Response property of the Web Form. The **ASP Redirect Method** is used to redirect the client to a different URL. It is a predefined method of the Response Object.
- The Response property gets the Http Response object associated with the Page.
- The HTTP Response object allows you to send HTTP response data to a client.
- The Redirect method of the HTTP Response class redirects a client to a new URL.
  // Redirect to another page within the same application
  Syntax: Response.Redirect (path)
  Response.Redirect("HomePage.aspx");
  // Redirect to an external website
  Response.Redirect("https://www.google.com");

❖ Server.Transfer Method:

- Server.Transfer is a method used in ASP.NET to move from one page to another on the server without going back to the user's browser.
- When you call the Transfer method, ASP.NET terminates processing of the current request (that is: stops processing the code; as if you called "GoTo") and simply transfers the user to another page.
- The ASP Server.Transfer Method is used to send all the information that has been created in one asp file to another .asp file.
  Syntax: Server.Transfer(path)

❖ Difference between Response.Redirect and Server.Transfer

Feature	Response.Redirect	Server.Transfer
Type of Redirect	Client-side redirect (sends a new request to the browser).	Server-side redirect (transfers control to another page on the server).
URL Change	The URL in the browser changes to the new page.	The URL in the browser remains the same (original page's URL).
HTTP Request	Sends a new HTTP request to the browser (round-trip).	Does not send a new HTTP request (no round-trip).
Scope	Can redirect to any page, including external websites.	Only works within the same application (same server).
Performance	Slightly slower due to an additional HTTP request/response cycle.	Faster because it doesn't require a new HTTP request.

Page Data	The data from the current page is lost (except through mechanisms like query strings or session).	Preserves form data, request objects, and view state (optional).
Postbacks and Query Strings	Does not maintain the previous page's state (new request).	Can maintain the state of controls and form data (optional).
Memory Management	Frees up memory associated with the previous page after the new request is made.	Previous page data is still in memory until the transfer is complete, leading to higher memory usage.
Syntax	Response.Redirect("Page2.aspx");	Server.Transfer("Page2.aspx");

❖ Cross Page Posting in ASP.NET / Cross Page Postback

- PostBack is the process of submitting an ASP.NET page to the server for processing.
- The PostBackUrl property allows you to perform a cross-page post using the Button control.
- Set the PostBackUrl property to the URL of the Web page to post to when the Button control is clicked.
- For example, specifying Page2.aspx causes the page that contains the Button control to post to Page2.aspx.
- If you do not specify a value for the PostBackUrl property, the page posts back to itself.

Important:

- When performing a cross-page postback with controls with server-side validation, you should check that the page's IsValid property is true before processing the postback.
- In the case of a cross-page postback, the page to check is the PreviousPage.

Example:

```
// Page1.aspx
<form id="form1" runat="server">
  <asp:TextBox runat="server" ID="_tb1"></asp:TextBox>
  +
  <asp:TextBox runat="server" ID="_tb2"></asp:TextBox>
  <asp:Button runat="server" ID="_button" PostBackUrl="~/Page2.aspx" Text=" = " />
</form>
// Page2.aspx
<script runat="server">
protected void Page_Load(object sender, EventArgs e)
{
        TextBox tb1 = (TextBox)PreviousPage.FindControl("_tb1");
        TextBox tb2 = (TextBox)PreviousPage.FindControl("_tb2");
        int sum = Int32.Parse(tb1.Text) + Int32.Parse(tb2.Text);
        _result.Text = sum.ToString();
}
</script>
<form id="form1" runat="server">
        Answer is: <asp:Label runat="server" ID="_result"></asp:Label>
</form>
```

❖ ASP.NET State Management:

ASP.NET State management preserves state control and objects in an application because ASP.NET web applications are stateless. All ASP.NET web applications are stateless i.e., by default, for each page posted to the server, the state of controls is lost. In a single line, State management maintains and stores the information of any user till the end of the user session.

There are two types of state management techniques: Client side and Server side.

1) Client side:

In ASP.NET, Client-Side State Management refers to methods used to store data directly on the client's browser or machine, rather than on the server. This approach can reduce the load on the server and is useful for lightweight data, but it comes with security and size limitations.

Here are the main client-side state management techniques in ASP.NET:

1) Hidden Field
2) View State
3) Cookies
4) Control State

1) Hidden Field:

- Hidden fields are used to store value to client side.
- A Hidden Field in ASP.NET is a server-side control that allows you to store data on the web page without displaying it to the user.
- It is commonly used to store small amounts of data between page postbacks.
- The data in the hidden field is not visible in the browser but can be seen if a user views the page source.

  aspx Code:

  ```
  <asp:HiddenField ID="HiddenFieldUserID" runat="server" Value="1001" />
  ```

C# Code:

```
protected void ButtonSubmit_Click(object sender, EventArgs e)
{
   // Retrieve value from the hidden field
   string userId = HiddenFieldUserID.Value;

   // Display or use the hidden field value
   LabelMessage.Text = "Your User ID is: " + userId;
}
```

2) View State:

- Viewstate is a very useful client side property.
- View state is another client side state management mechanism provided by ASP.NET to store user's data
- It is used for page level state management.
- Viewstate stores any type of data and used for sending and receiving information,
- View State provides page level state management i.e., as long as the user is on the current page, state is available and the user redirects to the next page and the current page state is lost.
- View state is enabled by default for all server side controls of ASP.NET with a property EnableviewState set to true.

Example: ASP.NET Code

```
<form id="form1" runat="server">
   <asp:Label ID="Label1" runat="server" Text="Enter Name"></asp:Label>
   <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
   <asp:Button ID="Button1" runat="server" onclick="Button1_Click"
      Text="Clear &amp; Submit" />
   <asp:Button ID="Button2" runat="server" onclick="Button2_Click"
      Text="Restore" />
</form>
```

C#.NET Code:

```
protected void Button1_Click(object sender, EventArgs e)
{
    ViewState["myValue"] = TextBox1.Text;
    TextBox1.Text = String.Empty;
}
```

```
protected void Button2_Click(object sender, EventArgs e)
{
    TextBox1.Text = ViewState["myValue"].ToString();
}
```

3) Cookies:

- Cookie is a small text file which is created by the client's browser and also stored on the client hard disk by the browser.
- It does not use server memory. Generally a cookie is used to identify users.
- A cookie is a small file that stores user information.
- Whenever a user makes a request for a page the first time, the server creates a cookie and sends it to the client along with the requested page and the client browser receives that cookie and stores it on the client machine either permanently or temporarily.

  Types of Cookies: Persistence & Non-Persistence Cookies

  1) Persistence Cookie:

  Cookies which you can set an expiry date time are called persistence cookies. Persistence cookies are permanently stored till the time you set.

  Let us see how to create persistence cookies.

  Response.Cookies["CookieName"].Value="This is A Persistance Cookie";

  Response.Cookies["CookieName"].Expires=DateTime.Now.AddSeconds(10);

  2) Non-Persistence Cookie:

  Non persistence cookies are not permanently stored on the user client hard disk folder. It maintains user information as long as the user accesses the same browser. When user closes the browser the cookie will be discarded.

  Let us see how to create non persistence cookies.

  Response.Cookies["CookieName"].Value = "This is A Non Persistance Cookie";

  How to read a cookie:

  ```
  if (Request.Cookies["CookieName"] != null)
  {
      Label1.Text = Request.Cookies["CookieName"].Value;
  }
  ```

4) Control State:

Control State is another client side state management technique. This is used when there is a need to store control data related to Custom control. Whenever we develop a custom control and want to preserve some information then Control State is used, Control state is separate from view state.

2) Server side:

Server-side state management involves storing state information on the server, rather than relying on the client's browser or device.

1) Session State:

Session State is one of the most commonly used server-side state management techniques. Session management is a very strong technique to maintain state. Generally session is used to store user's information and/or uniquely identify a user (or say browser). Session State that allows you to store and retrieve user-specific data across multiple requests and pages during a user's session. The server maintains the state of user information by using a session ID.

Methods:

Methods	Description
Add(name, value)	Adds an item to the session state collection.
Clear	Removes all the items from session state collection.
Remove(name)	Removes the specified item from the session state collection.
RemoveAll	Removes all keys and values from the session-state collection.
RemoveAt	Deletes an item at a specified index from the session-state collection.

How to get and set value in Session:

```
Session["Username"] = "admin"; //Set Value to The Session
Label1.Text = Session["Username"].ToString(); //Get Value from the Sesion
```

Removing Data from Session:

```
// Remove a specific item
Session.Remove("UserName");
// Remove all items
Session.Clear();
// Abandon the entire session
Session.Abandon();
```

2) Application State:

The data stored in application state is common for all users of that particular ASP.NET application and can be accessed anywhere in the application. It is also called application level state management. Data stored in the application should be of small size.

Methods	**Description**
Add(name, value)	Adds an item to the application state collection.
Clear	Removes all the items from the application state collection.
Remove(name)	Removes the specified item from the application state collection.
RemoveAll	Removes all objects from an HttpApplicationState collection.
RemoveAt	Removes an HttpApplicationState object from a collection by index.
Lock()	Locks the application state collection so only the current user can access it.
Unlock()	Unlocks the application state collection so all the users can access it.

How to get and set a value in the application object:

```
//Set Value to The  Application Object
Application["Count"] = Convert.ToInt32(Application["Count"]) + 1;
Label1.Text = Application["Count"].ToString(); //Get Value from the Application Object
```

Removing Data from Application State:

```
// Remove a specific item
Application.Remove("TotalUsers");
// Clear all items
Application.Clear();
```

****

UNIT-IV

ADO.NET

❖ Data Controls in ASP.NET:

Data Controls are used to display table format of data. The data controls are used in ASP.NET to display data in various forms and do various database activities such as Add, Edit, Update and Delete operations. One of the important goals of the Asp.net language is code minimization. Data controls play an important role in this purpose. Data controls used to display the records in the form of reports.

There are 2 types of controls to display the table format of data.

1) Single Row Data Controls
2) Multiple Row Data Controls

1) **Single Row Data Controls**: Single row data controls display single record in a Table. ASP.NET provides following single row data controls.

   i) **Detailsview Data Control**: The DetailsView Control enable us to work with a single data item at a time. This control enable us to display, edit, insert, and delete a single record in a database. **asp:BoundField** is used to bind data in DetailsView.

   The following are the some important templates of DetailsView Control:

   <HeaderTemplate>: Displays Header text for a Data Source collection and applies a different style for the Header text.

   <FooterTemplate>: Changes the background color or style of alternating items in a Data Source collection.

   <PagerTemplate>

   <Fields>

Example: ASP.NET Code

```
<asp:DetailsView ID="DetailsView2" runat="server" Height="50px" Width="125px"
    AllowPaging="True" onpageindexchanging="DetailsView2_PageIndexChanging">
</asp:DetailsView>
```

C#.NET Code:

```
OleDbConnection cn=new OleDbConnection(@"Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=D:\02 Dot Net Notes - NEP 2020\DataControlsASP.NET\Employee.mdb");
int currentPageIndex = 0;

protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        Display();
    }
}
```

```
private void Display()
{
    cn.Open();
    DataSet ds = new DataSet();
    string strDisplay = "select * from EmpInfo";
    OleDbDataAdapter da = new OleDbDataAdapter(strDisplay, cn);
    da.Fill(ds);
    DetailsView2.DataSource = ds;
    DetailsView2.DataBind();
}
protected void DetailsView2_PageIndexChanging(object sender,
DetailsViewPageEventArgs e)
{
    // Set the current page index based on user interaction
    DetailsView2.PageIndex = e.NewPageIndex;
    currentPageIndex = e.NewPageIndex;

    // Rebind the data to reflect the new page
    Display();
}
```

ii) **FormView Data Control:** This control displays a single record of data at a time like DetailsView control and supports the editing of record. A FormView is a databound control used to insert, display, edit, update and delete data in ASP.NET that renders a single record at a time. A FormView control is similar to a DetailView in ASP.NET but the only difference is that a DetailsView has a built-in tabular rendering whereas a FormView requires a user-defined template to insert, display, edit, update and delete data.

The FormView control supports the following features:

- Template driven
- Supports column layout
- Built-in support for paging and grouping
- Built-in support for insert, edit and delete capabilities

Example: ASP.NET Code

```
<asp:FormView ID="FormView2" runat="server"
    onpageindexchanging="FormView2_PageIndexChanging">
</asp:FormView>
```

C#.NET Code:

```
OleDbConnection cn = new OleDbConnection(@"Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=D:\02 Dot Net Notes - NEP 2020\DataControlsASP.NET\Employee.mdb");
int currentPageIndex = 0;

protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        Display();
    }
}
```

```
private void Display()
{
    cn.Open();
    DataSet ds = new DataSet();
    string strDisplay = "select * from EmpInfo";
    OleDbDataAdapter da = new OleDbDataAdapter(strDisplay, cn);
    da.Fill(ds);
    FormView2.DataSource = ds;
    FormView2.DataBind();
}
protected void FormView2_PageIndexChanging(object sender, FormViewPageEventArgs e)
{
    // Set the current page index based on user interaction
    FormView2.PageIndex = e.NewPageIndex;
    currentPageIndex = e.NewPageIndex;

    // Rebind the data to reflect the new page
    Display();
}
```

2) Multiple Row Data Controls:

Multiple row data controls display multiple records in a Table. ASP.NET provides following single row data controls.

i) Repeater data control

ii) GridView data control

iii) DataList data control

i) Repeater Data Control: A Repeater is a Data-bound control. Repeater control is used to show a repeated list of items from data source like data table or database. It creates a link between the Data Source and the presentation UI to display the data.

A Repeater has five templates to format it:

1. <HeaderTemplate> - Displays Header text for a Data Source collection and applies a different style for the Header text.
2. <AlternatingItemTemplate> - Changes the background color or style of alternating items in a Data Source collection.
3. <Itemtemplate> - It defines how the each item is rendered from the Data Source collection.
4. <SeperatorTemplate> - It will determine the separator element that separates each item in the item collection. It will be a <br> or <Hr> HTML element.
5. <FooterTemplate> - Displays a footer element for the Data Source collection.

Example:

```
<asp:Repeater ID="Repeater1" runat="server">
<HeaderTemplate>
<table border="2">
<tr>
    <td>ID</td>
    <td>Name</td>
    <td>Address</td>
</tr>
</HeaderTemplate>
<ItemTemplate>
<tr>
    <td><%# Eval("EId") %></td>
    <td><%# Eval("EName") %></td>
    <td><%# Eval("Address") %></td>
</tr>
</ItemTemplate>
<FooterTemplate>
</table>
</FooterTemplate>
</asp:Repeater>
```

C#.NET Code:

```
OleDbConnection cn = new OleDbConnection(@"Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=D:\02 Dot Net Notes - NEP 2020\DataControlsASP.NET\Employee.mdb");

protected void Page_Load(object sender, EventArgs e)
{
    Display();
}

private void Display()
{
    cn.Open();
    DataSet ds = new DataSet();
    string strDisplay = "select * from EmpInfo";
    OleDbDataAdapter da = new OleDbDataAdapter(strDisplay, cn);
    da.Fill(ds);
    Repeater1.DataSource = ds;
    Repeater1.DataBind();
}
```

ii) GridView data control: GridView is a control used to display data in tables on a web page. It displays data in both rows and columns, where each column represents a field, and each row represents a record.

The GridView control supports the following features:

- Binding to data source controls, such as SqlDataSource.
- Built-in sort capabilities.
- Built-in update and delete capabilities.
- Built-in paging capabilities.
- Built-in row selection capabilities.
- Programmatic access to the GridView object model to dynamically set properties, handle events, and so on.
- Multiple key fields.
- Multiple data fields for the hyperlink columns.

Example: ASP.NET Code:

```
<asp:GridView ID="GridView2" runat="server"
    onpageindexchanging="GridView2_PageIndexChanging"
    onselectedindexchanged="GridView2_SelectedIndexChanged" PageSize="2"
    AllowPaging="True">
</asp:GridView>
```

C#.NET Code:

```
OleDbConnection cn = new OleDbConnection(@"Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=D:\02 Dot Net Notes - NEP 2020\DataControlsASP.NET\Employee.mdb");
int currentPageIndex = 0;

protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        Display();
    }
}
private void Display()
{
    cn.Open();
    DataSet ds = new DataSet();
    string strDisplay = "select * from EmpInfo";
    OleDbDataAdapter da = new OleDbDataAdapter(strDisplay, cn);
    da.Fill(ds);
    GridView1.DataSource = ds;
    GridView1.DataBind();
}
protected void GridView2_PageIndexChanging(object sender, GridViewPageEventArgs e)
{
    // Set the current page index based on user interaction
    GridView1.PageIndex = e.NewPageIndex;
    currentPageIndex = e.NewPageIndex;

    // Rebind the data to reflect the new page
    Display();
}
```

iii) DataList Control:

DataList is a Databound control to display and manipulate data in a web application. It is a composite control that can combine other ASP.Net controls and it is present in the form. The DataList appearance is controlled by its template fields.

The following template fields are supported by the DataList control:

- Itemtemplate: It specifies the Items present in the Datasource, it renders itself in the browser as many rows present in the data source collection.
- EditItemTemplate: Used to provide edit permissions to the user.
- HeaderTemplate: Used to display header text to the data source collection.
- FooterTemplate: Used to display footer text to the data source collection.
- ItemStyle: Used to apply styles to an ItemTemplate.

- EditStyle: Used to apply styles to an EditItemTemplate
- HeaderStyle: Used to apply styles to a HeaderTemplate
- FooterStyle: Used to apply styles to a FooterTemplate.

Example: ASP.NET Code:

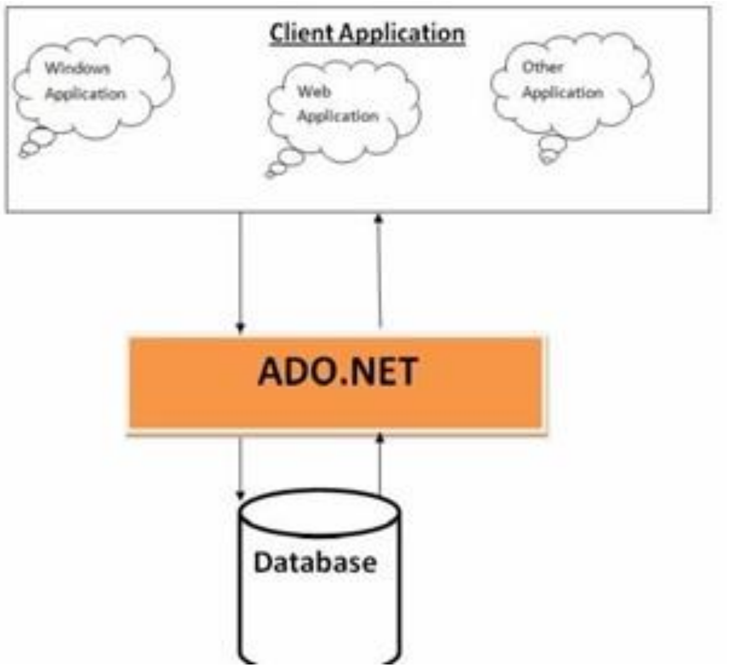
```
<asp:DataList ID="DataList2" runat="server">
<HeaderTemplate>
<table border="2">
<tr>
    <td>EID</td>
    <td>Name</td>
    <td>Address</td>
</tr>
</HeaderTemplate>
<ItemTemplate>
<tr>
    <td><%# Eval("EId") %></td>
    <td><%# Eval("EName") %></td>
    <td><%# Eval("Address") %></td>
</tr>
</ItemTemplate>
<FooterTemplate>
</table>
</FooterTemplate>
</asp:DataList>
```

C#.NET Code:

```
OleDbConnection cn = new OleDbConnection(@"Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=D:\02 Dot Net Notes - NEP 2020\DataControlsASP.NET\Employee.mdb");
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        Display();
    }
}
private void Display()
{
    cn.Open();
    DataSet ds = new DataSet();
    string strDisplay = "select * from EmpInfo";
    OleDbDataAdapter da = new OleDbDataAdapter(strDisplay, cn);
    da.Fill(ds);
    DataList1.DataSource = ds;
    DataList1.DataBind();
}
```

❖ Introduction to ADO.NET:

ADO.NET is an object-oriented set of libraries that allows you to interact with data sources. Commonly, the data source is a database, but it could also be a text file or an Excel spreadsheet. ADO.NET is a large set of .NET classes that enable us to retrieve and manipulate data, and update data sources, in very many ways. ADO.NET is a model used by .NET applications to communicate with a database for retrieving, accessing, and updating data.



In ADO.NET you create and manage connections using different classes and objects

❖ ADO.NET Classes:

ADO.NET is a set of classes (a framework) to interact with data sources such as databases and XML files. ADO is known as ActiveX Data Objects. It allows us to connect to the databases. It has classes and methods to retrieve and manipulate data.

The following are a few of the .NET applications that use ADO.NET to connect to a database, execute commands and retrieve data from the database.

- ASP.NET Web Applications
- Console Applications
- Windows Applications

ADO.NET provides various classes. They are:

1) Connection Class
2) Command Class
3) DataReader Class
4) DataAdaptor Class
5) DataSet.Class

1) Connection Class:

In ADO.NET, we use the connection class to connect to the database. The Connection object creates the connection to the database. When the connection of an object is created, it takes a connection string that contains the information about the database server, server type, database name, connection type, and database user. A connection string is usually stored in the web.config file or app.config file of an application.

ADO.NET provides connection to multiple providers. Each provider has a functionality to connect with different database. Here is a list of data providers in ADO.NET.

- Data Provider for SQL Server (System.Data.SqlClient)
- Data Provider for MS ACCESS (System.Data.OleDb)
- Data Provider for MYSQL (System.Data.Odbc)
- Data Provider for ORACLE (System.Data.OracleClient)

How to use connection class with this provider is given below-

- Connection Class for SQL Server (SqlConnection)
- Connection Class for MSACCESS (OleDbConnection)
- Connection Class for MYSQL (OdbcConnaction)
- Connection Class for ORACLE (OracleConnection)

Common Properties:

➢ ConnectionString: A string that contains information about the data source, such as the server name, database name, authentication details, etc.

➢ State: Indicates the current state of the connection (e.g., Closed, Open, Connecting, Executing, Fetching).

➢ Database: Gets the name of the current database or the database to be used after a connection is opened.

Common Methods:

➢ Open(): Opens a database connection with the settings specified by the ConnectionString.

➢ Close(): Closes the connection to the database.

➢ Dispose(): Releases all resources used by the connection.

Example:

OleDbConnection cn = new OleDbConnection(@"Data Source=ServerName;Initial Catalog=Database;User ID=Username;Password=password");

2) Command Class:

The Command class provides methods for storing and executing SQL statements and Stored Procedures. Command class is essential for performing operations such as:

➢ Retrieving Data: Executing SELECT statements to fetch data.

➢ Modifying Data: Performing INSERT, UPDATE, or DELETE operations.

➢ Executing Stored Procedures: Running precompiled SQL code stored within the database.

➢ Executing Scalar Queries: Retrieving a single value from the database.

The following are the various commands that are executed by the Command Class.

- **ExecuteReader:** Returns data to the client as rows. This would typically be an SQL select statement or a Stored Procedure that contains one or more select statements.
- **ExecuteNonQuery:** Executes a command that changes the data in the database, such as an update, delete, or insert statement, or a Stored Procedure that contains one or more of these statements.
- **ExecuteScalar:** This method only returns a single value. This kind of query returns a count of rows or a calculated value.

Example:

```
OleDbCommand cmdSave=new OleDbCommand("INSERT INTO EMP
VALUES(1,'Amol','Karad')",cn);
cmdSave. ExecuteNonQuery();
```

3) DataReader Class:

The DataReader is used to retrieve data. It is used in conjunction with the Command class to execute an SQL Select statement and then access the returned rows. ADO.NET DataReader object is used for accessing data from the database. DataReader object provides a read only, forward only to retrieve data from a data. The .NET framework provides following DataReader Classes.

- SqlDataReader
- OleDbDataReader
- OdbcDataReader

The main methods are:

1. Read: Returns true until there is a next row.

2. GetValue: Returns the stored value at the specified index of the currently selected row.

3. GetValues: Saves the values of the current row into an array.

4. NextResult: Moves the cursor to the next result set (if any).

5. Close: Closes the reader.

4) DataAdapter Class:

The DataAdapter is used to connect DataSets to databases. The DataAdapter is most useful when using data-bound controls in Windows Forms. The DataAdapter is used either to fill a DataTable or DataSet with data from the database with its Fill method.

The DataAdapter is the class at the core of ADO .NET's disconnected data access. It is essentially the middleman facilitating all communication between the database and a DataSet.

The most important methods of the DataAdapter are as follows:

1) **Fill**: adds a DataTable to the DataSet by executing the SelectCommand of the DataAdapter.

2) **FillSchema**: adds a DataTable to a DataSet by executing the SelectCommand of the DataAdapter, but builds only the schema of the table, no data will be added.
3) **Update**: updates the data source with the modified data in the current DataTable.

5) DataSet:

The DataSet is the heart of ADO.NET. The dataset is a disconnected, in-memory representation of data. It can be considered as a local copy of the relevant portions of the database. It does not have a continuous connection to the database. To update the database a reconnection is required.

The DataSet contains a collection of one or more DataTable objects consisting of rows and columns of data. The data in DataSet can be loaded from any valid data source like Microsoft SQL serverdatabase, an Oracle database or from a Microsoft Access database.

Example: DataSet ds = new DataSet();

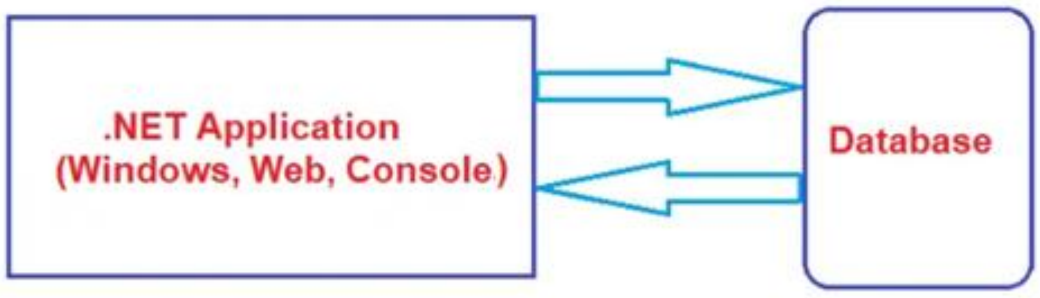
❖ Connected and Disconnected architecture in ADO.NET:

Ado.net is a data access technology that allows interaction between applications and databases. The Ado.net framework supports two models of data access architecture i.e. Connection Oriented Data Access Architecture and Disconnected Data Access Architecture.

1) Connection-Oriented Data Access Architecture:

In the case of Connection Oriented Data Access Architecture, always an open and active connection is required in between the .NET Application and the database. An example is Data Reader and when we are accessing the data from the database, the Data Reader object requires an active and open connection to access the data, if the connection is closed then we cannot access the data from the database and in that case, we will get the runtime error.

The Connection Oriented Data Access Architecture is always forward only. That means using this architecture mode, we can only access the data in the forward direction. Once we read a row, then it will move to the next data row and there is no chance to move back to the previous row.



Connection Oriented Data Access Architecture

i) DataReader:

- It is a connected architecture, which means when you require data from the database you need to connect with database and fetch the data from there.
- DataReaders are read-only, forward-only cursors.
- They hold only one row of the result set at a given time.
- It is optimized for speed. The DataReader is instantiated through a Command object.

The main methods are:

1) Read: Returns true until there is a next row.

2) GetValue: Returns the stored value at the specified index of the currently selected row.

3) GetValues: Saves the values of the current row into an array.

4) NextResult: Moves the cursor to the next result set (if any).

5) Close: Closes the reader.

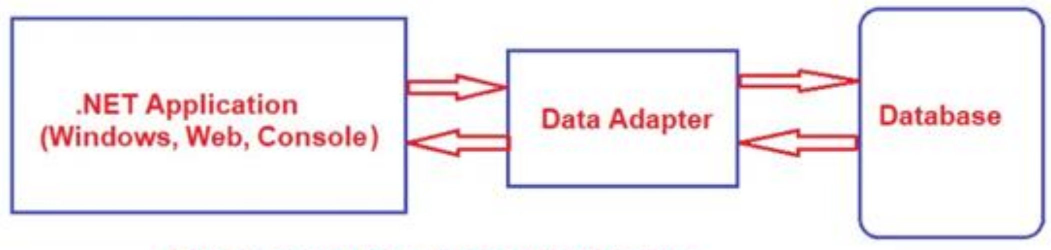
Example:

```
OleDbCommand cmd=new OleDbCommand("SELECT * FROM EMP",cn);
OleDbDataReader dr = cmd.ExecuteReader();
if (dr.HasRows)
{
    while (dr.Read())
    {
            Console.WriteLine(dr[0].ToString());
    }
}
else
{
    Console.WriteLine("No rows found.");
}
dr.Close();
```

2) Disconnection-Oriented Data Access Architecture:

In the case of Disconnection Oriented Data Access Architecture, always an open and active connection is not required in between the .NET Application and the database. In this architecture, Connectivity is required only to read the data from the database and to update the data within the database.

An example is DataAdapter and DataSet or DataTable classes. Here, using the DataAdapter object we can retrieve the data from the database and store the data in a DataSet or DataTable. The DataSets or DataTables are in-memory objects or you can say they store the data temporarily within .NET Application. Then whenever required in our .NET Application, we can fetch the data from the dataset or data table and process the data. Here, we can modify the data, we can insert new data, can delete the data from within the dataset or data tables.



Disconnected Data Access Architecture

i) DataAdapter:

- The DataAdapter is the class at the core of ADO .NET's disconnected data access.
- It is essentially the middleman facilitating all communication between the database and a DataSet.
- The DataAdapter is used either to fill a DataTable or DataSet with data from the database with its Fill method.
- The DataAdapter provides four properties that represent database commands: SelectCommand, InsertCommand, DeleteCommand and UpdateCommand
- The most important methods of the DataAdapter are as follows:
  1) Fill: adds a DataTable to the DataSet by executing the SelectCommand of the DataAdapter.
  2) FillSchema: adds a DataTable to a DataSet by executing the SelectCommand of the DataAdapter, but builds only the schema of the table, no data will be added.
  3) Update: updates the data source with the modified data in the current DataTable.
  For example to fill a DataSet, use the following code:
  OleDbDataAdapter da = new OleDbDataAdapter(“SELECT * FROM EMP”,cn);
  DataSet ds = new DataSet();
  da.Fill(ds, “EMP”);

ii) DataSet:

The DataSet is the heart of ADO.NET. The dataset is a disconnected, in-memory representation of data. It can be considered as a local copy of the relevant portions of the database. It does not have a continuous connection to the database. To update the database a reconnection is required. The DataSet contains a collection of one or more DataTable objects consisting of rows and columns of data. The data in DataSet can be loaded from any valid data source like Microsoft SQL serverdatabase, an Oracle database or from a Microsoft Access database.

Example: DataSet ds = new DataSet();

iii) DataTable:

- DataTable represents relational data into tabular form.
- ADO.NET provides a DataTable class to create and use data table independently.
- The DataTable is a database table representation and provides a collection of columns and rows to store data in a grid form.
- Each column defines a data type and each row represents a data record.

DataTable Properties:

Property	**Description**
Columns	It is used to get the collection of columns that belong to this table.
Rows	It is used to get the collection of rows that belong to this table.
TableName	It is used to get or set the name of the DataTable.

DataTable Methods:

Method	**Description**
Clear()	It is used to clear the DataTable of all data.
Clone()	It is used to clone the structure of the DataTable.
Copy()	It is used to copy both the structure and data of the DataTable.
NewRow()	It is used to create a new DataRow with the same schema as the table.

Example: ASP.NET Code to add Gridview

```
<asp:GridView ID="GridView1" runat="server">
</asp:GridView>
```

C#.NET Code:

```
using System.Data;
protected void Page_Load(object sender, EventArgs e)
{
    DataTable dt = new DataTable();
    dt.Columns.Add("ID");
    dt.Columns.Add("Name");
    dt.Rows.Add("1", "Vishal");
    dt.Rows.Add("2", "Rahul");
    GridView1.DataSource = dt;
    GridView1.DataBind();
}
```

❖ Data Binding in ADO.NET:

Data Binding is a fundamental concept that allows developers to connect UI elements such as GridView, Repeater, DropDownList, etc. to different data sources like databases such as MS-Access, SQL Server, Oracle, MySQL etc. Data binding can be categorized into two: simple data binding and complex data binding.

1) Simple Data Binding:

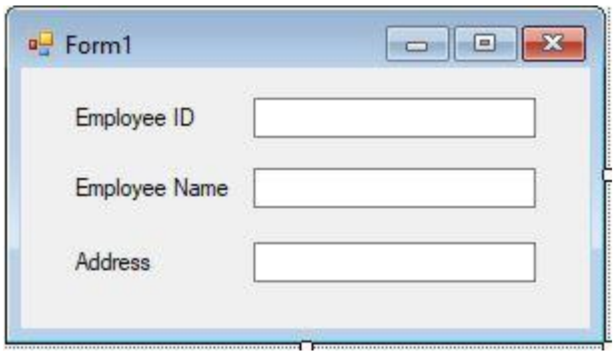
The Simple Data Binding is the process of binding the control with the single value in the dataset.

The controls like text box, label can be bound to the control through the control properties.

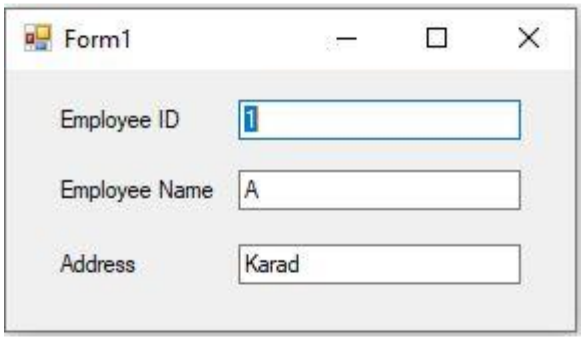
Consider an example to display the employee details. The details are added in the following format.

Steps:

- Create a Windows Form Application in Visual Studio .NET. The following customized format is created for user.



- Select the first text box and the properties for it appear in the window.
- Expand the DataBindings property
- Select the Text property for enabling the drop down list.
- Click the Add Project Data Source from the drop down list
- Make a connection with the Employee database and select the Employee table
- Select the Other Data Sources, Project Data Sources, EmployeeDataSet, Employee table.
- Select the Name column and bind it with the textbox.
- Bind all the other text boxes with the database values.
- Press F5 and execute the Windows Form.
- The following output is displayed to the user.



Form1

Employee ID: 1

Employee Name: A

Address: Karad

2) Complex Data Binding:

The Complex Data Binding is the process of binding the component with the Database. The controls can be GridView, Dropdown list, or combo box. Multiple values can be displayed from the dataset through the binding.

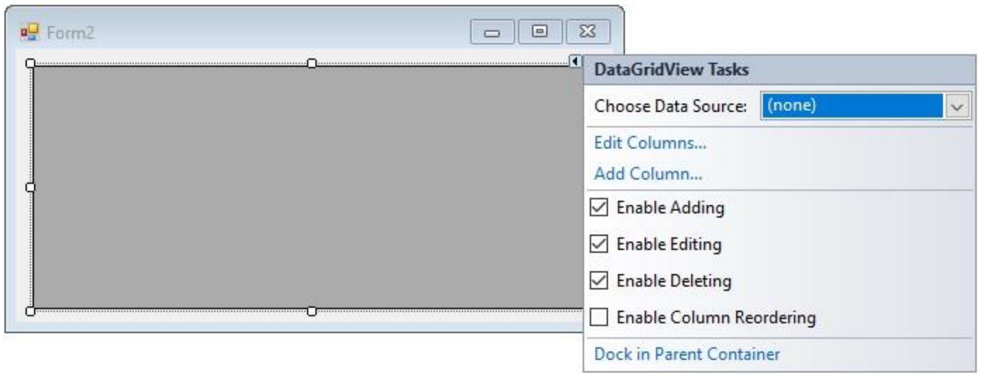
The controls that can be used for binding the multiple values from the database to the Windows Form are listed below.

- DataGridView: It is used to display the multiple records and columns. The DataSource property of the DataGridView control is used for binding the specific data element.

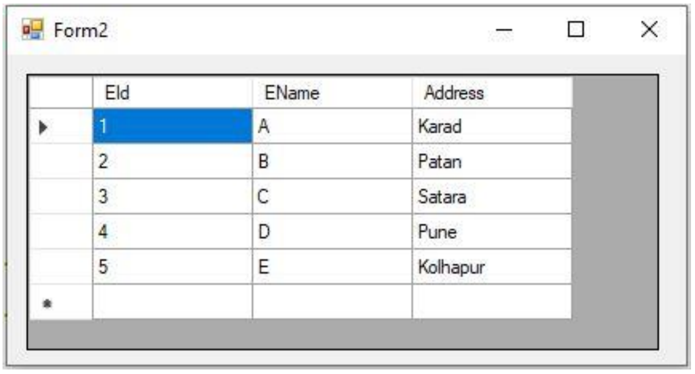
- ComboBox: The control contains a text box for entering the data and drop down list for displaying the values. The DataSource property is useful for binding the control. The element specific information can be bind through the DisplayMember property
- ListBox: It is used for displaying the data for the column from several records of datasets. The DataSource property is used for binding the control to the data source.
- The DisplayMember property is used for binding the control to the specific data element.

Steps to bind data to DataGridView Control:

- Create a Windows Form Application in Visual Studio .NET. Add dataGridView Control in ToolBox.Click on DataGridView Task



- Click the Choose Data Source ->Add Project Data Source from the drop down list
- Make a connection with the Employee database and select the Employee table
- Select the Other Data Sources, Project Data Sources, EmployeeDataSet, Employee table.
- Select your Database Table -> Employee
- Press F5 and execute the Windows Form.
- The following output is displayed to the user.



EId	EName	Address
1	A	Karad
2	B	Patan
3	C	Satara
4	D	Pune
5	E	Kolhapur

❖ Report Generation:

A **report** is a document that shares important information in an organized way. In .NET applications, reports serve as a bridge between raw data and user-friendly presentations. Depending on the requirements, reports can range from simple tabular displays the documents with dynamic content based on user inputs.

Microsoft Report Viewer (RDLC): A versatile tool integrated with Visual Studio, allowing the creation of rich, interactive reports with features like grouping, sorting, and parameterization.

Microsoft Report Viewer is a control provided by Microsoft that allows embedding reports within .NET applications. Reports are defined using RDLC (Report Definition Language Client-side) files, which are designed using Visual Studio's integrated designer.

Types of Report: 1) **Simple Report** **2) Parameterized Report**

1) Simple Report:

A **simple report** displays data in a straightforward manner without user interaction or dynamic content changes.

Step-by-Step Implementation:

1) **Create an ASP.NET Web Forms Application**
   - Open Visual Studio.
   - Navigate to **File** > **New** > **Project**.
   - Select **ASP.NET Web Application (.NET Framework)**.
   - Name the project (e.g., SimpleReportApp) and click **OK**.
   - Choose the **Web Forms** template and click **Create**.
2) **Add a Report (RDLC) to the Project**
   - In **Solution Explorer**, right-click the project, select **Add** > **New Item**.
   - Choose **Report** from the list, name it (e.g., EmployeeReport.rdlc), and click **Add**.
3) **Design the Report**
   - The RDLC designer opens automatically.
   - **Add a Data Source:**
     - In **Report Data** pane (if not visible, go to **View** > **Report Data**), right-click **Datasets** and select **Add Dataset**.
     - Name the dataset (e.g., EmployeeDataSet).
     - Choose **Use a dataset embedded in my report**.
     - Click **OK**.
   - **Insert a Table:**

        - Drag a **Table** control from the **Toolbox** onto the report.
        - Define columns corresponding to your data fields (e.g., EmpID, EmpName, Department, Salary).

4) **Add a ReportViewer Control to the Web Form**
    - Open **Default.aspx**.
    - Switch to **Design** view.
    - From the **Toolbox**, drag a **ReportViewer** control onto the page.
    - Set its properties:
        - **Name:** ReportViewer1
        - **Dock:** Fill
    - Configure the ReportViewer:
        - Click the smart tag (small arrow) on the ReportViewer.
        - Select **Local Report** > **Choose Report**.
        - Browse to and select Report.rdlc.
5) **Add ScriptManager Control on Web Form**
    - Open **Default.aspx.cs**.
6) **Run the Application**
    - Press **F5** or click **Start**.
    - The browser displays the report with the employee data in a tabular format.

2) Parameterized Report:

A **parameterized report** allows users to filter or customize the report's content based on input parameters, enhancing interactivity and relevance.

Step-by-Step Implementation

1) **Create or Open your Project** in Visual Studio (Windows Forms or ASP.NET).
2) **Add your Employee Database** as a data source:
    - Right-click the project in **Solution Explorer** > **Add** > **New Item** > **DataSet**.
    - Set up the dataset to connect to your database and select the `Employee` table, which contains the `empid` field.
3) **Add a new RDLC report**:
    - Right-click your project > **Add** > **New Item** > **Report** (RDLC) > Name it (e.g., `EmployeeReport.rdlc`).
4) **Design the Report Layout:**
    - In the Report Data panel, add the fields you want to display (like empid, name, etc.) from your Employee table.

- Drag these fields onto the report design surface to create a table layout.

5) **Add a Parameter**:
   - In the **Report Data** panel, right-click on **Parameters** > **Add Parameter**.
   - Name the parameter (e.g., EmployeeID), set the **Data Type** to Integer (if empid is an integer), and leave other settings as default.

6) **Use the Parameter to Filter Data:**
   - Click on the dataset's query (from the **Report Data** panel).
   - Modify the query to filter based on the `empid` parameter:
   - `SELECT * FROM Employee WHERE empid = @EmployeeID`

7) **Pass the Parameter from Code:**
   - Add a ReportViewer Control to your form or page.
   - Drag and drop the ReportViewer control onto the form.
   - Set the LocalReport.ReportPath to your EmployeeReport.rdlc.
   - Pass the Parameter Value to the Report in your code: For **ASP.NET**: csharp

     using Microsoft.Reporting.WebForms;

     // Create the parameter

     ReportParameter empIDParam = new ReportParameter("EmployeeID", "123"); // Example empid

     // Pass the parameter to the ReportViewer

     ReportViewer1.LocalReport.SetParameters(empIDParam);

     // Refresh the report

     ReportViewer1.LocalReport.Refresh();

8) **Run the Application:**
   - Press **F5** or click **Start**.

****