

17/12/19

19PC222

Dot Net Programming

Unit 1: The .NET Framework - Learning  
the .NET languages - Introduction .NET  
evolution - .NET & its architecture - CLR -  
What is assembly - components of assembly  
- DLL Hell and Assembly Version -  
objects & namespaces. - Setting up ASP.NET  
& IIS.

Unit - 1.

What is .NET?

.NET is a free, cross cross  
Platform, open source developer platform  
for building many different types of  
application.

With .NET we can use  
multiple languages, editors and libraries  
to build for web, mobile, desktop, IOT,  
Gaming. as well as web services  
Lang. Interoperability.  
Platform independent Partially.

2  
CLASS  
Glass Platform:

The code will run natively on any compatible operating system (OS)

The .Net Framework:

- \* Form based Application / window
- \* Web based Application
- \* Web Services

1. Basic func. of OS?
2. console.
3. JDK Version.

Basic func. of OS:

- \* Memory Management
- \* I/O
- \* Process
- \* Device

1) Basic functions of OS:

- \* Booting
- \* Memory Management
- \* Loading & Execution
- \* Data Security
- \* Disk Management
- \* Process Management
- \* Device controlling
- \* Printing controlling

2) Console:

- \* Console don't have user interface
- \* Run in command prompt.
- \* Combination of monitor and keyboard.

\* Monitor provides O/P & keybo used for i/p.

Eg: unix, Dos.

### 3) Java version History.

\* JDK Alpha and Beta (1995)

First release but they have highly unstable.

\* JDK 1.0 (Jan 23, 1996)

Name : Oak.

First stable version of JDK as JDK 1.0.2 is called Java1.

upto JDK 1.0.2, Private, Protected keywords used together to create another form of protection.

In JDK 1.0.2, it is removed.

\* JDK 1.1 (Feb 19, 1997)

It includes ;

\* Concept of inner class

\* JavaBeans

\* JDBC

\* RMI

\* AWT event model

It replaced JDK to recognize the base platform from J2SE (Java 2 platform, Enterprise edition) & J2ME (Java 2 platform, Micro edition)

1520 classes in 59 packages.

Additions were included in this version;

1. Java Plug In

2. Java IDL

3. Collections framework

4. Swing graphical API

5. Sun's JVM with JIT compiler

\* J2SE 1.3 (May 8, 2000)

Codename: Mestral.

It includes;

HotSpot JVM

RMI Support

JNDI (Java naming & directory interface)

J7DA.

JavaSound.

Synthetic proxy class.

\* J2SE 1.4 (Feb 6, 2002)

codename: Merlin.

Improved Libraries

Logging API

XML Parser

Security & Cryptography

Extensions.

\* J2SE 5.0 (Sep 30, 2004)

Codename: Tiger

Used Metadata

Autoboxing

Enumerations

Static imports

\* Java SE 6 (Dec 11, 2006)

Codename: Mustang

JDBC 4.0 support

7

Scripting lang. support

After Java 6, Sun released many updates to fix bugs.

\* Java SE 7 (July 28, 2011)

Codename: Dolphin.

JVM support for dynamic lang.

compressed 64 bit pointer

Strings added to switch.

Multiple exception.

\* Java SE 8 (March 18, 2014)

Codename: Spider.

Lang. level support for lambda exp.

Embed javascript code within app.

Launching JavaFX Apps.

\* Java SE 9 (Sep 21, 2017).

Supports gigabyte heaps

Garbage collector

Self tuning JVM

Provides Java Linker.

8

Automatic Scaling & sizing

\* Java SE 10 (Mar 20, 2018)

Remove Primitive datatype

Root certificates

Time based release versioning

Consolidate JDK Forest into

Single repository

Heap allocation.

\* Java SE 11 - upcoming.

• Net Revolution:

• Net Framework Timeline

• Net 1.0 (Feb. 2002)  
• Net 1.1 (Apr 3, 2003)  
• Net 2.0 (Jan 22, 2006)

• Net 4.6 (Jul, 2015)



## .NET Framework Timeline

• Net 1.0 (Feb 13, 2002)	• Net 1.1 (Apr 3, 2003)	Net 2.0 Jan 23, 2006	Net 3.0 Nov 21, 2006
• Net 4.6 Jul 20, 2015	Net 4.5 Aug 15, 2012	Net 4.0 Apr 12, 2010	Net 3.5 (Nov 19, 2007)

Net 4.6.1

Nov.

\* Microsoft initiated .Net

framework development process in 1999

The first Beta of version released in

late 2000. On Feb 13, 2002 .Net 1.0

was released for Windows 98, Windows  
Windows ME, Windows 2000 & Windows

\* .Net 1.1 Supported Windows ME

(upgraded 1.0).

\* .Net 2.0 released on Jan 29, 2006  
The SDK was launched in 2006.

\* on NOV 21, 2006, .Net 3.0 was launched  
and it is also called WinLX.

\* On NOV 19, 2007, version 3.5 was  
launched. This version 3.5 uses CLR 2.0  
which is the same version as .Net 2.0.  
.Net 3.5 supported windows Vista,  
Windows XP, Windows 7, Windows Server 2008.

\* .Net 4.0 released on <sup>APR</sup> May 12, 2010  
This version introduces CLR 4.0.

\* .Net 4.5 released on Aug 15, 2012.

Further .Net 4.6 released on July 29, 2015.

It supports RyuJIT for 64 bit.

\* Microsoft released Visual Studio  
on APR 5, 2017

\* .Net 4.7 released on May 9, 2017.

\* .Net 4.8 released on APR 18, 2019.

MSIL - CIL (new name)  
(old name) ↓

Common intermediate lang.

• Net 1.0 - CLR 1.0.

Use of DLL class libraries

Support for object oriented

Web developer (OWD)

• Net 1.1 - CLR 1.1

Visual Studio .Net 2003 (D#)

ASP.NET Mobile controls.

ADO.NET for ODBC

• Net 2.0 - CLR 2.0

Visual Studio 2005 Released.

Enhancement to ASP.NET &  
ADO.NET

Data Protection API's

FPP support.

Generic collection.

Serialization

Generic Collection :

A generic collection is

strongly typed, so that we can eliminate

runtime type mismatches, it improves

the performance by avoiding boxing & unboxing

## Serialization :-

12

is a process of converting an object into a stream of data, so that it can be easily transmittable over the network or can be stored in a persistent storage location.

12020

• Net Framework 3.0.

- CLR Version 2.0.

- IDE Released Visual Studio .NET 2003



Integrated Development Environment.

Rich GUI Capabilities

DICOM → Medical Images.

Web Server.

Database Server.

IIS.

↳ Web Server.

The four core components :

Four components of this version

are :

1. Windows Presentation Foundation (WPF)

This component mainly used for rich client apps & web apps windows

21 employ XML (extended application Markup lang). to build rich user interface. - 2D/3D graphic effects also

CSS → Making layout Page

create browser layout.

2. Windows Communication Foundation <sup>Foundation.</sup> Effects. (WCF)

- To develop Service Oriented applets.

- The distributed applets that run btw servers & clients called Indigo.

3. Windows Workflow Foundation (WWF).

- To implement workflow within .Net applets.

API → Applet. Program Interface.

4. Windows CardSpace

- To control & manage personal information.

11

### Web server

Web server is server slw on H/W dedicated to running said slw that can satisfy World wide web client request. It contains one or more websites

### Data server

Data server is the phrase used to describe computer slw and H/W (a database platform) that delivers DB services. It performs tasks such as data analysis, storage, data manipulation, archiving & client/server architecture.

020

### Features:

Lambda expression.

Extension Methods.

Object, and collection

Implicit typed local variable.

Net Version 3.5

CLR Version 2.0

Visual Studio

VS-Net 2008 (IDE)

## Features:

NET ajax enabled website  
 LINQ lang. Integrated Query become  
 accessible to ADO.NET.

Latency mode in Garbage collection  
 Peer to peer networking  
 webservice interoperability.

## • NET 4.0

CLR version 4.0

VS .Net 2010 (IDE)

## Features:

Dynamic lang. runtime

Expanded baseclass

BigInteger & Complex numbers.

Parallel Computing

## • NET 4.5

CLR version 4.0

VS .Net 2012 (IDE)

Support for Windows Store

appln. develop.

Built in support for asynchro.

Enhanced WCF, WPF, WWF

• Net F/W 4.6.2  
Aug. 2, 2016.

Element Initializer

Cryptography enhancement

Indexed members.

• Net F/W 4.7  
CLR 4.0.

ASVS - Net 2017

Improved TLS (Transport

Enhanced cryptography

High DPI Support.

• Net F/W 4.7.1

Oct 17, 2017.

Support for Portable PDP

Improved Garbage Collection

Runtime feature reduction.

• Net F/W 4.7.2.

Apr. 30, 2018

New overloads of DSA Dot Create

& RSA Dot Create

Ephemeral Keys

Support for Same Site Cookies.

• Net F/W 4.8

JIT improvements

Anti-manual Scanning for all

Assemblies.



- Oct.

Improved performance & reliability  
Support for automatic binding

Redirection.

Object heap - compact large

Object heaps.

• Net Framework 4.5.9

May 5 2014.

Enhancement of High DPI

Extended Expanded resizing

Windows form.

• Net FW 4.6.

CLR 4.0.

Introduced JIT compiler

called RyuJIT

Event tracking enhancement

HTTP 2 Support

• Net FW 4.6.1

NOV 30, 2015.

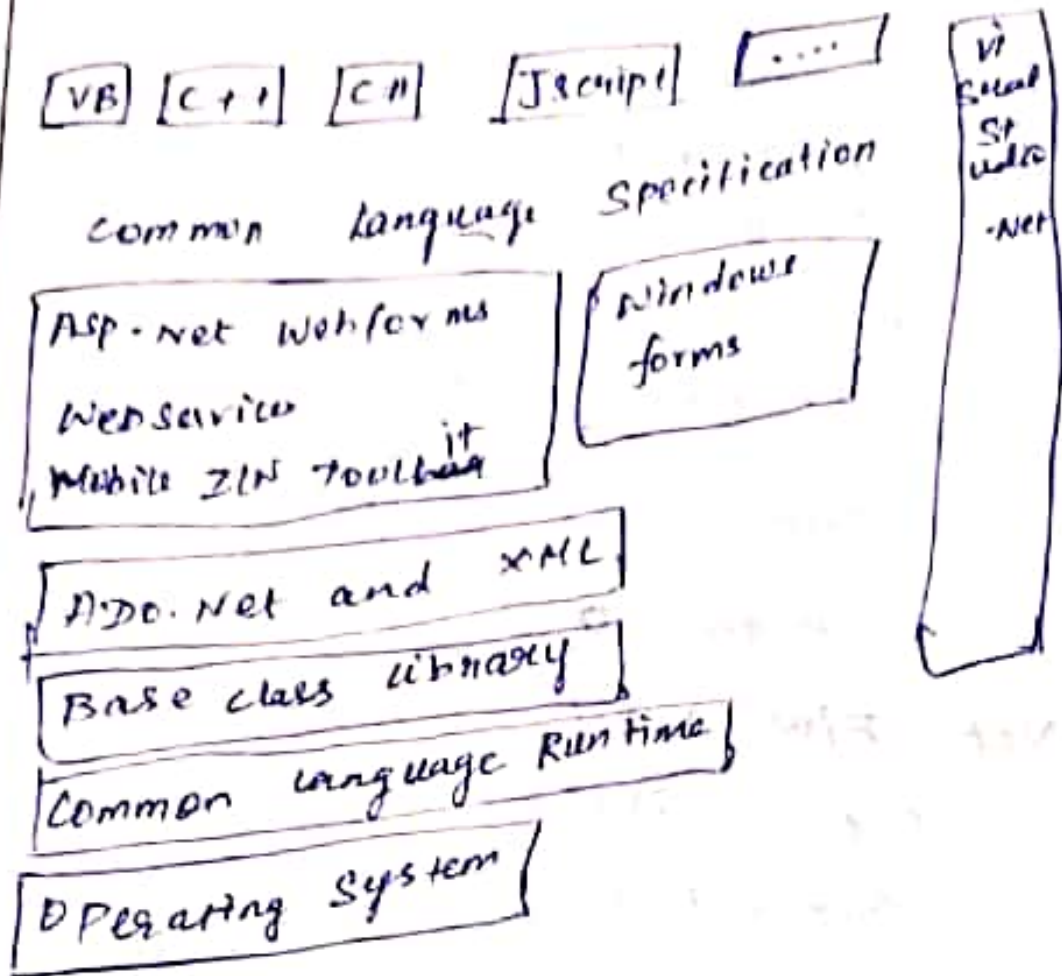
~~Native~~ Native image generator.

(NIG)

Small checking def developer

• Accessibility enhancement.

• Net Architecture:



Bottom to Top.

OS  $\rightarrow$  CLR (JIT) MSIL to Native code <sup>using JIT</sup>  
CTS (Common Type System). It allocates  
its own size for incoming code.  
Allocates memory in .NET.

CIL  $\Rightarrow$  Common Intermediate  
lang.

## Components of .NET Framework

Common Language Runtime (CLR)

CLR monitors execution of .NET application and provides essential services.

It manages code at execution time. It provides co-services like remote communication, memory & thread management.

## CLR Features

Exception Handling

Garbage Collection

Working with various platform.

Developer can make an application in variety of .NET applications programming

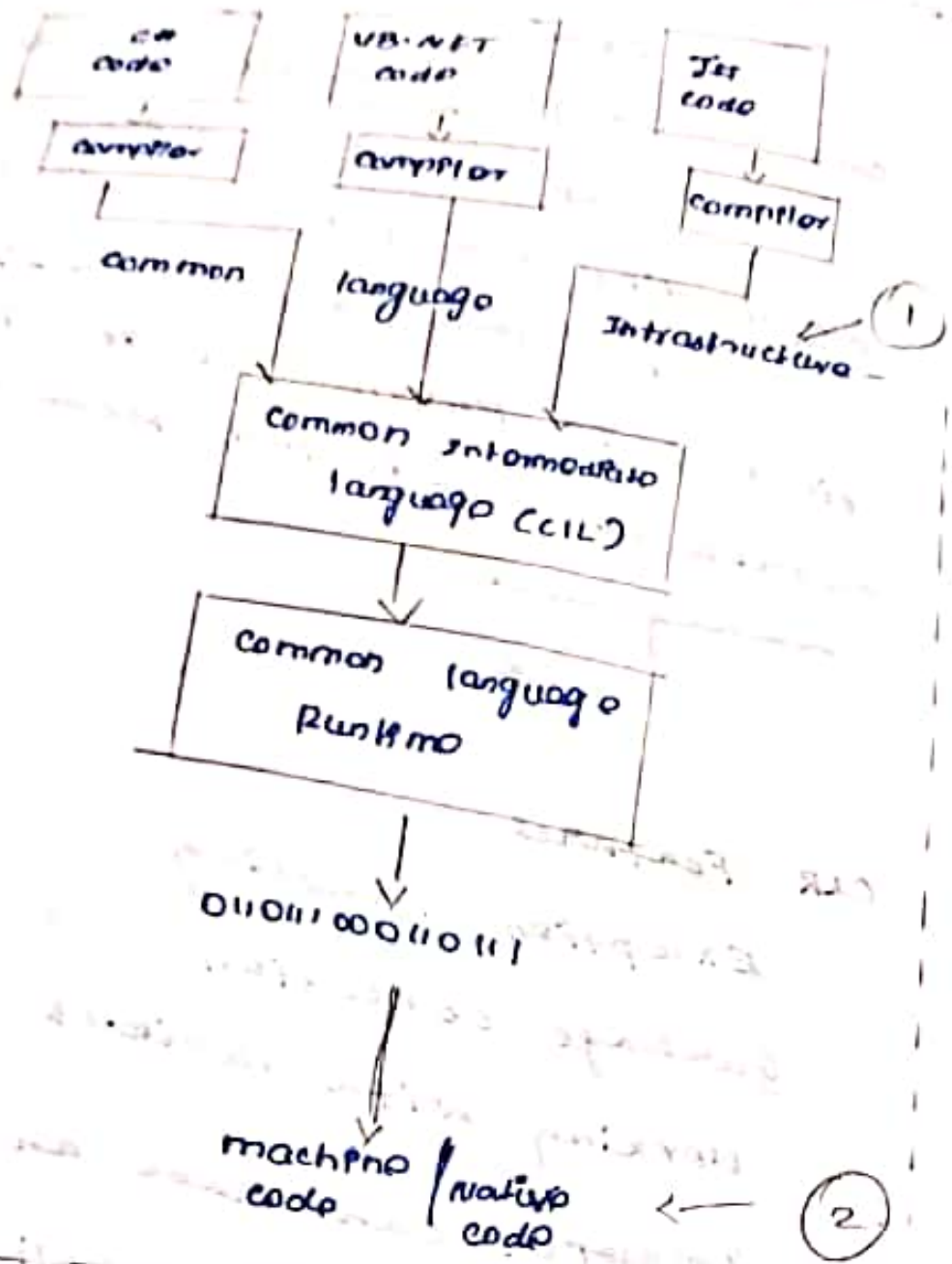
lang. The lang. may be VB.NET or

C#.NET.

the subsequent compiler (VB-compiler) sent to

Program to CLI - Common Lang. Infrastructure

to run the app.



1. .NET compiler lang. compiler to another platform. Called CLR Neutral lang.

2. The platform specific CLR compiler

CLR into machine readable code that

enables executed in current platform.

The compiler translates source code into MSIL which is a CPU independent set of instructions.

MSIL includes instructions for loading, storing, initializing & calling methods on object as well as instructions for arithmetic & logical operations control to direct memory access & exception value.

Before code can be executed MSIL must be converted into CPU specific code by JIT compilation.

CLR manages the common type system (CTS)

Managed code:

The code whose execution is managed by CLR is called managed code. The code does not rely on the CLR. It is called unmanaged code.

VES: Virtual Execution System.

It is a part of the CLI which provides environment for executing managed code.

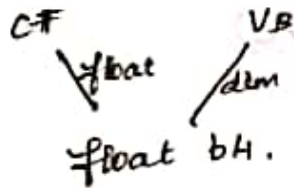
VES provides support for a set of built-in data types flow control constructs and exception handling modules.

Common Type Systems: (CTS)

CTS prevents broadcasting. It specifies rules related to datatype that lang. must follow.

CLS is a subset of CTS that allows different languages to interoperate.

CTS is a data log of .NET types



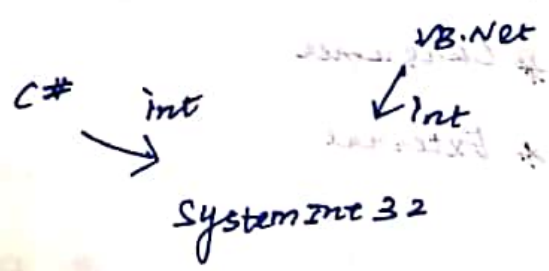
It defines how types are declared used and managed in CLR it supports cross language integration.

Functions:

Establishes framework which support cross language, under integration types & high performance code integration.

Provides object oriented model that support complete implementation of many programming language. Defines rules that language must follow which ensures that objects return same different language with each other. Provides a library that contain the primitive data type such as boolean, byte, char, Int32, UInt64 used in application development.

CTS & Standardised are all programming lang. using .NET under the umbrella of .NET of a common data type for lazy & smooth communication among the .NET lang.



## CLS (Common Lang. Specification)

27

CLS is a collection of rules & restrictions that allows interoperability among between lang. Interoperability among appls. CLS is a set of constraints (Basic lang. features) & constraints that serves as library writers & compiler

writers. CLS defines subsets of CTS. CLS is a set of restriction on the CTS. CLS defines not only the types allowed in external calls but the rules for using them. CLS is a simply a contract

between programming lang design and class library are author.

CLS is a set of rules that lang. compiler must adhere to in order to create. Three levels of comparability are;

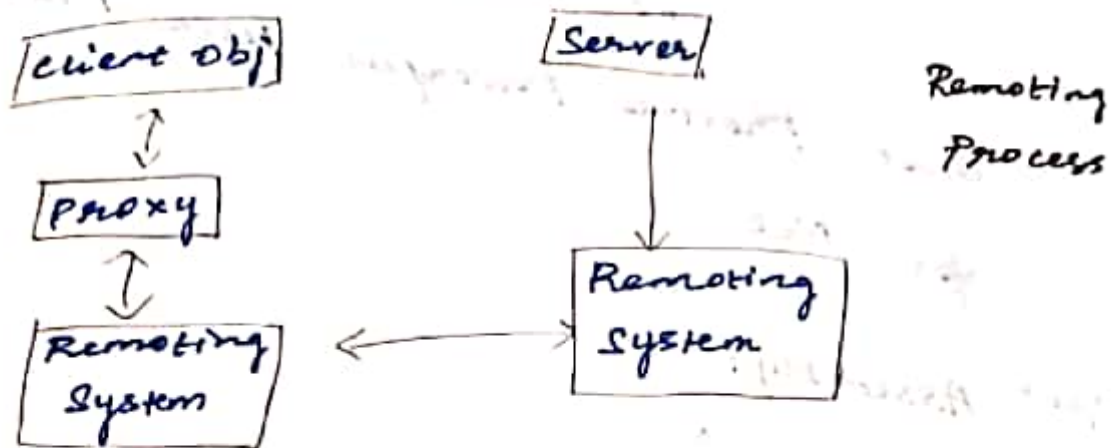
- \* Producer
- \* Consumer
- \* External



It is generic System for communication b/w different application.

These applns. are applied on same computers, different computers on same net.

It is a inter process communication



- .Net Framework class library:

It is a managed classes that provide access to system services.

It is comprehensive & object oriented code of reusable types that can be used to develop UI or web applications.

File I/O, Socket Programming, database access, remoting are few

Services of FCL

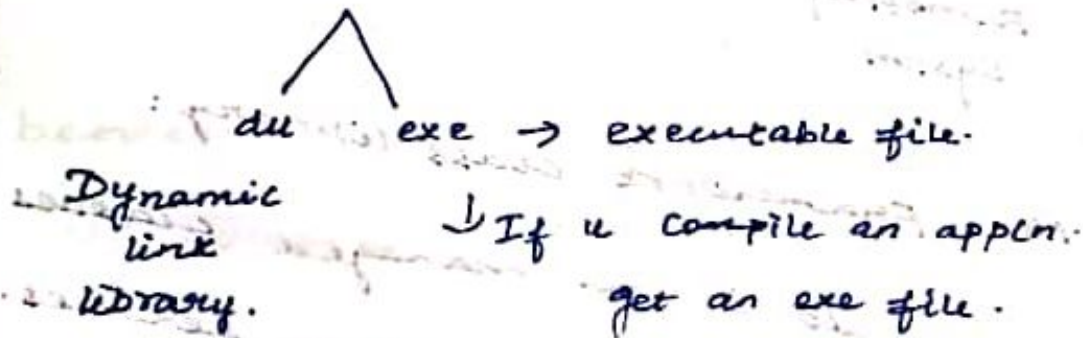
and Standardised attacks rang. by  
bringing them under the control of  
.Net framework.

User & Program Interface:

It includes windows forms  
that provide powerful user interface  
for web.

22/11/2020

Assembly:



↓  
If u compile class files  
get an dll file.

Assembly is a self description.

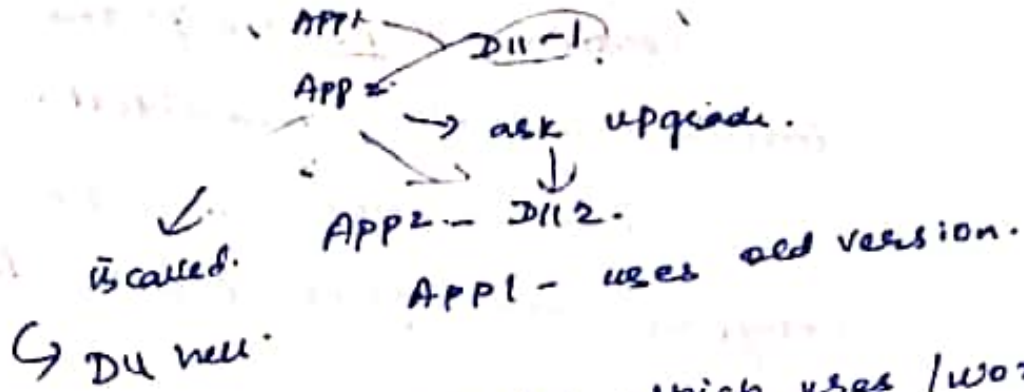
Fundamental unit of deployment.

Implement ⇒ type the code.

Put the project to client ⇒ Deployment.

# 1. Version control

→ Versioning



Develop a new version which uses / work both apps. A control should pass to the assembly. called versioning.

This problem is called DLL-hell.

## Assembly

Fundamental unit of deployment, version control, reuse, security permission for a .NET based app.

logical unit of distribution.

It takes a form of .dll or .exe file. It is a building block of .NET apps.

It can consist of single or multiple files called modules.

A module may be generated from different languages.

## Features of Assembly:

Assemblies are self describing.

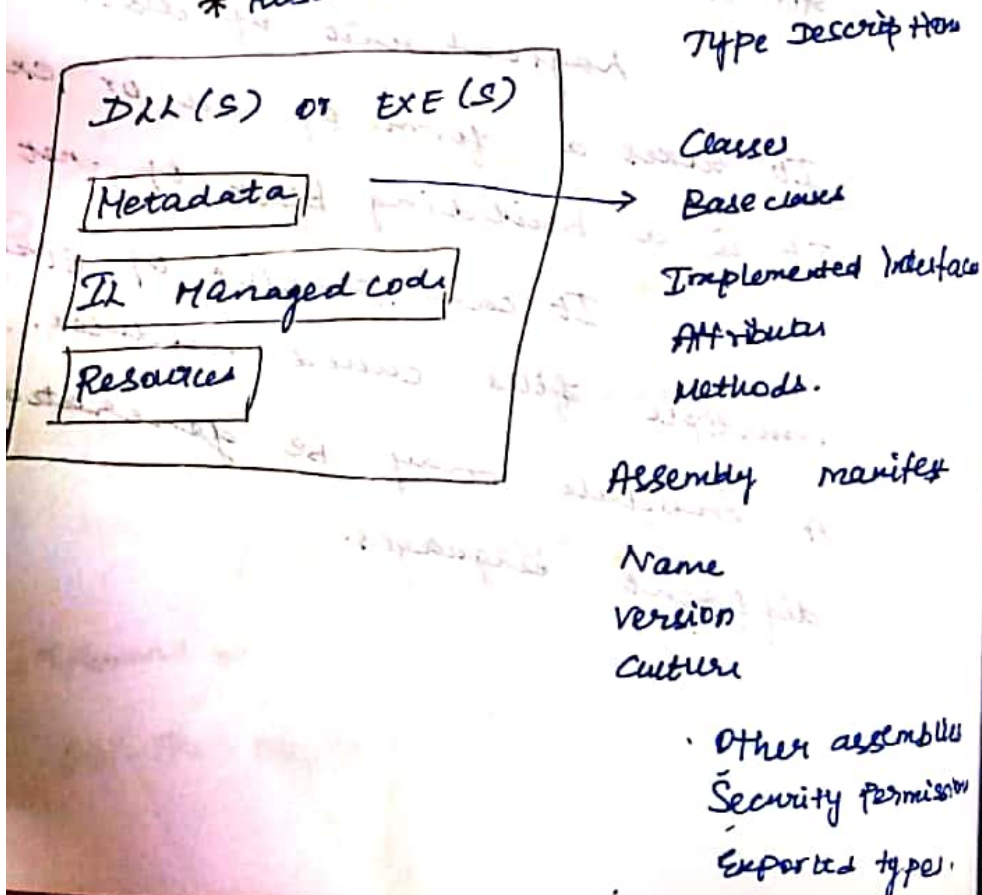
Version dependencies are recorded inside assembly manifest.

Installation can be easy by copying the files that belong to an assembly.

Assembly can be private or shared.

## Components of Assembly:

- \* Assembly Metadata
- \* Type Metadata
- \* CIL code
- \* Resources



## Assembly Metadata :

1. Assembly name
2. Version Information
3. Culture
4. Strong name information
5. List of files that makeup assembly
6. Type reference information
7. Reference assembly information.

## Type Metadata

- \* Information about types
- \* classes, interfaces, structs, events indexes that are used.

i.e. methods, attributes, properties.

## Features of Metadata :

\* JIT compiler gets everything it needs from compiler

(385) \* Used for garbage collection

\* .Net provides classes to read Metadata (Reflection API).

CLI/MSIL code:

\* code generated by an appropriate  
of net language compiler (C# compiler)

\* MSIL shipped in portable  
Executable (PE) units.

\* Later JIT compiled to native  
code.

\* Full IL code can be  
compiled to native code using  
NGEN.exe.

Resources:

\* Non executable data that is  
part of application.

Eg: images, sounds, videos etc.

Integrated Development Environment (IDE) for  
VB.NET

↳ Env. to develop a PP  
Start from IDE  
command prompt/dcmd

- \* Visual Studio 2010 (VS)
- \* Visual Basic 2010 Express (VBE)
- \* Visual Web Developer.

microsoft

31

Writing VB-Net Programs on Linux & MacOS.

Mono is the Open-Source version of .Net

Flow:

- \* Runs on several OS.
- \* VB 2012 (part of Visual Studio)
- \* Cross platform (distributed app)
- \* Brings better development tools
- \* Mono runs on

Android, iOS, Linux, Windows, Solaris

unix.

Basic building blocks / programs structure

of VB-Net:

VB-Net program consist of

\* Namespace declaration.

\* A class or module

\* One or more procedures

\* Variables

\* The main procedures

\* Statements & expressions

\* Comments.

Func -> return value

Sub -> don't return value

↳ Assign some values.

- collection of classes
1. Func.
  2. Sub
  3. Operator
  4. Get
  5. Set
  6. Add item
  7. Remove item
  8. Raise event

(2m)  
(10T)  
(3m)

Imports System

Module Module1

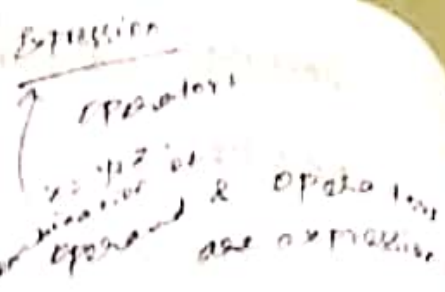
Sub Main

Console.WriteLine("Hello world")

Console.ReadKey() → for until the o/p is displayed press any key

End Sub

End Module



O/P: Hello world.

Compile & Execute VB.Net

\* Start Visual Studio

\* On menu → choose file → New →

Project

\* Choose VB template

\* Choose Console Application.

\* Specify Name & location to project.

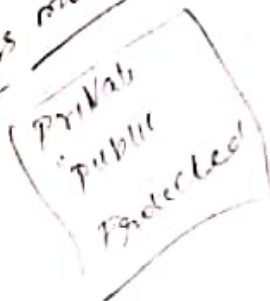
\* Then Press ok.

\* New project appear in Solution Explorer.

\* Write code in code editor & press

F5 button.

Access modifiers





objects:

objects have states & behaviours

Ex: Dog has state - colour, name, breed etc..

Behaviour - walking, eating etc..

class:

class can be defined as a template / blueprint that describes states / behaviours that object of its type support.

Method: Methods are basically behaviours. Logics are written, data is manipulated, the actions are routed.

Instance variable: Each object has its unique set of instance variables. The

objects - state is created by the values assigned to those instance variable

Program:

```
Imports System
```

```
public class Rectangle
```

```
    private length, breadth as Double.
```

```
    public methods
```

function getArea() as Double

GetArea = length \* breadth

End function

Public Sub display()

C.W ("length: {0}", length)

C.W ("width: {0}", breadth);

C.W ("Area: {0}", GetArea())

End Sub

Shared Sub main()

Dim r as New Rectangle()

r.AcceptDetails()

r.Display()

C- Readline

O/P: length = 4.5

breadth = 3.5

End Sub

End class

Area = 15.75

Dim

↳ can declare

any variable

28/1/2020

Objects:

2m  
or  
5m  
35

Real time entities that have identity, State & behaviour.

It is instance of class

class:

Group of

class is a description which have commonality.

class is a

CIA - Confidential Integrity Availability

Objects: Objects are real time entities that have state, behaviour & Identity.

Eg: A tree, a car, a person, a dog

Not an object: affection, feelings/love.

Adv: of Object oriented:

\* Reusability.

\*

OOPS principle:

Inheritance

Polymorphism

Encapsulation

Data hiding

Abstraction.

9/1/2020

Namespace:

collection of classes.

2m  
(or)  
5m

Inheritance is one relationship

IIS - Internet Information Services.

A Sale	Sale no: 1A3H3	Earn loyalty points	unsold, opened, empty
--------	-------------------	---------------------------	-----------------------------

State:

A condition of object at any moment, affecting how it can behave

Behaviour:

What an object can do, How it can respond to events and stimuli

Identity:

Each object is unique,

Class & Instance

All objects are instances of a class.

class

37

A description of a set of objects with similar attributes, operations, methods, relationships and sy.

Semantics.

Namespace:

Namespace is logical division of class, structure and interface.

Namespaces are way of grouping, typings<sup>& name</sup> and reducing chance of name

Collision.

Namespace with all the builtin functionality comes under 'System' namespace.

Namespace declaration in VB:

```
Imports System;
```

```
Imports System.Data;
```

In C#

```
using System;
```

```
using System.Data;
```

Eg: for Namespace.

```
namespace namespace
```

```
{
    ↪ keyword
```

```
class TestSample
```

```
{
```

```
    public void message()
```

```
{
```

```
    console.WriteLine("This is namespace");
```

```
}
```

```
}
```

```
}
```

### Advantage of Namespace

\* We can establish security, version, reference and deployment boundaries using namespaces.

\* Because of grouping namespace we can create hierarchy which is easy to identify classes by fully qualified names.

\* We can avoid name collisions.

VB-Net :

Lang. that support .net <sup>framework</sup> base.  
class library] is called VB-Net.

VB is not support by .net

VB-net support .net framework

Released 2003.

Features of VB-Net :

It is object oriented Program  
OR multi-paradigm language.

Real time Programs => OOPS

It supports OOP &

Automated garbage collection.

Can have GUI.

Disadv :

\* No pointer

\* Challenging secure job

Bez of security lack pointer

dropped in Java.

Features:

- \* Not case sensitive
- \* Object oriented.
- \* Automatic code formatting
- \* XML design, improved object browser
- \* Automated Garbage collection.
- \* ~~Multi~~ Multithreading support
- \* A standard library
- \* Simple generics  $\rightarrow$  class library. (store)
- \* Windows form.
- \* Event Management

DB is reference  $\leftarrow$  original obj

\* References. Referencing an external object.

\* Attributes = Pages provide additional information.  $\rightarrow$  Add info.

\* Indexes & versioning

VB-Net:

VB-Net stands for visual

basic-net which is a programming lang. developed by microsoft released first in 2002 to replace visual basic. It is a object oriented programming lang., supports OO concepts that include encapsulation, polymorphism, abstraction.



VB-Net concern .Net framework which means it has full access to .Net libraries. It helps in rapid creation of web, windows, office and mobile applications. It is improper with interoperability with other .net langs. like Visual C++, VC#, VJ#

VB-Net is multi paradigm language. First version is VB-Net 7.0

Year	Name	Version
2002	VB-Net 7.0	.Net version 1.0
2003	VB-Net 7.1	.Net version 1.1
2005	VB-Net 8.0	-
	<u>update</u> : Partial class, generics, operator overloading etc.	
2008	9.0	.Net 3.5
	Linq, lambda exp, type inference etc.	
2010	VB-Net 10.0	VB-Net C# class data
	Dynamic language runtime,	

2012 VB.Net 11.0, Net 4.5 42

Global keyword, Namespace etc.

2015 VB.Net 14.0 VB.Net 2015

formatting string, '?' operator etc.

2017 VB.Net 15

Organising source code in just single action was introduced.

Adv. of VB.Net :

- \* Code formatted automatically
- \* Enterprise class code can be generated.
- \* web with performance counter, event logs the system.
- \* Drag & drop capability for elements (Java -> Swing)
- \* Connecting applications
- \* Docking, automatic control anchoring.
- \* A Form can be divided into blocks and called multithread.

## Identifiers :

43

Name used to identify a class, variable, func, or any other user defined item.

- A name must begin with a letter and followed by digit (0-9)

the first character of an identifier cannot be a digit.

No embedded symbols like ? - + !

@ # % ^ & \* ( ) [ ] { } ~ , : ' ' / and \.

It should not be a reserved

## Keyword.

### VB.Net Keywords:

Address	Function	Continue
class	friend	
byte	of	
double	object	
error	string	
Alias	try	
And	type	
As		

Data types in VB.NET

Data type	Storage Allocation	Value Range
Byte	1 byte	0-255 (unsigned)
Char	2 byte	0-65535 (unsigned)
Date	8 byte	0:00:00 (midnight) on Jan 1, 0001 to 11:59:59 on Dec 31, 9999
Decimal	16 byte	-1.7976931348623157E+28 through +1.7976931348623157E+28 neg. values through -1.7976931348623157E+28 for the value through +1.79E+28 28/28 decimal places to the right of the decimal
Decimal	16 byte	through to the right of the decimal
Integer	4 byte	-2147483648 through 2147483647 (Signed)
Long	8 byte	-9223372036854775808 through 9223372036854775807 (Signed)
Object	4 byte on 32 bit	Any type can be stored in a variable of type object.

Byte	1 byte	-128 to 127	45
Short	2 bytes	-32768 to 32767 (signed)	
String	Represents on platform	0 to 2 billion unicode characters.	
Integer	4 bytes	0 - 4294967295	(Unsigned)
Userdefined	Structure		

types of .

=> Plenty of data are available.

=> Don't want to store data in same data so can use dif. data types.

=> Easily retrieve data are called data structure.

Signed => -255 to 0.

Unsigned => 0 to 255.

unboxing => obj to var.

boxing => var to obj.

Program for demonstrating datatypes:

Module Datatypes

Sub main ()

Dim b as Byte

Dim n as Integer

Dim Si as Single

Dim d as Double

Dim dn as Date

Dim c as Char

Dim s as String

Dim bl as Boolean

b = 1

n = 1234567

Si = 0.12345678901234566

d = 0.12345678901234566

dn = ~~Thursday~~ Today

c = "U" &

s = "Kumar"

If ScriptEngine = "VB" then

bl = True

If Else

bl = false

End if

If ... then

C. N (C & " and " & S2 .val (vlt)

C. N (". declaring on the day of { 0 }", da)

C. W (" the single : { 0 } , the double { 1 } ,  
S1, d.)

End if.

C. Readkey()

End sub

end module

o/p

V and Kumar

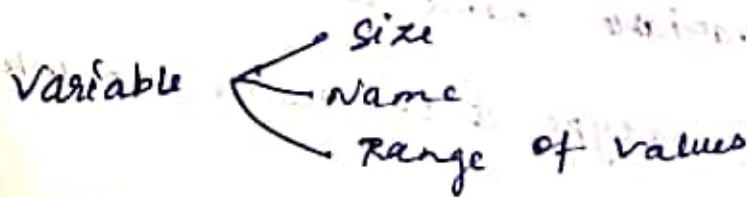
Declaring on the day of 04.2.2020 11AM

The single = 0.1234568

The double = 0.123456789012346.

### Variable Declaration:

↓  
A name given to a storage area.



### Basic Value Types:

Integral Types => Sbyte, Byte, Short  
 ushort, Integer, UInteger  
 Long, ulong, char

types => 'True' / False

Datatype => Date

Other types:

Enum => Value type

class => Reference types of variable

Declaration:

Dim => Statement - variable declaration  
& storage allocation for one/more variables.

Dim => module, class, structure, procedure  
or block level.

Syntax:

Dim Variable - Name as Datatype.

Variable - name is the variable name and

Datatype is the name to which variable

belongs.

Dim X as Integer

Variable name  
Variable datatype



## Variable Initialization

49

```
Dim x As Integer  
x = 10.
```

```
Dim name As String  
name = "John"
```

Accepting user values

```
Dim username As String  
username = Console.ReadLine  
↓  
Read from console.
```

// Program for Declaration of variables

```
Module module1
```

```
Sub main()
```

```
Dim username As String
```

```
username = Console.ReadLine ("Enter n
```

```
username = C.R.
```

```
C.W()
```

```
C.W()
```

```
C.ReadLine()
```

```
End Sub.
```

```
End module.
```

Declaration of variables: 50  
Constants: Fixed values that the program may not alter during its execution.

Const average AS long = 11999  
Private Const private AS double = 3.1415  
Enum: is a set of named integer constants  
Module ConstantEnum.  
Enum color = 0 red = 1 green = 2  
Blue = 3.  
End Enum. End module

Scope of variable:

The scope (visibility) of a variable is the section of the application that can see and manipulate the variable. (lifetime) variable.  
→ block, func, module.  
→ can change variable or not if visible & change

Local variable:

The scope is limited to a Procedure (Procedural-level) Block-level variables introduced in a block of code.  
Eg: If statement or a loop. - The variable is invisible outside the loop block.

module variable.

51

A variable that is available to all procedures. That must be declared outside the procedure.

Public variable:

Global scope - They are visible from any part of the application.

// Program:

Private Sub button\_click (Byval sender As object, Byval AS System EventArgs) Handles Buttonclick.

Dim i as Integer

Dim sum as Integer

For i = 0 to 100 Step 2

sum = sum + i;

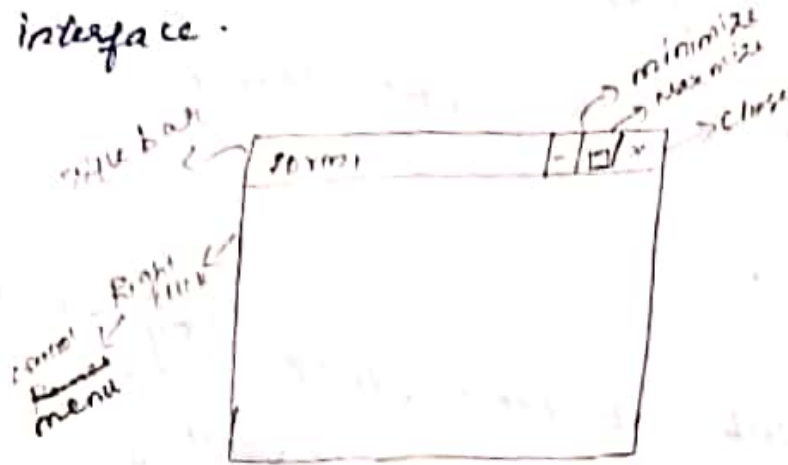
Next

msgbox

"The sum is " + sum.ToString

End sub.

interface.



MS Visual Studio → File → New Project -  
Windows Form applications → OK.

Form Properties:

The Properties can be set/  
read during the application execution.

AcceptButton → OK button → Automatically activated  
while pressing enter key.

CancelButton → Cancel button → 'Esc' key.

AutoSize → Automatically scaled to a  
height based on current font.

AutoSize

Autoscroll minimize	BorderStyle	HelpButton	
AutoScroll Position	ControlBox	Height	53
BackColor	Enabled	MinimizeBox	
Topmost	Font	MaximizeBox	
Width	Text	Name	
	TopLeft	StartPosition	

Form Methods:

Activate - Activates the form and gives it focus.

ActivateMdiChild - Activates MDI child of a form.

- AddNewForm
- BringToFront
- CenterToParent
- CenterToScreen
- Close
- Contains
- Focus
- SetAutoScrollMargin
- Show
- ShowDialog
- SetDesktopBorder
- Hide
- Refresh
- Scale (Size F)
- ScaleControl
- ScaleCore
- Select
- SendToBack
- SendAutoScrollToBack
- SetDesktopBackLocation

File -> Add new form  
 ↓ is called MDI child.  
 ↓ multi document interface

Properties which describe the object 54  
 Methods cause an object to do something  
 Events are actions when an object does something.

Property can be set at design time  
 by using Properties window.  
 or at run time by using statements

Object.Property = value.  
 ↙ ↘  
 Name of the object      characteristic  
 You can change      You can't change  
 Form1.Caption = "Hello"  
 ↘ ↙  
 New Property Setting.

Message Box control:

Public class Form1.

Private sub Button1\_Click (ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click

MessageBox.Show("Hello");

End sub.

End class.

↖ eg. for adding control.  
 (event handling)

Reference  
 ↓  
 obj pass.

e As EventArgs  
 ↓  
 Mouse location  
 ↘ Right / left.

Object is a control.

Form events:

- |             |                   |                |
|-------------|-------------------|----------------|
| Activated   | HelpButtonClicked | MouseLeave     |
| click       | KeyDown           | MouseMove      |
| closed      | KeyPress          | MouseUp        |
| Closing     | KeyUp             | MouseWheel     |
| DoubleClick | Load              | Move           |
| DragDrop    | LostFocus         | Resize         |
| Enter       | MouseDown         | Scroll         |
| GetFocus    | MouseEnter        | Shown          |
|             | Mouseover         | VisibleChanged |

// Program:

Public class Form1

private Sub Form1\_Load (Sender As Object, e As EventArgs)

Dim Button1 As New Button()

Dim Button2 As New Button()

Button1.Text = "OK"

Button2.Text = "Cancel"

Button2.Location = New Point (Button1.Left, Button1.Height + Button1.Top + 10)

Me.Text = "HSC Student"

56

Me.FormBorderStyle = FormBorderStyle.FixedDialog

Me.Height = 300

Me.Width = 560

Me.Controls.Add(Button1)

Me.Controls.Add(Button2)

End Sub

End class

## Exception Handling

(An exception is a response to an exceptional circumstance that arise while a program is running, such as attempt to divide by zero.)

Four keywords to handle exception:

\* Try => A try block identifies a block of code for which particular exceptions activated.

\* Catch => Program catches an exception with an exception handler at the place in a program where you want to handle the problem.



\* Finally => Executes a given set of 5-7 statements whether an exception is thrown or not thrown.

\* Throw => Program throws an exception when a problem shows up.

Syntax:

Try [try statement]

Exit Try

[Catch [Exception [AS type] [when expression]

[Catch statements]

[Exit Try]

[Catch ...]

[Finally

[Finally statement]

End Try.

Exception classes

( ) SystemException class

SystemApplicationException

System SystemException are derived from

SystemException class

```

Sub main ( )
  division (75, 0)
  Console.ReadKey ( )
End Sub
End module.

```

DIP: Exception caught  
 System.DivideByZeroException  
 Attempted to divide by zero.  
 Result = 0.

On Error GOTO Statement :

'On Error' GOTO Statement is an example for unstructured exception handling. Try-catch - Finally - Structured Exception handling.

Error GOTO redirect the flow of the program to a given location.

```

Public class Form1
Sub Button1_Click (Byval sender As System.Object, Byval e As System.EventArgs)
  Handler Button1_Click.

```

Exception class	Description
System.IO.IOException	Handles I/O events
System.IndexOutOfRangeException	Array index out of range.
System.DivideByZeroException	dividing by zero error
System.OutOfMemoryException	Insufficient free memory.

// Program for Exceptional Handling:

Module Exception prog

Sub division (By val num1 AS Integer, By val num2 AS Integer)

Dim result as Integer

Try

result = num1 / num2 ;

Catch e AS DivideByZeroException

C.W ("Exception caught. {0}", e)

Finally

C.W ("Result: {0}", result)

End Try

End Sub

OnError Goto nextStep

Dim result as Integer

Dim num as Integer

num = 100

result = num/10

nextStep :

msgbox ("control here")

} Label Statement

End Sub

End class.

### Control Statement:

Simple If.

→ Keyenter  
Keyboard event.  
wid prgm.

If then

If else.

If elseif.

Inputbox → eg.

Class

Object.

Exception handling.

ADO.Net architecture

CTS, CLR.

Dll hell problem.

msgbox, Assembly.

Event handling.

EventArgs.

eg: prgm.

Mouse Event.

click.

$$6 \times 2 = 12.$$

$$3 \times 6 = 18$$

$$2 \times 10 = \frac{20}{50}$$

1. Form.

2. ADO.Net.

Event

An event is an action that calls a function or may cause another event.

Event Handler: Event handler are functions that tell how to respond to an event.

Eg: clicking a button, entering some text in a text box, clicking a menu.

VB.NET is an event-driven language

- mouse event
  - keyboard event
- } two types of event.

Handling mouse event:

Mouse down

Mouse enter

Mouse over

Mouse leave

Mouse move

Mouse up

Mouse wheel

The event handler is mouse event args. which has properties

1. Button => pressed / not.

2. click => No. of clicks

3. Delta - no. of dents the mouse wheel rotated.

4. X => Indicate X coordinate

5. Y => " Y "

Public class form1

Private sub form1\_load (Sender As Object, e As EventArgs) Handles MyBase.Load

Me.Text = "

End Sub

Private sub txt1\_TextChanged (Sender As Object, e As EventArgs) Handles txt1.TextChanged  
Me.Text = txt1.Text

txt1.BackColor = Color.Blue

txt1.ForeColor = Color.White

End Sub

Private sub txt1\_MouseEnter (Sender As Object, e As EventArgs) Handles txt1.MouseEnter  
txt1.BackColor = Color.White

txt1.ForeColor = Color.Blue

End Sub

Private sub txt2\_TextChanged (Sender As Object, e As EventArgs) Handles txt2.TextChanged

txt2.BackColor = Color.Blue

txt2.ForeColor = Color.White

End Sub

Sub txtname - MouseLeave (Sender As Object, e As EventArgs) Handles txtname.MouseLeave.

txtname.BackColor = Color.White.

txtname.ForeColor = Color.Blue.

Private Sub txtAddress - MouseEnter (Sender As Object, e As EventArgs) Handles txtAddress.MouseEnter  
txtAddress.BackColor = Color.White  
txtAddress.ForeColor = Color.Blue.

End Sub.

Private Sub txtAddress - MouseLeave (Sender As Object, e As EventArgs) - Handles txtAddress.MouseLeave  
txtAddress.BackColor = Color.White

Unit - 2

Intro. to VB .Net .

Net Framework & CLR .

Building VB .NET appln .

VB IDE

Forms

Properties

Events

Datatype .

Declaring variable

Scope of variable

Operators & Statement

Window APP form

46 windows app .

22/12/2020

# Introduction to C# - Overview

## Features of C# :

First Release 2002

C# 1.0	- Net 1.0 / 1.1	VS. Net 2002.
2.0	- Net 2.0	VS " 2005
3.0	" 3.0 / 3.5	" " 2008.
4.0	- Net 4.0	" 2010.
" 5.0	" 4.5	" 2012 / 2013
6.0	4.6	" 2013 / 2015
7.0	- Net Core 2.0	" 2017.
8.0	- Net Core 3.0	" 2019

- Net core is used for

distributed operating system. like DOS, linux, unix.

C# is a very simple language.

Syntax is similar to Java.

using System (package is used)

Namespace is used. object orient

Prog. lang.





Abstraction  
Inheritance  
Polymorphism

Features  
of oops.

It is a type safe.  
C++ only access memory directly.

(Own memory).

Features of C++: (S, P, M) . P.M)

Simple:

It is a model prog. lang.

Object oriented.

Speedy.

Typesafe

Interoperability.

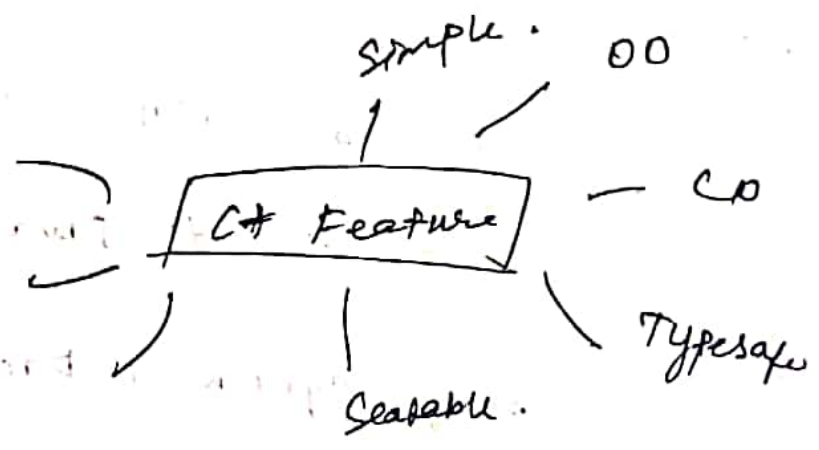
Scalable & updatable.

Component oriented

~~Structure prog~~

Structured programming lang.

Rich library



It is a simple lang. & It provide  
structured approach.

It is based upon the current trend  
& it is very powerfull.

An. OOP. & hence makes maintainence  
As a Where procedure oriented programming  
lang. is not easy to manage while the  
code grows.

C# type safe <sup>code</sup> can only access  
the memory location, that it has  
permission to execute. Therefore it improves  
security of the program.

C# is automatic Scalable &  
updatable.

It is the predominant S/W  
development methodology to develop more  
robust & highly Scalable applications.

We can break the program <sup>into</sup> parts using functions. So it is easy to understand and modify.

It has lots of built-in functions that makes the development faster.

Compilation & execution time is fast.

Creating C# Sample Program:

Class Program

{

Static void main (String [] args)

{

System.Console.WriteLine ("Hi Students");

}

}

O/P:

Hi Students.

0 1 1 2 3 5

68

4/1/2020

Program for fibonacci series in C#.

✘

class fibonacci

{  
static void main (string [] args)

{

int n1 = 0, n2 = 1, n3, i, no

System.Console.WriteLine ("Enter n  
value = ");

no = Console.ReadLine ();

for (i = 2; i < no; i++)

{

n3 = n1 + n2;

Console.Write (n3 + " ");

n1 = n2;

n2 = n3;

n3 = n1;

Console.ReadKey ();

$$n1 = 0, \quad n2 = 1$$

$n0 = 5$

$$r = 2 < 5.$$

$$n3 = 0 + 1 = 1.$$


$$n1 = 1$$

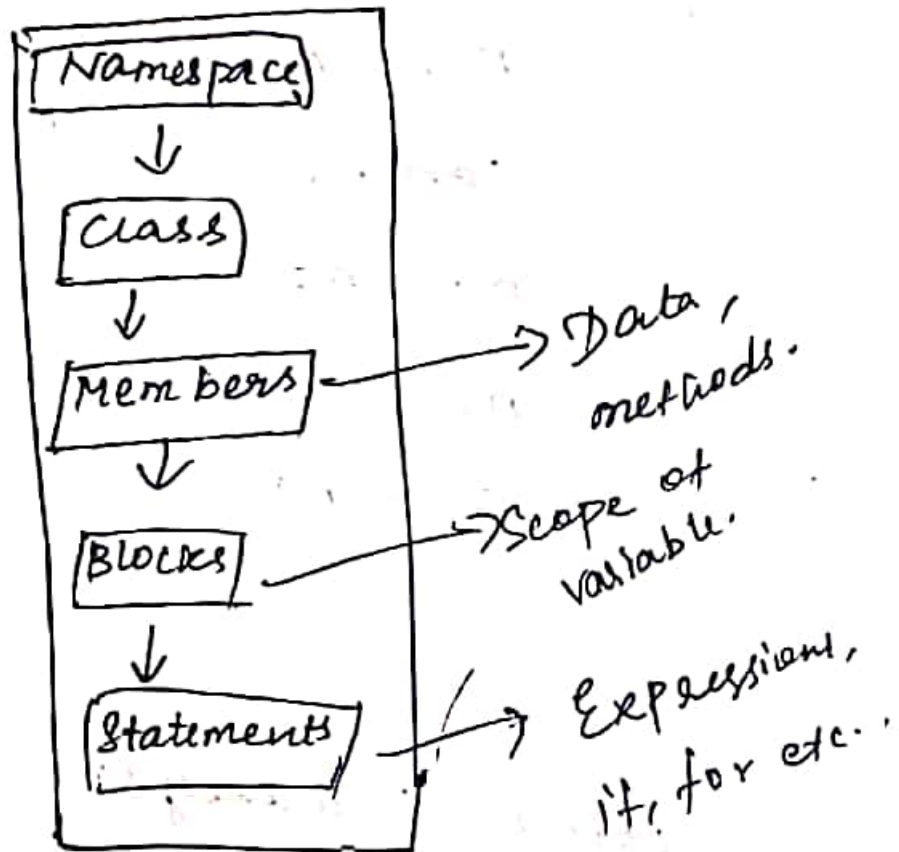
$$n2 = 1$$

$$n3 = 0.$$

BCL (Base class library)  
Several namespaces  
" classes  
sub namespaces  
sub classes.  
⇒ using dot operator  
Whole class library  
Supporting

3/2020.

Basic Structure of C# 



using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

To  
BCL  
Importing  
namespace.

namespace FirstPrgrm → Namespace declaration

{

class Program

{

static void main (String [] args)

{

Console.WriteLine ("Hi Students");

}

}

}

Namespaces are given to avoid name collision.

### Importing Section:

This section contains importing statement to import base class library. If required namespace is a member of another namespace we have to specify a parent/child namespaces separated by dot (.).

Eg: using System. Data 71  
using System. IO.

### Namespace Declaration:

Here a user defined namespace is to be declared.

All classes related to a project should be declare inside one namespace.

Generally namespace name will be same as project name.

### Class Declaration:

This is to declare the startup class. In every .Net applns. like console applns & windows applns there should be a startup class. Here

Program is using a startup classname.

A startup class or main class contains main method.

Main Method : 72

main () method is the starting execution point of the applns.

"Using" is a keyword using this refer to .net .pcl in C# applns. in .net the base class libraries are divided into collection of namespace.

Each namespace contains a set of predefined classes & sub namespaces.

Namespace contain another namespaces is called

Variables & Data Types :

Integral types - sbyte, byte, short, int, long, ushort & char.

Floating point types - float, double.

Decimal points - Decimal.

Boolean types - True/False

Nullable types - Nullable data types.



Defining variables → int can accept variable value varying

Data-type variable list; 73

int i, j, k;

char c, ch;

float f, salary;

int i = 100; // initialization

Accepting values from users!

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

Boxing / unboxing

any value  
convert into  
object type.

↑ object type is  
convert into  
value type.

(-3) Switch (ch)

ch = 1

2

3

Unit – II - Developing VB.NET Applications - Introduction to VB.Net, The .Net Frame work and Common language runtime, Building VB. Net Application, VB IDE, forms, properties, events, VB language-console application and windows application, data type, declaring variable, scope of variable, operators and statements - Windows Applications-forms, adding controls to forms, handling events, MsgBox, Input Box, multiple forms, handling mouse and Keyboard events, object oriented programming creating and using classes and objects, Handling Exceptions- on Error Goto.

## Operators and Statements

### ARITHMETIC OPERATORS

Operator	Mathematical Function	Example
+	Addition	1+2=3
-	Subtraction	10-4=6
^	Exponential	3^2=9
*	Multiplication	5*6=30
/	Division	21/7=3
Mod	Modulus(returns the remainder of an integer division)	15 Mod 4=3
\	Integer Division(discards the decimal places)	19/4=4

```
Private Sub BtnCal_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles BtnCal.Click
```

```
Dim num1, num2, difference, product, quotient As Single
```

```
Dim num1 As Single, num2 As Single
```

```
Dim sum As Single, diff As Single, pdct As Double, quo As Double
```

```
num1 = TxtNum1.Text
```

```
num2 = TxtNum2.Text
```

```
sum=num1+num2
```

```
difference=num1-num2
```

```
product = num1 * num2
```

```
quotient=num1/num2
```

```
LblSum.Text=sum
```

```
LblDiff.Text=difference
```

```
LblPdct.Text = product
```

```
LblQuo.Text = quotient
```

```
End Sub
```

Upon running the program, the user may enter two numbers and click on the calculate button to perform the four basic arithmetic operations. The results will be displayed the on the four labels, as shown in Figure

A screenshot of a Windows application window titled "Aritmetic Operations". The window has a standard Windows title bar with minimize, maximize, and close buttons. Inside the window, there are six input fields, each with a label to its left: "Number 1" (value: 436), "Number 2" (value: 24), "Sum" (value: 460), "Difference" (value: 412), "Product" (value: 10464), and "Quotient" (value: 18.1666666666). At the bottom of the window is a "Calculate" button.

### Conditional Operators

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Equal to or Greater than
<=	Less than or Equal to
<>	Not equal to

### Logical Operators

we might need to make more than one comparisons to arrive at a decision. In this case, using numerical comparison operators alone might not be sufficient and we need to use the logical operators.

Operator	Description
And	Both sides must be true
Or	One side or other must be true
Xor	One side or other must be true but not both
Not	Negates true

## Using the If control structure with the Comparison Operators

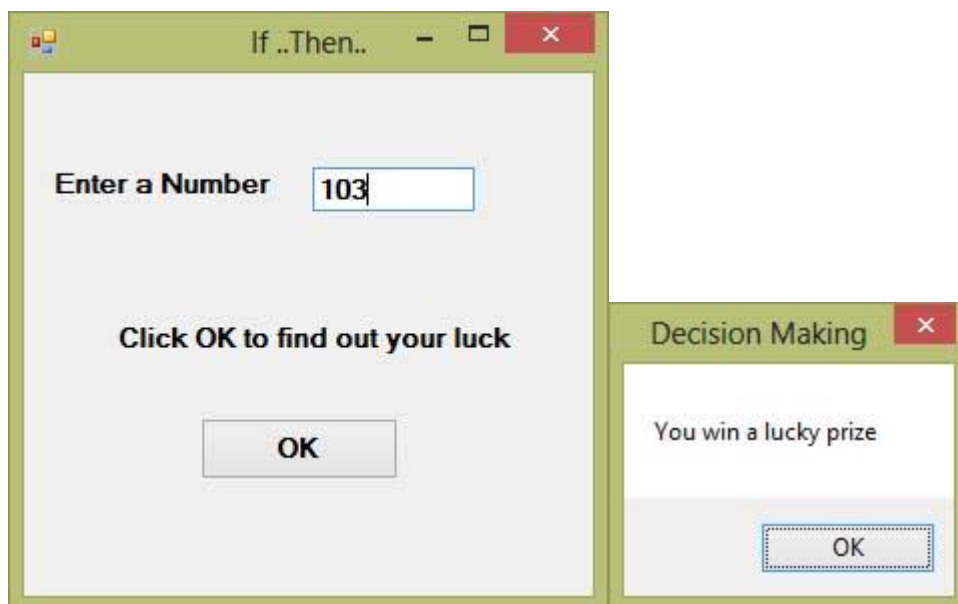
### IF...THEN STATEMENT

This is the simplest control structure which instructs the computer to perform a certain action specified by the Visual Basic 2015 expression if the condition is true. However, when the condition is false, no action will be performed.

```

If condition Then
Visual Basic expressions
End If
Private Sub OK_Click(sender As Object, e As EventArgs) Handles OK.Click
Dim myNumber As Integer
myNumber = TxtNum.Text
If myNumber > 100
Then
MsgBox(" You win a lucky prize")
End If
End Sub

```



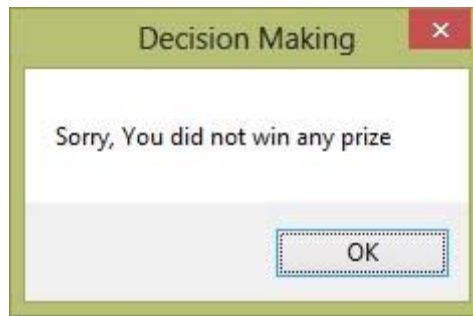
### IF...THEN...ELSE STATEMENT

In order to provide an alternative output, we need to use the **If...Then...Else** Statement.

```

Private Sub OK_Click(sender As Object, e As EventArgs) Handles OK.Click
Dim myNumber As Integer
myNumber = TxtNum.Text
If myNumber > 100 Then
MsgBox( " Congratulation! You win a lucky prize") Else MsgBox( " Sorry, You did not win any prize")
End If
End Sub

```



### IF...THEN...ELSEIF STATEMENT

In circumstances where there are more than two alternative conditions, using just If...Then...Else statement will not be enough.

```
Private Sub OK_Click(sender As Object, e As EventArgs) Handles OK.Click
```

```
Dim Mark As Integer
```

```
Dim Grade As String
```

```
Mark = TxtMark.Text
```

```
If Mark >= 80 And Mark <= 100 Then
```

```
Grade = "A"
```

```
Elseif Mark >= 60 And Mark < 80 Then
```

```
Grade = "B"
```

```
Elseif Mark >= 40 And Mark < 60
```

```
Grade = "C"
```

```
Elseif Mark >= 0 And Mark < 40
```

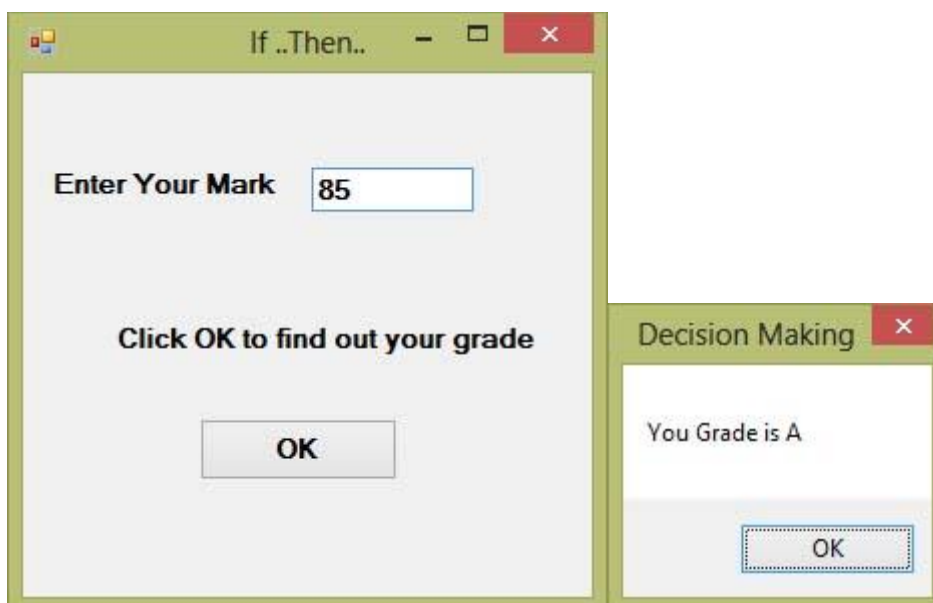
```
Grade = "D"
```

```
Else Grade = "Out of Range"
```

```
End If
```

```
MsgBox("You Grade is " & Grade)
```

```
End Sub
```



## Select case

The Select Case control structure also involves decisions making but it slightly different from the If...ElseIf control structure .The If ... Then...ElseIf statement control structure evaluates only one expression but each ElseIf statement computes different values for the expression. On the other hand, the Select Case control structure evaluates one expression for multiple values. Select Case is preferred when there exist multiple conditions as using If...Then..ElseIf statements will become too messy.

### Syntax

Select Case expression

Case value1

Block VB statements

Case value2

Block VB Statements

Case value3

.

.

Case Else

Block VB Statements

End Select

```
Private Sub BtnShow_Click(sender As Object, e As EventArgs) Handles BtnShow.Click
```

```
Dim grade As String
```

```
grade = TxtGrade.Text
```

```
Select Case grade
```

```
Case "A"
```

```
MsgBox("High Distinction")
```

```
Case "A-"
```

```
MsgBox("Distinction")
```

```
Case "B"
```

```
MsgBox("Credit")
```

```
Case "C"
```

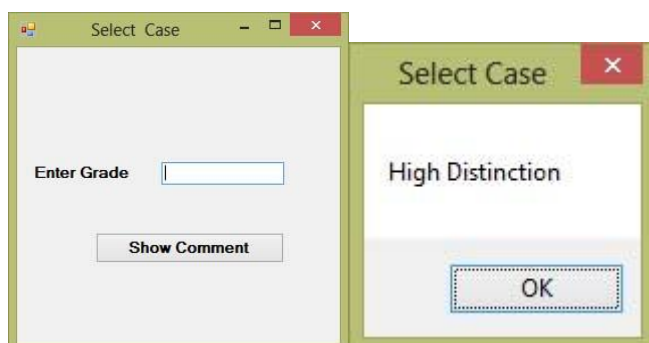
```
MsgBox("Pass")
```

```
Case Else
```

```
MsgBox("Fail")
```

```
End Select
```

```
End Sub
```



## Looping

### Forms

**Visual Basic Form** is the container for all the controls that make up the user interface.

### Scope of variable

<https://www.vbtutor.net/index.php/visual-basic-2017-lesson-9-working-variables-constants/>

In Visual Basic.NET, the **Dim** keyword is used to declare the data.

However, you can also use other keywords to declare the data. Three other keywords are **private**, **static** and **public**.

The forms are as shown below:

- Private VariableName as Datatype
- Static VariableName as Datatype
- Public VariableName as Datatype

The above keywords indicate the scope of the declaration.

**Private** declares a local variable or a variable that is local to a procedure or module. However, Private is rarely used, we normally use Dim to declare a local variable.

**Static** keyword declares a variable that is being used multiple times, even after a procedure has been terminated. Most variables created inside a procedure are discarded by Visual Basic when the procedure is terminated. Static keyword preserves the value of a variable even after the procedure is terminated.

**Public** is the keyword that declares a global variable, which means it can be used by all the procedures and modules of the whole Visual Basic program.

## Object oriented programming creating and using classes and objects

In order for a programming language to qualify as an object oriented programming language, it must have three core technologies namely **encapsulation**, **inheritance** and **polymorphism**. These three terms are explained below:

### Encapsulation

- **Encapsulation** is a mechanism to wrap the data (variables) and code acting on the data (methods) together as a single unit.
- The data components of a class are called instance variables and one instance of a class is an object. For example, in a library system, a class could be a member, and John and Sharon could be two instances (two objects) of the library class.

### Inheritance

- **Inheritance** is a mechanism in which one object acquires all the properties and behaviors of a parent object.
- The idea behind **inheritance** is that you can create new classes that are built upon existing classes.
- Advantage: Less programming is required when adding functions to complex systems (reusability).

### Polymorphism

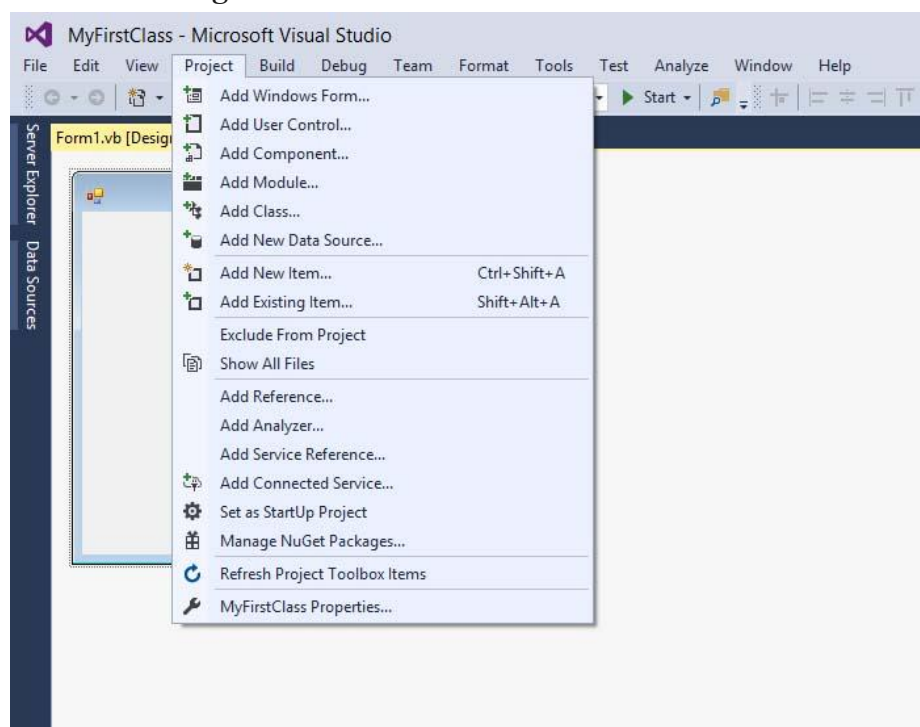
- Polymorphism is the ability of an object to take on many forms.
- Object-oriented programming allows procedures about objects to be created whose exact type is not known until runtime.
- For example, a screen cursor may change its shape from an arrow to a line depending on the program mode.

**Class:** A class consists of data members as well as methods.

## VB.Net window program using class and objects

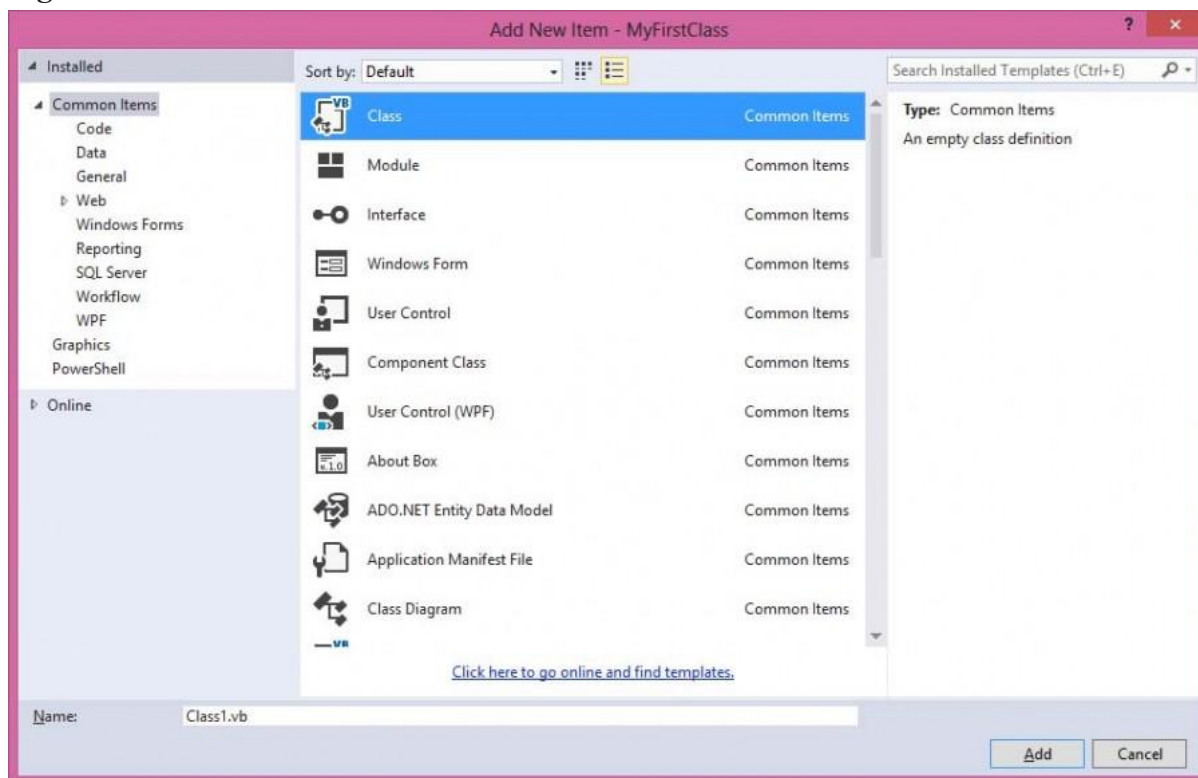
**The following Program shows you how to create a class that can calculate your BMI (Body Mass Index).**

To create a class, start Visual Basic 2017 as usual and choose Windows Applications. In the Visual Basic 2015 IDE, click on Project on the menu bar and select Add Class, as shown in Figure.





After clicking the Add Class item, the Add New Item dialog appears, as shown in Figure.



Click on the Class item and the default class Class1.vb will appear as a new tab in a code window. Rename the class as MyClass.vb. Rename the form as MyFirstClass.vb. Now, in the MyClass.vb window, create a new class MyClass1 and enter the following code

```
Public Class MyClass1
    Public Function BMI(ByVal height As Single, ByVal weight As Single)
        BMI = Format((weight) / (height ^ 2), "0.00")
    End Function
End Class
```

Now you have created a class (an object) called MyClass1 with a method known as BMI.

In order to use the BMI class, insert a button into the form and click on the button to enter the following code:

```
Private Sub BtnBMI_Click(sender As Object, e As EventArgs) Handles BtnBMI.Click
    Dim MyObjectAs Object
        Dim h, w As Single
        MyObject = New MyClass1()
        h = InputBox("What is your height in meter")
        w = InputBox("What is your weight in kg")
        MessageBox.Show(MyObject.BMI(h, w), "Your BMI")
    End Sub
```

When you run this program and click the button, the user will be presented with two input boxes to enter his or her height and weight subsequently and the value of BMI will be shown in a pop-up message box, as shown in the figures below:

MyFirstClass ✕

What is your height in meter

OK

Cancel

1.80

MyFirstClass ✕

What is your weight in kg

OK

Cancel

80

Your BMI ✕

24.69

OK

## Exception Handling

An exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

Exceptions provide a way to transfer control from one part of a program to another. VB.Net exception handling is built upon four keywords - **Try**, **Catch**, **Finally** and **Throw**.

- **Try** – A Try block identifies a block of code for which particular exceptions will be activated. It's followed by one or more Catch blocks.
- **Catch** – A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The Catch keyword indicates the catching of an exception.
- **Finally** – The Finally block is used to execute a given set of statements, whether an exception is thrown or not thrown. For example, if you open a file, it must be closed whether an exception is raised or not.
- **Throw** – A program throws an exception when a problem shows up. This is done using a Throw keyword.

### Syntax

```
Try
[ tryStatements ]
[ Exit Try ]
[ Catch [ exception [ As type ] ] [ When expression ]
[ catchStatements ]
[ Exit Try ] ]
[ Catch ... ]
[ Finally
[ finallyStatements ] ]
End Try
```

### Exception Classes in .Net Framework

- The exception classes in .Net Framework are mainly directly or indirectly derived from the **System.Exception** class.
- **System.ApplicationException** and **System.SystemException** classes are derived System.Exception class.

Exception Class	Description
System.IO.IOException	Handles I/O errors.
System.IndexOutOfRangeException	Handles errors generated when a method refers to an array index out of range.
System.ArrayTypeMismatchException	Handles errors generated when type is mismatched with the array type.

System.NullReferenceException	Handles errors generated from dereferencing a null object.
System.DivideByZeroException	Handles errors generated from dividing a dividend with zero.
System.InvalidCastException	Handles errors generated during typecasting.
System.OutOfMemoryException	Handles errors generated from insufficient free memory.
System.StackOverflowException	Handles errors generated from stack overflow.

Following is an example of throwing an exception when dividing by zero condition occurs

```
Module exceptionProg
Subdivision(ByVal num1 As Integer, ByVal num2 As Integer)
Dim result As Integer
Try
result = num1 \ num2
Catch e As DivideByZeroException
Console.WriteLine("Exception caught: {0}", e)
Finally
Console.WriteLine("Result: {0}", result)
EndTry
EndSub
Sub Main()
division(25, 0)
Console.ReadKey()
EndSub
EndModule
Output
```

```
Exception caught: System.DivideByZeroException: Attempted to divide by zero.
at ...
Result: 0
```

## Event Handling

Event: An event is an action that calls a function or may cause another event.

Event Handler: Event handlers are functions that tell how to respond to an event.

Clicking on a button, or entering some text in a text box, or clicking on a menu item, all are examples of events.

VB.Net is an event-driven language. There are mainly two types of events –

- Mouse events
- Keyboard events

## Handling Mouse Events

Mouse events occur with mouse movements in forms and controls.

Following are the various mouse events related with a Control class –

- **MouseDown** – it occurs when a mouse button is pressed
- **MouseEnter** – it occurs when the mouse pointer enters the control
- **MouseHover** – it occurs when the mouse pointer hovers over the control
- **MouseLeave** – it occurs when the mouse pointer leaves the control
- **MouseMove** – it occurs when the mouse pointer moves over the control
- **MouseUp** – it occurs when the mouse pointer is over the control and the mouse button is released
- **MouseWheel** – it occurs when the mouse wheel moves and the control has focus

The event handlers of the mouse events get an argument of type **MouseEventArgs**. The **MouseEventArgs** object is used for handling mouse events. It has the following properties –

- **Buttons** – indicates the mouse button pressed
- **Clicks** – indicates the number of clicks
- **Delta** – indicates the number of detents the mouse wheel rotated
- **X** – indicates the x-coordinate of mouse click
- **Y** – indicates the y-coordinate of mouse click

## Example

Following is an example, which shows how to handle mouse events. Take the following steps –

- Add three labels, three text boxes and a button control in the form.
- Change the text properties of the labels to - Customer ID, Name and Address, respectively.
- Change the name properties of the text boxes to txtID, txtName and txtAddress, respectively.
- Change the text property of the button to 'Submit'.
- Add the following code in the code editor window

```
PublicClassForm1
PrivateSubForm1_Load(sender AsObject, e AsEventArgs)Handles MyBase.Load
' Set the caption bar text of the form.
Me.Text = "tutorialspont.com"
End Sub

Private Sub txtID_MouseEnter(sender As Object, e As EventArgs)_
Handles txtID.MouseEnter
'codefor handling mouse enter on ID textbox
txtID.BackColor=Color.CornflowerBlue
txtID.ForeColor=Color.White
EndSub

PrivateSubtxtID_MouseLeave(sender AsObject, e AsEventArgs) _
HandlestxtID.MouseLeave
'code for handling mouse leave on ID textbox
txtID.BackColor = Color.White
txtID.ForeColor = Color.Blue
End Sub

Private Sub txtName_MouseEnter(sender As Object, e As EventArgs) _
Handles txtName.MouseEnter
'codefor handling mouse enter on Name textbox
txtName.BackColor=Color.CornflowerBlue
txtName.ForeColor=Color.White
EndSub

PrivateSubtxtName_MouseLeave(sender AsObject, e AsEventArgs) _
HandlestxtName.MouseLeave
'code for handling mouse leave on Name textbox
txtName.BackColor = Color.White
txtName.ForeColor = Color.Blue
End Sub

Private Sub txtAddress_MouseEnter(sender As Object, e As EventArgs) _
Handles txtAddress.MouseEnter
'codefor handling mouse enter on Address textbox
txtAddress.BackColor=Color.CornflowerBlue
txtAddress.ForeColor=Color.White
EndSub

PrivateSubtxtAddress_MouseLeave(sender AsObject, e AsEventArgs) _
```

```
HandletxtAddress.MouseLeave
```

```
'code for handling mouse leave on Address textbox
```

```
txtAddress.BackColor = Color.White
```

```
txtAddress.ForeColor = Color.Blue
```

```
End Sub
```

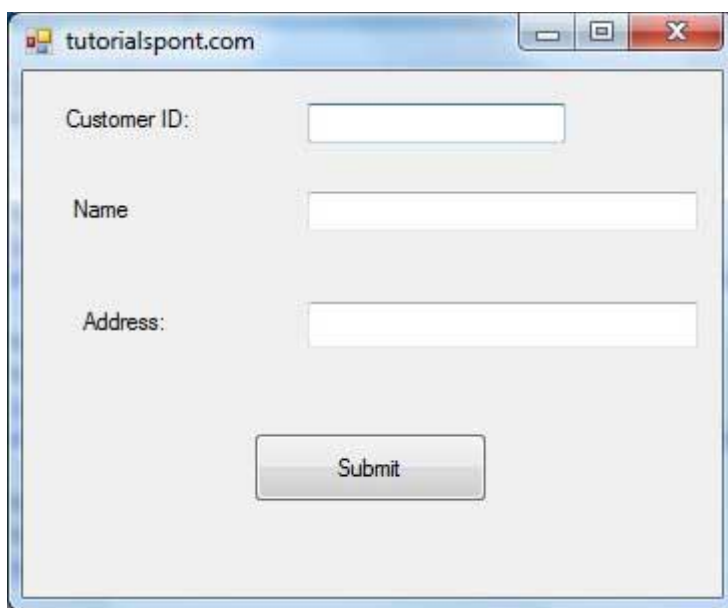
```
Private Sub Button1_Click(sender As Object, e As EventArgs) _
```

```
Handles Button1.Click
```

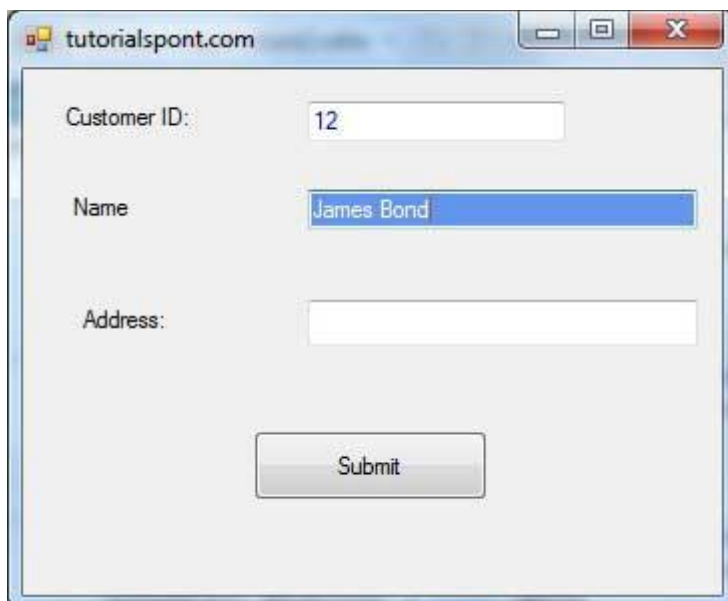
```
MsgBox("Thank you " & txtName.Text & ", for your kind cooperation")
```

```
End Sub
```

```
End Class
```



While entering the text in the text boxes and check the mouse events.



## Handling Keyboard Events

Following are the various keyboard events related with a Control class –

- **KeyDown** – occurs when a key is pressed down and the control has focus
- **KeyPress** – occurs when a key is pressed and the control has focus
- **KeyUp** – occurs when a key is released while the control has focus

The event handlers of the KeyDown and KeyUp events get an argument of type **EventArgs**. This object has the following properties –

- **Alt** – it indicates whether the ALT key is pressed
- **Control** – it indicates whether the CTRL key is pressed
- **Handled** – it indicates whether the event is handled
- **KeyCode** – stores the keyboard code for the event
- **KeyData** – stores the keyboard data for the event
- **KeyValue** – stores the keyboard value for the event
- **Modifiers** – it indicates which modifier keys (Ctrl, Shift, and/or Alt) are pressed
- **Shift** – it indicates if the Shift key is pressed

The event handlers of the KeyDown and KeyUp events get an argument of type **EventArgs**. This object has the following properties –

- **Handled** – indicates if the KeyPress event is handled
- **KeyChar** – stores the character corresponding to the key pressed

Example

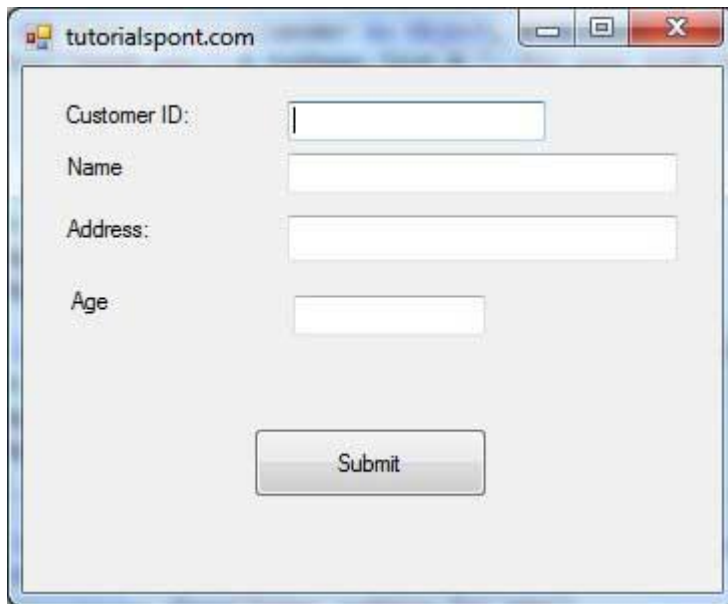
- Add a label with text Property as 'Age' and add a corresponding text box named txtAge.
- Add the following codes for handling the KeyUP events of the text box txtID.

```
PrivateSub txtID_KeyUP(sender As Object, e As EventArgs) _
    Handle txtID.KeyUp
    If (Not Char.IsNumber(ChrW(e.KeyCode))) Then
        MessageBox.Show("Enter numbers for your Customer ID")
        txtID.Text = " "
    EndIf
EndSub
```

- Add the following codes for handling the KeyUP events of the text box txtAge.

```
PrivateSub txtAge_KeyUP(sender As Object, e As EventArgs) _
    Handle txtAge.KeyUp
    If (Not Char.IsNumber(ChrW(e.keyCode))) Then
        MessageBox.Show("Enter numbers for age")
        txtAge.Text = " "
    EndIf
EndSub
```





A screenshot of a web browser window titled "tutorialspont.com". The page contains a form with four input fields: "Customer ID:", "Name", "Address:", and "Age:". Each field is currently empty. Below the fields is a "Submit" button.

If you leave the text for age or ID as blank or enter some non-numeric data, it gives a warning message box and clears the respective text –



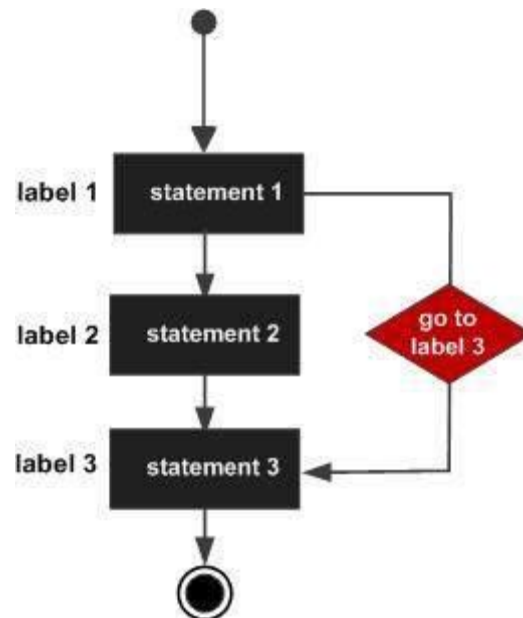
A screenshot of the same web browser window. The form fields now contain the following text: "Customer ID:" has "12", "Name" has "James Bond", "Address:" has "California, US", and "Age" has "1n". A warning message box is overlaid on the form, containing the text "Enter numbers for age" and an "OK" button. The "Submit" button is still visible on the form.

## GoTo Statement

The GoTo statement transfers control unconditionally to a specified line in a procedure.

Syntax

GoTo label



Program

Module loops

SubMain()

' local variable definition

Dim a As Integer = 10

Line1:

Do

If (a = 15) Then

' skip the iteration '

a = a + 1

GoTo Line1

End If

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

a = a + 1

Loop While (a < 20)

Console.ReadLine()

End Sub

End Module

Output

value of a: 10

value of a: 11

value of a: 12

value of a: 13

value of a: 14

value of a: 16

value of a: 17

value of a: 18

value of a: 19

While a value is 15 the printing of a is skipped by a GoTo statement.

**on Error Goto.****On Error Statement****Syntax**

On Error GoTo *line*

On Error Resume Next

On Error GoTo 0

On Error GoTo statements is an example of Vb.Net's Unstructured Exception Handling .

VB.NET has two types of Exception handling .

- Structured Error Handling and
- Unstructured Error handling .

VB.NET using Try..Catch statement for Structured Error handling and On Error GoTo statement is using for Unstructured Error handling.

Error GoTo redirect the flow of the program in a given location.

On Error Resume Next - whenever an error occurred in runtime, skip the statement and continue execution on following statements.

Take a look at the following program

**VB.NET Source Code**

```
Public Class Form1
```

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As _System.EventArgs) Handles Button1.Click
```

```
    Dim result As Integer
```

```
    Dim num As Integer
```

```
num = 100
```

```
result = num / 0
```

```
MsgBox("here")
```

```
End Sub
```

```
End Class
```

when u execute this program you will get error message like Arithmetic operation resulted in an overflow .

See the program we put an On Error GoTo statement.

```
Public Class Form1
```

```
Private Sub Button1_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Button1.Click
```

```
    On Error GoTo nextstep
```

```
    Dim result As Integer
```

```
    Dim num As Integer
```

```
num = 100
```

```
result = num / 0
nextstep:
MsgBox("Control Here")
End Sub
End Class
```

When you execute the program you will get the message box "Control Here" . Because the On Error statement redirect the exception to the Label statement.

**Sources:**

1. [https://www.tutorialspoint.com/vb.net/vb.net\\_exception\\_handling.htm](https://www.tutorialspoint.com/vb.net/vb.net_exception_handling.htm)
2. <https://dotnettutorials.net/course/asp-net-core-tutorials/#>

**Unit – III - Developing - ASP.NET Applications** - ASP.NET Applications – Understanding ASP.NET Controls - Overview of ASP.NET framework, Web Form fundamentals - Web control classes – Using Visual Studio.NET - Validation and Rich Controls - State management – Tracing, Logging, and Error Handling.

### **Developing ASP.NET Applications**

#### **What is ASP.NET?**

- ASP.NET is a web development platform.
- It is a complete software infrastructure and various services required to build up web applications for PC, as well as mobile devices.
- ASP.NET works on top of the HTTP protocol, and uses the HTTP commands and policies to set a browser-to-server bilateral communication and cooperation.

#### **Development**

- ASP.NET is a part of Microsoft .Net platform.
- ASP.NET applications are compiled codes. It is written using the extensible and reusable components or objects present in .Net framework.
- ASP.NET is used to produce interactive, data-driven web applications over the internet. It consists of a large number of controls such as text boxes, buttons, and labels for assembling, configuring, and manipulating code to create HTML pages.

An ASP.NET application consists of two major parts:

- The .aspx file: this is essentially the GUI that you see on the web page.
- The .cs file (code behind): this is essentially the code that executes the logic (calculations) associated with the GUI of the web page.

**The ASP.NET application codes can be written in any of the following languages:**

- C#
- Visual Basic.Net
- Jscript
- J#

## Understanding ASP.NET Controls

### What are controls?

- Controls are small building blocks of the graphical user interface, which include text boxes, buttons, check boxes, list boxes, labels, and numerous other tools.
- Using these tools, the users can enter data, make selections and indicate their preferences, etc.
- Controls are also used for structural jobs, like validation, data access, security, creating master pages, and data manipulation.
- An ASP.NET control is a .NET class that executes on the server and renders certain content to the browser.

For example, a Label control was used to display the current date and time. The ASP.NET framework includes more than 90 controls, which enable you to do everything from displaying a list of database records to displaying a randomly rotating banner advertisement.

### Overview of ASP.NET Controls

The ASP.NET Framework contains more than 90 controls. These controls can be divided into seven groups:

- **Standard Controls**—Enable you to render standard form elements such as buttons, input fields, and labels.
- **Validation Controls**—Enable you to validate form data before you submit the data to the server. For example, you can use a 'RequiredFieldValidator' control to check whether a user entered a value for a required input field.
- **Rich Controls**—Enable you to render things such as calendars, file upload buttons, rotating banner advertisements, and multistep wizards.
- **Data Controls**—Enable you to work with data such as database data. For example, you can use these controls to submit new records to a database table or display a list of database records.
- **Navigation Controls**—Enable you to display standard navigation elements such as menus, tree views, and bread crumb trails.
- **Login Controls**—Enables you to display login, change password, and registration forms.
- **HTML Controls**—Enable you to convert any HTML tag into a server-side control.

With the exception of the HTML controls, you declare and use all ASP.NET controls in a page in exactly the same way. For example, if you want to display a text input field in a page, you can declare a TextBox control like this:

```
<asp:TextBox id="TextBox1" runat="Server" />
```

- This control declaration looks like the declaration for an HTML tag. Remember, however, unlike an HTML tag, a control is a .NET class that executes on the server and not in the web browser.

When the TextBox control is rendered to the browser, it renders the following content:

```
<input name="TextBox1" type="text" id="TextBox1" />
```

- The first part of the control declaration, the `asp:` prefix, indicates the namespace for the control. All the standard ASP.NET controls are contained in the `System.Web.UI.WebControls` namespace. The prefix `asp:` represents this namespace.
- Next, the declaration contains the name of the control being declared. In this case, a `TextBox` control is declared.
- This declaration also includes an ID attribute. You use the ID to refer to the control in the page within your code.
- Every control must have a unique ID.

The declaration also includes a `runat="Server"` attribute. This attribute marks the tag as representing a server-side control. If you neglect to include this attribute, the `TextBox` tag would be passed to the browser.

### Understanding HTML Controls

HTML controls in a different way than you declare standard ASP.NET controls. The ASP.NET Framework enables you to take any HTML tag (real or imaginary) and add a `runat="server"` attribute to the tag. The `runat="server"` attribute converts the HTML tag into a server-side ASP.NET control.

### Understanding and Handling Control Events

The majority of ASP.NET controls support one or more events. For example, the ASP.NET Button control supports the Click event. The Click event is raised on the server after you click the button rendered by the Button control in the browser.

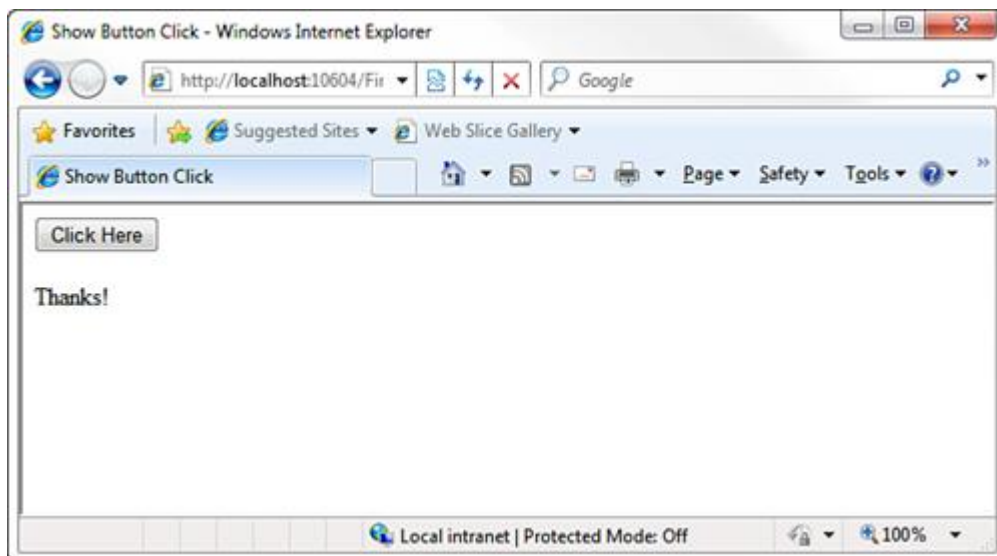
The following code illustrates how you can write code that executes when a user clicks the button rendered by the Button control (in other words, it illustrates how you can create a Click event handler):

#### Code. ShowButtonClick.aspx

```
<%@ Page Language="C#" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">
    protected void btnSubmit_Click(object sender, EventArgs e)
    {
```

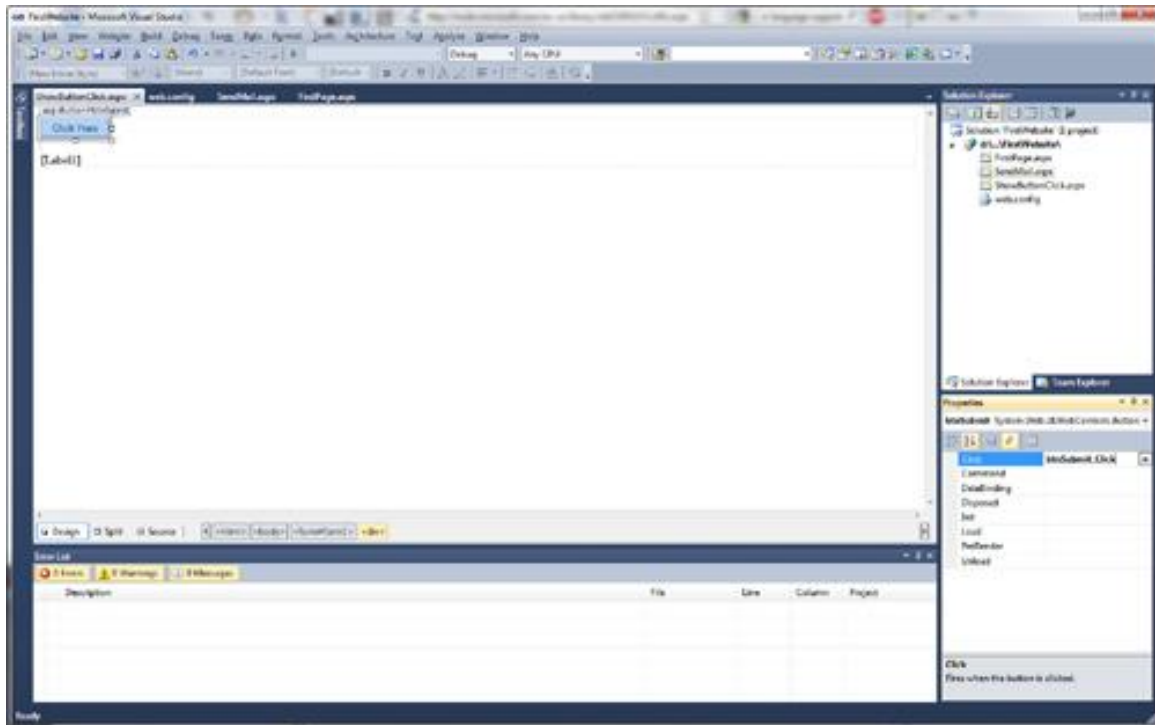


```
        Label1.Text = "Thanks!";
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>Show Button Click</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Button
                id="btnSubmit"
                Text="Click Here"
                OnClick="btnSubmit_Click"
                Runat="server" />
            <br /><br />
            <asp:Label
                id="Label1"
                Runat="server" />
        </div>
    </form>
</body>
</html>
```



- You can add an event handler automatically to a control in multiple ways when using Visual Web Developer.
- In Design view, you can double-click a control to add a handler for the control's default event.
- Double-clicking a control switches you to Source view and adds the event handler.

- Finally, from Design view, after selecting a control on the designer surface, you can add an event handler from the Properties window by clicking the Events button (the lightning bolt) and double-clicking next to the name of any of the events



**Adding an event handler from the Properties window.**

## ASP.NET validation controls

ASP.NET validation controls validate the user input data to ensure that useless, unauthenticated, or contradictory data don't get stored.

ASP.NET provides the following validation controls:

- RequiredFieldValidator
- RangeValidator
- CompareValidator
- RegularExpressionValidator
- CustomValidator
- ValidationSummary

### BaseValidator Class

- The validation control classes are inherited from the BaseValidator class.
- Therefore, it would help to take a look at the properties and the methods of this base class, which are common for all the validation controls.

Members	Description
ControlToValidate	Indicates the input control to validate.
Display	Indicates how the error message is shown.
EnableClientScript	Indicates whether client side validation will take.
Enabled	Enables or disables the validator.
ErrorMessage	Indicates error string.
Text	Error text to be shown if validation fails.
IsValid	Indicates whether the value of the control is valid.
SetFocusOnError	It indicates whether in case of an invalid control, the focus should switch to the related input control.
ValidationGroup	The logical group of multiple validators, where this control belongs.
Validate()	This method revalidates the control and updates the IsValid property.

### RequiredFieldValidator Control

The RequiredFieldValidator control ensures that the required field is not empty. It is generally tied to a text box to force input into the text box.

## Syntax

```
<asp:RequiredFieldValidator ID="rfvcandidate"
  runat="server" ControlToValidate="ddlcandidate"
  ErrorMessage="Please choose a candidate"
  InitialValue="Please choose a candidate">
</asp:RequiredFieldValidator>
```

## CompareValidator Control

The CompareValidator control compares a value in one control with a fixed value or a value in another control.

It has the following specific properties:

Properties	Description
Type	It specifies the data type.
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.

## Syntax

```
<asp:CompareValidator ID="CompareValidator1" runat="server"
  ErrorMessage="CompareValidator">
</asp:CompareValidator>
```

## RegularExpressionValidator

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

The following table summarizes the commonly used syntax constructs for regular expressions:

Character Escapes	Description
\b	Matches a backspace.
\t	Matches a tab.
\r	Matches a carriage return.
\v	Matches a vertical tab.

\f	Matches a form feed.
\n	Matches a new line.
\	Escape character.

Apart from single character match, a class of characters could be specified that can be matched, called the metacharacters.

Meta characters	Description
.	Matches any character except \n.
[abcd]	Matches any character in the set.
[^abcd]	Excludes any character in the set.
[2-7a-mA-M]	Matches any character specified in the range.
\w	Matches any alphanumeric character and underscore.
\W	Matches any non-word character.
\s	Matches whitespace characters like, space, tab, new line etc.
\S	Matches any non-whitespace character.
\d	Matches any decimal character.
\D	Matches any non-decimal character.

Quantifiers could be added to specify number of times a character could appear.

Quantifier	Description
*	Zero or more matches.
+	One or more matches.
?	Zero or one matches.
{N}	N matches.
{N,}	N or more matches.
{N,M}	Between N and M matches.

Syntax

```
<asp:RegularExpressionValidator ID="string" runat="server" ErrorMessage="string"
  ValidationExpression="string" ValidationGroup="string">
</asp:RegularExpressionValidator>
```

### CustomValidator

- The CustomValidator control allows writing application specific custom validation routines 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.

### Syntax

```
<asp:CustomValidator ID="CustomValidator1" runat="server"
  ClientValidationFunction=.cvf_func. ErrorMessage="CustomValidator">
</asp:CustomValidator>
```

### 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 mutually inclusive properties list out the error message:

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

### Syntax

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

### Validation Groups

- Complex pages have different groups of information provided in different panels. In such situation, a need might arise for performing validation separately for separate group. This kind of situation is handled using validation groups.
- To create a validation group, you should put the input controls and the validation controls into the same logical group by setting their *ValidationGroup* property.

## Example Program for Validation Control

The following example describes a form to be filled up by all the students of a school, divided into four houses, for electing the school president. Here, we use the validation controls to validate the user input.

This is the form in design view:

The content file code is as given:

```
<form id="form1" runat="server">
  <table style="width: 66%;">
    <tr>
      <td class="style1" colspan="3" align="center">
        <asp:Label ID="lblmsg"
          Text="President Election Form : Choose your president"
          runat="server" />
      </td>
    </tr>
    <tr>
      <td class="style3">
        Candidate:
      </td>
      <td class="style2">
        <asp:DropDownList ID="ddlcandidate" runat="server" style="width:239px">
          <asp:ListItem>Please Choose a Candidate</asp:ListItem>
          <asp:ListItem>M H Kabir</asp:ListItem>
          <asp:ListItem>Steve Taylor</asp:ListItem>
          <asp:ListItem>John Abraham</asp:ListItem>
          <asp:ListItem>Venus Williams</asp:ListItem>
        </asp:DropDownList>
      </td>
      <td>
        <asp:RequiredFieldValidator ID="rfvcandidate">

```

```

        runat="server" ControlToValidate ="ddlcandidate"
        ErrorMessage="Please choose a candidate"
        InitialValue="Please choose a candidate">
    </asp:RequiredFieldValidator>
</td>
</tr>
<tr>
<td class="style3">
    House:
</td>
<td class="style2">
    <asp:RadioButtonList ID="rblhouse" runat="server" RepeatLayout="Flow">
        <asp:ListItem>Red</asp:ListItem>
        <asp:ListItem>Blue</asp:ListItem>
        <asp:ListItem>Yellow</asp:ListItem>
        <asp:ListItem>Green</asp:ListItem>
    </asp:RadioButtonList>
</td>
<td>
    <asp:RequiredFieldValidator ID="rfvhouse" runat="server"
        ControlToValidate="rblhouse" ErrorMessage="Enter your house name" >
    </asp:RequiredFieldValidator>
    <br />
</td>
</tr>
<tr>
<td class="style3">
    Class:
</td>
<td class="style2">
    <asp:TextBox ID="txtclass" runat="server"></asp:TextBox>
</td>
<td>
    <asp:RangeValidator ID="rvclass"
        runat="server" ControlToValidate="txtclass"
        ErrorMessage="Enter your class (6 - 12)" MaximumValue="12"
        MinimumValue="6" Type="Integer">
    </asp:RangeValidator>
</td>
</tr>
<tr>
<td class="style3">
    Email:

```



```

</td>
<td class="style2">
  <asp:TextBox ID="txtemail" runat="server" style="width:250px">
  </asp:TextBox>
</td>
<td>
  <asp:RegularExpressionValidator ID="remail" runat="server"
    ControlToValidate="txtemail" ErrorMessage="Enter your email"
    ValidationExpression="\w+([-+.']\w+)*@\w+([-.]w+)*\.\w+([-.]w+)*">
  </asp:RegularExpressionValidator>
</td>
</tr>
<tr>
  <td class="style3" align="center" colspan="3">
    <asp:Button ID="btnsubmit" runat="server" onclick="btnsubmit_Click"
      style="text-align: center" Text="Submit" style="width:140px" />
  </td>
</tr>
</table>
<asp:ValidationSummary ID="ValidationSummary1" runat="server"
  DisplayMode ="BulletList" ShowSummary ="true" HeaderText="Errors:" />
</form>

```

*The code behind the submit button*

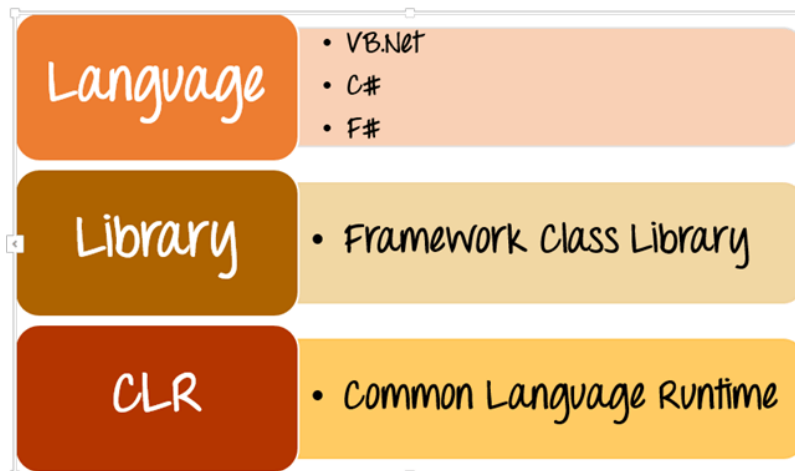
```

protected void btnsubmit_Click(object sender, EventArgs e)
{
  if (Page.IsValid)
  {
    lblmsg.Text = "Thank You";
  }
  else
  {
    lblmsg.Text = "Fill up all the fields";
  }
}

```

## ASP.NET Architecture and its Components

ASP.Net is a framework which is used to develop a Web-based application.



**ASP.NET Architecture Diagram**

The architecture of the .Net framework is based on the following key components

- **Language** – A variety of languages exists for .net framework. They are VB.net and C#. These can be used to develop web applications.
- **Library** - The .NET Framework includes a set of standard class libraries. The most common library used for web applications in .net is the Web library. The web library has all the necessary components used to develop .Net web-based applications.
- **Common Language Runtime** - The Common Language Infrastructure or CLI is a platform. .Net programs are executed on this platform. The CLR is used for performing key activities. Activities include Exception handling and Garbage collection.

Below are some of the key characteristics of the ASP.Net framework

- **Code Behind Mode** – This is the concept of **separation of design and code**. By making this separation, it becomes easier to maintain the ASP.Net application. The general file type of an ASP.Net file is **aspx**. Assume we have a web page called MyPage.aspx. There will be another file called **MyPage.aspx.cs** which would denote the code part of the page. So Visual Studio creates separate files for each web page, one for the design part and the other for the code.
- **State Management** – ASP.Net has the facility to control state management. HTTP is known as a stateless protocol. Let's take an example of a shopping cart application. Now, when a user decides what he wants to buy from the site, he will press the submit button.
- The application needs to remember the items the **user choose for the purchase**. This is known as remembering the state of an application at a current point in time. HTTP

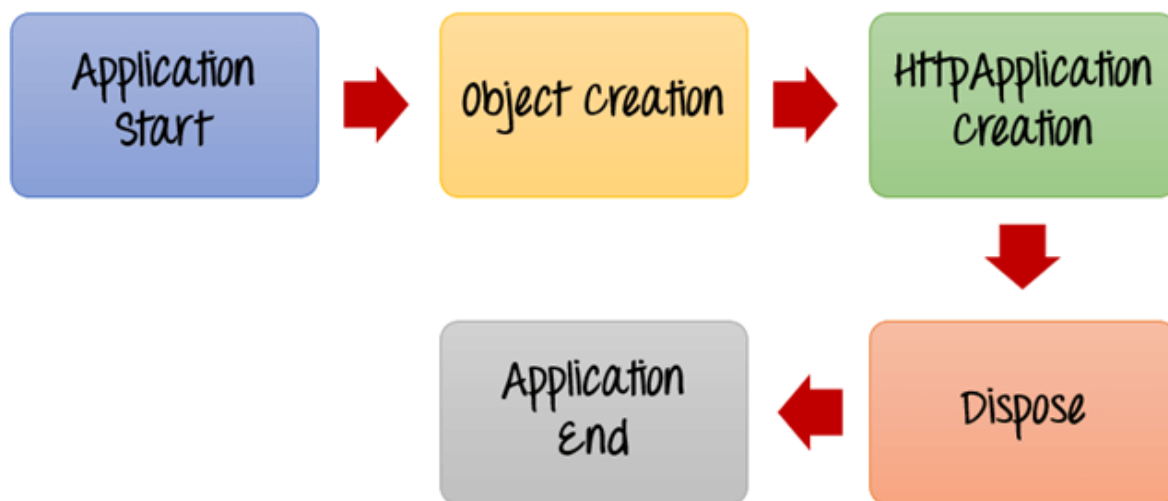
is a stateless protocol. When the user goes to the purchase page, HTTP will not store the information on the cart items. **Additional coding needs to be done** to ensure that the cart items can be carried forward to the purchase page. Such an implementation can become complex at times. But ASP.Net can do **state management** on your behalf. So ASP.Net can remember the cart items and pass it over to the purchase page.

- **Caching** – ASP.Net can implement the concept of Caching. This improve's the performance of the application. By caching those **pages which are often requested by the user can be stored in a temporary location**. These pages can be retrieved faster and better responses can be sent to the user. So caching can significantly improve the performance of an application.

ASP.Net is a development language used for constructing web-based applications. ASP.Net is designed to work with the standard HTTP protocol.

### What is ASP.Net Lifecycle?

- When an ASP.Net application is launched, there are series of steps which are carried out. These series of steps make up the lifecycle of the application.
- Let's look at the various stages of a typical page lifecycle of an ASP.Net Web Application.



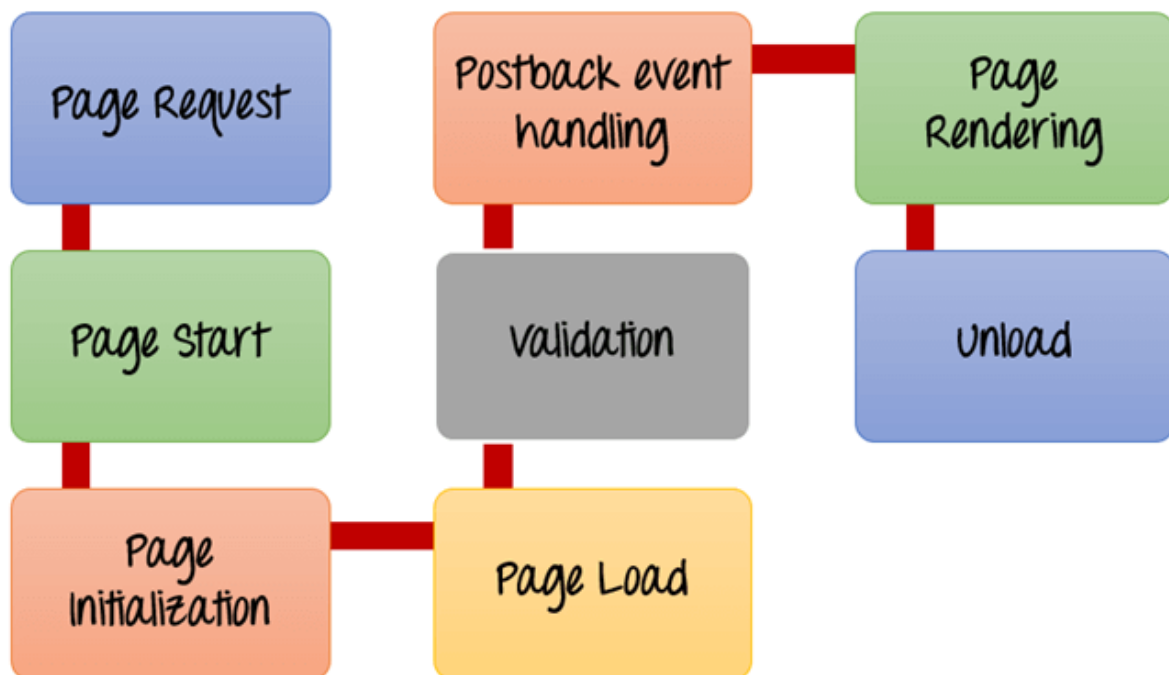
- **Application Start** - The life cycle of an ASP.NET application starts when a request is made by a user. This request is to the Web server for the ASP.Net Application. This happens when the first user normally goes to the home page for the application for the first time. During this time, there is a method called Application\_start which is executed by the web server. Usually, in this method, all global variables are set to their default values.
- **Object creation** - The next stage is the creation of the HttpContext, HttpRequest & HttpResponse by the web server. The HttpContext is just the container for the HttpRequest and HttpResponse objects. The HttpRequest object contains information

about the current request, including cookies and browser information. The `HttpResponse` object contains the response that is sent to the client.

- **HttpApplication creation** - This object is created by the web server. It is this object that is used to process each subsequent request sent to the application. For example, let's assume we have 2 web applications. One is a shopping cart application, and the other is a news website. For each application, we would have 2 `HttpApplication` objects created. Any further requests to each website would be processed by each `HttpApplication` respectively.
- **Dispose** - This event is called before the application instance is destroyed. During this time, one can use this method to manually release any unmanaged resources.
- **Application End** - This is the final part of the application. In this part, the application is finally unloaded from memory.

### What is ASP.Net Page Lifecycle?

When an ASP.Net page is called, it goes through a particular lifecycle. This is done before the response is sent to the user. There are series of steps which are followed for the processing of an ASP.Net page.



### ASP.Net Page Lifecycle

1. **Page Request**- This is when the page is first requested from the server. When the page is requested, the server checks if it is requested for the first time. If so, then it needs to compile the page, parse the response and send it across to the user. If it is not

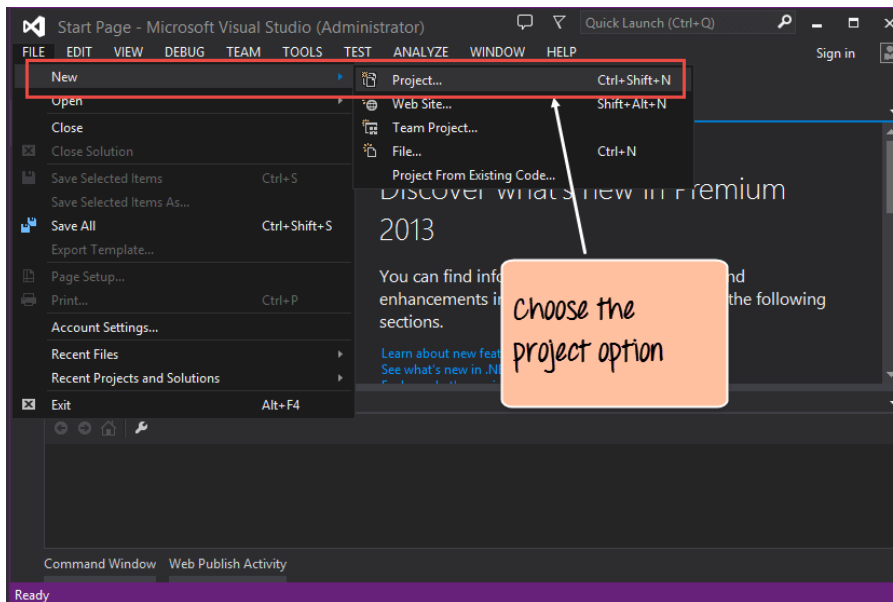
the first time the page is requested, the cache is checked to see if the page output exists. If so, that response is sent to the user.

2. **Page Start** – During this time, 2 objects, known as the Request and Response object are created. The Request object is used to hold all the information which was sent when the page was requested. The Response object is used to hold the information which is sent back to the user.
3. **Page Initialization** – During this time, all the controls on a web page is initialized. So if you have any label, textbox or any other controls on the web form, they are all initialized.
4. **Page Load** – This is when the page is actually loaded with all the default values. So if a textbox is supposed to have a default value, that value is loaded during the page load time.
5. **Validation** – Sometimes there can be some validation set on the form. For example, there can be a validation which says that a list box should have a certain set of values. If the condition is false, then there should be an error in loading the page.
6. **PostBack event handling** – This event is triggered if the same page is being loaded again. This happens in response to an earlier event. Sometimes there can be a situation that a user clicks on a submit button on the page. In this case, the same page is displayed again. In such a case, the Postback event handler is called.
7. **Page Rendering** – This happens just before all the response information is sent to the user. All the information on the form is saved, and the result is sent to the user as a complete web page.
8. **Unload** – Once the page output is sent to the user, there is no need to keep the ASP.net web form objects in memory. So the unloading process involves removing all unwanted objects from memory.

## ASP.NET First Program Example to display your information

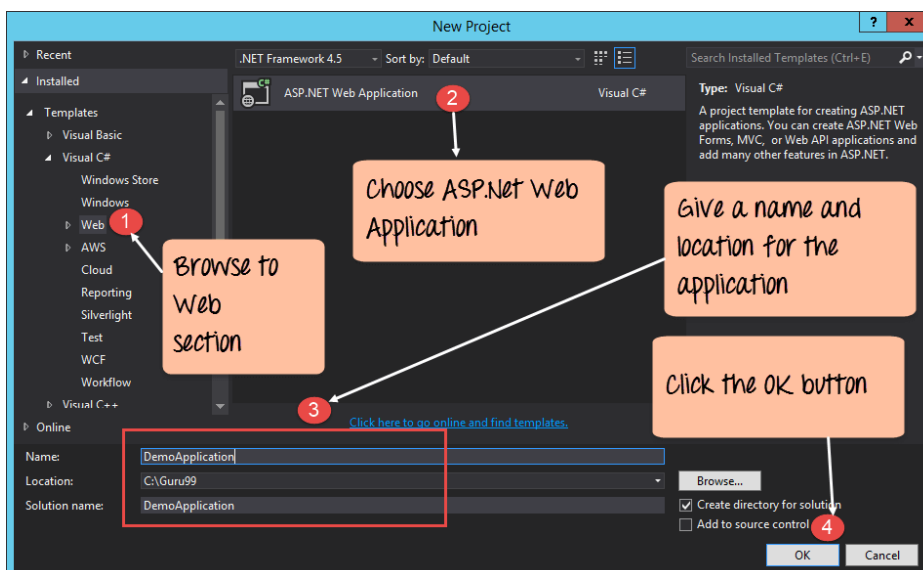
(The students may watch the video lesson provided by me)

Step 1) The first step involves the creation of a new project in Visual Studio. After launching Visual Studio, you need to choose the menu option New->Project.



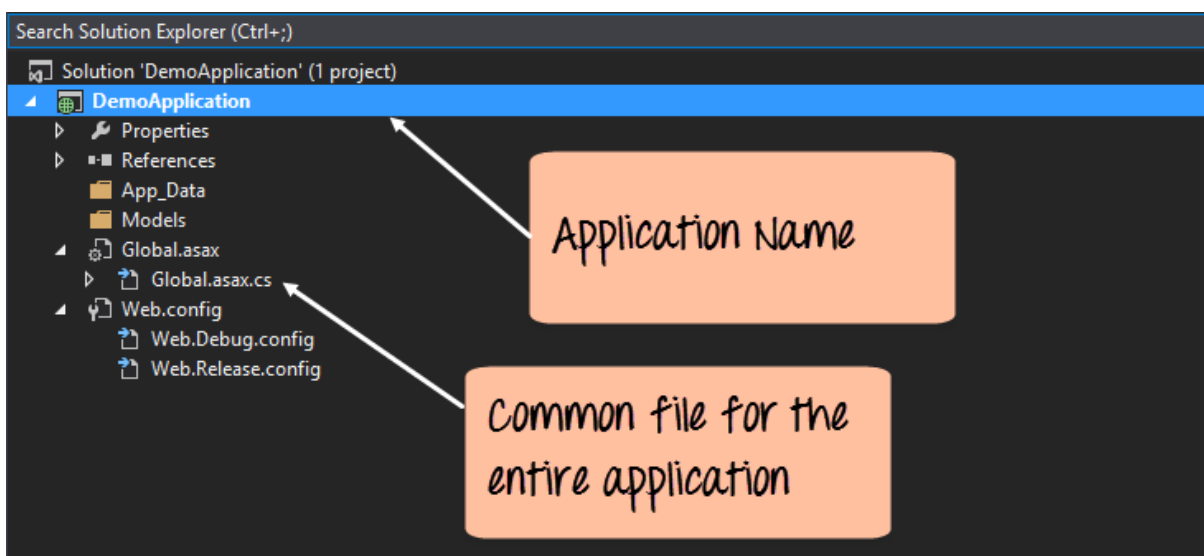
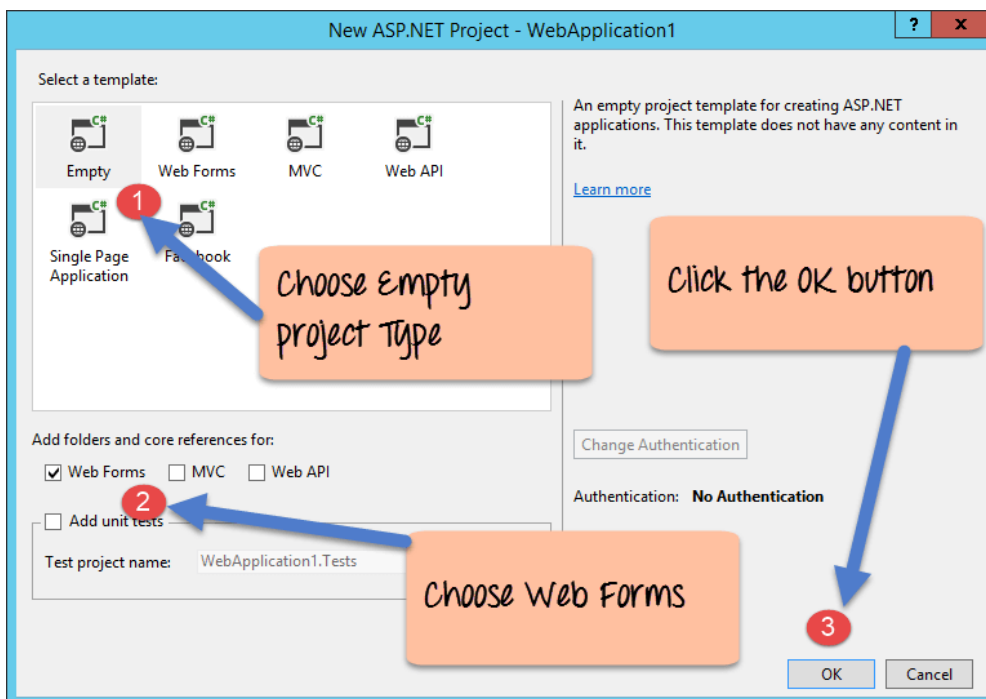
Step 2) The next step is to choose the project type as an ASP.Net Web application. Here we also need to mention the name and location of our project.

- In the project dialog box, you can see various options for creating different types of projects. Click the Web option on the left-hand side.
- When we click the Web option in the previous step, we will be able to see an option for ASP.Net Web Application. Click this option.
- We then give a name for the application, which in our case is DemoApplication. We also need to provide a location to store our application.
- Finally, we click the 'OK' button to let Visual Studio to create our project.



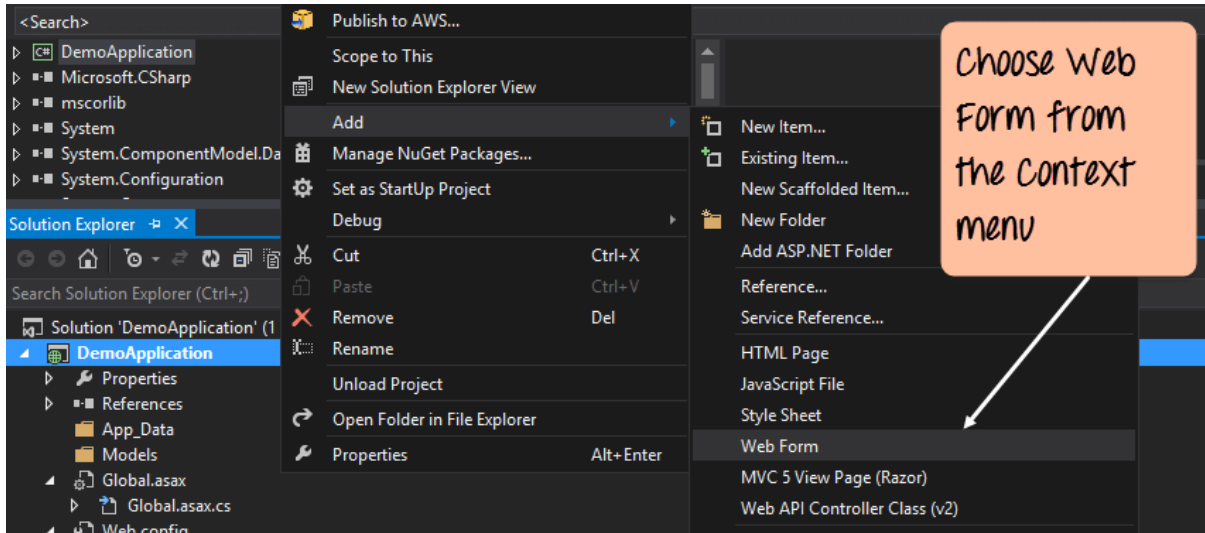
Step 3) In the next screen, you have to choose the type of ASP.net web application that needs to be created. In our case, we are going to create a simple Web Form application.

- First, choose the project type as 'Empty'. This will ensure that we start with a basic application which is simple to understand.
- We choose the option "web Forms". This adds the basic folders. These are required for a basic Web Forms Application.
- Finally, we click the 'OK' button to allow Visual Studio to create our application.
- In the Solution Explorer, you will be able to see the DemoApplication Solution. This solution will contain 2 project files as shown above. At the moment, one of the key files in the project is the 'Global.asax.cs'. This file contains application specific information. In this file, you would initialize all application specific variables to their default values.



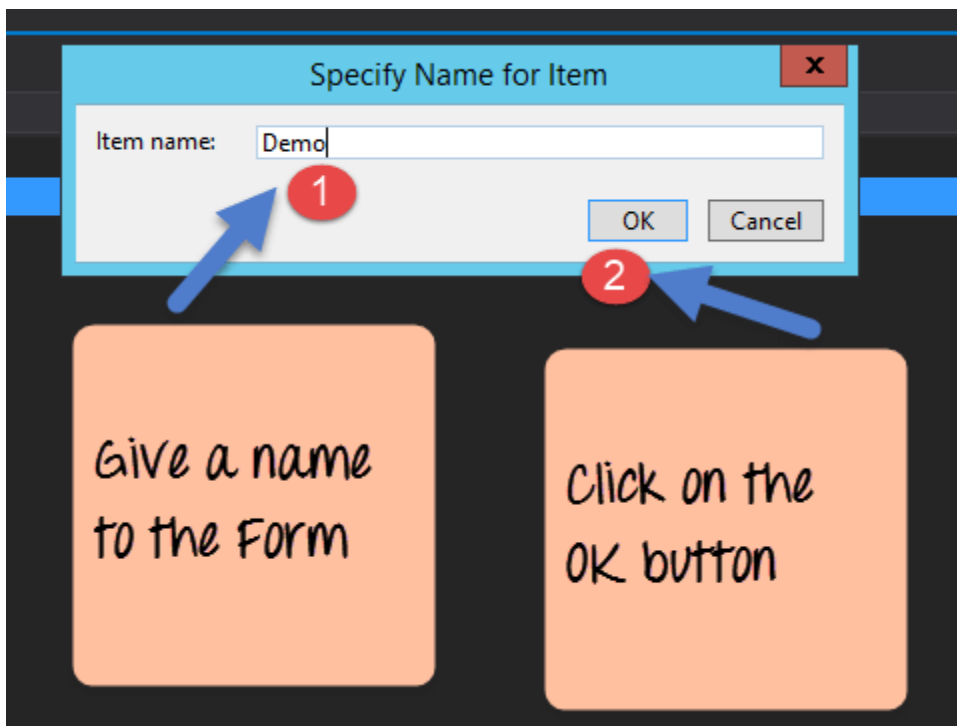
Step 4) Now, it's time to add a Web Form file to the project. This is the file which will contain all the web-specific code for our project.

- Right-click on the DemoApplication project and
- Choose Add->Web Form from the context menu.



Step 5) In the next screen we are going to be prompted to provide a name for the web form.

- Give a name for the Web Form. In our case, we are giving it a name of Demo.
- Click the Ok button.



Automatically Visual Studio will create the Demo Web Form and will open it in Visual Studio.



Step 6) The next step is to add the code, which will do the work of displaying your personal details This can be done by just adding one line of code to the Demo.aspx file.

```
<html xmlns="www.w3.org/1999/xhtml">
<head runat="server">
    <title>Personal Details</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <%Response. Write( "Name: Ravichandran"); %>
            <%Response. Write( "Class: MCA"); %>
            <%Response. Write( "Year: Third Year"); %>
            <%Response. Write( "Roll No: 1234999"); %>
        </div>
    </form>
</body>
</html>
```

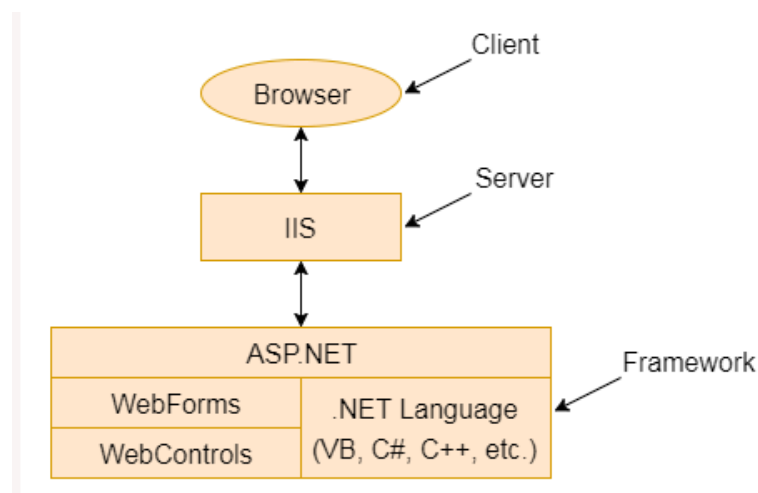
### **Output**

The Response object in ASP.Net is used to send information back to the user. So in our case, we are using the method "Write" of the Response object to write the text. The <% and %> markers are used to add ASP.net specific code.

If you follow all of the above steps and run your program in Visual Studio, you will get the following output.

## Web Form fundamentals

- Web Forms are web pages built on the ASP.NET Technology.
- It executes on the server and generates output to the browser. It is compatible to any browser to any language supported by .NET common language runtime.
- It is flexible and allows us to create and add custom controls.
- We can use Visual Studio **to create ASP.NET Web Forms**. It is an IDE (Integrated Development Environment) that allows us to drag and drop server controls to the web forms. It also allows us to set properties, events and methods for the controls.
- **To write business logic**, we can choose any .NET language like: **Visual Basic or Visual C#**.
- Web Forms are made up of two components: the **visual portion (the ASPX file)**, and the **code behind the form, which resides in a separate class file**.



### The components of the ASP.NET

The main purpose of Web Forms is to overcome the limitations of ASP and separate view from the application logic.

## Web Forms Features

ASP.NET is full of features and provides an awesome platform to create and develop web application. Here, we are discussing these features of Web Forms.

- Server Controls
- Master Pages
- Working with data
- Membership
- Client Script and Client Frameworks
- Routing
- State Management
- Security
- Performance
- Error Handling

## **Server Controls**

Web Forms provides rich set of server controls. These controls are objects that run when the page is requested and render markup to the browser. Some Web server controls are similar to familiar HTML elements, such as buttons and text boxes. It also provides controls that we can use to connect to data sources and display data.

## **Master Pages**

It allows us to create a consistent layout for the pages in our application. This page defines the look and feel and standard behavior that we want for all of the pages in our application. When users request the content pages, they merge with the master page to produce output that combines the layout of the master page with the content from the content page.

## **Working with Data**

In an ASP.NET Web Forms application, we use data-bound controls to automate the presentation or input of data in web page UI elements such as tables and text boxes and drop-down lists.

## **Membership**

Project's Account folder contains the files that implement the various parts of membership: registering, logging in, changing a password, and authorizing access. Additionally, ASP.NET Web Forms supports OAuth and OpenID. These authentication enhancements allow users to log into your site using existing credentials, from such accounts as Facebook, Twitter and Google.

## **Client Script and Client Frameworks**

We can enhance the server-based features of ASP.NET by including client-script functionality in ASP.NET Web Form pages. We can use client script to provide a richer, more responsive user interface to the users. We can also use client script to make asynchronous calls to the Web server while a page is running in the browser.

## **Routing**

We can configure URL routing of our application. A request URL is simply the URL a user enters into their browser to find a page on our web site. We use routing to define URLs that are semantically meaningful to users and that can help with search-engine optimization (SEO).

## **State Management**

ASP.NET Web Forms includes several options that help you preserve data on both a per-page basis and an application-wide basis.

## **Security**

Developing a secure application is most important aspect of software development process. ASP.NET Web Forms allow us to add extensibility points and configuration options that enable us to customize various security behaviors in the application.

## Performance

Web Forms provides good performance and allows us to modify performance related to page and server control processing, state management, data access, application configuration and loading, and efficient coding practices.

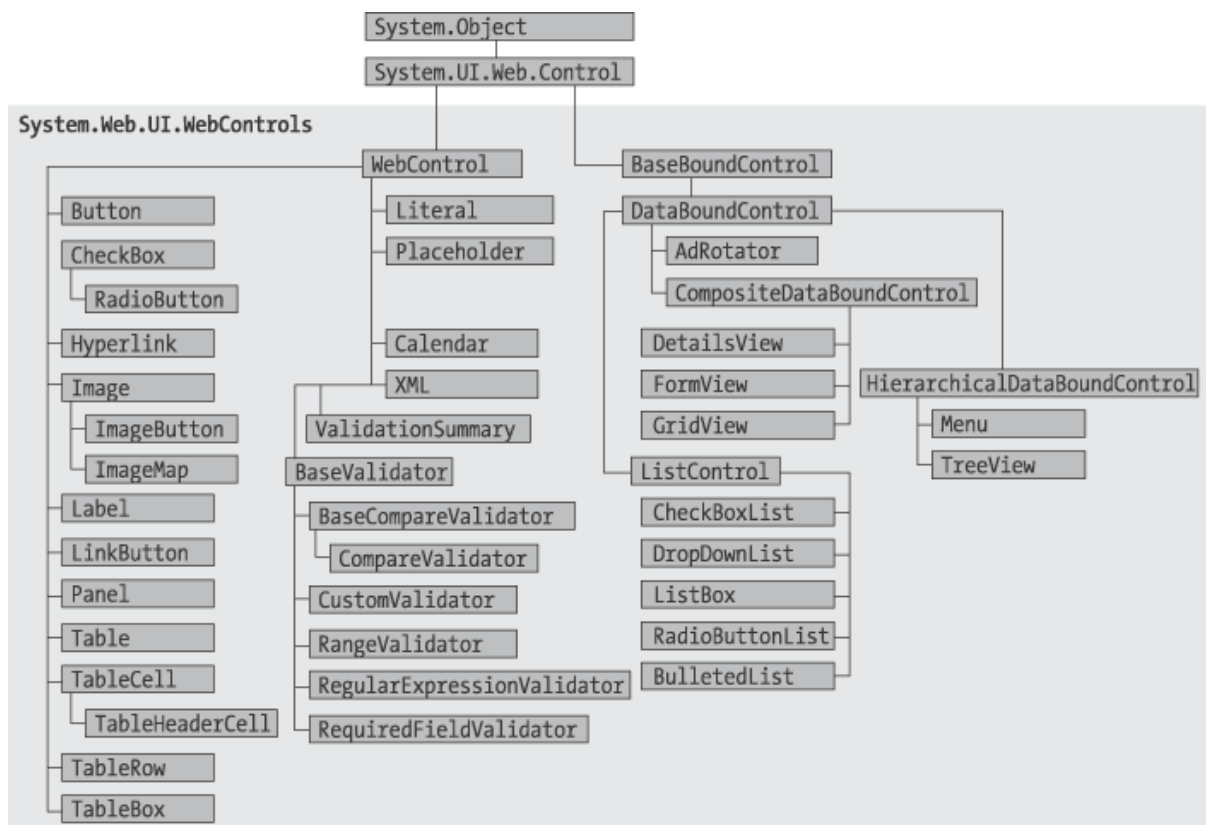
## Debugging and Error Handling

We can diagnose problems that occur in our Web Forms application. Debugging and error handling are well supported within ASP.NET Web Forms so that our applications compile and run effectively.

**ASP.NET provides various controls like:** server controls and HTML controls for the Web Forms. We have tables all these controls below.

## Web control classes

Web control classes are defined in the **System.Web.UI.WebControls** namespace. They follow a slightly more tangled object hierarchy than HTML server controls, as shown in Figure.



**The web control hierarchy**

Server Controls are the tags that are understood by the server. There are basically three types of server controls.

- **HTML Server Controls** - Traditional HTML tags
- **Web Server Controls** - New ASP. NET tags
- **Validation Server Controls** - For input validation

### ASP.NET HTML Server Controls

ASP.NET provides a way to work with HTML Server controls on the server side; programming with a set of controls collectively is called HTML Controls.

- These controls are grouped together in the Visual Studio Toolbox in the the HTML Control tab. The markup of the controls are similar to the HTML control.
- These controls are basically the original HTML controls but enhanced to enable server side processing.
- HTML elements in ASP. NET files are, by default, treated as text. To make these elements programmable, add a `runat="server"` attribute to the HTML element. This attribute indicates that the element should be treated as a server control.

### ASP.NET 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.
- However, Web server controls do not necessarily map to any existing HTML elements and they may represent more complex elements.
- Mostly all Web Server controls inherit from a common base class, namely the **WebControl** class defined in the **System.Web.UI.WebControls** namespace.

### ASP.NET Validation Server Controls

- After you create a web form, you should make sure that mandatory fields of the form elements such as login name and password are not left blank; data inserted is correct and is within the specified range. Validation is the method of scrutinizing (observing) that the user has entered the correct values in input fields.
- A Validation server control is used to validate the data of an input control. If the data does not pass validation, it will display an error message to the user.
- In ASP. NET you can use ASP. NET Validation Controls while creating the form and specify what ASP. NET Validation Controls you want to use and to which server control you want bind this.
- Validation Controls are derived from a common base class and share a common set of properties and methods. You just have to drag and drop the ASP. NET Validation Control in the web form and write one line of code to describe its functionality.

- This reduces the developer time from writing JavaScript for each type of validation. Moreover, through ASP. NET Validation Controls if any invalid data is entered the browser itself detects the error on the client side and displays the error without requesting the server. This is another advantage because it reduces the server load.

### Rich Controls

ASP.NET provides large set of controls. These controls are divided into different categories, depends upon their functionalities. The followings control comes under the rich controls category.

- FileUpload control
- Calendar control
- AdRotator control
- MultiView control
- Wizard control

#### *FileUpload control*

FileUpload control is used to browse and upload files. After the file is uploaded, you can store the file on any drive or database. FileUpload control is the combination of a browse button and a text box for entering the filename.

**The FileUpload control supports the following important properties.**

- **FileBytes:** It returns the contents of uploaded file as a byte array
- **FileContent:** You can get the uploaded file contents as a stream.
- **FileName:** Provides the name of uploaded file.
- **HasFile:** It is a Boolean property that checks whether particular file is available or not.
- **PostedFile:** Gets the uploaded file wrapped in the HttpPostedFile object.

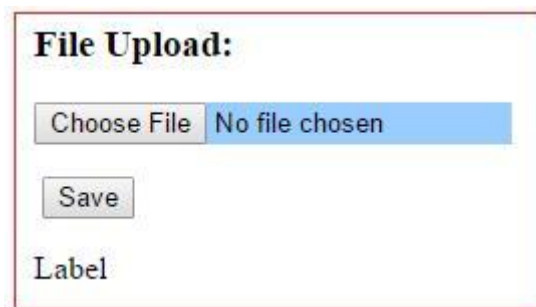
#### **Example**

```
using System;
using System.Text;
public partial class RichControl : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    protected void btnSave_Click(object sender, EventArgs e)
    {
        StringBuilder sb = new StringBuilder();
        if (FileUpload1.HasFile)
        {
            try
            {
                sb.AppendFormat(" Uploaded file: {0}", FileUpload1.FileName);
                //save the file
                FileUpload1.SaveAs(@"C:\\" + FileUpload1.FileName);
                //Showing the file information
            }
            catch { }
        }
    }
}
```

```

sb.Append("<br/> File Name: {0}" + FileUpload1.PostedFile.FileName);
sb.Append("<br/> File type: {0}" + FileUpload1.PostedFile.ContentType);
sb.Append("<br/> File length: {0}" + FileUpload1.FileBytes.Length);
Label1.Text = sb.ToString();
}
catch (Exception ex)
{
    sb.Append("<br/> Error <br/>");
    sb.Append(ex.Message);
    Label1.Text = sb.ToString();
}
}
else
{
    Label1.Text = sb.ToString();
}
}
}
}

```



### ***Calendar control***

Calendar control provides you lots of property and events. By using these properties and events you can perform the following task with calendar control.

- Select date.
- Selecting a day, a week or a month.
- Customize the calendar's appearance.

### **The Calendar control supports three important events:**

Event	Description
SelectionChanged	This event is fired when you select a day, a week or an entire month.
DayRender	This event is fired when each data cell of the calendar control is rendered.
VisibleMonthChanged	It is raised when user changes a month.

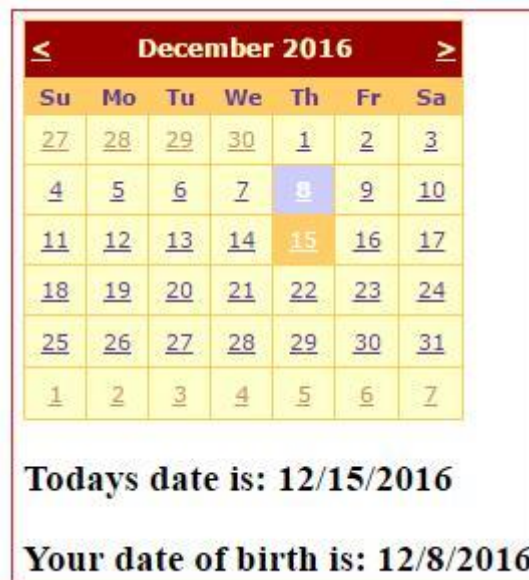
Calendar control supports SelectionMode property that allows you to select a single day, week, or entire month.

### Example

```
using System;
using System.Text;
public partial class RichControl : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    protected void Calendar1_SelectionChanged(object sender, EventArgs e)
    {
        Label1.Text ="Todays date is: " + Calendar1.TodaysDate.ToShortDateString();
        Label2.Text = "Your date of birth is: " + Calendar1.SelectedDate.ToShortDateString();
    }
}
```

When you select a date, SelectionChanged event will fired and displays the date in a label controls.

In this example the date format is MM/DD/YYYY.



### AdRotator control

- AdRotator control is used to display different advertisements randomly in a page.
- The list of advertisements is stored in either an XML file or in a database table.
- Lots of websites uses AdRotator control to display the advertisements on the web page.



To create an advertisement list, first add an XML file to your project.

### Code for XML file

```
<?xml version="1.0" encoding="utf-8" ?>

<Advertisements>
  <Ad>
    <ImageUrl>~/Images/logo1.png</ImageUrl>
    <NavigateUrl>http://www.TutorialRide.com</NavigateUrl>
    <AlternateText>Advertisement</AlternateText>
    <Impressions>100</Impressions>
    <Keyword>banner</Keyword>
  </Ad>
  <Ad>
    <ImageUrl>~/Images/logo2.png</ImageUrl>
    <NavigateUrl>http://www.TutorialRide.com</NavigateUrl>
    <AlternateText>Advertisement</AlternateText>
    <Impressions>100</Impressions>
    <Keyword>banner</Keyword>
  </Ad>
  <Ad>
    <ImageUrl>~/Images/logo3.png</ImageUrl>
    <NavigateUrl>http://www.CareerRide.com</NavigateUrl>
    <AlternateText>Advertisement</AlternateText>
    <Impressions>100</Impressions>
    <Keyword>banner</Keyword>
  </Ad>
  <Ad>
    <ImageUrl>~/Images/logo4.png</ImageUrl>
    <NavigateUrl>http://www.TutorialRide.com</NavigateUrl>
    <AlternateText>Advertisement</AlternateText>
    <Impressions>50</Impressions>
    <Keyword>banner</Keyword>
  </Ad>
</Advertisements>
```

In the given XML file 'Images' is the name of the folder, where we stored all the images to display. Now set the AdRotator control's AdvertisementFile property. Set the path of the XML file that you created above to AdRotator control's AdvertisementFile property.

### Important properties of AdRotator control.

- **ImageUrl:** The URL of the image that will be displayed through AdRotator control.
- **NavigateUrl:** If the user clicks the banner or ad then the new page is opened according to given URL.
- **AlternateText:** It is used for displaying text instead of the picture if picture is not displayed. It is also used as a tooltip.
- **Impressions:** It is a number that sets how frequently an advertisement will appear.
- **Keyword:** It is used to filter ads or identifies a group of advertisement.

### **MultiView control**

- MultiView control can be used when you want to create a tabbed page.
- In many situations, a web form may be very long, and then you can divide a long form into multiple sub forms. MultiView control is made up of multiple view controls. You can put multiple ASP.NET controls inside view controls. One View control is displayed at a time and it is called as the active view. View control does not work separately. It is always used with a Multiview control.
- If working with Visual Studio 2010 or later, you can drag and drop a MultiView control onto the form. You can drag and drop any number of View controls inside the MultiView control. The number of view controls is depends upon the need of your application.

### **The MultiView control supports the following important properties**

- **ActiveViewIndex:** It is used to determine which view will be active or visible.
- **Views:** It provides the collection of View controls contained in the MultiView control.
- For understand the Multiview control, first we will create a user interface as given below.

In the given example, in Multiview control, we have taken three separate View control.

1. In First step we will design to capture Product details.
2. In Second step we will design to capture Order details.
3. Next we will show summary for confirmation.

MultiView1	
<div style="border: 1px solid gray; padding: 5px;"> <p><b>Step 1 - Product Details</b></p> <p>Product ID <input type="text"/></p> <p>Product Name <input type="text"/></p> <p>Price/Unit <input type="text"/></p> <p style="text-align: right;"><input type="button" value="Next &gt;&gt;"/></p> </div>	View1
<div style="border: 1px solid gray; padding: 5px;"> <p><b>Step 2 - Order Details</b></p> <p>Order ID <input type="text"/></p> <p>Quantity <input type="text"/></p> <p style="text-align: left;"><input type="button" value=" &lt;&lt; Previous"/></p> <p style="text-align: right;"><input type="button" value="Next &gt;&gt;"/></p> </div>	View2
<div style="border: 1px solid gray; padding: 5px;"> <p><b>Step 3 - Summary</b></p> <p><b>Product Details</b></p> <p>Product ID : [lblProductID]</p> <p>Product Name : [lblProductName]</p> <p>Price/Unit : [lblPrice]</p> <p><b>Order Details</b></p> <p>Order ID : [lblOrderID]</p> <p>Quantity : [lblQuantity]</p> <p style="text-align: left;"><input type="button" value=" &lt;&lt;Previous"/></p> <p style="text-align: right;"><input type="button" value="Submit &gt;&gt;"/></p> </div>	View3

MultiViewControlDemo.aspx file

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="RichControl.aspx.cs" Inherits="RichControl" %>
<! DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title></title>
</head>
<body>
  <form id="form1" runat="server">
    <div>
      <asp:MultiView ID="MultiView1" runat="server">
        <asp:View ID="View1" runat="server">
          <table style="border:1px solid black">
            <tr>
              <td colspan="2">
                <h2>Step 1 - Product Details</h2>
              </td>
            </tr>
            <tr>
              <td>Product ID</td>
              <td>
                <asp:TextBox ID="txtProductID" runat="server"></asp:TextBox>
              </td>
            </tr>
            <tr>
              <td>Product Name</td>
              <td>
                <asp:TextBox ID="txtProductName" runat="server"></asp:TextBox>
              </td>
            </tr>
            <tr>
              <td>Price/Unit</td>
              <td>
                <asp:TextBox ID="txtProductPrice" runat="server"></asp:TextBox>
              </td>
            </tr>
            <tr>
              <td colspan="2" style="text-align:right">
                <asp:Button ID="btnStep2" runat="server"
                  Text="Next >>" onclick="btnStep2_Click" />
              </td>
            </tr>
          </table>
        </asp:View>
        <asp:View ID="View2" runat="server">
          <table style="border:1px solid black">
            <tr>
              <td colspan="2">
```

```

        <h2>Step 2 - Order Details</h2>
    </td>
</tr>
<tr>
    <td>Order ID</td>
    <td>
        <asp:TextBox ID="txtOrderID" runat="server"></asp:TextBox>
    </td>
</tr>
<tr>
    <td>Quantity</td>
    <td>
        <asp:TextBox ID="txtQuantity" runat="server"></asp:TextBox>
    </td>
</tr>
<tr>
    <td>
        <asp:Button ID="btnBackToStep1" runat="server" Text="<< Previous"
            onclick="btnBackToStep1_Click" />
    </td>
    <td style="text-align:right">
        <asp:Button ID="btnStep3" runat="server" Text="Next >>"
            onclick="btnGoToStep3_Click" />
    </td>
</tr>
</table>
</asp:View>
<asp:View ID="View3" runat="server">

    <table style="border:1px solid black">

    <tr>
        <td colspan="2"><h2>Step 3 - Summary</h2></td>

    </tr>
    <tr>
        <td colspan="2"><h3>Product Details</h3></td>
    </tr>
    <tr>
        <td>Product ID</td>
        <td>
            <asp:Label ID="lblProductID" runat="server"></asp:Label>
        </td>
    </tr>
    <tr>
        <td>Product Name</td>

```

```

        <td>
            <asp:Label ID="lblProductName" runat="server"></asp:Label>
        </td>
    </tr>
    <tr>
        <td>Price/Unit</td>
        <td>
            <asp:Label ID="lblPrice" runat="server"></asp:Label>
        </td>
    </tr>
    <tr>
        <td colspan="2"><h3>Order Details</h3></td>
    </tr>
    <tr>
        <td>Order ID</td>
        <td>
            <asp:Label ID="lblOrderID" runat="server"></asp:Label>
        </td>
    </tr>
    <tr>
        <td>Quantity</td>
        <td>
            <asp:Label ID="lblQuantity" runat="server"></asp:Label>
        </td>
    </tr>
    <tr>
        <td>
            <asp:Button ID="btnBackToStep2" runat="server" OnClick="btnBackToStep2_Click" style="height:
26px" Text="<<Previous" />
        </td>
        <td style="text-align:right"
            <asp:Button ID="btnSubmit" runat="server" Text="Submit >>" OnClick="btnSubmit_Click"
                />
        </td>
    </tr>
</table>
</asp:View>
</asp:MultiView>
</div>
</form>
</body>
</html>

```

### **MultiViewControlDemo.aspx.cs file**

```

using System;
using System.Text;
public partial class RichControl : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)

```

```
{
    if (! IsPostBack)
    {
        MultiView1.ActiveViewIndex = 0;
    }
}
protected void btnStep2_Click(object sender, EventArgs e)
{
    MultiView1.ActiveViewIndex = 1;
}
protected void btnBackToStep1_Click(object sender, EventArgs e)
{
    MultiView1.ActiveViewIndex = 0;
}
protected void btnGoToStep3_Click(object sender, EventArgs e)
{
    MultiView1.ActiveViewIndex = 2;
    lblProductID.Text = txtProductID.Text;
    lblProductName.Text = txtProductName.Text;
    lblPrice.Text = txtProductPrice.Text;
    lblOrderID.Text = txtOrderID.Text;
    lblQuantity.Text = txtQuantity.Text;
}
protected void btnSubmit_Click(object sender, EventArgs e)
{
    Response.Redirect("SaveData.aspx");
}
protected void btnBackToStep2_Click(object sender, EventArgs e)
{
    MultiView1.ActiveViewIndex = 1;
}
}
```

ActiveViewIndex property of MultiView control is zero based.

## Wizard Control

- This control is same as MultiView control but the main difference is that, it has inbuilt navigation buttons.
- The wizard control enables you to design a long form in such a way that you can work in multiple sub form. You can perform the task in a step by step process. It reduces the work of developers to design multiple forms. It enables you to create multi step user interface. Wizard control provides with built-in previous/next functionality.
- The Wizard control can contains one or more WizardStep as child controls. Only one WizardStep is displayed at a time. WizardStep control has an important property called as StepType. The StepType property determines the type of navigation buttons that will be displayed for that step. The possible values are:
- The StepType associated with each WizardStep determines the type of navigation buttons that will be displayed for that step.

### The StepTypes are:

- Start:
- Step:
- Finish:
- Complete:
- Auto:

Drag the Wizard control on the web page from toolbox, you will get the following code. You can put WizardStep according to application need.

### Important events of Wizard control are as follows:

- ActiveStepChanged:
- CancelButtonClick:
- FinishButtonClick:
- NextButtonClick:
- PreviousButtonClick:

Now we will create an application as we had done with MultiView control. We will create three different WizardStep in Wizard control.

1. In First step we will design to capture Product details.
2. In Second step we will design to capture Order details.
3. Next we will show summary for confirmation.

### WizardControlDemo.aspx.cs file

```
using System;
using System.Web.UI.WebControls;

public partial class WizardControl : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }
}
```

```

protected void Wizard1_FinishButtonClick(object sender, WizardNavigationEventArgs e)
{
    Response.Redirect("SaveData.aspx");
}
protected void Wizard1_NextButtonClick(object sender, WizardNavigationEventArgs e)
{
    if (e.NextStepIndex == 2)
    {
        lblProductID.Text = txtProductID.Text;
        lblProductName.Text = txtProductName.Text;
        lblPrice.Text = txtProductPrice.Text;
        lblOrderID.Text = txtOrderID.Text;
        lblQuantity.Text = txtQuantity.Text;
    }
}
}

```

The diagram illustrates the progression of a wizard through three steps:

- Step 1 - Product Details:** This step contains three text input fields labeled "Product ID", "Product Name", and "Price/Unit". A "Next" button is located at the bottom right of the step's content area.
- Step 2 - Order Details:** This step contains two text input fields labeled "Order ID" and "Quantity". It features "Previous" and "Next" buttons at the bottom.

A blue arrow points from the "Next" button of Step 1 to the "Previous" button of Step 2, indicating the navigation flow.

Register the event for Next button by using property window with event tab. In the given example, for going third step from second we have to set `e.NextStepIndex == 2`. Here StepIndex is zero based.

FinishButtonClick event performs the final WizardStep with a summary of the answers entered in the previous WizardStep controls.



## State management

- Maintaining state is an important part of any web application. State Management System is a mechanism to track the user state, or data, which is significant with particular application. State management manages the state of an object on different request.
- The HTTP protocol is the fundamental protocol of the World Wide Web. HTTP is a stateless protocol means every request is from new user with respect to web server. HTTP protocol does not provide any method of determining whether any two requests are made by the same person.

### There are two types of state management system in ASP.NET.

- Client-side state management
- Server-side state management

Client-side state management stores information on the client's computer and server-side state management stores the information in the server's memory or a database.

Client side state management system

ASP.NET provides several techniques for storing state information on the client. These include the following:

- **View state:** ASP.NET uses view state to track values in controls between page requests. It works within the page only. You cannot use view state value in next page. ASP.NET page contains a hidden form field named `__VIEWSTATE`. This hidden form field stores the value of the control's property. When the page is posted back to the server, then the value of `__VIEWSTATE` is pulled out and re-creates the values of all the properties stored in View State.
- **Control state:** The data that is associated with the server controls is called as control state. You can persist information about a control that is not part of the view state. If view state is disabled for a control or the page, the control state will still work.
- **Hidden fields:** It store data without displaying that control and data to the user's browser. This data is presented back to the server and is available when the form is processed. Hidden fields data is available within the page only (page-scoped data). It is rendered as an `<input type="hidden"/>` HTML tag. Hidden field should not be used to store confidential data.
- **Cookies:** Cookies are small piece of information that server creates on the browser. Cookies store a value in the user's browser that the browser sends with every page request to the web server. It works on key/value pair.

There are two types of cookies:

- Session cookies
- Persistent cookies

- **Query strings:** In query strings values are stored at the end of the URL. These values are visible to the user through his or her browser's address bar. Query strings are not secure. You should not send secret information through the query string.

### ***View state***

- View state is an inbuilt feature of ASP.NET that retains values between multiple requests for the same page. ASP.NET page contains a hidden form field named `__VIEWSTATE`.
- This hidden form field stores the value of the control's property. By default view state is enabled for page and its controls.
- You can disable view state by setting the property `EnableViewState` as false. Storing too much data into View State can hamper the performance of web page.

Therefore we should take care while enabling and disabling the property `EnableViewState`.

### **Example**

```
//writing information to view state
ViewState.Add("MyInfo", "Welcome");
//read information from view state
if (ViewState["MyInfo"] != null)
{
    string data = (string)ViewState["MyInfo"];
}
```

### ***Hidden fields***

Hidden fields in HTML are simply input fields and not visible on the browser during execution. Hidden fields are used to store data at the page level. Hidden fields are simple to implement for a page specific data and stores small amount of data. We should not use hidden fields for sensitive data. It has no built-in compression, encryption technique.

```
<asp:HiddenField ID="HiddenField1" runat="server" />
```

### **Example**

```
//writing information to Hidden field
HiddenField1.Value = "Welcome";
//read information from Hidden field
string str = HiddenField1.Value;
```

### **Cookies**

A cookie is a small amount of data that server creates on the client. Cookie is small text information. You can store only string values when using a cookie. When a request sent to web server, server creates a cookie, and sent to browser with an additional HTTP header.

### **The HTTP header looks like this:**

```
Set-Cookie: message=Hello.
```

Here cookie name is message and value is hello.

If the cookies has created on a browser and user requests a page from the same application, then the browser sends a header that looks like this:

Cookie: message=Hello

### **There are two types of cookies:**

- Session cookies: A session cookie exists only till the user closes the web browser, the session cookie deleted permanently.
- Persistent cookies: A persistent cookie, on the other hand, can available for months or even years. When you create a persistent cookie, the cookie is stored permanently by the user's browser on the user's computer.

### *Use of Cookies*

#### **Some common uses of cookies are:**

- Authentication of user.
- Identification of a user session.
- User's preferences.
- Shopping cart contents.
- Remember users between visits.

### *Creating and reading cookies*

#### **We can create cookies in different ways.**

##### Example 1

```
Response.Cookies["Message"].Value = TextBox1.Text;
string msg = Request.Cookies["Message"].Value;
```

##### *Example 2*

```
HttpCookie UserCookies = new HttpCookie("Message");
UserCookies.Value = TextBox1.Text;
Response.Cookies.Add(UserCookies);
// Reading the cookie.
string roll = Request.Cookies["Message"].Value;
```

##### *Example 3*

```
//Writing Multiple values in single cookie
Response.Cookies["EmpCookies"]["EmpID"] = txtID.Text;
Response.Cookies["EmpCookies"]["FirstName"] = txtFirstName.Text;
Response.Cookies["EmpCookies"]["LastName"] = txtLastName.Text;
Response.Cookies["EmpCookies"]["Address"] = txtAddress.Text;

//Reading Cookie.
string info;
if (Request.Cookies["EmpCookies"] != null)
{
    info = Request.Cookies["EmpCookies"]["EmpID"] + "<br>";
    info += Request.Cookies["EmpCookies"]["FirstName"] + "<br>";
    info += Request.Cookies["EmpCookies"]["LastName"] + "<br>";
}
```

```
info += Request.Cookies["EmpCookies"]["Address"] + "<br>";
```

```
Label1.Text = info;
```

```
}
```

// cookie names are case sensitive. Cookie named EmpCookies is different from setting a cookie named empcookies.

The above examples create a session cookie. The cookie disappears when you close your web browser. If you want to create a persistent cookie, then you need to specify an expiration date for the cookie.

```
Response.Cookies["message"].Expires = DateTime.Now.AddYears(1);
```

### Limitation of cookies

- Cookie can store only string value.
- Cookies are browser dependent.
- Cookies are not secure.
- Cookies can store small amount of data.
- Size of cookies is limited to 4096 bytes.

#### *Important properties of HttpCookie*

- **Domain:** Enables you to get or set the domain of the cookie.
- **Expires:** It contains the expiration time of the cookie.
- **HasKeys:** Returns bool value, indicating whether the cookie has subkeys.
- **Name:** Provides the name of the cookie.
- **Path:** Enables you to get or set the virtual path to submit with the cookie.
- **Secure:** It contains true if the cookie is to be passed with SSL.
- **Value:** It contains the value of the cookie.

#### **Example**

```
using System;
using System.Web;
public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        HttpCookie obj = new HttpCookie("MyCookie");
        obj.Value="Welcome !!";
        Response.Cookies.Add(obj);
        string info;
        info = "Domain =: " + obj.Domain + "<br>";
        info += "Name =: " + obj.Name + "<br>";
        info += "Path =: " + obj.Path+"<br>";
        info += "Value =: " + obj.Value + "<br>";
        info += "HasKeys =: " + obj.HasKeys + "<br>";
        info += "Secure =: " + obj.Secure + "<br>";
        Label1.Text = info;}
}
```

### **Query strings**

- Query String object is helpful when we want to transfer a value from one page to another. Query String is very easy to use. Query string values are appended to the end of the page URL. It uses a question mark (?), followed by the parameter name followed by an equal sign (=) and its value.
- You can append multiple query string parameters using the ampersand (&) sign.
- Always remember, we should not send lots of data through QueryString. Another limitation is that information we send through QueryString is visible on the address bar.

### **Example**

```
Response.Redirect("Default.aspx?msg="+txtMessage.Text);
```

In the example, the Response.Redirect method requests the Default.aspx page. The query string contains a single parameter named msg. The value for that parameter is set at run time by entering the data into textbox control. In this example the query string has one parameter but we can pass more than one parameter as given below.

```
Response.Redirect("Default2.aspx?ID=" + txtID.Text + "&Name=" + txtFirstName.Text);
```

### **Reading values from QueryString**

```
Label1.Text = "ID: " + Server.HtmlEncode(Request.QueryString["ID"]) + ", Name: " +  
Server.HtmlEncode(Request.QueryString["Name"]);
```

We should use Server.HtmlEncode method while using QueryString. Server.HtmlEncode method encode the "<" sign with "<." Special characters that a Web browser cannot process, it helps to process that browser understands easily.

### **Important points about QueryString**

- It is easy to use.
- Sensitive data should not pass using QueryString.
- Browsers have 2,083-character limits on URLs. Therefore there is limit to pass the data.
- QueryString is a part of URL.
- It uses one or more than one parameter.
- It uses "&" sign while using more than one parameter.
- SPACE is encoded as '+' or '%20'

## Server side state management system - ASP.NET

**There are two important objects which work on server.**

- Session
- Application

State management is the technique that is used to maintain user and page information over multiple requests while browsing the web.

- HTTP is a stateless protocol. It does not store any information about user on web page. It is a general requirement that information should be maintained while navigating the website.
- Session provides that facility to store information on server memory not browse. It stores the user's specific information. It can store any type of object. For every user Session data store separately, means session is user specific.

### **Storing the data in Session object**

```
Session ["UserName"] = txtName.Text;
```

### **Retrieving the data from Session object**

```
Label1.Text = Session ["UserName"].ToString();
```

When we store data to Session state, a session cookie named is ASP.NET\_SessionId is created automatically. It contains a unique identifier that is used to track the user while moving from one page to another page.

### **Important properties of Session object**

Session Properties	Description
CookieMode	It specifies whether cookieless sessions are enabled. Possible values are AutoDetect, UseCookies, UseDeviceProfile, and UseUri.
SessionID	It provides the unique session identifier. It is secure enough and can't be decoded or hampered. When client communicate with server, only session id is transmitted, between them.
Count	It provides the number of items in Session state.
IsCookieless	Provides the information whether sessions are cookieless or not.
IsNewSession	It determines whether session is new or not.
IsReadOnly	It determines whether the Session state is read-only.
Keys	Provides the list of item names stored in Session state.
Mode	It determines the current Session state store provider. Possible values are Custom, InProc, Off, SqlServer, and StateServer.

### Important methods of Session object

- **Abandon:** It is used to end a user session.
- **Clear:** It clears all items from Session state.
- **Remove:** This method is used to remove a particular item from Session state.

### Example

```
using System;
using System.Web;
public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        string info;
        info = "CookieMode =:" + Session.CookieMode.ToString() + "</br>";
        info += "Count =:" + Session.Count.ToString() + "</br>";
        info += "IsCookieless =:" + Session.IsCookieless.ToString() + "</br>";
        info += "IsNewSession =:" + Session.IsNewSession.ToString() + "</br>";
        info += "IsReadOnly =:" + Session.IsReadOnly.ToString() + "</br>";
        info += "Keys =:" + Session.Keys.Count + "</br>";
        info += "Mode =:" + Session.Mode.ToString() + "</br>";
        info += "SessionID =:" + Session.SessionID.ToString() + "</br>";
        Label1.Text = info;
    }
}
```

### Session Events

There are two events that session object supports. These two events are handled in Global.aspx file.

- Session\_Start
- Session\_End

Whenever a new user sessions starts, Session\_Start events fires. The Session\_End event is raised when a session ends.

Example: Global.asax file

```
<%@ Application Language="C#" %>
<script runat="server">
    void Application_Start(object sender, EventArgs e)
    {
        Application["UserCount"] = 0;
    }
    void Application_End(object sender, EventArgs e)
    {
        // Code that runs on application shutdown
    }
    void Application_Error(object sender, EventArgs e)
```

```

{
    // Code that runs when an unhandled error occurs
}

void Session_Start(object sender, EventArgs e)
{
    Application.Lock();
    int count = (int)Application["UserCount"];
    Application["UserCount"] = count + 1;
    Application.Unlock();
}
void Session_End(object sender, EventArgs e)
{
    Application.Lock();
    int count = (int)Application["UserCount"];
    Application["UserCount"] = count - 1;
    Application.Unlock();
}
}
</script>

```

In the above example, the variable `UserCount` is incremented by one, whenever a new session begins.

The `Session_End` event is raised, when a session ends and the `UserCount` variable is decremented by one.

**We can display the result on web page as follows:**

```

void Page_Load()
{
    Label1.Text = Application["UserCount"].ToString();
}

```

### ***Session Times Out property***

By default, the ASP.NET Framework provides 20 minutes as session timeout. We can change this time according to application need.

Be aware that when you increase the value of session timeout property more memory is consumed by your application.

You can specify the Session timeout in the web configuration file or you can do it programmatically.

```

<configuration>
  <system.web>
    <sessionState timeout="60" />
  </system.web>
</configuration>

```



### ***Session Mode***

In ASP.NET there are following session modes available,

- InProc
- StateServer
- SQLServer
- Custom
- Off

By default, the Session state mode is InProc means Session state is stored in memory in the same process as the ASP.NET process. So accessing data is very fast. Another advantage is that there are no requirements of serialization to store data in InProc Session Mode.

There are two main disadvantages to storing Session state in the ASP.NET process.

- We can't use in-process Session state with a web farm.
- All Session state is lost, if application restarts.

You can store Session data out-of-process. You can choose **StateServer** option for storing session data. It stores Session state in a Windows NT process. **SqlServer** mode stores Session state in a SQL Server database. It is the most reliable and secure session management and Session data do not affected if we restart the IIS.

**Custom** mode stores Session state in a custom location. If we set Session **Mode="off"** in web.config, Session will be disabled for the application. For this we need to configure web.config in following way.

```
<configuration>
  <system.web>
    <sessionState mode="Off"></sessionState>
  </system.web>
</configuration>
```

Session State Mode	State Provider
InProc	In-Memory Object
StateServer	Aspnet_state.exe
SQLServer	DataBase
Custom	CustomProvider

### **Cookieless Session State**

If a user disables cookies in the browser, then Session state doesn't work because by default, Session state depends on cookies. The ASP.NET Framework uses the ASP.NET\_SessionId cookie identifier to identify the user while browsing the web. If you want that Session state should work even when cookies are disabled, then you can use cookieless sessions.

You can enable cookieless sessions by adjusting the sessionState element in the web configuration file as.

```
<configuration>
  <system.web>
    <sessionState cookieless="AutoDetect" regenerateExpiredSessionId="true" />
  </system.web>
</configuration>
```

### **Advantages and disadvantages of Session**

Following are the basic advantages and disadvantages of using session.

#### **Advantages:**

- It stores user states and data to all over the application.
- Easy mechanism to implement and we can store any kind of object.
- Stores every user data separately.
- Session is secure and transparent from user because session object is stored on the server.

#### **Disadvantages:**

- Performance overhead in case of big number of user, because of session data stored in server memory.
- Overhead involved in serializing and De-Serializing session Data. Because In case of StateServer and SQLServer session mode we need to serialize the object before store.

#### ***Application State***

Application object is used to store information at application level rather than user level. All pages of your application can access the Application object. Application variables are stored on a web server.

If you are using Application object, then you may face concurrency problem. To avoid this problem we should use the lock and unlock methods. Therefore if multiple thread requests came for same data then only one thread can do the work.

#### **Writing data to Application object**

```
Application["Message"] = "Hello to all";
```

We can use Application object in a scenario where we want to count the number of visitors of web site.

Application State variables are empty, when the process hosting the application is restarted.

**Difference between session state and application state**

Application	Session
It works at application level rather than user level.	Session object is user specific.
Application state is stored only in the memory on the server.	Session state is stored in inProc and outProc
Application state does not depends upon client's cookies	Session object depends upon cookie or can be cookieless.
Application state does not depend upon the current browser.	Session state has scope to the current browser only.

## Tracing, Debugging, Error Handling

In any application, errors are bound to occur during the development process. It is important to be able to discover errors at an early stage.

In Visual Studio, it is possible to do this for ASP.Net applications. Visual Studio is used for Debugging and has error handling techniques for ASP.Net.

### What is Debugging in ASP.NET?

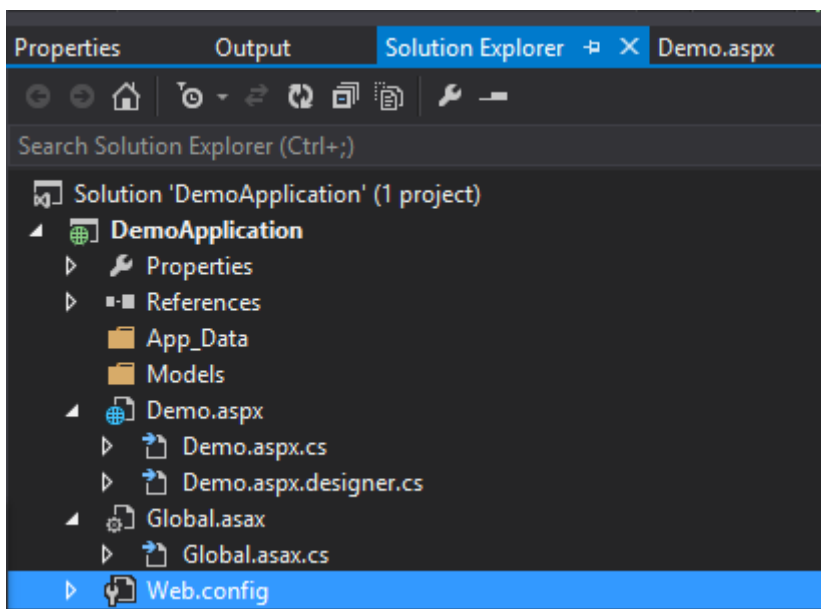
Debugging is the process of adding breakpoints to an application. These breakpoints are used to pause the execution of a running program. This allows the developer to understand what is happening in a program at a particular point in time.

Let's take an example of a program. The program displays a string "We are debugging" to the user. Suppose when we run the application, for some reason, the string is not displayed. To identify the problem we need to add a breakpoint. We can add a breakpoint to the code line which displays the string. This breakpoint will pause the execution of the program. At this point, the programmer can see what is possibly going wrong. The programmer rectifies the program accordingly.

Here in the example, we will use our 'DemoApplication' that was created in earlier chapters. In the following example, we will see

- How to make the demo application display a string.
- How to add breakpoints to an application.
- How to debug the application using this breakpoint.

**Step 1)** Let's first ensure we have our web application open in Visual Studio. Ensure the DemoApplication is open in Visual Studio.



**Step 2)** Now open the Demo.aspx.cs file and add the below code line.

- We are just adding the code line Response.Write to display a string.
- So when the application executes, it should display the string "We are debugging" in the web browser.

```

public partial class Demo : System.Web.UI.Page
{

    protected void Page_Load(object sender, EventArgs e)
    {
        Response.Write("We are debugging");
    }
}

```

Display a string

```

namespace DemoApplication
{
    public partial class Demo : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            Response.Write("We are debugging");
        }
    }
}

```

**Step 3)** Now let's add a breakpoint. A breakpoint is a point in Visual Studio where you want the execution of the program to stop.

```

public partial class Demo : System.Web.UI.Page
{

    protected void Page_Load(object sender, EventArgs e)
    {
        Response.Write("We are debugging");
    }
}

```

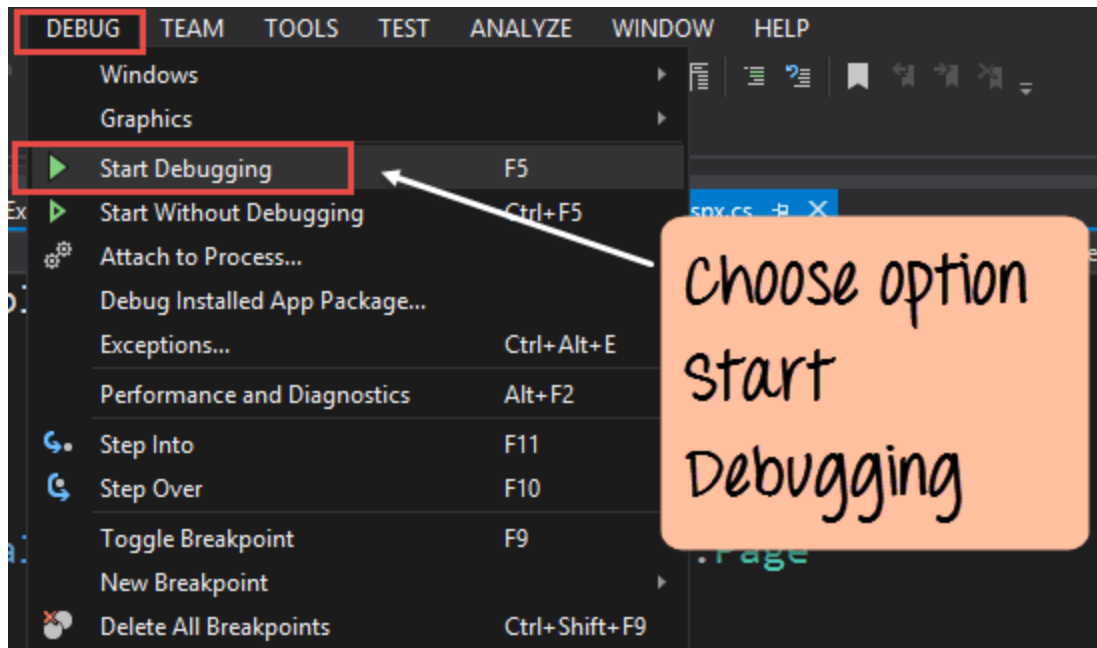
Click on the column

Code line will now become red

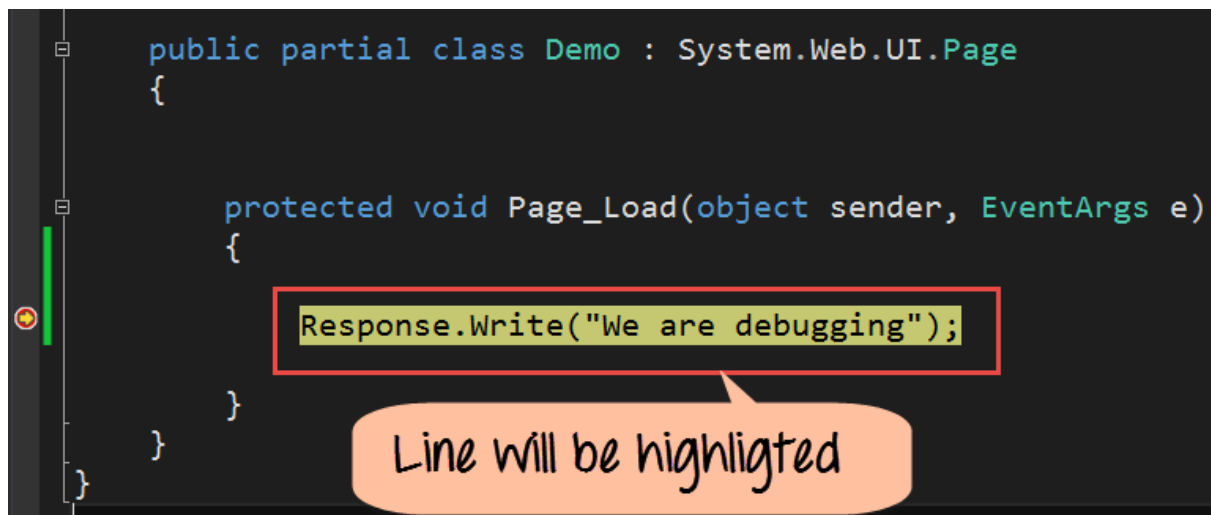
1. To add a breakpoint, you need to click the column where you want the breakpoint to be inserted. So in our case, we want our program to stop at the code line "Response.Write". You don't need to add any command to add a breakpoint. You just need to click on the line on which you want to add a breakpoint.
2. Once this is done, you will notice that the code gets marked in red. Also, a red bubble comes up in the column next to the code line.

**Note:** - You can add multiple breakpoints in an application

**Step 4)** Now you need to run your application using Debugging Mode. In Visual Studio, choose the menu option Debug->Start Debugging.



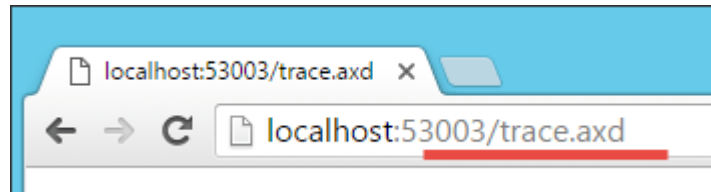
**Output:-**



When you perform all the steps correctly, the execution of the program will break. Visual Studio will go to the breakpoint and mark the line of code in yellow.

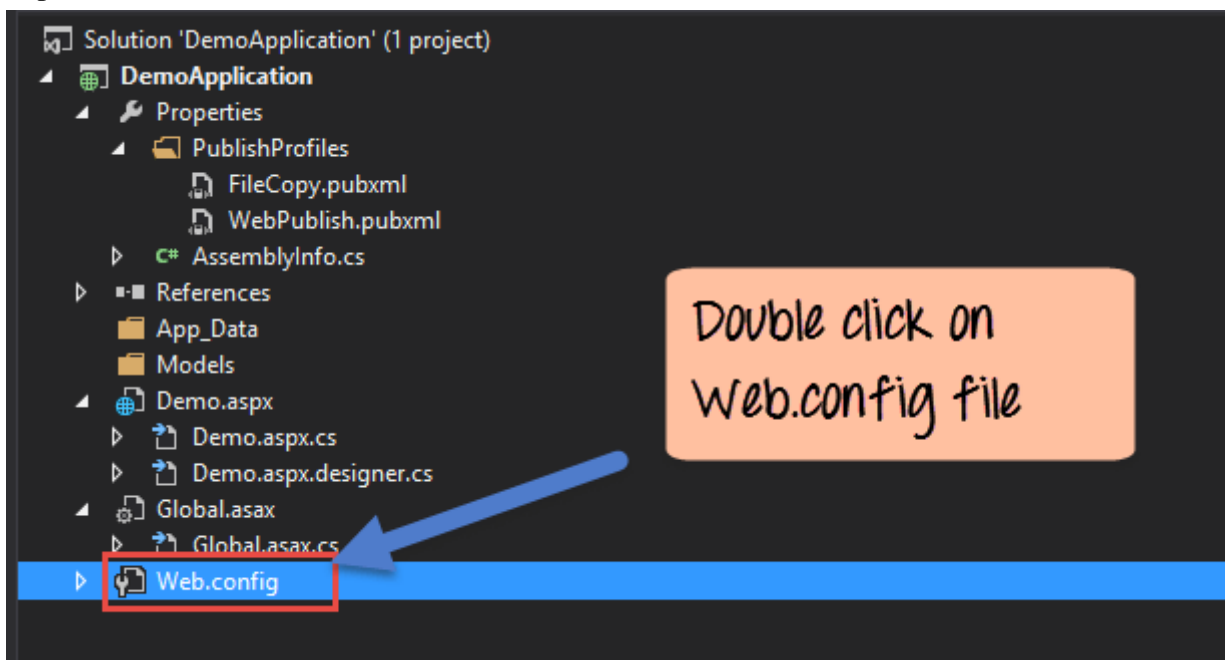
## What is Tracing in ASP.NET?

Application tracing allows one to see if any pages requested results in an error. When tracing is enabled, an extra page called trace.axd is added to the application. (See image below). This page is attached to the application. This page will show all the requests and their status.



Let's look at how to enable tracing for an application.

**Step 1)** Let's work on our 'DemoApplication'. Open the web.config file from the Solution Explorer.



**Step 2)** Add the below line of code to the Web.config file.

The trace statement is used to enable tracing for the application.

- The 'requestLimit' in trace statement is used. It specifies the number of page requests that has to be traced.
- In our example, we are giving a limit of 40. We give limit because a higher value will degrade the performance of the application.

```

<!--
For more information on how to configure your ASP.NET application, please visit
http://go.microsoft.com/fwlink/?LinkId=169433
-->
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.0" />
    <httpRuntime targetFramework="4.0" />

    <trace enabled="true" pageOutput="false" requestLimit="40" localOnly="false"/>

  </system.web>
</configuration>

```

To enable tracing

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<!--
```

For more information on how to configure your ASP.NET application, please visit

<http://go.microsoft.com/fwlink/?LinkId=169433>

```
-->
```

```
<configuration>
```

```
  <system.web>
```

```
    <compilation debug="true" targetFramework="4.0" />
```

```
    <httpRuntime targetFramework="4.0" />
```

```
    <trace enable="true" pageOutput="false" requestLimit="40" localOnly="false"/>
```

```
  </system.web>
```

```
</configuration>
```

Run the "demoapplication" in Visual Studio.

**Output:-**

Application Trace

[ [clear current trace](#) ]  
Physical Directory: C:\Guru99\DemoApplication\DemoApplication\

No.	Time of Request	File	Status Code	Verb	View Details
1	5/26/2016 12:21:57 AM		200	GET	<a href="#">View Details</a>
2	5/26/2016 12:22:04 AM	Demo.aspx	200	GET	<a href="#">View Details</a>
3	5/26/2016 12:22:06 AM	__browserLink/requestData/78dd65b7dc3a46c3ae4915043b26f688	200	GET	<a href="#">View Details</a>

Microsoft .NET Framework Version:4.0.30319; ASP.NET Version:4.0.30319.34274



If you now browse to the URL – **http://localhost:53003/trace.axd** , you will see the information for each request. Here you can see if any errors occur in an application. The following types of information are shown on the above page

1. The time of the request for the web page.
2. The Name of the web page being requested.
3. The status code of the web request. (status code of 200 means that the request is successful).
4. The View details which you allow to view more details about the web request. An example of this is shown below. One important detailed information provided is the header information. This information shows what is the information sent in the header of each web request.

The screenshot displays a web browser window with the following content:

localhost:53003/Trace.axd X  
localhost:53003/Trace.axd?id=0

**Request Cookies Collection**

Name	Value	Size
------	-------	------

**Response Cookies Collection**

Name	Value	Size
------	-------	------

**Headers Collection**

Name	Value
Connection	keep-alive
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding	gzip, deflate, sdch
Accept-Language	en-US,en;q=0.8
Host	localhost:53003
User-Agent	Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.102 Safari
Upgrade-Insecure-Requests	1

**Form Collection**

Name	Value
------	-------

## Error Handling: Displaying a Custom Error Page

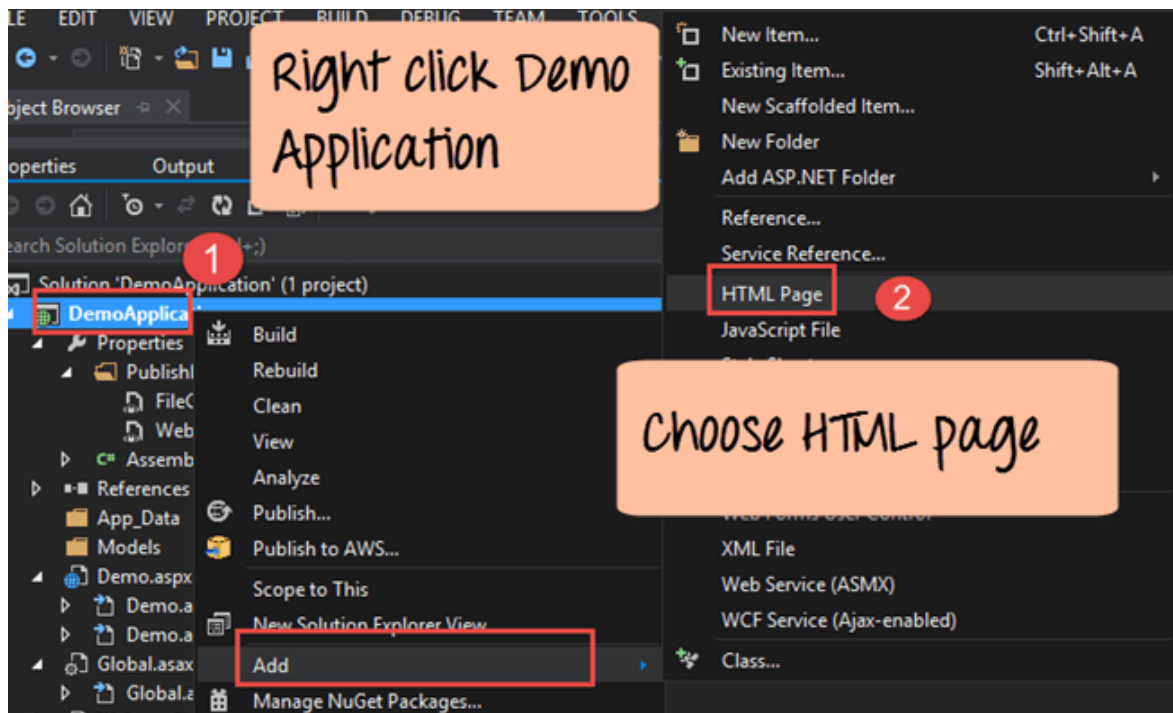
In ASP.Net, you can have custom error pages displayed to the users. If an application contains any sort of error, a custom page will display this error to the user.

In our example, we are first going to add an HTML page. This page will display a string to the user "We are looking into the problem". We will then add some error code to our demo.aspx page so that the error page is shown.

Let's follow the below mentioned steps

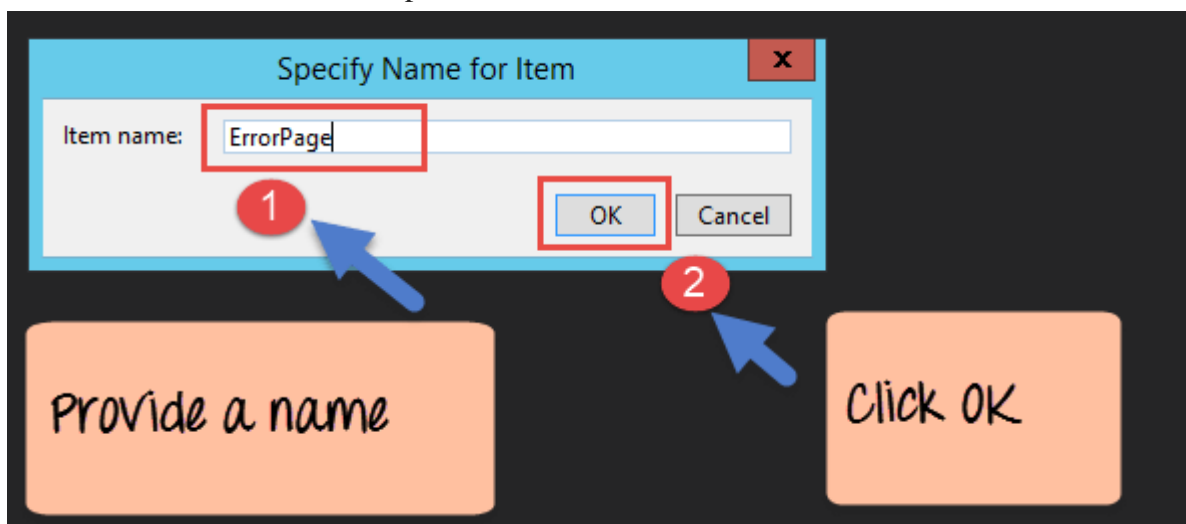
**Step 1)** Let's work on our DemoApplication. Let's add an HTML page to the application

1. Right-click on the DemoApplication in Solution Explorer
2. Choose the menu option 'Add'-'>'HTML Page

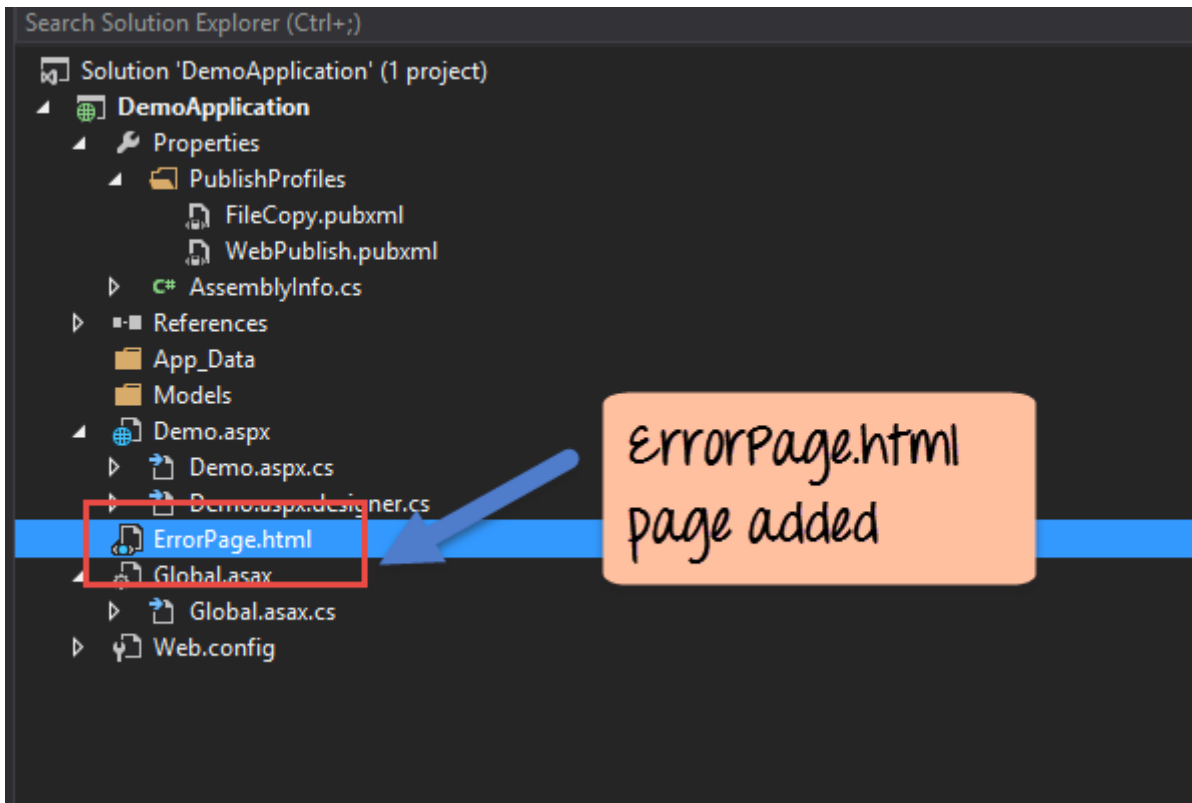


**Step 2)** In the next step, we need to provide a name to the new HTML page.

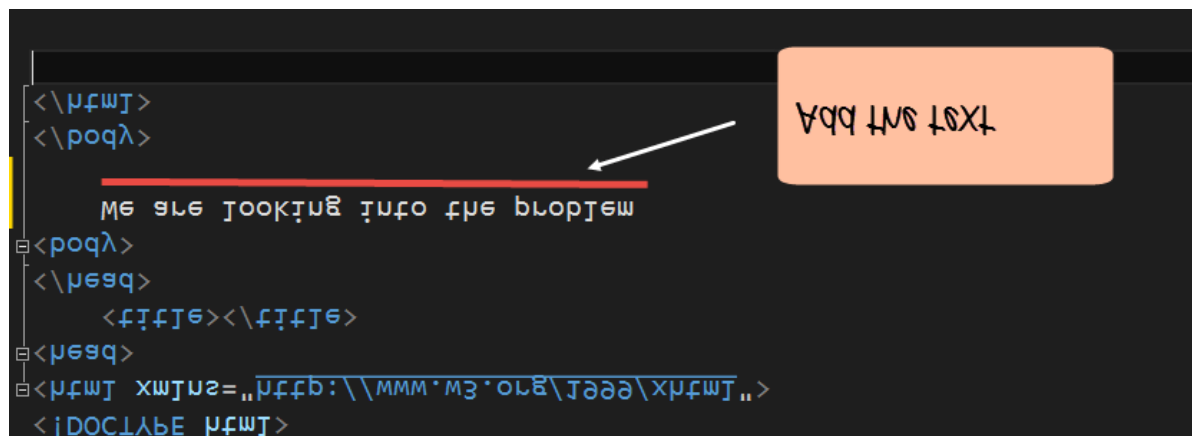
1. Provide the name as 'ErrorPage.'
2. Click the 'OK' button to proceed.



**Step 3)** The Errorpage will automatically open in Visual Studio. If you go to the Solution Explorer, you will see the file added.



Add the code line "We are looking into the problem" to the HTML page. You don't need to close the HTML file before making the change to the web.config file.



```

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title></title>
</head>
  <body>
    We are looking into the problem
  </body>
</html>

```

**Step 4)** Now you need to make a change in the web.config file. This change will notify that whenever an error occurs in the application, the custom error page needs to be displayed.

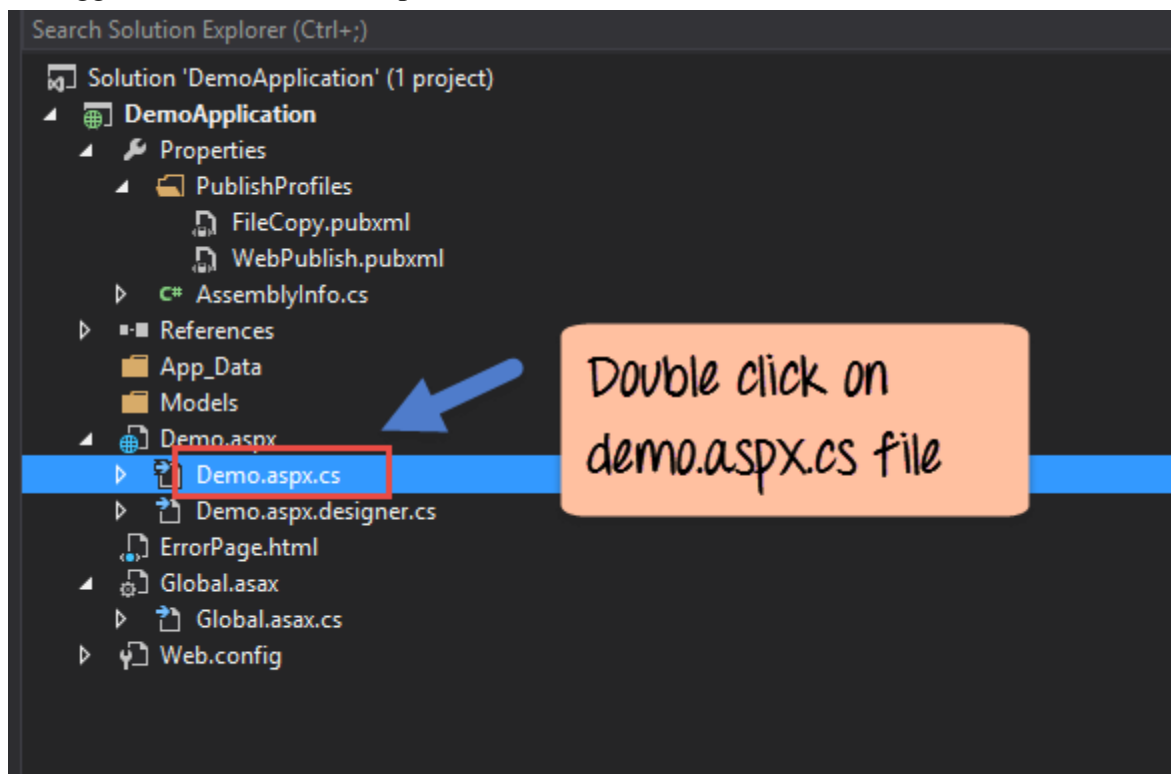
The 'customErrors' tag allows defining a custom error page. The defaultRedirect property is set to the name of our custom error's page created in the previous step.

```
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.0" />
    <httpRuntime targetFramework="4.0" />
    <customErrors mode="On" defaultRedirect="ErrorPage.html" />
  </system.web>
</configuration>
```

Custom error block

```
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.0" />
    <httpRuntime targetFramework="4.0" />
    <customErrors mode="On" defaultRedirect="ErrorPage.html" />
  </system.web>
</configuration>
```

**Step 5)** Now let's add some faulty code to the demo.aspx.cs page. Open this page by double-clicking the file in Solution Explorer



Add the below code to the Demo.aspx.cs file.

- These lines of code are designed to read the lines of a text from a file.
- The file is supposed to be located in the D drive with the name 'Example.txt.'

- But in our situation, this file does not really exist. So this code will result in an error when the application runs.

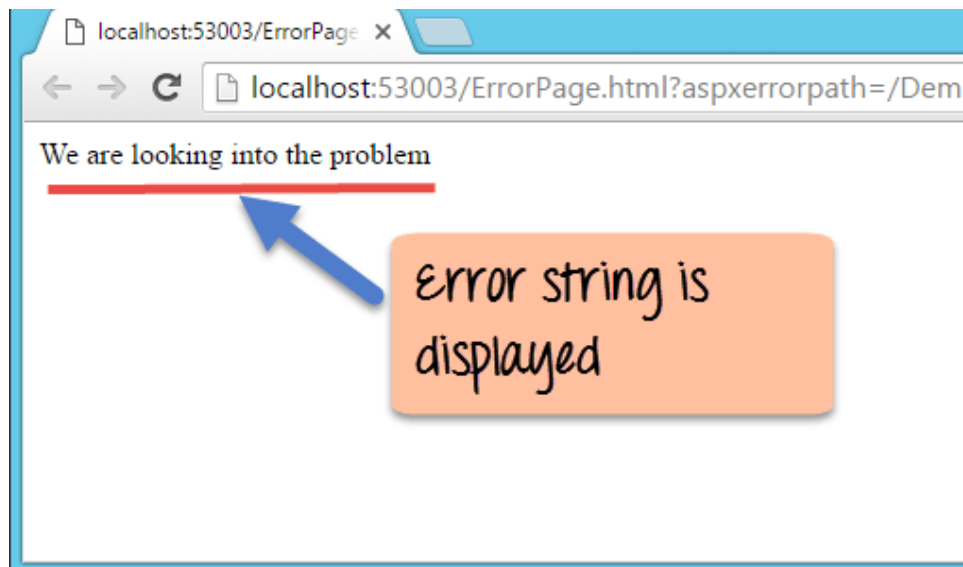
```
public partial class Demo : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        string path = @"D:\Example.txt";
        string[] lines;
        lines = File.ReadAllLines(path);
    }
}
```

Code which results in an error

```
namespace DemoApplication
{
    public partial class Demo : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            String path = @"D:\Example.txt";
            string[] lines;
            lines = File.ReadAllLines(path);
        }
    }
}
```

Now execute the code in Visual Studio and you should get the below output.

### Output:-



The above page shows that an error was triggered in the application. As a result, the Error.html page is displayed to the user.

## ASP.NET Unhandled Exception

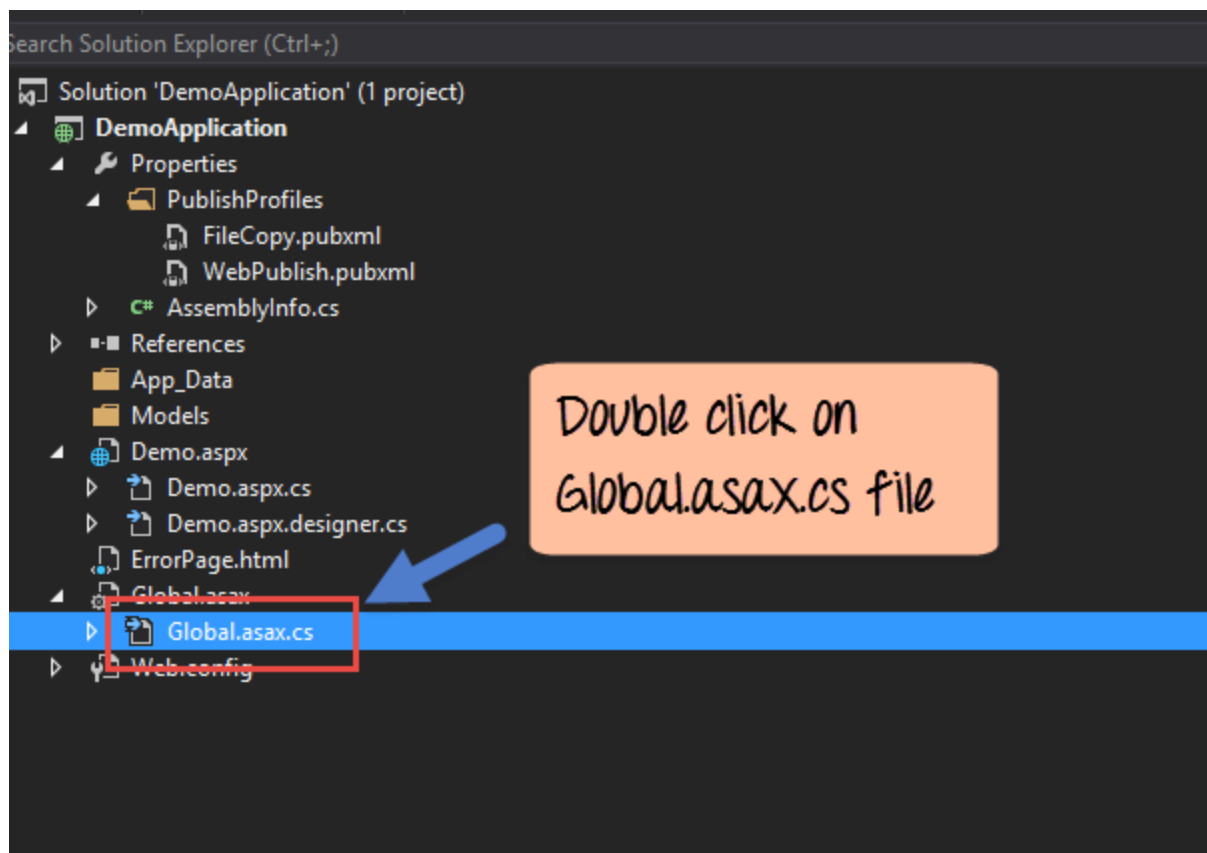
Even in the best of scenarios, there can be cases of errors which are just not foreseen.

Suppose if a user browses to the wrong page in the application. This is something that cannot be predicted. In such cases, ASP.Net can redirect the user to the errorpage.html.

Let's see an example on this.

- We are going to use our same 'DemoApplication' which has the Errorpage.html.
- And we will try to view a web page which does not exist in our application.
- We should be redirected to our ErrorPage.html page in this case. Let's see the steps to achieve this.

**Step 1)** Let's work on our DemoApplication. Open the Global.asax.cs file from the Solution Explorer



**NOTE:** The global.asax.cs file is used to add code that will be applicable throughout all pages in the application.

**Step 2)** Add the below line of code to the global.asax.cs. These lines will be used to check for errors and display the ErrorPage.html page accordingly.

```

{
}

protected void Application_Error(object sender, EventArgs e)
{
    HttpException lastErrorWrapper = Server.GetLastError() as HttpException;
    if (lastErrorWrapper.GetHttpCode() == 404)
        Server.Transfer("~/ErrorPage.html");
}
}

```

```

namespace DemoApplication
{

    public partial class Demo : System.Web.UI.Page
    {
        protected void Application_Error(object sender, EventArgs e)
        {
            HttpException lastErrorWrapper = Server.GetLastError() as
HttpException;

            if(lastErrorWrapper.GetHttpCode() == 404)
                Server.T ransfer("~/ErrorPage.html");
        }
    }
}

```

### Code Explanation:-

1. The first line is the Application\_Error event handler. This event is called whenever an error occurs in an application. Note that the event name has to be 'Application\_Error'. And the parameters should be as shown above.
2. Next, we define an object of the class type HttpException. This is a standard object which will hold all the details of the error. We then use the Server.GetLastError method to get all the details of the last error which occurred in the application.
3. We then check if the error code of the last error is 404. (The error code 404 is the standard code returned when a user browses to a page which is not found). We then transfer the user to the ErrorPage.html page if the error code matches.

Now run the code in Visual Studio and you should get the below output

### Output:-

Browse the page <http://localhost:53003/Demo1.aspx> . Remember that Demo1.aspx does not exist in our application. You will then get the below output.



The above page shows that an error was triggered in the application. As a result, the Error.html page is displayed to the user.

### **ASP.NET Error logging**

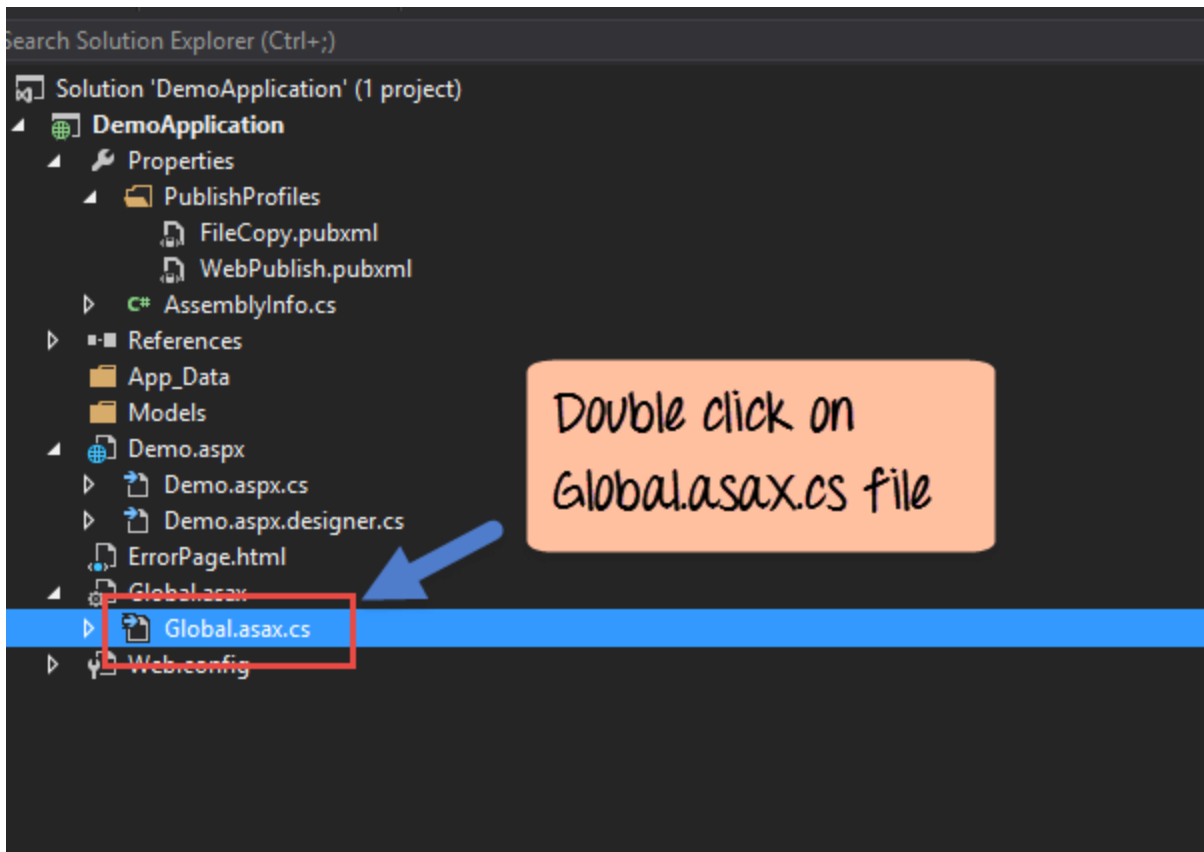
By logging application errors, it helps the developer to debug and resolve the error at a later point of time. ASP.Net has the facility to log errors. This is done in the Global.asax.cs file when the error is captured. During the capturing process, the error message can be written into a log file.

Let's see an example on this.

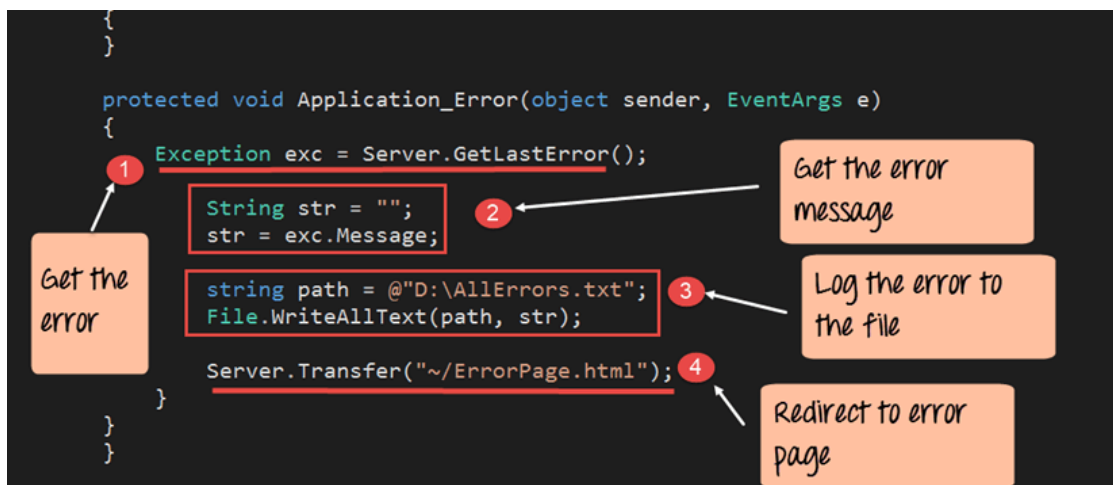
- We are going to use our same DemoApplication which has the Errorpage.html.
- And we will try to view a web page which does not exist in our application.
- We should be redirected to our ErrorPage.html page in this case.
- And at the same time, we will write the error message to a log file. Let's see the steps to achieve this.

**Step 1)** Let's work on our DemoApplication. Open the Global.asax.cs file from the Solution Explorer





**Step 2)** Add the below line of code to the global.asax.cs. It will check for errors and display the ErrorPage.html page accordingly. Also at the same time, we will log the error details in a file called 'AllErrors.txt.' For our example, we will write code to have this file created on the D drive.



```

namespace DemoApplication
{
    public partial class Demo : System.Web.UI.Page
    {
        protected void Application_Error(object sender, EventArgs e)
        {

```

```
String str ="";
str = exc.Message;

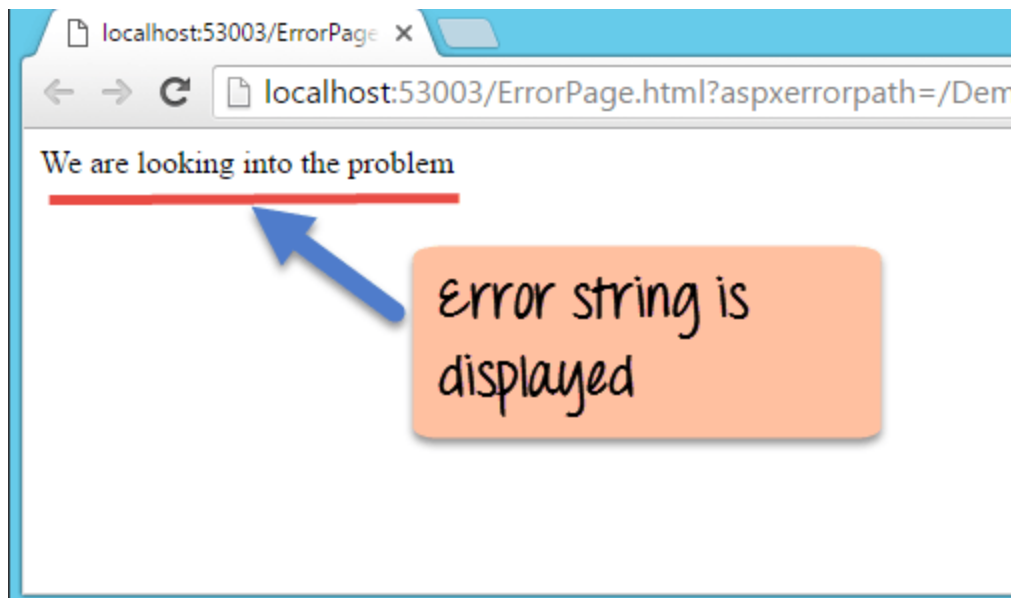
String path = @"D:\AllErrors.txt";
File.WriteAllTest(path,str);
Server.trasfer("~/ErrorPage.html");
}
}
}
```

### Code Explanation:-

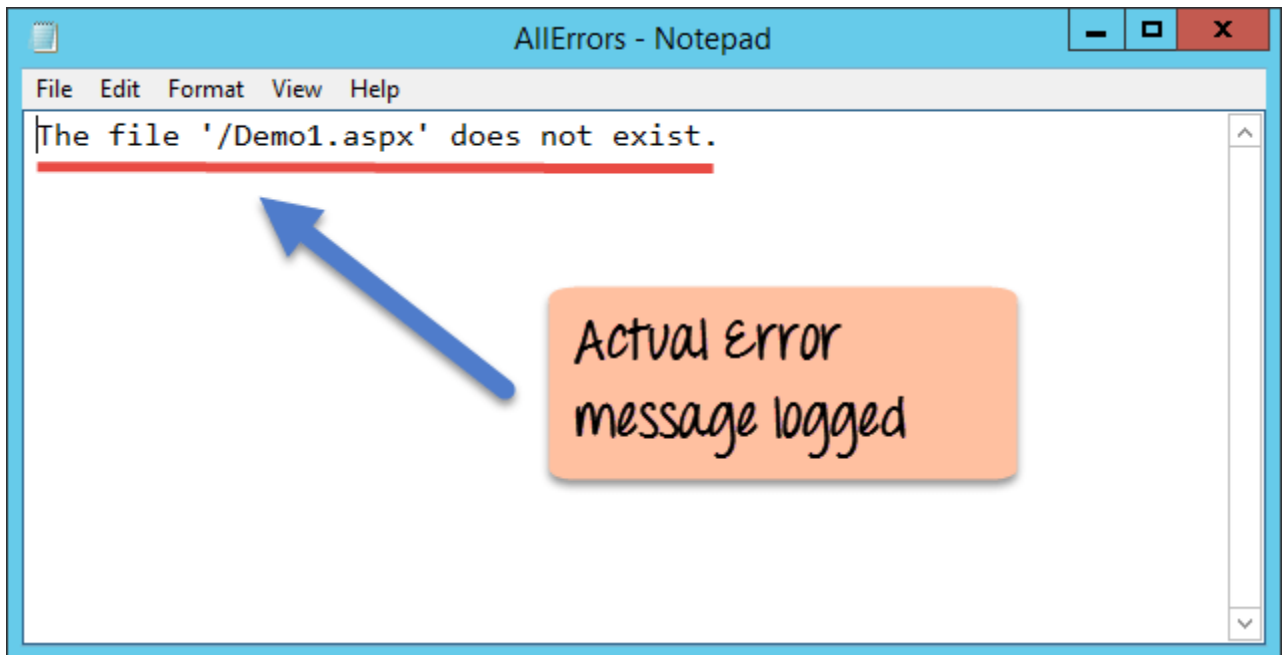
1. The first line is to get the error itself by using the 'Server.GetLastError' method. This is then assigned to the variable 'exc'.
2. We then create an empty string variable called 'str'. We get the actual error message using the 'exc.Message' property. The exc.Message property will have the exact message for any error which occurs when running the application. This is then assigned to the string variable.
3. Next, we define the file called 'AllErrors.txt.' This is where all the error messages will be sent. We write the string 'str' which contains all the error messages to this file.
4. Finally, we transfer the user to the ErrorPage.html file.

### Output:-

Browse the page <http://localhost:53003/Demo1.aspx> . Remember that Demo1.aspx does not exist in our application. You will then get the below output.



And at the same time, if you open the 'AllErrors.txt' file you will see the below information.



The error message can then be passed on to the developer at a later point in time for debugging purposes.

### Summary

- ASP.Net has the facility to perform debugging and Error handling.
- Debugging can be achieved by adding breakpoints to the code. One then runs the Start with Debugging option in Visual Studio to debug the code.
- Tracing is the facility to provide more information while running the application. This can be done at the application or page level.
- At the page level, the code `Trace=true` needs to be added to the page directive.
- At the application level, an extra page called `Trace.axd` is created for the application. This provides all the necessary tracing information.

## Unit IV – Missing Topics

**Unit – IV - Developing C#.NET Applications** - Introducing C# - overview of C# - Literals, Variables- Data Types, -Operators, -checked and unchecked operators – Expressions – Branching -Looping-*Object Oriented Aspects Of C#*: Class – Objects – Constructors and its types- inheritance, properties, indexers, index overloading – polymorphism - sealed class and methods - interface, - abstract class, operator overloading, - delegates, events, errors and exception - Threading.

### 1. Branching Statements

#### ➤ *if Statement* :

It executes its block only if the condition is true

#### **Syntax:**

```
if ( condition )
{
    //statement;
}
```

#### **Program**

```
public static void Main(string[] args)
{
    int number = 3;
    if (number < 10)
    {
        Console.WriteLine("{0} is less than 10", number);
    }

    Console.WriteLine("This statement is always executed.");
}
```

#### ➤ *if else Statement*

It executes if block if condition is true; otherwise it will execute the else block

#### **Syntax:**

```
if (condition )
{
    //statement;
}
```

```

else
{
//statement;
}

```

### Program

```

// Determine if a value is positive or negative.
using System;
class PosNeg {
static void Main() {
int i;
for(i=-3; i <= 3; i++) {
Console.Write("Testing " + i + ": ");
if(i < 0)
Console.WriteLine("negative");
else Console.WriteLine("positive");
}
}
}

```

#### Output

```

Testing -3: negative
Testing -2: negative
Testing -1: negative
Testing 0: positive
Testing 1: positive
Testing 2: positive
Testing 3: positive

```

In this example, if *i* is less than zero, then the target of the *if* is executed. Otherwise, the target of the *else* is executed. In no case are both executed.

#### ➤ *else if Statement:*

1. It checks the condition of both *if* and *else if* block and executes the respective block; otherwise it will execute the *else* block.

##### 1. **Syntax:**

```

if ( condition)
{
//statement;
}

```

```

        else if( condition)
        {
            //statement;
        }
        else
        {
            //statement;
        }
    }

public static void Main(string[] args)
    {
        int number = 12;

        if (number < 5)
        {
            Console.WriteLine("{0} is less than 5", number);
        }
        else if (number > 5)
        {
            Console.WriteLine("{0} is greater than 5", number);
        }
        else
        {
            Console.WriteLine("{0} is equal to 5");
        }
    }
}

```

The value of number is initialized to 12. The first test expression `number < 5` is false, so the control will move to the else if block. The test expression `number > 5` is true hence the block of code inside else if will be executed.

Similarly, we can change the value of `number` to alter the flow of execution.

### ➤ *Switch Statement*

The switch block consists of several cases which includes a default case too.

Each case has break statement to jump out of switch block on its execution.

The cases are matched and then executed provided the condition for cases in switch statement..

#### **Syntax:**

```

switch (variable)
{

```

```
    case 1:  
        //statement;  
        break;  
    case 2:  
        //statement;  
        break;  
    default:  
        //statement;  
        break;  
}
```

### Program

```
int day = 4;  
switch (day)  
{  
    case 1:  
        Console.WriteLine("Monday");  
        break;  
    case 2:  
        Console.WriteLine("Tuesday");  
        break;  
    case 3:  
        Console.WriteLine("Wednesday");  
        break;  
    case 4:  
        Console.WriteLine("Thursday");  
        break;  
    case 5:  
        Console.WriteLine("Friday");  
        break;  
    case 6:  
        Console.WriteLine("Saturday");  
        break;  
    case 7:  
        Console.WriteLine("Sunday");  
        break;  
}  
// Outputs "Thursday" (day 4)
```

## 1. Looping Statements

### ➤ while Statement

1. It executes the block until the condition fails.
2. It will execute its block only if the condition is true and continues to loop

### 3. Syntax:

```
while ( condition )  
{  
    //statement;  
}
```

### Program

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

Output:

```
0  
1  
2  
3  
4
```

### ➤ do-while Statement

1. It executes its statements and checks the condition.
2. It continues looping if condition is true; else it aborts.

### 3. Syntax:

```
do  
{  
    //statement;  
}  
while ( condition );
```

### Program

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



```
    }  
    while (i < 5);
```

**Output**

```
0  
1  
2  
3  
4
```

**➤ for Statement**

1. It executes its block until the condition fails.
2. **Syntax:**

```
for( initialization; condition; iteration)  
{  
    //statement;  
}
```

**Program**

```
class Program  
{  
    static void Main(string[] args)  
    {  
        for (int i = 0; i < 5; i++)  
        {  
            Console.WriteLine(i);  
        }  
    }  
}
```

**Output**

```
0  
1  
2  
3  
4
```

➤ **foreach Statement**

It executes the block for each values.

**Syntax:**

```
foreach (datatype values in variable)
```

```
{
```

```
//statement;
```

```
}
```

The **foreach loop** in C# executes a block of code on each element in an array or a collection of items. When executing foreach loop it traversing items in a collection or an array.

**Program**

```
string[] days = { "Sunday", "Monday", "TuesDay"};
```

```
foreach (string day in days)
```

```
{
```

```
    MessageBox.Show("The day is : " + day);
```

```
}
```

Output

Sunday

Monday

TuesDay

## 4. Jumping Statements

### ➤ *goto Statement*

It defines a region with a label; on goto execution the region is called and executed respectively.

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.

It can be used to transfer control from deeply nested loop or switch case label.

#### **Syntax:**

Label:

//statements

Program

```
public class GotoExample
{
    public static void Main(string[] args)
    {
        ineligible:
            Console.WriteLine("You are not eligible to vote!");
            Console.WriteLine("Enter your age:\n");
            int age = Convert.ToInt32(Console.ReadLine());
            if (age < 18){
                goto ineligible;
            }
            else
            {
                Console.WriteLine("You are eligible to vote!");
            }
        }
    }
}
```

Output

You are not eligible to vote!

Enter your age:

11

You are not eligible to vote!

Enter your age:

5

You are not eligible to vote!

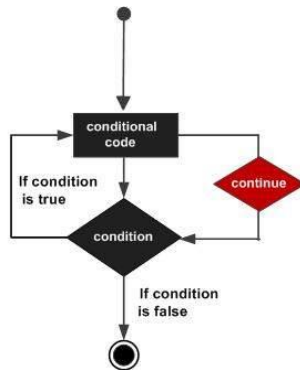
Enter your age:

26

You are eligible to vote!

➤ **continue Statement :**

The continue statement is used to execute the current block sequentially.



**Program**

```

static void Main(string[] args) {
    /* local variable definition */
    int a = 10;
    /* do loop execution */
    do {
        if (a == 15) {
            /* skip the iteration */
            a = a + 1;
            continue;
        }
        Console.WriteLine("value of a: {0}", a);
        a++;
    }
    while (a < 20);
    Console.ReadLine();
}
  
```

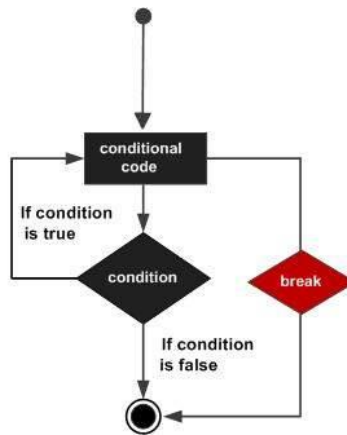
**Output**

```

value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
  
```

➤ **break Statement :**

The break Statement is used to jump out the current block after its execution.



**Program**

```

class Program {
    static void Main(string[] args) {
        /* local variable definition */
        int a = 10;

        /* while loop execution */
        while (a < 20) {
            Console.WriteLine("value of a: {0}", a);
            a++;

            if (a > 15) {
                /* terminate the loop using break statement */
                break;
            }
        }
        Console.ReadLine();
    }
}
  
```

**Output**

```

value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
  
```

## C# - Classes

A class definition starts with the keyword `class` followed by the class name; and the class body enclosed by a pair of curly braces.

```
<access specifier> class class_name {
    // member variables
    <access specifier> <data type> variable1;
    <access specifier> <data type> variable2;
    ...
    <access specifier> <data type> variableN;
    // member methods
    <access specifier> <return type> method1(parameter_list) {
        // method body
    }
    <access specifier> <return type> method2(parameter_list) {
        // method body
    }
    ...
    <access specifier> <return type> methodN(parameter_list) {
        // method body
    }
}
```

- Access specifiers specify the access rules for the members as well as the class itself. If not mentioned, then the default access specifier for a class type is *internal*. Default access for the members is *private*.
- Data type specifies the type of variable, and return type specifies the data type of the data the method returns, if any.
- To access the class members, you use the dot (.) operator.
- The dot operator links the name of an object with the name of a member.

Program

```
using System;
namespace BoxApplication {
    class Box {
        public double length; // Length of a box
        public double breadth; // Breadth of a box
        public double height; // Height of a box
    }
    class Boxtester {
        static void Main(string[] args) {
```

```
Box Box1 = new Box(); // Declare Box1 of type Box
Box Box2 = new Box(); // Declare Box2 of type Box
double volume = 0.0; // Store the volume of a box here

// box 1 specification
Box1.height = 5.0;
Box1.length = 6.0;
Box1.breadth = 7.0;

// box 2 specification
Box2.height = 10.0;
Box2.length = 12.0;
Box2.breadth = 13.0;
// volume of box 1
volume = Box1.height * Box1.length * Box1.breadth;
Console.WriteLine("Volume of Box1 : {0}", volume);

// volume of box 2
volume = Box2.height * Box2.length * Box2.breadth;
Console.WriteLine("Volume of Box2 : {0}", volume);
Console.ReadKey();
}
}
}
```

#### Output

Volume of Box1 : 210

Volume of Box2 : 1560

## C# Constructors

- A class constructor is a special member function of a class that is executed whenever we create new objects of that class.
- A constructor has exactly the same name as that of class and it does not have any return type.
- There can be two types of constructors in C#.
  - Default constructor
  - Parameterized constructor

### C# Default Constructor

A constructor which has no argument is known as default constructor. It is invoked at the time of creating object.

using System;

```
public class Employee
{
    public Employee()
    {
        Console.WriteLine("Default Constructor Invoked");
    }
}

class TestEmployee{
    public static void Main(string[] args)
    {
        Employee e1 = new Employee();
        Employee e2 = new Employee();
    }
}
```

Output:

Default Constructor Invoked

Default Constructor Invoked

### C# Parameterized Constructor

A constructor which has parameters is called parameterized constructor. It is used to provide different values to distinct objects.

using System;

```
public class Employee
{
    public int id;
    public String name;
```



```
public float salary;
public Employee(int i, String n,float s)
{
    id = i;
    name = n;
    salary = s;
}
public void display()
{
    Console.WriteLine(id + " " + name+" "+salary);
}
}
class TestEmployee{
    public static void Main(string[] args)
    {
        Employee e1 = new Employee(101, "Sonoo", 890000f);
        Employee e2 = new Employee(102, "Mahesh", 490000f);
        e1.display();
        e2.display();

    }
}
```

Output:

101 Sonoo 890000

102 Mahesh 490000

## Inheritance

- When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class.
- This existing class is called the base class, and the new class is referred to as the derived class.
- The idea of inheritance implements the IS-A relationship. For example, mammal IS A animal, dog IS-A mammal hence dog IS-A animal as well, and so on.

### Base and derived class

A class can be derived from more than one class or interface, which means that it can inherit data and functions from multiple base classes or interfaces.

```
<access-specifier> class <base_class> {
    ...
}
```

```
class <derived_class> : <base_class> {
    ...
}
```

Program

```
using System;
```

```
namespace InheritanceApplication {
    class Shape {
        public void setWidth(int w) {
            width = w;
        }
        public void setHeight(int h) {
            height = h;
        }
        protected int width;
        protected int height;
    }
}
```

```
// Derived class
```

```
class Rectangle: Shape {
    public int getArea() {
        return (width * height);
    }
}
```

```
}  
class RectangleTester {  
    static void Main(string[] args) {  
        Rectangle Rect = new Rectangle();  
  
        Rect.setWidth(5);  
        Rect.setHeight(7);  
  
        // Print the area of the object.  
        Console.WriteLine("Total area: {0}", Rect.getArea());  
        Console.ReadKey();  
    }  
}
```

Output

Total area: 35

## Interface

C# does not support multiple inheritance. However, you can use interfaces to implement multiple inheritance.

```
using System;
namespace InheritanceApplication {
    class Shape {
        public void setWidth(int w) {
            width = w;
        }
        public void setHeight(int h) {
            height = h;
        }
        protected int width;
        protected int height;
    }

    // Base class PaintCost
    public interface PaintCost {
        int getCost(int area);
    }

    // Derived class
    class Rectangle : Shape, PaintCost {
        public int getArea() {
            return (width * height);
        }
        public int getCost(int area) {
            return area * 70;
        }
    }
    class RectangleTester {
        static void Main(string[] args) {
            Rectangle Rect = new Rectangle();
            int area;
            Rect.setWidth(5);
            Rect.setHeight(7);
            area = Rect.getArea();
        }
    }
}
```

```
// Print the area of the object.  
Console.WriteLine("Total area: {0}", Rect.getArea());  
Console.WriteLine("Total paint cost: ${0}" , Rect.getCost(area));  
Console.ReadKey();  
}  
}  
}
```

#### Output

Total area: 35

Total paint cost: \$2450

## Properties

- C# Properties doesn't have storage location. C# Properties are extension of fields and accessed like fields.
- The Properties have accessors that are used to set, get or compute their values.

### Usage of Properties

- C# Properties can be read-only or write-only.
- We can have logic while setting values in the C# Properties.
- We make fields of the class private, so that fields can't be accessed from outside the class directly. Now we are forced to use C# properties for setting or getting values.

### C# Properties Example

```
using System;
```

```
public class Employee
{
    private string name;

    public string Name
    {
        get
        {
            return name;
        }
        set
        {
            name = value;
        }
    }
}
```

```
class TestEmployee{
    public static void Main(string[] args)
    {
        Employee e1 = new Employee();
        e1.Name = "Sonoo Jaiswal";
        Console.WriteLine("Employee Name: " + e1.Name);
    }
}
```

### Output

Employee Name: Sonoo Jaiswal

## Indexer

- An indexer allows an object to be indexed such as an array.
- When you define an indexer for a class, this class behaves similar to a virtual array.
- You can then access the instance of this class using the array access operator ([ ]).

## Syntax

```

element-type this[int index] {
    // The get accessor.
    get {
        // return the value specified by index
    }

    // The set accessor.
    set {
        // set the value specified by index
    }
}

```

- Declaration of behavior of an indexer is to some extent similar to a property.
- Similar to the properties, you use get and set accessors for defining an indexer.
- However, properties return or set a specific data member, whereas indexers returns or sets a particular value from the object instance. In other words, it breaks the instance data into smaller parts and indexes each part, gets or sets each part.

Defining a property involves providing a property name. **Indexers are not defined with names, but with the ‘this’ keyword, which refers to the object instance.**

Program

```

using System;
namespace IndexerApplication {
    class IndexedNames {
        private string[] namelist = new string[size];
        static public int size = 10;
        public IndexedNames() {
            for (int i = 0; i < size; i++)
                namelist[i] = "N. A.";
        }
        public string this[int index] {
            get {
                string tmp;
                if( index >= 0 && index <= size-1 ) {

```

```
        tmp = namelist[index];
    } else {
        tmp = "";
    }
    return ( tmp );
}
set {
    if( index >= 0 && index <= size-1 ) {
        namelist[index] = value;
    }
}
}
static void Main(string[] args) {
    IndexedNames names = new IndexedNames();
    names[0] = "Zara";
    names[1] = "Riz";
    names[2] = "Nuha";
    names[3] = "Asif";
    names[4] = "Davinder";
    names[5] = "Sunil";
    names[6] = "Rubic";

    for ( int i = 0; i < IndexedNames.size; i++ ) {
        Console.WriteLine(names[i]);
    }
    Console.ReadKey();
}
}
```

**Output**

Zara

Riz

Nuha

Asif

Davinder

Sunil

Rubic



N. A.

N. A.

N. A.

## Index overloading

- Indexers can be overloaded.
- Indexers can also be declared with multiple parameters and each parameter may be a different type.
- It is not necessary that the indexes have to be integers. C# allows indexes to be of other types, for example, a string.

using System;

```
namespace IndexerApplication {
    class IndexedNames {
        private string[] namelist = new string[size];
        static public int size = 10;
        public IndexedNames() {
            for (int i = 0; i < size; i++) {
                namelist[i] = "N. A.";
            }
        }
        public string this[int index] {
            get {
                string tmp;

                if( index >= 0 && index <= size-1 ) {
                    tmp = namelist[index];
                } else {
                    tmp = "";
                }
                return ( tmp );
            }
            set {
                if( index >= 0 && index <= size-1 ) {
                    namelist[index] = value;
                }
            }
        }
        public int this[string name] {
            get {
                int index = 0;
                while(index < size) {
                    if (namelist[index] == name) {
                        return index;
                    }
                }
            }
        }
    }
}
```

```

    }
    index++;
}
return index;
}
}
static void Main(string[] args) {
    IndexedNames names = new IndexedNames();
    names[0] = "Zara";
    names[1] = "Riz";
    names[2] = "Nuha";
    names[3] = "Asif";
    names[4] = "Davinder";
    names[5] = "Sunil";
    names[6] = "Rubic";
    //using the first indexer with int parameter
    for (int i = 0; i < IndexedNames.size; i++) {
        Console.WriteLine(names[i]);
    }
    //using the second indexer with the string parameter
    Console.WriteLine(names["Nuha"]);
    Console.ReadKey();
}
}
}

```

### Output

Zara

Riz

Nuha

Asif

Davinder

Sunil

Rubic

N. A.

N. A.

N. A.

2

## Polymorphism

- The word polymorphism means having many forms.
- In object-oriented programming paradigm, polymorphism is often expressed as 'one interface, multiple functions'.
- Polymorphism can be static or dynamic.
- In static polymorphism, the response to a function is determined at the compile time. In dynamic polymorphism, it is decided at run-time.

## Static Polymorphism

The mechanism of linking a function with an object during compile time is called early binding. It is also called static binding. C# provides two techniques to implement static polymorphism. They are –

- Function overloading
- Operator overloading

## Function Overloading

- You can have multiple definitions for the same function name in the same scope.
- The definition of the function must differ from each other by the types and/or the number of arguments in the argument list.
- You cannot overload function declarations that differ only by return type.

The following example shows using function **print()** to print different data types –

using System;

```
namespace PolymorphismApplication {
    class Printdata {
        void print(int i) {
            Console.WriteLine("Printing int: {0}", i);
        }
        void print(double f) {
            Console.WriteLine("Printing float: {0}", f);
        }
        void print(string s) {
            Console.WriteLine("Printing string: {0}", s);
        }
    }
    static void Main(string[] args) {
        Printdata p = new Printdata();
        // Call print to print integer
        p.print(5);
        // Call print to print float
        p.print(500.263);
    }
}
```

```

// Call print to print string
p.print("Hello C++");
Console.ReadKey();
}
}
}

```

#### Output

```

Printing int: 5
Printing float: 500.263
Printing string: Hello C++

```

### Dynamic Polymorphisms

- C# allows you to create abstract classes that are used to provide partial class implementation of an interface.
- Implementation is completed when a derived class inherits from it. Abstract classes contain abstract methods, which are implemented by the derived class.
- The derived classes have more specialized functionality.

Here are the rules about abstract classes –

- You cannot create an instance of an abstract class
- You cannot declare an abstract method outside an abstract class
- When a class is declared sealed, it cannot be inherited, abstract classes cannot be declared sealed.

The following program demonstrates an abstract class –

```

using System;
namespace PolymorphismApplication {
    abstract class Shape {
        public abstract int area();
    }

    class Rectangle: Shape {
        private int length;
        private int width;

        public Rectangle( int a = 0, int b = 0) {
            length = a;
            width = b;
        }
    }
}

```

```

public override int area () {
    Console.WriteLine("Rectangle class area :");
    return (width * length);
}
}
class RectangleTester {
    static void Main(string[] args) {
        Rectangle r = new Rectangle(10, 7);
        double a = r.area();
        Console.WriteLine("Area: {0}",a);
        Console.ReadKey();
    }
}
}

```

Output

```

    Rectangle class area :
    Area: 70

```

When you have a function defined in a class that you want to be implemented in an inherited class(es), you use **virtual** functions. The virtual functions could be implemented differently in different inherited class and the call to these functions will be decided at runtime.

Dynamic polymorphism is implemented by **abstract classes** and **virtual functions**.

The following program demonstrates this –

```

using System;
namespace PolymorphismApplication {
    class Shape {
        protected int width, height;

        public Shape( int a = 0, int b = 0) {
            width = a;
            height = b;
        }
        public virtual int area() {
            Console.WriteLine("Parent class area :");
            return 0;
        }
    }
}

```

```
class Rectangle: Shape {
    public Rectangle( int a = 0, int b = 0): base(a, b) {

    }
    public override int area () {
        Console.WriteLine("Rectangle class area :");
        return (width * height);
    }
}
class Triangle: Shape {
    public Triangle(int a = 0, int b = 0): base(a, b) {
    }
    public override int area() {
        Console.WriteLine("Triangle class area :");
        return (width * height / 2);
    }
}
class Caller {
    public void CallArea(Shape sh) {
        int a;
        a = sh.area();
        Console.WriteLine("Area: {0}", a);
    }
}
class Tester {
    static void Main(string[] args) {
        Caller c = new Caller();
        Rectangle r = new Rectangle(10, 7);
        Triangle t = new Triangle(10, 5);

        c.CallArea(r);
        c.CallArea(t);
        Console.ReadKey();
    }
}
}
```

**Output**

Rectangle class area:

Area: 70

Triangle class area:

Area: 25



## Operator Overloading

- You can redefine or overload most of the built-in operators available in C#.
- Thus a programmer can use operators with user-defined types as well.
- Overloaded operators are functions with special names the keyword operator followed by the symbol for the operator being defined. similar to any other function, an overloaded operator has a return type and a parameter list.

For example, go through the following function –

```
public static Box operator+ (Box b, Box c) {
    Box box = new Box();
    box.length = b.length + c.length;
    box.breadth = b.breadth + c.breadth;
    box.height = b.height + c.height;
    return box;
}
```

The above function implements the addition operator (+) for a user-defined class Box. It adds the attributes of two Box objects and returns the resultant Box object.

Implementing the Operator Overloading

The following program shows the complete implementation –

```
using System;
namespace OperatorOvlApplication {
    class Box {
        private double length; // Length of a box
        private double breadth; // Breadth of a box
        private double height; // Height of a box

        public double getVolume() {
            return length * breadth * height;
        }
        public void setLength( double len ) {
            length = len;
        }
        public void setBreadth( double bre ) {
            breadth = bre;
        }
        public void setHeight( double hei ) {
            height = hei;
        }
    }
}
```

```
}  
// Overload + operator to add two Box objects.  
public static Box operator+ (Box b, Box c) {  
    Box box = new Box();  
    box.length = b.length + c.length;  
    box.breadth = b.breadth + c.breadth;  
    box.height = b.height + c.height;  
    return box;  
}  
}  
class Tester {  
    static void Main(string[] args) {  
        Box Box1 = new Box(); // Declare Box1 of type Box  
        Box Box2 = new Box(); // Declare Box2 of type Box  
        Box Box3 = new Box(); // Declare Box3 of type Box  
        double volume = 0.0; // Store the volume of a box here  
  
        // box 1 specification  
        Box1.setLength(6.0);  
        Box1.setBreadth(7.0);  
        Box1.setHeight(5.0);  
  
        // box 2 specification  
        Box2.setLength(12.0);  
        Box2.setBreadth(13.0);  
        Box2.setHeight(10.0);  
  
        // volume of box 1  
        volume = Box1.getVolume();  
        Console.WriteLine("Volume of Box1 : {0}", volume);  
  
        // volume of box 2  
        volume = Box2.getVolume();  
        Console.WriteLine("Volume of Box2 : {0}", volume);  
  
        // Add two object as follows:
```

```
Box3 = Box1 + Box2;  
// volume of box 3  
volume = Box3.getVolume();  
Console.WriteLine("Volume of Box3 : {0}", volume);  
Console.ReadKey();  
}  
}  
}
```

When the above code is compiled and executed, it produces the following result –

Volume of Box1 : 210

Volume of Box2 : 1560

Volume of Box3 : 5400

## Sealed Classes

- Sealed classes are used to restrict the inheritance feature of object oriented programming. Once a class is defined as a sealed class, this class cannot be inherited.
- In C#, the sealed modifier is used to declare a class as sealed. In Visual Basic .NET, NotInheritable keyword serves the purpose of sealed. If a class is derived from a sealed class, compiler throws an error.
- If you have ever noticed, structs are sealed. You cannot derive a class from a struct. The following class definition defines a sealed class in C#:

```
// Sealed class
```

```
sealed class SealedClass
```

```
{
}
```

In the following code, I create a sealed class **SealedClass** and use it from Class1. If you run this code, it will work just fine. But if you try to derive a class from the SealedClass, you will get an error.

```
using System;
```

```
class Class1
```

```
{
```

```
    static void Main(string[] args)
```

```
    {
```

```
        SealedClass sealedCls = new SealedClass();
```

```
        int total = sealedCls.Add(4, 5);
```

```
        Console.WriteLine("Total = " + total.ToString());
```

```
    }
```

```
}
```

```
// Sealed class
```

```
sealed class SealedClass
```

```
{
```

```
    public int Add(int x, int y)
```

```
    {
```

```
        return x + y;
```

```
    }
```

```
}
```

### Why Sealed Classes?

- The main purpose of a sealed class is to take away the inheritance feature from the class users so they cannot derive a class from it.
- One of the best usage of sealed classes is when you have a class with static members. For example, the **Pens** and **Brushes** classes of the **System.Drawing** namespace.
- The Pens class represents the pens with standard colors. This class has only static members. For example, Pens.Blue represents a pen with blue color.
- Similarly, the Brushes class represents standard brushes. The Brushes.Blue represents a brush with blue color.

So when you're designing a class library and want to **restrict your classes not to be derived by developers**, you may want to use sealed classes.

- A sealed class is completely opposite to an abstract class.
- This sealed class cannot contain abstract methods.
- It should be the bottom most class within the inheritance hierarchy.
- A sealed class can never be used as a base class.
- This sealed class is specially used to avoid further inheritance.
- The keyword sealed can be used with classes, instance methods, and properties.

### Sealed Methods in C#

- The method that is defined in a parent class, if that method cannot be overridden under a child class, we call it a sealed method.
- By default, every method is a sealed method because overriding is not possible unless the method is not declared as virtual in the parent class.
- If a method is declared as virtual in a class, any child class of it can have the rights to override that method.

For Example:

```
namespace SealedDemo
{
    class class1
    {
        public virtual void show() { }
    }
    class class2 : class1
    {
        public override void show() { }
    }
    class class3 : class2
    {
        public override void show() { }
    }
}
```

In the above case even if the first child is not overriding the method the second child can still override the method.

When a child class is overriding its parent class virtual methods the child class uses the sealed modifier in the method so that further overriding of the method will not be possible i.e. child classes cannot override the methods.

For example:

```
namespace SealedDemo
{
    class class1
    {
        public virtual void show() { }
    }
    class class2 : class1
    {

```

```
public sealed override void show() { }  
}  
class class3 : class2  
{  
//'class3.show()': cannot override inherited member 'class2.show()' because it is sealed  
public override void show() { } //Invalid  
}  
}
```

## Delegates

- In C#, delegate is a reference to the method. It works like function pointer in C and C++. But it is objected-oriented, secured and type-safe than function pointer.
- For static method, delegate encapsulates method only. But for instance method, it encapsulates method and instance both.
- The best use of delegate is to use as event.
- Provides a good way to encapsulate the methods.
- Delegates are the library class in System namespace.
- These are the type-safe pointer of any method.
- Delegates are mainly used in implementing the call-back methods and events.
- Delegates can be chained together as two or more methods can be called on a single event.
- It doesn't care about the class of the object that it references.
- Internally a delegate declaration defines a class which is the derived class of System.Delegate.

### Example

Let's see a simple example of delegate in C# which calls add() and mul() methods.

```
using System;
```

```
delegate int Calculator(int n); //declaring delegate
```

```
public class DelegateExample
{
    static int number = 100;
    public static int add(int n)
    {
        number = number + n;
        return number;
    }
    public static int mul(int n)
    {
        number = number * n;
        return number;
    }
    public static int getNumber()
    {
        return number;
    }
    public static void Main(string[] args)
```



```
{  
    Calculator c1 = new Calculator(add); //instantiating delegate  
    Calculator c2 = new Calculator(mul);  
    c1(20); //calling method using delegate  
    Console.WriteLine("After c1 delegate, Number is: " + getNumber());  
    c2(3);  
    Console.WriteLine("After c2 delegate, Number is: " + getNumber());  
  
}
```

Output:

After c1 delegate, Number is: 120

After c2 delegate, Number is: 360

## Events

- The Event is something special that is going to happen.
- example of Button control in Windows. Button performs multiple events such as click, mouseover, etc.
- The event is an encapsulated delegate. C# and .NET both support the events with the delegates. When the state of the application changes, events and delegates give the notification to the client application. Delegates and Events both are tightly coupled for dispatching the events, and event handling require the implementation of the delegates. The sending event class is known as the publisher, and the receiver class or handling the Event is known as a subscriber.

The key points about the events are as:

- In C#, event handler will take the two parameters as input and return the void.
- The first parameter of the Event is also known as the source, which will publish the object.
- The publisher will decide when we have to raise the Event, and the subscriber will determine what response we have to give.
- Event can contain many subscribers.
- Generally, we used the Event for the single user action like clicking on the button.
- If the Event includes the multiple subscribers, then synchronously event handler invoked.

## Declaration of the Event

### Syntax

```
public event EventHandler CellEvent;
```

## Steps for implementing the Event

For the declaration of the Event in the class, firstly, the event type of the delegate must be declared.

```
public delegate void CellEventHandler(object sender, EventArgs e);
```

## Declaration of the Event

```
public event CellEventHandler CellEvent;
```

## Invocation of the Event

```
if (CellEvent != null) CellEvent(this, e);
```

We can invoke the Event only from within the class where we declared the Event.

## Hooking up the Event

```
OurEventClass.OurEvent += new ChangedEventHandler(OurEventChanged);
```

## Detach the Event

```
OurEventClass.OurEvent -= new ChangedEventHandler(OurEventChanged);
```

- Delegates work as pointer to a function. It is a reference data type and it holds the reference of the method. System.Delegate class implicitly derived all the delegates.
- Delegate can be declared using the delegate keyword which is followed by the signature

## Syntax of Delegates

<access modifier> delegate <return type> <delegate\_name>(<parameters>)

## Example of Delegates

```
public delegate void PrintWord(int value);
```

- The above PrintWord delegate can be used to point any method which has the same return type and declared parameters with PrintWord.
- Here we will take an example that declares and uses the PrintWord delegates.

```
class Program1
```

```
{
    // declare delegate
    public delegate void PrintWord(int value);
    static void Main(string[] args)
    {
        // Print delegate points to PrintNum
        PrintWord printDel = PrintNum;
        // or
        // Print printDel = new Print(PrintNumber);
        printDel(100000);
        printDel(200);
        // Print delegate points to PrintMoney
        printDel = PrintMoney;
        printDel(10000);
        printDel(200);
    }
    public static void PrintNum(int num)
    {
        Console.WriteLine("Number: {0,-12:N0}", num);
    }
    public static void PrintMoney(int money)
    {
        Console.WriteLine("Money: {0:C}", money);
    }
}
```

Output:

Number: 100,000

Number: 200

Money: \$10,000.00

Money: \$200.00

In the above example, we declared the `PrintWord` delegates, which accepts the `int` type parameter and returns the `void`. In the `main()` method, we declare the `PrintWord` type method and assigned the `PrintNum` name method. Now we will invoke the `PrintWord` delegate, which in-turn invokes the `PrintNum` method. In the same way, if the `PrintWord` delegates variable is assigned to the `PrintMoney` method, then this will invoke the `PrintMoney` method.

Also, we can create the delegate object by using the `new` operator and specify the name of the method, as shown below:

```
PrintWord printDel = new PrintWord(PrintNum);
```

Delegates can be declared, as shown below:

```
public delegate void someEvent();
```

```
public organize event
```

## Exception Handling

- Exception Handling in C# is a process to handle runtime errors. We perform exception handling so that normal flow of the application can be maintained even after runtime errors.
- In C#, exception is an event or object which is thrown at runtime. All exceptions are derived from System.Exception class. It is a runtime error which can be handled. If we don't handle the exception, it prints exception message and terminates the program.

### Advantage

It maintains the normal flow of the application. In such case, rest of the code is executed even after exception.

### C# Exception Classes

All the exception classes in C# are derived from System.Exception class. Let's see the list of C# common exception classes.

Exception	Description
System.DivideByZeroException	handles the error generated by dividing a number with zero.
System.NullReferenceException	handles the error generated by referencing the null object.
System.InvalidCastException	handles the error generated by invalid typecasting.
System.IO.IOException	handles the Input Output errors.
System.FieldAccessException	handles the error generated by invalid private or protected field access.

### C# Exception Handling Keywords

In C#, we use 4 keywords to perform exception handling:

- try
- catch
- finally, and
- throw

### C# example without try/catch

```
using System;
public class ExExample
{
    public static void Main(string[] args)
    {
        int a = 10;
```

```

    int b = 0;
    int x = a/b;
    Console.WriteLine("Rest of the code");
}
}

```

Output:

Unhandled Exception: System.DivideByZeroException: Attempted to divide by zero.

### C# finally example if exception is handled

```

using System;
public class ExExample
{
    public static void Main(string[] args)
    {
        try
        {
            int a = 10;
            int b = 0;
            int x = a / b;
        }
        catch (Exception e) { Console.WriteLine(e); }
        finally { Console.WriteLine("Finally block is executed"); }
        Console.WriteLine("Rest of the code");
    }
}

```

Output:

System.DivideByZeroException: Attempted to divide by zero.

Finally block is executed

### C# User-Defined Exceptions

C# allows us to create user-defined or custom exception. It is used to make the meaningful exception. To do this, we need to inherit Exception class.

C# user-defined exception example

```

using System;
public class InvalidAgeException : Exception
{
    public InvalidAgeException(String message)
        : base(message)
    {
    }
}

```

```
public class TestUserDefinedException
{
    static void validate(int age)
    {
        if (age < 18)
        {
            throw new InvalidAgeException("Sorry, Age must be greater than 18");
        }
    }
    public static void Main(string[] args)
    {
        try
        {
            validate(12);
        }
        catch (InvalidAgeException e) { Console.WriteLine(e); }
        Console.WriteLine("Rest of the code");
    }
}
```

Output:

InvalidAgeException: Sorry, Age must be greater than 18

Rest of the code

## C# Checked and Unchecked

- C# provides checked and unchecked keyword to handle integral type exceptions. Checked and unchecked keywords specify checked context and unchecked context respectively.
- In checked context, arithmetic overflow raises an exception whereas, in an unchecked context, arithmetic overflow is ignored and result is truncated.

### C# Checked

The checked keyword is used to explicitly check overflow and conversion of integral type values at compile time.

Let's first see an example that does not use checked keyword.

### C# Checked Example without using checked

```
using System;
namespace CSharpProgram
{
    class Program
    {
        static void Main(string[] args)
        {
            int val = int.MaxValue;
            Console.WriteLine(val + 2);
        }
    }
}
```

Output:

-2147483647

See, the above program produces the wrong result and does not throw any overflow exception.

### C# Checked Example using checked

This program throws an exception and stops program execution.

```
using System;
namespace CSharpProgram
{
    class Program
    {
        static void Main(string[] args)
        {
            checked
            {
```



```
        int val = int.MaxValue;
        Console.WriteLine(val + 2);
    }
}
}
```

Output:

Unhandled Exception: System.OverflowException: Arithmetic operation resulted in an overflow.

### **C# Unchecked**

The Unchecked keyword ignores the integral type arithmetic exceptions. It does not check explicitly and produce result that may be truncated or wrong.

Example

```
using System;
namespace CSharpProgram
{
    class Program
    {
        static void Main(string[] args)
        {
            unchecked
            {
                int val = int.MaxValue;
                Console.WriteLine(val + 2);
            }
        }
    }
}
```

Output:

-2147483647

### C# SystemException class

- The SystemException is a predefined exception class in C#. It is used to handle system related exceptions.
- It works as base class for system exception namespace. It has various child classes like: ValidationException, ArgumentException, ArithmeticException, DataException, StackOverflowException etc.

It consists of rich constructors, properties and methods that we have tabled below.

### C# SystemException Signature

[SerializableAttribute]

[ComVisibleAttribute(**true**)]

**public class** SystemException : Exception

using System;

namespace CSharpProgram

```
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                int[] arr = new int[5];
                arr[10] = 25;
            }
            catch (SystemException e)
            {
                Console.WriteLine(e);
            }
        }
    }
}
```

Output:

System.IndexOutOfRangeException: Index was outside the bounds of the array.

## Multithreading

- A thread is defined as the execution path of a program. Each thread defines a unique flow of control.
- If your application involves complicated and time consuming operations, then it is often helpful to set different execution paths or threads, with each thread performing a particular job.
- Threads are lightweight processes. One common example of use of thread is implementation of concurrent programming by modern operating systems. Use of threads saves wastage of CPU cycle and increase efficiency of an application.

So far we wrote the programs where a single thread runs as a single process which is the running instance of the application. However, this way the application can perform one job at a time. To make it execute more than one task at a time, it could be divided into smaller threads.

## Thread Life Cycle

The life cycle of a thread starts when an object of the **System.Threading.Thread** class is created and ends when the thread is terminated or completes execution.

Following are the various states in the life cycle of a thread –

The Unstarted State – It is the situation when the instance of the thread is created but the Start method is not called.

The Ready State – It is the situation when the thread is ready to run and waiting CPU cycle.

The Not Runnable State – A thread is not executable, when Sleep method has been called

Wait method has been called

Blocked by I/O operations

The Dead State – It is the situation when the thread completes execution or is aborted.

The Main Thread

In C#, the **System.Threading.Thread** class is used for working with threads. It allows creating and accessing individual threads in a multithreaded application. The first thread to be executed in a process is called the main thread.

When a C# program starts execution, the main thread is automatically created. The threads created using the Thread class are called the child threads of the main thread. You can access a thread using the **CurrentThread** property of the Thread class.

The following program demonstrates main thread execution –

```
using System;
using System.Threading;
namespace MultithreadingApplication {
    class MainThreadProgram {
        static void Main(string[] args) {
```

```

Thread th = Thread.CurrentThread;
th.Name = "MainThread";
Console.WriteLine("This is {0}", th.Name);
Console.ReadKey();
}
}
}

```

When the above code is compiled and executed, it produces the following result –

```

    This is MainThread

```

### **Complete program for multithreading**

```

using System;
using System.Threading;
namespace MultithreadingApplication {
class ThreadCreationProgram {
public static void CallToChildThread() {
try {
Console.WriteLine("Child thread starts");
// do some work, like counting to 10
for (int counter = 0; counter <= 10; counter++) {
Thread.Sleep(500);
Console.WriteLine(counter);
}
Console.WriteLine("Child Thread Completed");
} catch (ThreadAbortException e) {
Console.WriteLine("Thread Abort Exception");
} finally {
Console.WriteLine("Couldn't catch the Thread Exception");
}
}
static void Main(string[] args) {
ThreadStart childref = new ThreadStart(CallToChildThread);
Console.WriteLine("In Main: Creating the Child thread");
Thread childThread = new Thread(childref);
childThread.Start();
//stop the main thread for some time
Thread.Sleep(2000);

//now abort the child

```

```

        Console.WriteLine("In Main: Aborting the Child thread");
        childThread.Abort();
        Console.ReadKey();
    }
}
}

```

### Output

```

In Main: Creating the Child thread
Child thread starts
0
1
2
In Main: Aborting the Child thread
Thread Abort Exception
Couldn't catch the Thread Exception

```

### Creating Threads

Threads are created by extending the Thread class. The extended Thread class then calls the Start() method to begin the child thread execution.

### Managing Threads

The Thread class provides various methods for managing threads.

The following example demonstrates the use of the sleep() method for making a thread pause for a specific period of time.

### Destroying Threads

The Abort() method is used for destroying threads.

The runtime aborts the thread by throwing a ThreadAbortException. This exception cannot be caught, the control is sent to the *finally* block, if any.

### Thread Life Cycle - states

- Unstarted State
  - When the instance of Thread class is created, it is in unstarted state by default.
- Runnable State
  - When start() method on the thread is called, it is in runnable or ready to run state.
- Running State
  - Only one thread within a process can be executed at a time. At the time of execution, thread is in running state.
- Not Runnable State
  - The thread is in not runnable state, if sleep() or wait() method is called on the thread, or input/output operation is blocked.
- Dead State
  - After completing the task, thread enters into dead or terminated state.

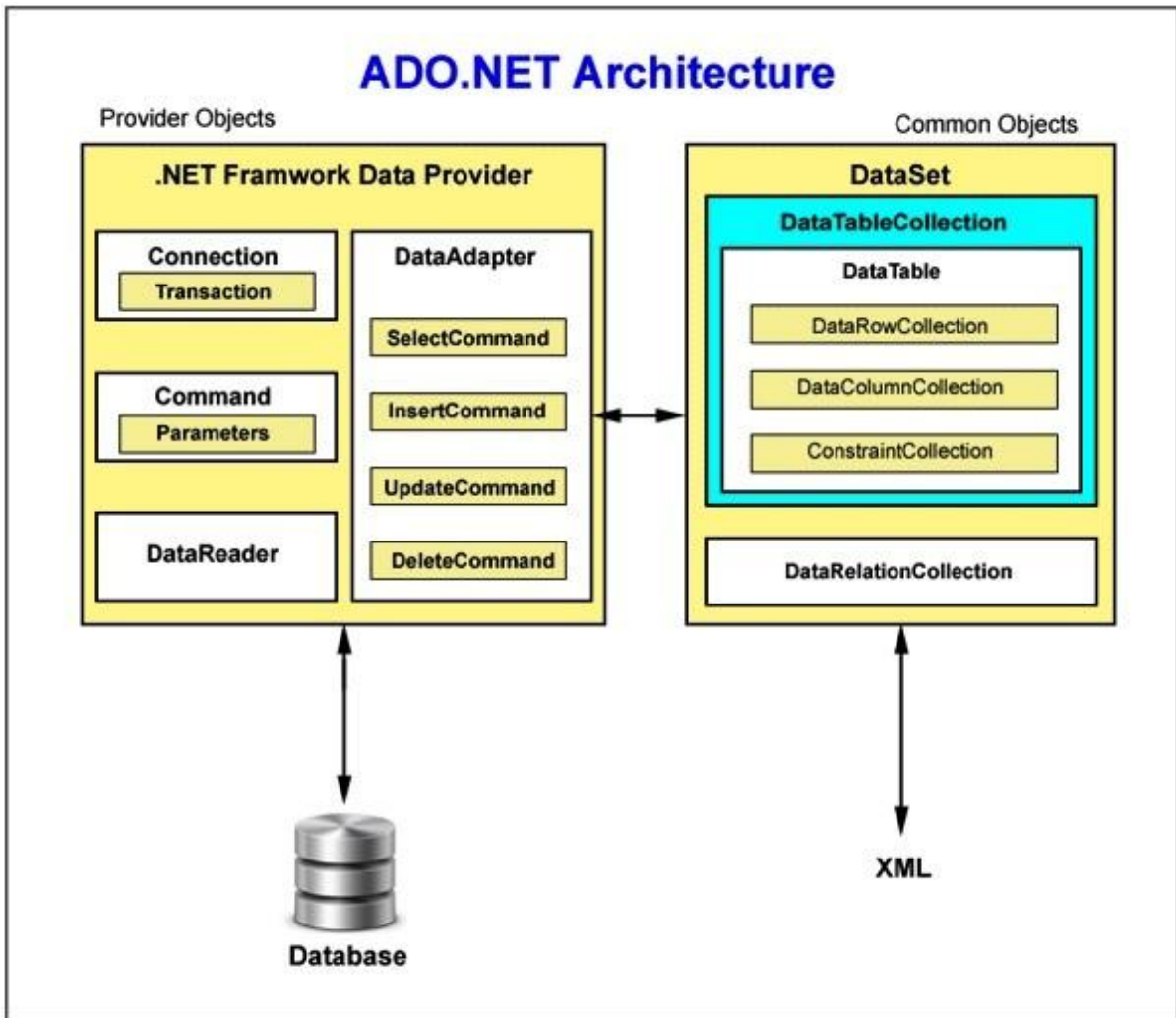
**Unit – V - ADO.NET** - Overview of ADO.NET - ADO.NET data access – Connected and Disconnected Database, Create Connection using ADO.NET Object Model, Connection Class, Command Class - Data binding – Data list – Data grid – Repeater – Files, Streams and Email – Using XML.

### **Overview of ADO.NET**

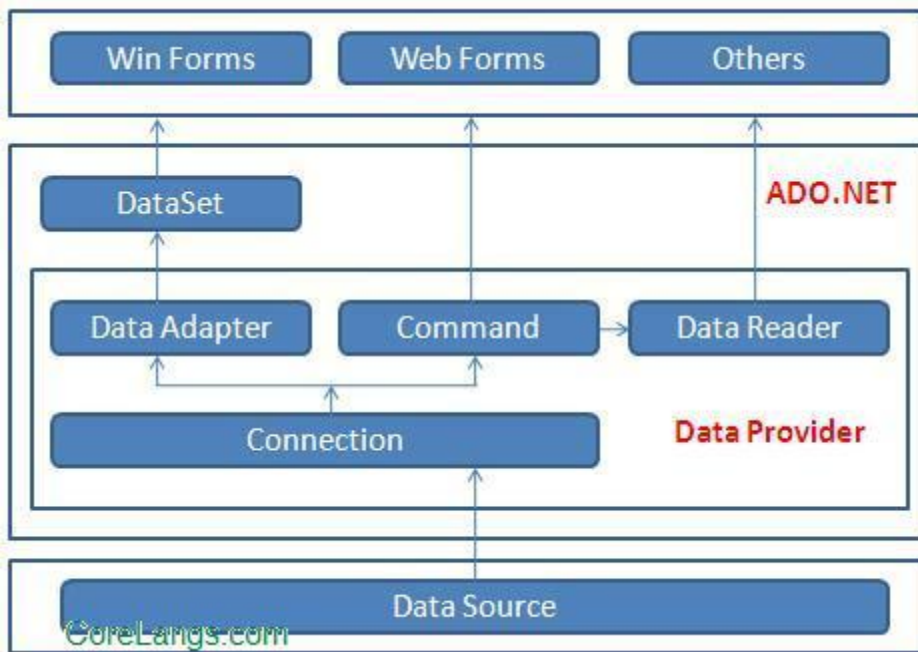
#### **What is ADO.NET?**

- ADO.NET (ActiveX Data Objects) is a module of .Net Framework which is used to establish connection between application and data sources. Data sources can be such as SQL Server and XML. ADO.NET consists of classes that can be used to connect, retrieve, insert and delete data.
- **System.Data** namespace is the core of ADO.NET and contains classes used by all data providers.
- All the **ADO.NET classes** are located in **System.Data.dll** and integrated with XML classes located in **System.Xml.dll**.
- ADO.NET has two main components that are used for accessing and manipulating data are the .NET Framework **Data Provider** and the **DataSet**.
- These are the components that are designed for data manipulation and fast access to data. It provides various objects such as **Connection, Command, DataReader and DataAdapter** that are used to perform database operations.

*ADO.NET is a rich set of classes, interfaces, structures and enumerated types that manage data access from various types of data stores.*



### ADO.NET ARCHITECTURE



### ADO.NET Architecture

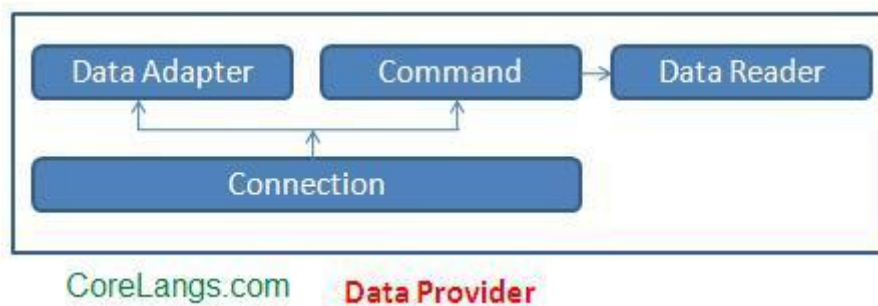
## Data providers

Different databases will have different storage formats. Different languages will support different data formats, this language formats will not be understandable to databases, this requires a translator between a language application and database. *This translator is called driver or provider.*

**Driver or provider is a software component, this act like mediator between application and database.** They are the

- Microsoft SQL Server Data Provider,
- OLEDB Data Provider and
- ODBC Data Provider .

SQL Server uses the SqlConnection object , OLEDB uses the OleDbConnection Object and ODBC uses OdbcConnection Object respectively.

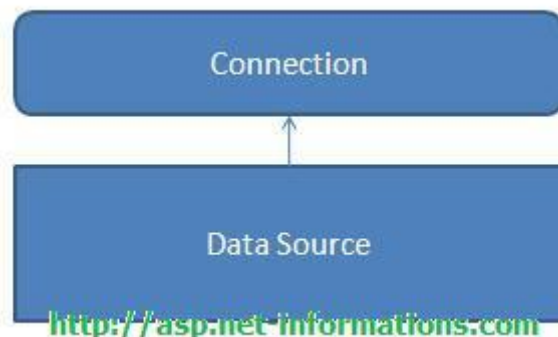


A data provider contains Connection, Command, DataAdapter, and DataReader objects. These four objects provide the functionality of Data Providers in the ADO.NET.



## CONNECTION Object

- The Connection Object provides physical connection and interaction with the Data Source.
- The .Net Framework provides two types of connection classes:
  - The **SqlConnection** object, that is designed specially to connect to Microsoft SQL Server and the **OleDbConnection** object, that is designed to provide connection to a wide range of databases, such as Microsoft Access and Oracle.
- A Connection object helps to identify the database server name, user name and password to connect to the database through **a connection string**.
- How to use the SqlConnection object:
  - Instantiate the SqlConnection class.
  - Open connection.
  - Pass the connection to ADO.NET objects.
  - Perform the database operations with ADO.NET object.
  - Close the connection.
- The connection string is different for each of the various data providers available in .NET.



No.	Connection String Parameter Name	Description
1	Data Source	Identify the server. Could be local machine, machine domain name, or IP Address.
2	Initial Catalog	Data base name.
3	Integrated Security	Set to SSIP to make connection with user's window login.
4	User ID	Name of user configured in SQL Server.
5	Password	Password matching SQL Server User ID

Code:

```
1. SqlConnection con;
2. con = new SqlConnection("Server=Krushna;Database=Anagha;Uid=sa;Pwd=sa");
```

## ASP.NET Sql Server Connection

The SqlConnection Object is Handling the part of physical communication between the ASP.NET application and the SQL Server Database. An instance of the SqlConnection class in ASP.NET is supported the Data Provider for SQL Server Database.

```
string connectionString=ConfigurationManager.ConnectionStrings["SQLDbConnection"].ToString();
```

When the connection is established, SQL Commands will execute with the help of the Command Object and retrieve or manipulate the data in the database. Once the Database activities is over , Connection should be closed and released with the Data Source resources. The Close() method in SqlConnection Class is used to close the Database Connection.

The following ASP.NET program connect to a database server and display the message in the Label control.

### Default.aspx program

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Button ID="Button1" runat="server" Text="Button" onclick="Button1_Click" />
        </div>
        <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
    </form>
</body>
</html>
```

### Default.aspx.cs

```
using System;
using System.Data ;
using System.Data.SqlClient ;
using System.Configuration;

public partial class _Default : System.Web.UI.Page
{
    protected void Button1_Click(object sender, EventArgs e)
    {
        string connectionString = ConfigurationManager.ConnectionStrings["SQLDbConnection"].ToString();
        SqlConnection connection = new SqlConnection(connectionString);
        connection.Open();
        Label1.Text = "Connected to Database Server !!";
        connection.Close();
    }
}
```

## COMMAND Object

- A Command object executes SQL statements and stored procedures on the database. These SQL statements can be SELECT, INSERT, UPDATE, or DELETE.
- It uses a connection object to perform these actions on the database.
- A Command object is used to perform various types of operations, like SELECT, INSERT, UPDATE, or DELETE on the database.
- SELECT

```
1. cmd =new SqlCommand("select * from Employee", con);
```

- The Command Object requires an instance of an Connection Object (con) for executing the SQL statements.
- INSERT

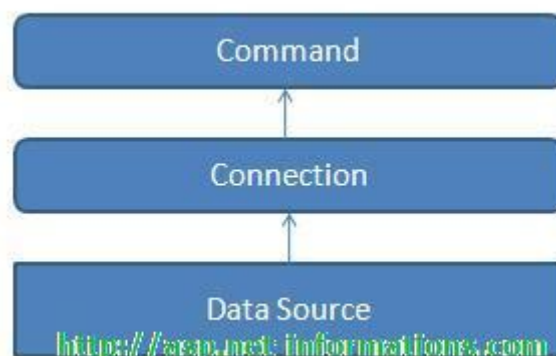
```
1. cmd = new SqlCommand("INSERT INTO Employee(Emp_ID, Emp_Name)VALUES ('" + aa + "', '" + bb + "')", con);
```

- UPDATE

```
1. SqlCommand cmd =new SqlCommand("UPDATE Employee SET Emp_ID ='" + aa + "', Emp_Name ='" + bb + "' WHERE Emp_ID = '" + aa + "'", con);
```

- DELETE

```
1. cmd =new SqlCommand("DELETE FROM Employee where Emp_ID='" + aa + "'", con);
```



- A Command object exposes several execute methods like:
  - **ExecuteScalar()**  
ExecuteScalar method uses to retrieve a single value from a database. Executes the query, and returns the first column of the first row in the result set returned by the query. Extra columns or rows are ignored.

```
int result = Convert.ToInt32(cmd.ExecuteScalar());
```

- It is very useful to use with aggregate functions like Count(\*) or Sum() etc.
- The following ASP.NET program find number of rows in the author table using ExecuteScalar method.

### Default.aspx

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Button ID="Button1" runat="server" Text="Button" onclick="Button1_Click" />
        </div>
        <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
    </form>
</body>
</html>
```

### Default.aspx.cs

```
using System;
using System.Data ;
using System.Data.SqlClient ;
using System.Configuration;

public partial class _Default : System.Web.UI.Page
{
    protected void Button1_Click(object sender, EventArgs e)
    {
        string connectionString = ConfigurationManager.ConnectionStrings["SQLDbConnection"].ToString();
        SqlConnection connection = new SqlConnection(connectionString);
        string sql = "select count(*) from authors";
        try
        {
            connection.Open();
            SqlCommand cmd = new SqlCommand(sql, connection);
            int result = Convert.ToInt32(cmd.ExecuteScalar());
            connection.Close();
            Label1.Text = "Number of rows in author table - " + result;
        }
        catch (Exception ex)
        {
            Label1.Text = "Error in ExecuteScalar " + ex.ToString();
        }
    }
}
```

- **ExecuteReader()**

Display all columns and all rows at client-side environment.

The ExecuteReader() in SqlCommand Object sends the SQL statements to the Connection Object and *populate a SqlDataReader Object* based on the SQL statement.

```
SqlDataReader reader = cmd.ExecuteReader();
```

### Default.aspx

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Button ID="Button1" runat="server" Text="Button" onclick="Button1_Click" />
            <br />
            <asp:ListBox ID="ListBox1" runat="server"></asp:ListBox>
            <br />
            <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
        </div>
    </form>
</body>
</html>
```

### Default.aspx.cs

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;

public partial class _Default : System.Web.UI.Page
{
    protected void Button1_Click(object sender, EventArgs e)
    {
        string connectionString = ConfigurationManager.ConnectionStrings["SQLDbConnection"].ToString();
        SqlConnection connection = new SqlConnection(connectionString);
        string sql = "select au_lname,au_fname from authors";
        try
        {
            connection.Open();
            SqlCommand cmd = new SqlCommand(sql, connection);
            SqlDataReader reader = cmd.ExecuteReader();
            while (reader.Read())
            {
                ListBox1.Items.Add(reader.GetValue(0) + " " + reader.GetValue(1));
            }
            connection.Close();
        }
        catch (Exception ex)
        {
            Label1.Text = "Error in ExecuteReader " + ex.ToString();
        }
    }
}
```

- **ExecuteNonQuery()**

*Something is done by the database but nothing is returned by the database.*

The ExecuteNonQuery() performs **Data Definition tasks** as well as **Data Manipulation tasks** also. The Data Definition tasks like creating Stored Procedures, Views etc. are performed by ExecuteNonQuery(). Also Data Manipulation tasks like Insert, Update , Delete etc. also perform by the ExecuteNonQuery() of SqlCommand Object. Although the ExecuteNonQuery **returns no rows, any output parameters or return values** mapped to parameters are populated with data.

### Default.aspx

```
using System;
using System.Data ;
using System.Data.SqlClient ;
using System.Configuration;

public partial class _Default : System.Web.UI.Page
{
    protected void Button1_Click(object sender, EventArgs e)
    {
        string connectionString = ConfigurationManager.ConnectionStrings["SQLDbConnection"].ToString();
        SqlConnection connection = new SqlConnection(connectionString);
        string sql = "insert into discounts values('New Discont',8042,1000,1000,5.25)";
        try
        {
            connection.Open();
            SqlCommand cmd = new SqlCommand(sql, connection);
            cmd.ExecuteNonQuery();
            connection.Close();
            Label1.Text = "Successfully Inserted !!";
        }
        catch (Exception ex)
        {
            Label1.Text = "Error inserting data" + ex.ToString();
        }
    }
}
```

## DataReader

- DataReader Object is a stream-based, forward-only, read-only retrieval of query results from the Data Sources, which do not update the data.
- A DataReader object is used to obtain the results of a SELECT statement from a command object.
- The DataReader cannot be created directly from code, they can be created only by calling the ExecuteReader() method of a Command Object.
- After creating an instance of the Command object, you have to create a DataReader by calling Command.ExecuteReader to retrieve rows from a data source.

```
1. dr = cmd.ExecuteReader();
2. DataTable dt = new DataTable();
3. dt.Load(dr);
```

```
SqlDataReader reader = cmd.ExecuteReader();
```

- You should always call the Close method when you have finished using the DataReader object.
- It is used in Connected architecture.
- Provide better performance.
- DataReader Object has Read-only access.
- DataReader Object supports a single table based on a single SQL query of one database
- DataReader Object is Bind to a single control.
- DataReader Object has faster access to data.
- DataReader cannot modify data.

The following ASP.NET program execute sql statement and read the data using SqlDataReader.

### Default.aspx

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>Untitled Page</title>
</head>
<body>
  <form id="form1" runat="server">
    <div>
      <asp:Button ID="Button1" runat="server" Text="Button" onclick="Button1_Click" />
      <br />
      <asp:ListBox ID="ListBox1" runat="server"></asp:ListBox>
      <br />
      <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
    </div>
```

```
</form>
</body>
</html>
```

### **Default.aspx.cs**

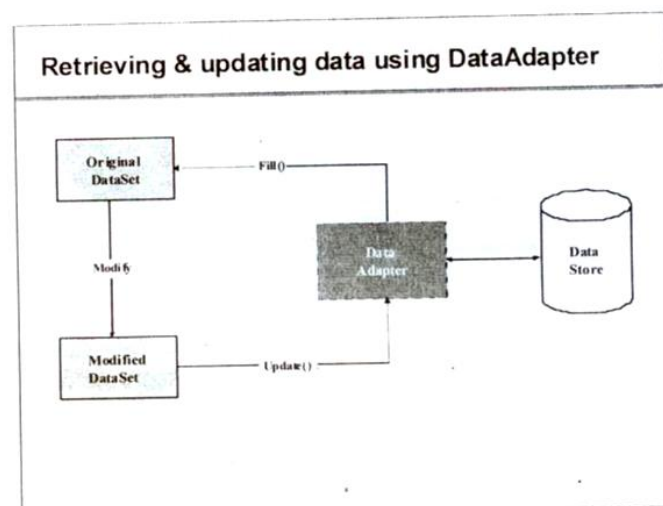
```
using System;
using System.Data ;
using System.Data.SqlClient ;
using System.Configuration;

public partial class _Default : System.Web.UI.Page
{
    protected void Button1_Click(object sender, EventArgs e)
    {
        string connectionString = ConfigurationManager.ConnectionStrings["SQLDbConnection"].ToString();
        SqlConnection connection = new SqlConnection(connectionString);
        string sql = "select pub_id, pub_name from publishers";
        try
        {
            connection.Open();
            SqlCommand cmd = new SqlCommand(sql, connection);
            SqlDataReader reader = cmd.ExecuteReader();
            while (reader.Read())
            {
                ListBox1.Items.Add(reader.GetValue(0) + " - " + reader.GetValue(1));
            }
            reader.Close();
            connection.Close();
        }
        catch (Exception ex)
        {
            Label1.Text = "Error in SqlDataReader " + ex.ToString();
        }
    }
}
```



## DataAdapter

- DataAdapter serves as a bridge between a DataSet and SQL Server for retrieving and saving data.
- A DataAdapter represents a set of data commands and a database connection **to fill the dataset and update a SQL Server database.**
- It maintains the data in a **DataSet object.**
- The user can read the data if required from the DataSet and write back the changes in a single batch to the database.
- Additionally, the Data Adapter contains a command object reference for **SELECT, INSERT, UPDATE, and DELETE operations on the data objects** and a data source.
- A Data Adapter mainly supports the following two methods:
  - **Fill ()**  
The Fill method populates a dataset or a data table object with data from the database.
  - **Update ()**  
The Update method commits the changes back to the database. It also analyzes the RowState of each record in the DataSet and calls the appropriate INSERT, UPDATE, and DELETE statements.
- A Data Adapter object is formed between a **disconnected ADO.NET object** and a data source.
- We can use SqlDataAdapter Object in combination with DataSet Object.



```

SqlDataAdapter adapter = new SqlDataAdapter(sql,connection );
adapter.Fill(ds);

```

1. SqlDataAdapter da=new SqlDataAdapter("Select \* from Employee", con);
2. da.Fill(ds, "Emp");

The following ASP.NET program shows a select operation using SqlDataAdapter.

### Default.aspx

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Button ID="Button1" runat="server" Text="Button" onclick="Button1_Click" />
            <br />
            <asp:ListBox ID="ListBox1" runat="server"></asp:ListBox>
            <br />
            <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
        </div>
    </form>
</body>
</html>
```

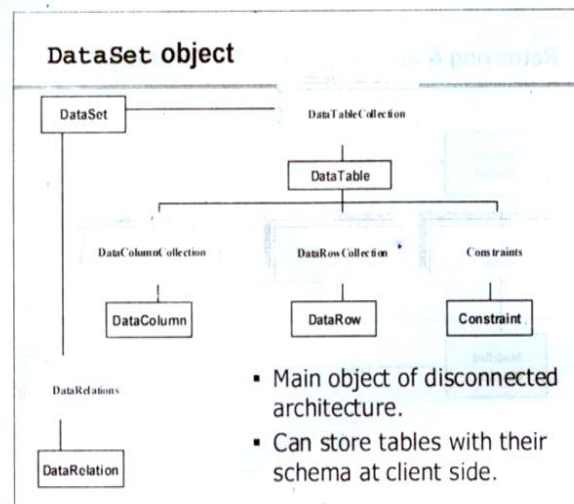
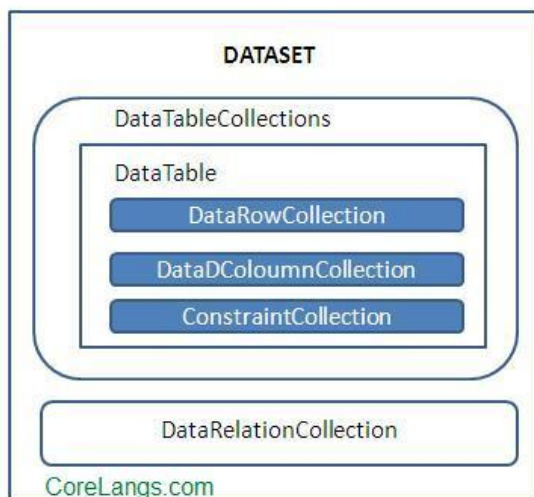
### Default.aspx.cs

```
using System;
using System.Data ;
using System.Data.SqlClient ;
using System.Configuration;

public partial class _Default : System.Web.UI.Page
{
    protected void Button1_Click(object sender, EventArgs e)
    {
        string connectionString = ConfigurationManager.ConnectionStrings["SQLDbConnection"].ToString();
        SqlConnection connection = new SqlConnection(connectionString);
        DataSet ds = new DataSet ();
        string sql = "select pub_name from publishers";
        try
        {
            connection.Open();
            SqlDataAdapter adapter = new SqlDataAdapter(sql,connection );
            adapter.Fill(ds);
            for (int i = 0; i < ds.Tables[0].Rows.Count -1; i++)
            {
                ListBox1.Items.Add(ds.Tables[0].Rows[i].ItemArray[0].ToString ());
            }
            connection.Close();
        }
        catch (Exception ex)
        {
            Label1.Text = "Error in execution " + ex.ToString();
        }
    }
}
```

## DataSet

- In the **disconnected scenario**, the data retrieved from the database is stored in a local buffer called **DataSet**.
- It is explicitly designed to access data from any data source. This class is defined in the **System.Data** namespace.
- A Data Set is a collection of **DataTable** and **DataRelations**. Each DataTable is a collection of **DataColumn**, **DataRow**s and **Constraints**.
- So it contains **rows, columns, primary keys, constraints, and relations** with other DataTable objects.
- DataSet consists of a collection of DataTable objects that you can relate to each other with DataRelation objects.
- The DataAdapter Object provides a bridge between the **DataSet** and the **Data Source**.



```

1. DataTable dt = new DataTable();
2. DataColumn col = new DataColumn();
3. Dt.columns.Add(col);
4. DataRow row = dt.newRow();

```

- It is used in a **disconnected architecture**.
- Provides **lower performance**.
- A DataSet object has **read/write access**.
- A DataSet object supports **multiple tables** from various databases.
- A DataSet object is bound to **multiple controls**.
- A DataSet object has **slower** access to data.
- A Dataset supports **integration with XML**.
- A DataSet communicates with the **Data Adapter** only.
- A DataSet can modify data.

**CommandBuilder Object**

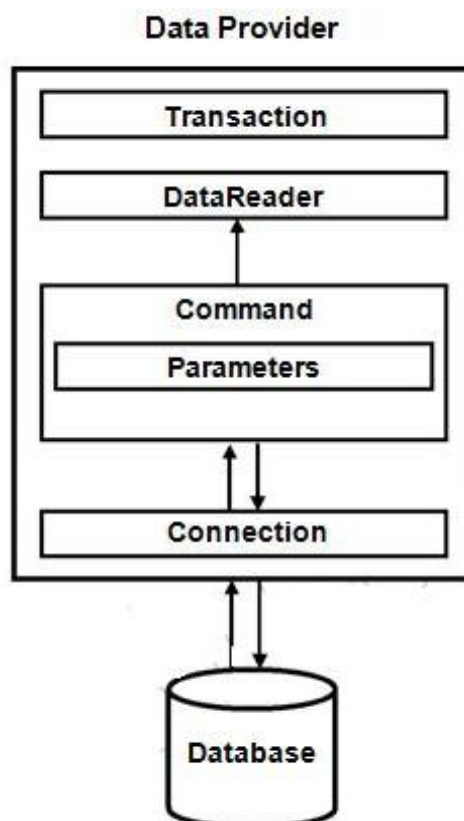
- By default DataAdapter contains only the select command and it doesn't contain insert, update and delete commands.
- To create insert, update and delete commands for the DataAdapter, CommandBuilder is used. It is used only to create these commands for the DataAdapter and has no other purpose.

### Differences Between DataReader and DataSet

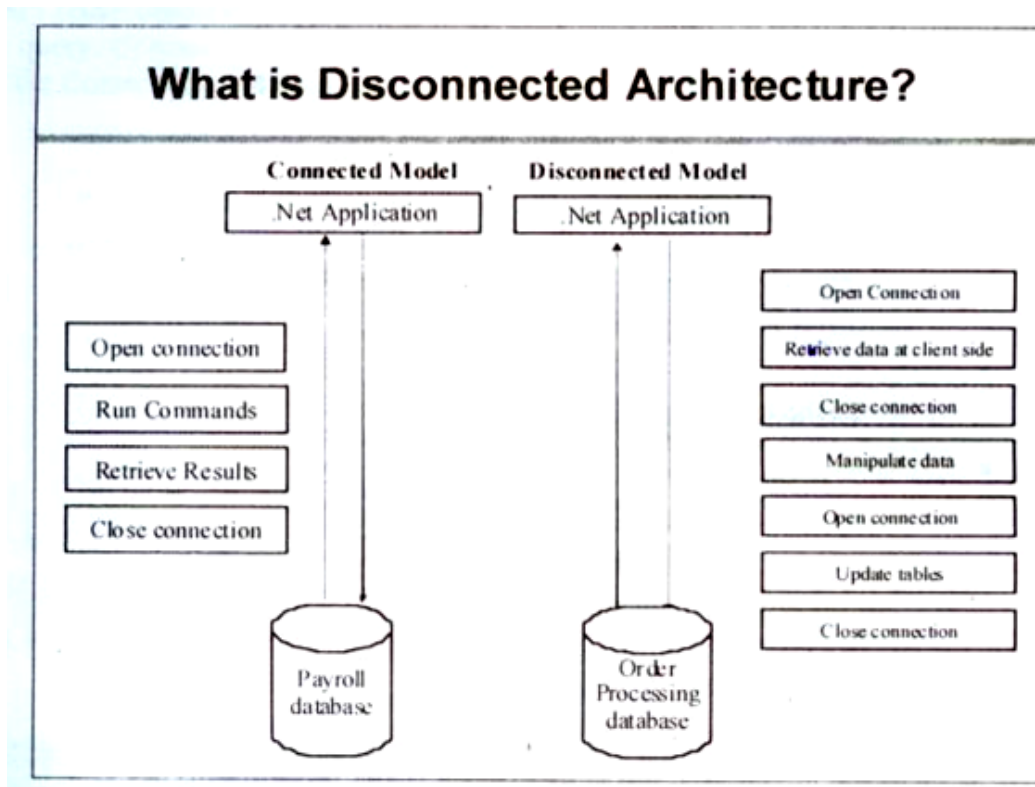
No.	Data Reader	DataSet
1	It is <b>connected object</b> and cannot provide access to data when connection to database was closed.	It is <b>disconnected object</b> and can provide access to data even when connection to database was closed
2	Provides <b>better</b> performance	Provides <b>lower</b> performance
3	DataReader object has <b>read-only</b> access	A DataSet object has <b>read/write</b> access
4	It can store data from <b>only one table.</b>	It can store data from <b>multiple tables.</b>
5	A DataReader object is bound to <b>a single control.</b> It can contain only one record at a time.	A DataSet object is bound to <b>multiple controls.</b> It can contain multiple records.
6	A DataReader object has <b>faster access</b> to data	A DataSet object has <b>slower access</b> to data
7	A DataReader object must be <b>manually coded</b>	A DataSet object is supported by <b>Visual Studio tools.</b>
8	We can't create <b>a relation</b> in a data reader	We can create <b>relations</b> in a dataset
9	Whereas a DataReader doesn't support data reader communicates with the <b>command object.</b>	A Dataset supports integration with XML. Dataset communicates with the <b>Data Adapter only.</b>
10	It is <b>read only</b> and it doesn't allow insert, update and delete on data.	It allows <b>insert, update and delete</b> on data
11	All the data of a DataReader will <b>be on server</b> and one record at a time is retrieved and stored in datareader when you call the Read() method of datareader.	All the data of a dataset will be <b>on client system.</b>

### Connected Architecture of ADO.NET

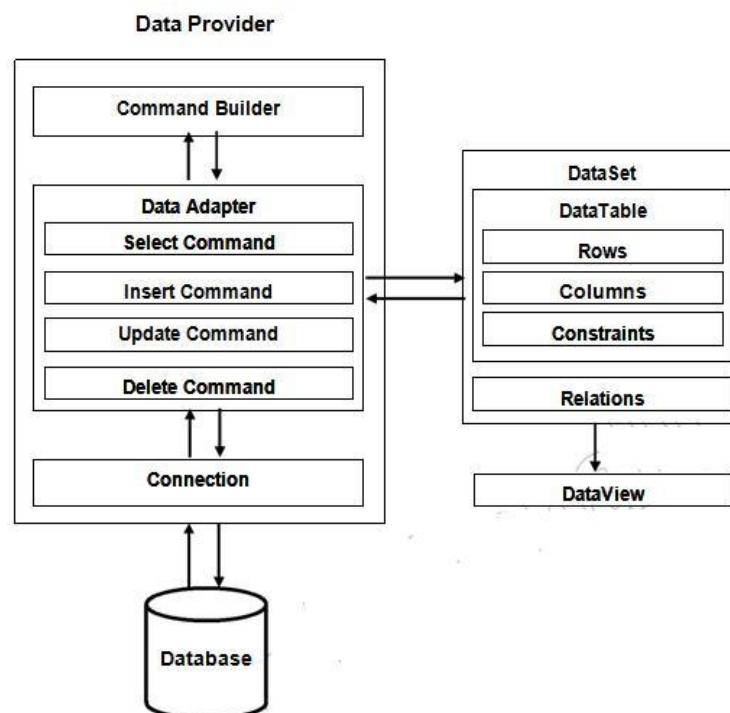
- In the connected architecture, **connection with a data source is kept open constantly** for data access as well as data manipulation operations.
- The ADO.NET Connected architecture considers mainly the following types of objects. Connected architecture was built on the classes connection, command, datareader and transaction.
  - SqlConnection con;
  - SqlCommand cmd;
  - SqlDataReader dr;
- **Connection** : in connected architecture also the purpose of connection is to just **establish a connection** to database and itself will not transfer any data.
- **DataReader** : DataReader is used to **store the data retrieved by command object** and make it available for .net application. Data in DataReader is read only and within the DataReader you can navigate only in forward direction and it also only one record at a time.
- To access one by one record from the DataReader, call Read() **method** of the DataReader whose return type is **bool**. When the next record was successfully read, the Read() method will return true and otherwise returns false



## Disconnected Architecture in ADO.NET



The architecture of ADO.net in which data retrieved from database can be accessed even when connection to database was closed is called as disconnected architecture. Disconnected architecture of ADO.net was built on classes connection, dataadapter, commandbuilder and dataset and dataview.



- The .NET application **does not always stay connected with the database**. The classes are designed in a way that they automatically open and close the connection. The data is stored **client-side** and is **updated** in the database whenever required.
- The ADO.NET Disconnected architecture considers primarily the following types of objects:
  - DataSet ds;
  - SqlDataAdapter da;
  - SqlConnection con;
  - SqlCommandBuilder bldr;
- **Connection** : Connection object is used to establish a connection to database and connection itself will not transfer any data.
- **DataAdapter** : DataAdapter is used to **transfer the data between database and dataset**. It has commands like **select, insert, update and delete**. Select command is used to retrieve data from database and insert, update and delete commands are used to send changes to the data in dataset to database. It needs a connection to transfer the data.
- **CommandBuilder** : By default dataadapter contains only the **select command** and **it doesn't contain insert, update and delete commands**. To create insert, update and delete commands for the dataadapter, commandbuilder is used. It is used only to create these commands for the dataadapter and has no other purpose.
- **DataSet** : Dataset is used to store the data retrieved from database by dataadapter and make it available for .net application.
- To fill data in to dataset **fill() method** of dataadapter is used and has the following syntax.

**Da.Fill(Ds,"TableName");**

- When fill method was called, dataadapter will open a connection to database, executes select command, stores the data retrieved by select command in to dataset and immediately closes the connection.
- As connection to database was closed, any changes to the data in dataset will not be directly sent to the database and will be made only in the dataset. To send changes made to data in dataset to the database, **Update()** method of the dataadapter is used that has the following syntax.

**Da.Update(Ds,"Tablename");**

- When Update method was called, dataadapter will again open the connection to database, executes insert, update and delete commands to send changes in dataset to database and immediately closes the connection. As connection is opened only when it is required and will be automatically closed when it was not required, this architecture is called disconnected architecture.
- A dataset can contain data in multiple tables.



## **Steps to Create a Database Using ADO.NET**

We should have the followings.

- .NET framework 4.5 or greater installed and ready to go.
- A text editor or visual studio.
- An ADO.NET Database Driver contained in products such as MySQL, PostgreSQL or RDM.

### **Steps to Creating your Application**

#### **Step 1 Open a command line prompt or visual studio**

Change to the directory in which you have installed the files for the sample.

#### **Step 2 Viewing your .cs file**

Using your text editor/visual studio, create the file “HelloWorldADO.NET.cs”.

#### **Step 3 Viewing your sample class**

The class can contain the same name as the .cs file containing the class. It should appear as follows:

```
Namespace HelloWorldApplication {
class HelloWorldADO.NET {
...
}
}
```

In this example everything is done within this class.

#### **Step 4 Examining the main method**

The main method is the entry point for your program. For this simple example, we are only using one .cs file. Therefore, the class will contain the main method as shown below. We will be accepting no arguments to this program.

```
static void main() {
...
}
```

#### **Step 5 Creating and initializing your Connection Object**

We have to initialize Connection object before you have access to any of the methods it contains.

Start a new try block for every object that you initialize. When you are done with the object, simply add a finally block that performs the corresponding close() method, and the outermost block will contain your catch block to handle all possible Exceptions.

RDM ADO.NET driver is being used so we have an RdmConnection object.

```
RdmConnection connection = new
RdmConnection("host=localhost;database=hello_worldADO");
try {
...
}
} catch (Exception exception) {
```

```

...
} finally {
Conn.close();
}

```

### Step 6 Creating your Statement Object

The newly created Connection object connection has a method in it called createCommand() that will return a RdmCommand object. You will then use that object with this Connection to the database.

```

RdmCommand command = connection.createCommand();
try {
...
} finally {
command.close();
}

```

### Step 7 Execute Statements to Create or Open the Database

Using the RdmCommand object command you just created, you can execute several different methods depending on the type of statement you want to execute.

For example, if you would like to execute a SQL statement such as “OPEN database\_name” or “DELETE \* FROM table\_name” you would perform a command.executeNonQuery() method. You can see executeNonQuery() used in the code snippet below. In this example, we will create the database programmatically.

In this example, the database is trivial, consisting of a single table named hello\_table containing a single character column named foo. The sequence will create a table if it doesn't yet exist, or just open it if it does exist.

```

try {
RdmTransaction rdmtans = connection.BeginTransaction();
command.CommandText = "CREATE TABLE hello_table (f00 char(31))";
command.ExecuteNonQuery();
rdmtans.commit();
// now the database physically exists
} catch (Exception exception) {
// we are here if database exists
}

```

### Step 8 Inserting a new Row using the Statement Object

To insert a single row into this database, we use the ExecuteNonQuery() method, which is used for complete (unprepared) INSERT, UPDATE or DELETE statements. This implicitly starts a transaction, which will be one unit of update work applied to the database atomically. One INSERT is shown below with a parameter binding, but more could be added at this point.

```

command.CommandText = "INSERT INTO hello_table(f00) VALUES(?)";

```

```

command.CommandText = insertString;
RdmParameter parameter = new RdmParameter();
parameter.RdmType = RdmType.AnsiString;
parameter.Direction = ParameterDirection.Input;
parameter.Value = "Hello World!";
command.Parameters.Add(parameter);
command.ExecuteNonQuery();

```

### Step 9 Committing Changes

In order to have your changes finalized in the database you must perform a transaction commit. In ADO.NET this is done through a method in the RdmTransaction object. The method we will be using is RdmTransaction.Commit() and that will finalize any changes you made during a transaction.

```
rdmtrans.Commit(); //Commits all changes
```

### Step 10 Creating your Result Set Object (retrieving data from the database)

In ADO.NET, when you want to retrieve data from the database, you perform a SQL SELECT statement using your Command object with an execute method that returns a Result Set object. This method is called Command.ExecuteReader(). This means it will execute the specified Query and return the Query results in the given Reader.

```
command.CommandText = "SELECT * FROM hello_table";
```

```

RdmDataReader reader = command.ExecuteReader();
try {
    ...
} finally {
    reader.Close();
}

```

### Step 11 Accessing the Result Set

In order to access every piece of data in your Result Set, you must iterate through it. A method is provided within the Result Set to check if the next result in the Result Set is NULL, meaning no more data.

If the method reader.Read() returns TRUE then there is data in the database and you can retrieve it from your result set.

To access the data inside the Result Set you must perform a getter method. There are numerous getter methods available to retrieve the specific data type from the Result Set.

In this example we want a string, therefore we use the reader.getString() method, with the parameter being the column (first/only column is 0) you are retrieving from. Take a look at the code below to see an example of how this can be done.

```

while(reader.Read() != false)
{
    Console.WriteLine(reader.GetString(0));
}

```

```
}
```

This loop will retrieve all rows in the result set. When this sample program is run for the first time, there will be only one row. If you run it multiple times, you will find one row for each time it has been run.

### Step 12 Deallocating Resources

Here you will deallocate all of the resources you used above. In this case, your resources are each object that you used above, being your Connection object, Statement, and Result Set objects. For each nested try block you will have a finally block, which performs the corresponding close method. These statements have been shown in context above, but here are the cleanup calls in sequence from the code.

```

} finally {
reader.Close ();
}
} finally {
command.Close ();
}
} catch (Exception exception) {
Console.WriteLine ("Exception : " + exception.ToString ());
} finally {
connection.Close ();
}

```

### Step 13 Final Catch and Finally block

The very last block contains both a catch block and a finally block. The catch block determines what to do if an exception was thrown in the code above. In this case just displaying the exception to standard out is sufficient. The finally block will be executed regardless of an exception being thrown. Here we will deallocate our Connection object. In this example, if your Connection object does throw an exception for some reason, it is “thrown” out of the method. In a normal situation you could either add another try catch block, or declare this program as a separate method and handle it elsewhere.

```

} catch (Exception exception) {
WriteLine("Exception : " + exception.ToString());
} finally {
connection.Close();
}

```

### Step 14 Compiling your application

### Step 15 Running the program

## Data Binding

User can bind the data with the controls of the forms. This process is known as data binding. There are two types of data binding in ASP.NET known as simple data binding and declarative data binding.

### Simple data binding

- In simple data binding, the control is bounded to a data set.
- The properties of the control are used for binding with the value.
- Depending on the control to be bounded, the binding's property is set.

Consider the following example where the Academic information of the student is bounded using various controls.

### Example to demonstrate the simple data binding in ASP.NET

Consider an example where a windows form is used for displaying the details. Create a Windows form application in Visual studio.

1. Add the windows form to the design view.
2. Add three labels and corresponding textboxes to it. Add the labels as Name, Age and Location.



3. Select the View, Properties Window in the application
4. Select the first textbox and navigate to the properties window.
5. Expand the DataBindings property.
6. From the drop down list, select the Text property.
7. Click on the Add Project Data Source from the drop down list.
8. Add a connection to the database and select the appropriate table.
9. Select Other Data Sources, Project data source, DataSet.
10. Select the appropriate value and bind the textbox control with it.
11. Press F5 or select Start debugging option. Execute the windows form and the following output is displayed.

### Declarative data binding

The process of binding a component like listbox, DataGrid, record list with the dataset is known as declarative binding. When there is more than one element in the database, the declarative binding is used.

Some of the controls used for the declarative data binding are listed below.

1. DataGrid: The data from multiple records is displayed using the DataGrid view. The DataSource property of the control is used for binding the specific element data.

2. **ListBox**: The data for a column from different dataset is displayed. The **DataSource property** is used for binding the control. The control binds to the specific element using the **DisplayMember property**.
3. **ComboBox**: The **DisplayMember property** is used for binding the control to the **specific data element**. The **DataSource property** is used for binding the control to the data source.

The following objects are needed for data binding in ASP.NET.

The data accessed from the database is stored in the dataset.

1. The data provider is used for accessing data through the command object
2. The data adapter is used for selecting, updating, inserting, deleting the data using commands.

**Consider the following example to demonstrate the declarative data binding**

Create an ASP.NET web application in visual studio.

1. Add a grid view control in the design view of the application
2. In the source view, add the following code

Code:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title></title>
</head>
<body>
  <form id="form1" runat="server">
    <div>

      <asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False"
      DataSourceID="SqlDataSource1" >
        <Columns>
          <asp:BoundField DataField="studid" HeaderText="studid" SortExpression="studid" />
          <asp:BoundField DataField="studname" HeaderText="studname" SortExpression="studname" />
          <asp:BoundField DataField="marks" HeaderText="marks" SortExpression="marks"/>
        </Columns>
      </asp:GridView>
      <asp:SqlDataSource ID="SqlDataSource1" runat="server"
        ConnectionString="<%$ConnectionStrings:demoConnectionString1 %>"
        SelectCommand="SELECT * FROM [ result ] "> </asp:SqlDataSource>

    </div>
  </form>
</body>
</html>
```

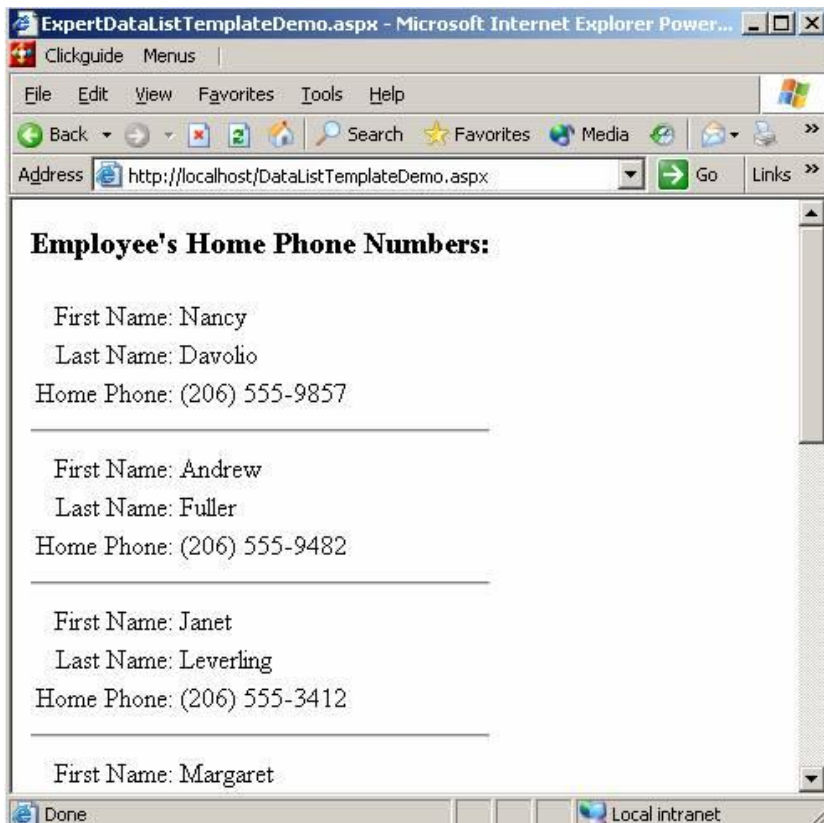
3. The above code is compiled and executed, the following output is generated (if the database is already created and inserted, we get the table).

studid	studname	marks
1	Dick	20
3	Peter	30
4	John	50

## DataList and DataGrid and Repeater

### What is DataList?

- The ASP.NET DataList control is a light weight server side control that works as a container for data items.
- It is used to display data into a list format to the web pages.
- It displays data from the data source.
- The data source can be either a DataTable or a table from database.



### What is DataGrid?

The DataGrid control binds to a single DataSet object. The DataSet object of the "DataGrid Application" is initially populated from a database using an DataAdapter object.



## What is repeater?

- Repeater control is used to show a repeated list of items from data source like data table, database, xml file, list etc.
- A Repeater is a Data-bound control.
- Data-bound controls are container controls.
- It creates a link between the Data Source and the presentation UI to display the data. The repeater control is used to display a repeated list of items.
- Repeater control provides us a table layout.
- Repeater is the fastest control in data controls available in Asp.Net. So, we can say performance of repeater control is far better than other data control like GridView.

There are different types of template exists in Repeater.

### **Header Template**

- It renders on the top of the control and show the header data.

### **Footer Template**

- It renders on the bottom of the control and use to show the footer data like paging.

### **Item Template**

- It is main template which is used to display the data from database, xml, list and data table.

### **Alternating Item Template**

- It is used as like Item Template but It renders once after other data display. Basically use of alternating item template is providing look and style of alternative rows like background color, font etc.

### **Separator Template**

- It renders after each item. For example a line after every record,

## DataList and DataGrid and Repeater

- DataList and GridView and Repeater are Data-bound controls that bound to a data source control like SqlDataSource, LinqDataSource to display and modify data in your Asp.Net web application.
- Data-bound controls are composite controls that contains others Asp.Net controls like as Label, TextBox, DropDownList etc. into a single layout. In this article, I am going to expose the difference among these three.



### Difference between DataList and Repeater

<b>DataList</b>	<b>Repeater</b>
Rendered as Table.	Template driven.
Automatically generates columns from the data source.	This features is not supported.
Selection of row is supported.	Selection of row is not supported.
Editing of contents is supported.	Editing of contents is not supported.
You can arrange data items horizontally or vertically in DataList by using property RepeatDirection.	This features is not supported.
Performance is slow as compared to Repeater	This is very light weight and fast data control among all the data control.

### Difference between GridView and Repeater

<b>GridView</b>	<b>Repeater</b>
It was introduced with Asp.Net 2.0.	It was introduced with Asp.Net 1.0
Rendered as Table.	Template driven.
Automatically generates columns from the data source.	This features is not supported.
Selection of row is supported.	Selection of row is not supported.
Editing of contents is supported.	Editing of contents is not supported.
Built-in Paging and Sorting is provided.	You need to write custom code.
Supports auto format or style features.	This has no this features.
Performance is very slow as compared to Repeater.	This is very light weight and fast data control among all the data control.

### Difference between GridView and DataList

<b>GridView</b>	<b>DataList</b>
It was introduced with Asp.Net 2.0.	It was introduced with Asp.Net 1.0.
Built-in Paging and Sorting is provided.	You need to write custom code.
Built-in supports for Update and Delete operations.	Need to write code for implementing Update and Delete operations.
Supports auto format or style features.	This features is not supported.
RepeatDirection property is not supported.	You can arrange data items horizontally or vertically in DataList by using property RepeatDirection.
Doesn't support customizable row separator.	Supports customizable row separator by using SeparatorTemplate.
Performance is slow as compared to DataList.	Performance is fast is compared to GridView.

## Files

- Reading and writing with streams
- The .NET supports to create simple “flat” files in text or binary format.
- Unlike a database these files does not have any internal structure.

## Text files

We can write to and read from from file using the special classes called stream writer and stream reader. The File class also support to read or write into a file.

```
Dim wr as StreamWriter
```

```
wr=File.CreateText("D:\ourFile.txt")
```

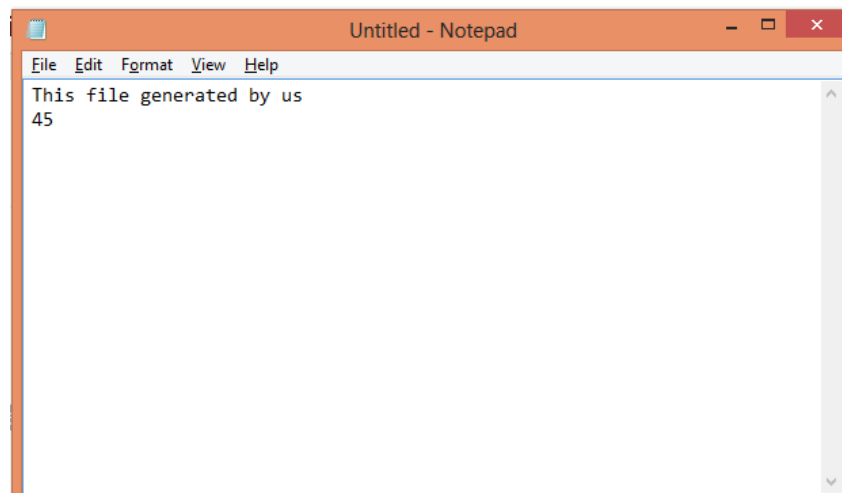
we can add information to the same file using WriteLine() method.

```
wr.WriteLine("This file generated by us")
```

```
wr.WriteLine(45)
```

```
wr.close()
```

## Output



To read the file, StreamReader class support by ReadLine() method.

```
Dim rr As StreamReader = File.OpenText("C:\ourFile.txt")
```

```
Dim InputS           tring As String
```

```
InputString = rr.ReadLine() 'This file generated by us
```

```
InputString = rr.ReadLine() '45
```

## XML

**XML classes supports communication between the applications or components.**

.NET provides five namespace - System.Xml, System.Xml.Schema, System.Xml.Serialization, System.Xml.XPath, and System.Xml.Xsl to support XML classes. The System.Xml namespace contains major XML classes. This namespace contains many classes to read and write XML documents. In this article, we are going to concentrate on reader and write class. These reader and writer classes are used to read and write XML documents. These classes are - XmlReader, XmlTextReader, XmlValidatingReader, XmlNodeReader, XmlWriter, and XmlTextWriter. As you can see there are four reader and two writer classes.

### Reading XML Documents

```
XmlTextReader textReader = new XmlTextReader("C:\\books.xml");
```

### Writing XML Documents

```
// Create a new file in C:\\ dir
XmlTextWriter textWriter = new XmlTextWriter("C:\\myXmlFile.xml", null);
// Opens the document
textWriter.WriteStartDocument();
// Write comments
textWriter.WriteComment("First Comment XmlTextWriter Sample Example");
textWriter.WriteComment("myXmlFile.xml in root dir");
// Write first element
textWriter.WriteStartElement("Student");
textWriter.WriteStartElement("r", "RECORD", "urn:record");
```

There are other classes to create an XML document, write and read.

### Program to create an XML document and write its contents to the XML document.

```
using System;
using System.Xml;
using System.Data;
using System.Data.OleDb;
namespace ReadingXML2 {
class Class1 {
static void Main(string[] args) {
// create a connection
OleDbConnection con = new OleDbConnection();
con.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\\Northwind.mdb";
// create a data adapter
OleDbDataAdapter da = new OleDbDataAdapter("Select * from Customers", con);
// create a new dataset
DataSet ds = new DataSet();
// fill dataset
da.Fill(ds, "Customers");
// write dataset contents to an xml file by calling WriteXml method
ds.WriteXml("C:\\OutputXML.xml");
}
}
}
```

\*\*\*\*\*

**Due to Corona outbreak during the lockdown period (March 24-April 21, 2020), I have collected the study materials for assisting students from the websites listed below.**

**Further, I acknowledge the sources thereto.**

<https://www.javatpoint.com/ado-net-command>

<https://www.c-sharpcorner.com/UploadFile/d0a1c8/database-programming-with-ado-net/>

<https://raima.com/architecture/>

<http://asp.net-informations.com/ado.net/ado-architecture.htm>

<http://www.programcall.com/9/aspnet/aspnet-validation-controls.aspx>

<https://www.go4expert.com/articles/data-binding-aspnet-t34155/>

<https://www.dotnettricks.com/learn/aspnet/difference-between-repeater-and-datalist-and-gridview>

<https://www.geeksforgeeks.org/c-sharp-delegates/>

<https://www.tutorialspoint.com/csharp/>