

Unit - 4 Advanced C#

Delegates; Events; Lambda Expressions; Exception Handling; Introduction to LINQ; Working with Databases; Web Applications using ASP.NET

Delegates

A delegate is an object that knows how to call a method. A delegate type defines the kind of method that delegate instances can call. Specifically, it defines the method's return type and its parameter types.

Delegates are especially used **for implementing events and the call-back methods**. All delegates are implicitly derived from the **System.Delegate** class. It provides a way which tells which method is to be called when an event is triggered.

For example, if you click an **Button** on a form (**Windows Form application**), the program would call a specific method.

In simple words, it is a type that represents references to methods with a particular parameter list and return type and then calls the method in a program for execution when it is needed.

Declaring Delegates

Delegate type can be declared using the **delegate** keyword. Once a delegate is declared, delegate instance will refer and call those methods whose return type and parameter-list matches with the delegate declaration.

Syntax:

```
[modifier] delegate [return_type] [delegate_name] ([parameter_list]);
```

- **modifier:** It is the required modifier which defines the access of delegate and it is optional to use.
- **delegate:** It is the keyword which is used to define the delegate.
- **return_type:** It is the type of value returned by the methods which the delegate will be going to call. It can be void. A method must have the same return type as the delegate.
- **delegate_name:** It is the user-defined name or identifier for the delegate.
- **parameter_list:** This contains the parameters which are required by the method when called through the delegate.

Example:

```
public delegate int DelegateTest(int x, int y, int z);
```

Note: A delegate will call only a method which agrees with its signature and return type. A method can be a static method associated with a class or can be an instance method associated with an object, it doesn't matter.

Instantiation & Invocation of Delegates

Once a delegate type is declared, a delegate object must be created with the new keyword and be associated with a particular method. When creating a delegate, the argument passed to the new expression is written similar to a method call, but without the arguments to the method.

For example

```
[delegate_name] [instance_name] = new [delegate_name] (method_name);
```

DelegateTest obj = new DelegateTest (MyMethod);

```
// here,  
// " DelegateTest" is delegate name.  
// " obj" is instance_name  
// " MyMethod" is the calling method.
```

Example of simple delegate is shown below:

```
using System;  
public delegate int MyDelegate(int x);  
class DelegateTest  
{  
    static int MyMethod(int x)  
    {  
        return x * x;  
    }  
  
    static void Main(string[] args)  
    {  
        MyDelegate del = new MyDelegate(MyMethod);  
        int res = del(5); //25  
        Console.WriteLine("Result is : "+res);  
        Console.ReadKey();  
    }  
}
```

Output:

Result is : 25

Example of delegate in multiple class

```
using System;
public delegate void MyDelegate(int a,int b);
class A{
    public void Add(int a,int b){
        int sum=a+b;
        Console.WriteLine("Sum is: {0}",sum);
    }
}
class Program{
    static void Main(){
        A obj=new A();
        MyDelegate del=new MyDelegate(obj.Add);
        del(20,30);
    }
}
```

Output:

Sum is: 50

Multicast Delegates

All delegate instances have multicast capability. This means that a delegate instance can reference not just a single target method, but also a list of target methods.

The + and += operators combine delegate instances.

For example:

```
SomeDelegate d = SomeMethod1;
d += SomeMethod2;
```

The last line is functionally the same as:

```
d = d + SomeMethod2;
```

Invoking d will now call both SomeMethod1 and SomeMethod2. **Delegates are invoked in the order they are added.**

The - and -= operators remove the right delegate operand from the left delegate operand.

For example:

```
d -= SomeMethod1;
```

Invoking d will now cause only SomeMethod2 to be invoked.

Calling + or += on a delegate variable with a null value works, and it is equivalent to assigning the variable to a new value:

```
SomeDelegate d = null;
d += SomeMethod1; // Equivalent (when d is null) to d =
SomeMethod1;
```

Similarly, calling -= on a delegate variable with a single target is equivalent to assigning null to that variable.

Example of Delegate Multicasting

```
using System;
public delegate void MyDelegate(int a,int b);
class A{

    public void Add(int a,int b){
        int sum=a+b;
        Console.WriteLine("Sum is: {0}",sum);
    }

    public void Diff(int a,int b){
        int diff=a-b;
        Console.WriteLine("Difference is: {0}",diff);
    }

    public void Multiply(int a,int b){
        int mul=a*b;
        Console.WriteLine("Product is: {0}",mul);
    }

}

class Program{
    static void Main(){
        A obj=new A();
        MyDelegate del=new MyDelegate(obj.Add);
        //Multicasting
        del+=obj.Diff;
        del+=obj.Multiply;

        del-=obj.Add;

        del(20,30);
    }
}
```

Output:

Difference is: -10
Product is: 600

Func Delegate

C# includes built-in generic delegate types Func and Action, so that you don't need to define custom delegates manually in most cases. Func is a generic delegate included in the System namespace.

It has zero or more input parameters and one out parameter. The last parameter is considered as an out parameter.

The Func delegate that takes one input parameter and one out parameter is defined in the System namespace, as shown below:

```
public delegate TResult Func<in T, out TResult>(T arg);
```

The following Func delegate takes two input parameters of int type and returns a value of int type:

```
Func<int, int, int> sum;
```

Example is shown below:

```
class Program
{
    static int Sum(int x, int y)
    {
        return x + y;
    }

    static void Main(string[] args)
    {
        Func<int,int, int> add = Sum;

        int result = add(10, 10);

        Console.WriteLine(result);
    }
}
```

Output:

20

Action Delegate

Action is a delegate type defined in the System namespace. **An Action type delegate is the same as Func delegate except that the Action delegate doesn't return a value.**

In other words, **an Action delegate can be used with a method that has a void return type.** An Action delegate can take up to **16 input parameters** of different types.

For example, the following delegate prints an int value.

```
public delegate void Print(int a);
static void ConsolePrint(int a)
{
    Console.WriteLine(a);
}
static void Main(string[] args)
{
    Print prnt = ConsolePrint;
    prnt(10);
}
```

Output

10

You can use an Action delegate instead of defining the above Print delegate, for example:

```
static void ConsolePrint(int a)
{
    Console.WriteLine(a);
}

static void Main(string[] args)
{
    Action<int> printActionDel = ConsolePrint;
    printActionDel(10);
}
```

Output:

10

Anonymous Method in C#

An anonymous method is a method which doesn't contain any name which is introduced in C# 2.0. It is useful when the user wants to create an inline method and also wants to pass parameter in the anonymous method like other methods.

An Anonymous method is defined using the delegate keyword and the user can assign this method to a variable of the delegate type.

```
delegate(parameter_list){
    // Code..
};
```

Following are the features of anonymous method:

1. Anonymous method can be defined using the delegate keyword
2. Anonymous method must be assigned to a delegate.
3. Anonymous method can access outer variables or functions.
4. Anonymous method can be passed as a parameter.
5. Anonymous method can be used as event handlers.

Example of Anonymous Method

```
using System;
class Program{
    //defining delegate
    public delegate void MyDel(int a,int b);

    static void Main(){
        //Example of anonymous method
        MyDel del=delegate(int a,int b){
            int sum=a+b;
            Console.WriteLine("Sum={0}",sum);
        };

        del(20,30);
    }
}
```

Output:

Sum=50

Lambda Expressions

Lambda expressions in C# are used like anonymous functions, with the difference that in Lambda expressions you don't need to specify the type of the value that you input thus making it more flexible to use.

The ' \Rightarrow ' is the lambda operator which is used in all lambda expressions. The Lambda expression is divided into two parts, the left side is the input and the right is the expression.

The Lambda Expressions can be of two types:

1. **Expression Lambda:** Consists of the input and the expression.

Syntax:

input \Rightarrow expression;

2. **Statement Lambda:** Consists of the input and a set of statements to be executed. It can be used along with delegates.

Syntax:

input \Rightarrow { statements };

Example of Expression Lambda

```
using System;
class Program{
    //Expression lambda
    public int Test(int a)=>a+10;
    public int Add(int a,int b)=>a+b;

    static void Main(){
        Program obj=new Program();
        int res;
        res=obj.Test(5); //returns 15
        Console.WriteLine("Result={0}",res);

        res=obj.Add(10,20); //returns 30
        Console.WriteLine("Result={0}",res);
    }
}
```

Output:

Result=15
Result=30

Example of Statement Lambda

```
using System;
class Program{
    //defining delegate
    public delegate void MyDel(int a,int b);

    static void Main(){
        //statement lambda
        MyDel del=(a,b)=>{
            int add=a+b;
            Console.WriteLine("Sum={0}",add);
        };

        del(10,20);
    }
}
```

Output:

Sum=30

Exception Handling

An exception is a problem that arises during the execution of a program. A C# 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. C# 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 is activated. It is 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.

Exception Classes in C#

C# exceptions are represented by classes. The exception classes in C# are mainly directly or indirectly derived from the **System.Exception** class. Some of the exception classes derived from the System.Exception class.

The **System.SystemException** class is the base class for all predefined system exception.

The following table provides some of the predefined exception classes derived from the Sytem.SystemException class –

Sr.No.	Exception Class & Description
1	System.IO.IOException Handles I/O errors.
2	System.IndexOutOfRangeException Handles errors generated when a method refers to an array index out of range.
3	System.ArrayTypeMismatchException Handles errors generated when type is mismatched with the array type.
4	System.NullReferenceException Handles errors generated from referencing a null object.
5	System.DivideByZeroException Handles errors generated from dividing a dividend with zero.
6	System.InvalidCastException Handles errors generated during typecasting.
7	System.OutOfMemoryException Handles errors generated from insufficient free memory.
8	System.StackOverflowException Handles errors generated from stack overflow.

Following is the syntax of exception handling.

```
try
{
    ... // exception may get thrown within execution of this block
}
catch (ExceptionA ex)
{
    ... // handle exception of type ExceptionA
}
catch (ExceptionB ex)
{
    ... // handle exception of type ExceptionB
}
finally
{
    ... // cleanup code
}
```

Try Block

Any suspected code that may raise exceptions should be put inside a try{ } block. During the execution, if an exception occurs, the flow of the control jumps to the first matching catch block. **Try block is used to monitor exception.**

A try block must be followed by catch or finally or both blocks. The try block without a catch or finally block will give a compile-time error.

Example – Divide by Zero Exception

```
using System;
class Program
{
    static void Main(string[] args)
    {
        int a = 10, b = 0;
        try
        {
            int result = a / b;
            Console.WriteLine("Result = {0}", result);
        }
        catch (DivideByZeroException ex)
        {
            Console.Write(ex);
        }

        Console.WriteLine("I am Outside try/catch block !");
    }
}
```

Output:

System.DivideByZeroException: Attempted to divide by zero.
I am Outside try/catch block !

Example – Index Out of Range

```
using System;
class Program{
    static void Main(){
        int[] arr=new int[2];
        try{
            //accessing array
            arr[3]=30;    //exception
        }catch(IndexOutOfRangeException ex){
            Console.WriteLine(ex);
        }
    }
}
```

Output:

System.IndexOutOfRangeException: Index was outside the bounds of the array.

Nested Try Block

C# allows nested try-catch blocks. **When using nested try-catch blocks, an exception will be caught in the first matching catch block that follows the try block where an exception occurred.**

```
using System;
public class Program
{
    public static void Main()
    {
        var divider = 0;

        try
        {
            try
            {
                var result = 100/divider;
            }
            Catch(Exception ex)
            {
                Console.WriteLine("Inner catch");
            }
        }
        Catch(Exception ex)
        {
            Console.WriteLine("Outer catch");
        }
    }
}
```

Output:

Inner catch

The catch Clause

A catch clause specifies what type of exception to catch. This must either be System.Exception or a subclass of System.Exception.

You can handle multiple exception types with multiple catch clauses:

```
using System;
class Program
{
    static void Main(string[] args)
    {
        Console.Write("Please enter a number to divide 100: ");

        try
        {
            int num = int.Parse(Console.ReadLine());

            int result = 100 / num;

            Console.WriteLine("100 / {0} = {1}", num, result);
        }
        catch(DivideByZeroException ex)
        {
            Console.Write("Cannot divide by zero. Please try again.");
        }
        catch(InvalidOperationException ex)
        {
            Console.Write("Invalid operation. Please try again.");
        }
        catch(FormatException ex)
        {
            Console.Write("Not a valid format. Please try again.");
        }
        catch(Exception ex)
        {
            Console.Write("Error occurred! Please try again.");
        }
    }
}
```

Output:

Please enter a number to divide 100: xy
Not a valid format. Please try again.

The finally Block

A finally block always executes—whether or not an exception is thrown and whether or not the try block runs to completion. finally blocks are typically used for clean-up code.

A finally block executes either:

- After a catch block finishes
- After control leaves the try block because of a jump statement (e.g., return or goto)
- After the try block ends

```
using System;
class Program
{
    static void Main(string[] args)
    {
        int a = 10, b = 0;
        try
        {
            int result = a / b;
            Console.WriteLine("Result = {1}", result);
        }
        catch (DivideByZeroException ex)
        {
            Console.Write(ex);
        }
        finally
        {
            Console.WriteLine("Finally always run !");
        }

        Console.WriteLine("I am Outside try/catch block !");
    }
}
```

Output:

System.DivideByZeroException: Attempted to divide by zero.

Finally always run !

I am Outside try/catch block !

Throwing Exceptions

We have seen in the previous section how to handle exceptions which are automatically raised by CLR. Here, we will see how to raise an exception manually.

An exception can be raised manually by using the throw keyword. Any type of exceptions which is derived from Exception class can be raised using the throw keyword.

```

class Program
{
    static void Check(int age)
    {
        if (age < 18)
            throw new ArithmeticException("Not Eligible to
                                           Vote !");
    }

    static void Main(string[] args)
    {
        try
        {
            Check(15);
        } catch (Exception ex)
        {
            Console.WriteLine(ex);
        }
    }
}

```

Output:

System.ArithmeticException: Not Eligible to Vote !

Re-throwing an exception

You can capture and re-throw an exception as follows:

```

try { ... }
catch (Exception ex)
{
    // Log error
    ...
    throw;           // Rethrow same exception
}

```

It is shown in below example:

```

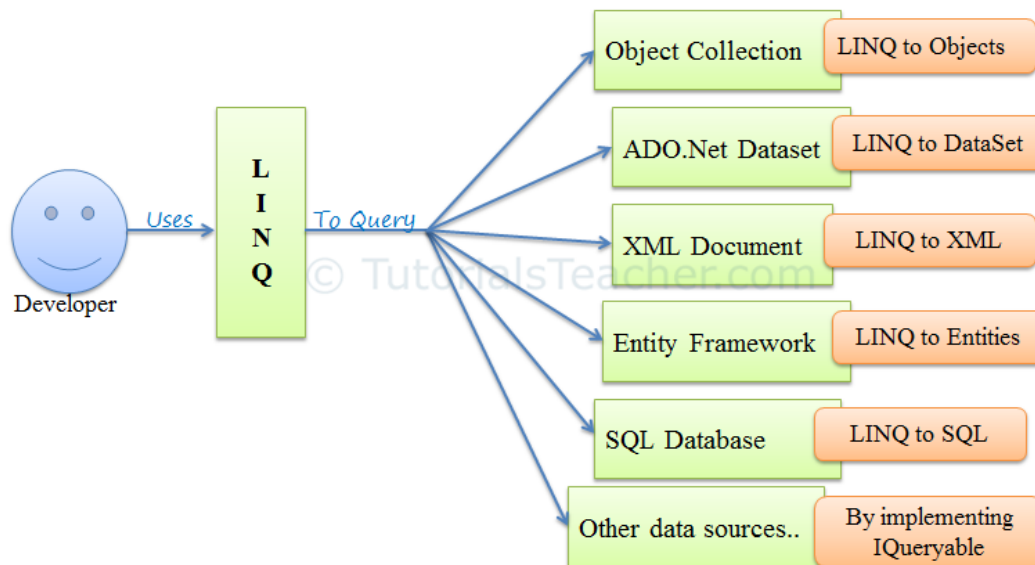
class Program
{
    static void Main(string[] args)
    {
        int a=10, b=0, res;
        try
        {
            res = a / b;
        } catch (ArithmeticException ex)
        {
            Console.WriteLine(ex);
            //rethrowing same exception
            throw;
        }
    }
}

```

Introduction to LINQ

LINQ (Language Integrated Query) is uniform query syntax in C# to retrieve data from different sources and formats. It is integrated in C#, thereby eliminating the mismatch between programming languages and databases, as well as providing a single querying interface for different types of data sources.

For example, SQL is a Structured Query Language used to save and retrieve data from a database. In the same way, LINQ is a structured query syntax built in C# to retrieve data from different types of data sources such as collections, ADO.Net DataSet, XML Docs, web service and MS SQL Server and other databases.



LINQ queries return results as objects. It enables you to use object-oriented approach on the result set and not to worry about transforming different formats of results into objects.



Syntax of LINQ

```
from variable in data_source
where condition
order by/group by
select variable;
```

where, data_source can be array, List, Database Table, XML, etc.

Example of LINQ in Array

```
using System;
using System.Linq;
class Program{
    static void Main(){

        //creating array with data
        int[] arr={10,20,23,5,43,6,29,18};

        //linq query
        var res=from x in arr
                where x>15 && x<30
                select x;

        //displaying data
        foreach(var item in res)
            Console.WriteLine(item);
    }
}
```

Output:

```
20
23
29
18
```

Example of LINQ in List

```
using System;
using System.Linq;
using System.Collections.Generic;
class Program{
    static void Main(){
        //creating list with data
        List<int> list=new List<int>(){
            10,20,23,5,43,6,29,18
        };

        //linq query
        var res=from x in list
                where x>15 && x<30
                orderby x descending
                select x;

        //displaying data
        foreach(var item in res)
            Console.WriteLine(item);
    }
}
```


Output:

29
23
20
18

LINQ Method

```
// Data source
string[] names = {"Bill", "Steve", "James", "Mohan" };
// LINQ Query
var myLinqQuery = from name in names
                  where name.Contains('a')
                  select name;
```

The above example demonstrates a simple LINQ query that gets all strings from an array which contains 'a'.

LINQ Operators

Classification	LINQ Operators
Filtering	Where, OfType
Sorting	OrderBy, OrderByDescending, ThenBy, ThenByDescending, Reverse
Grouping	GroupBy, ToLookup
Join	GroupJoin, Join
Projection	Select, SelectMany
Aggregation	Aggregate, Average, Count, LongCount, Max, Min, Sum
Quantifiers	All, Any, Contains
Elements	ElementAt, ElementAtOrDefault, First, FirstOrDefault, Last, LastOrDefault, Single, SingleOrDefault
Set	Distinct, Except, Intersect, Union
Partitioning	Skip, SkipWhile, Take, TakeWhile
Concatenation	Concat
Equality	Equals, SequenceEqual
Generation	DefaultEmpty, Empty, Range, Repeat
Conversion	AsEnumerable, AsQueryable, Cast, ToArray, ToDictionary, ToList

Using LINQ in Table like Data Source

```
using System;
using System.Linq;
using System.Collections.Generic;

class Employee{
    public int eid{get;set;}
    public string name{get;set;}
    public string address{get;set;}
    public double salary{get;set;}

    public Employee(int eid,string name,string address,double salary){
        this.eid=eid;
        this.name=name;
        this.address=address;
        this.salary=salary;
    }
}

class Program{
    static void Main(){
        //creating list for storing data
        List<Employee> list=new List<Employee>(){
            new Employee(101,"Ram","Btm",21000),
            new Employee(102,"Shyam","Ktm",15000),
            new Employee(103,"Hari","Btm",25000),
            new Employee(104,"Rita","Ktm",23000),

        };

        //linq query to select name and address of whose address is Btm
        var res=from x in list
                where x.address=="Btm"
                select new {x.name,x.address};
        //select x for *

        //displaying data in tabular form
        Console.WriteLine("Name\tAddress");
        foreach(var item in res)
            Console.WriteLine(item.name+"\t"+item.address);
    }
}
```

Output:

Name	Address
Ram	Btm
Hari	Btm

Select records of employees whose salary is between 15000 to 25000 and address is Ktm.

```
var res=from x in list
    where x.salary>=15000 && x.salary<=25000 &&
        x.address=="Ktm"
    select x;
```

Select name of employees whose address is Ktm order by salary.

```
var res=from x in list
    where x.address=="Ktm"
    orderby x.salary
    select new {x.name};
```

Select eid, name of employees whose address is Ktm order by salary descending.

```
var res=from x in list
    where x.address=="Ktm"
    orderby x.salary descending
    select new {x.eid,x.name};
```

StartsWith and EndsWith Method (equivalent to LIKE operator in SQL)

Select name, address of employees whose name starts with 'R' order by salary.

```
var res=from x in list
    where x.name.StartsWith('R')
    orderby x.salary
    select new {x.name,x.address};
```

Select name, address of employees whose name ends with 'm'.

```
var res=from x in list
    where x.name.EndsWith('R')
    select new {x.name,x.address};
```

Using aggregate functions

```
using System;
class LinqTest
{
    static void Main(string[] args)
    {
        List<int> marks = new List<int>() { 10,30,50,20,5};
        int max = marks.Max();
        int min = marks.Min();
        int sum = marks.Sum();
        int total = marks.Count();
        Console.WriteLine("Maximum marks="+max);
    }
}
```

```

        Console.WriteLine("Minimum marks=" + min);
        Console.WriteLine("Sum of marks=" + sum);
        Console.WriteLine("Total Count=" + total);

        Console.ReadLine();
    }
}

```

Output:

```

Maximum marks=50
Minimum marks=5
Sum of marks=115
Total Count=5

```

Sub Queries

Select name, salary of employee whose salary is maximum.

```

var res=from x in list
        where x.salary==(
            from y in list
            select y.salary
        ).Max()
        select new {x.name,x.salary};

```

Select records of employee whose salary is greater than the average salary of employee.

```

var res=from x in list
        where x.salary>(
            from y in list
            select y.salary
        ).Average()
        select x;

```

Using Group By

Select eid, name and salary of employees whose salary is less than 30000 group by address and order by salary in descending order.

```

var res=from x in list
        where x.salary<25000
        orderby x.salary descending
        group x by x.address;

```

Displaying Data

```

foreach(var data in res){
    Console.WriteLine(data.Key); //Group Key
    foreach(var item in data){
        Console.WriteLine(item.eid+"\t"+item.name+"\t"
            +item.salary);
    }
}

```

Output:

```

Ktm
104      Rita      23000
102      Shyam     15000
Btm
101      Ram       21000

```

Joining multiple lists – (join, concat, union)**Using Concat**

```

using System;
using System.Linq;
using System.Collections.Generic;

class Program{
    static void Main(){
        List<string> names = new List<string>()
            {"Ram", "Shyam", "Hari"};
        List<string> address = new List<string>()
            {"Btm", "Ktm", "Btm" };

        //using concat
        var result = names.Concat(address);

        //displaying data
        foreach (var res in result)
            Console.WriteLine(res);
    }
}

```

Output:

```

Ram
Shyam
Hari
Btm
Ktm
Btm

```

Using Union

```

//using union
var result = names.Union(address);

```

Output:

```

Ram
Shyam
Hari
Btm
Ktm

```

Difference:

- Union doesn't include duplication.

Join in LINQ

Syntax:

```
from ... in outerSequence
    join ... in innerSequence
    on outerKey equals innerKey
select ...
```

Example:

Select sid, name and cname whose course is Java.

```
using System;
using System.Linq;
using System.Collections.Generic;

class Course{
    public int cid{get;set;}
    public string cname{get;set;}

    public Course(int cid, string cname){
        this.cid=cid;
        this.cname=cname;
    }
}

class Student{
    public int sid{get;set;}
    public string name{get;set;}
    public int cid{get;set;}

    public Student(int sid,string name,int cid){
        this.sid=sid;
        this.name=name;
        this.cid=cid;
    }
}

class Program{
    static void Main(){
        //List for course
        List<Course> course=new List<Course>(){
            new Course(101,"Java"),
            new Course(102,"Dot Net")
        };

        //List for student
        List<Student> std=new List<Student>(){
            new Student(1001,"Ram",102),
            new Student(1002,"Shyam",101),
            new Student(1003,"Hari",101),
            new Student(1004,"Sita",102),
            new Student(1005,"Gita",101)
        };
    }
}
```

```

//select sid, name and cname whose course is Java
var res=from x in std
        join y in course
        on x.cid equals y.cid
        where y.cname=="Java"
        select new{x.sid,x.name,y.cname};

//displaying data
Console.WriteLine("Sid\tName\tCname");
foreach (var data in res)
    Console.WriteLine(data.sid+"\t"+data.name+"\t"+data.cname);
}
}

```

Output:

Sid	Name	Cname
1002	Shyam	Java
1003	Hari	Java
1005	Gita	Java

Working with Database

ADO.Net Basics

ADO.NET is a set of classes that expose data access services for .NET Framework programmers. ADO.NET provides a rich set of components for creating distributed, data-sharing applications. It is an integral part of the .NET Framework, providing access to relational, XML, and application data. ADO.NET supports a variety of development needs, including the creation of front-end database clients and middle-tier business objects used by applications, tools, languages, or Internet browsers.

Benefits of ADO.Net

1. **Ability to Cross:** Communicate between Heterogeneous Environments – ADO.NET has the exceptional benefit of establishing a connection between two heterogeneous environments. Once the connection is established, ADO.NET could easily communicate between these two heterogeneous environments.
2. **Easily Scalable:** ADO.NET is highly scalable, which means if requirement appears it is flexible enough to be expanded easily.
3. **High in Productivity:** ADO.NET is capable enough to build robust applications.
4. **Performance:** There is no lacking or delay and it can set connections quickly to fetch data.

Comparison between ADO.Net and Classic ADO

ADO

ADO was introduced in 1996 by Microsoft as its component of MDAC (Microsoft Data Access Components). It is based on COM (Component Object Modelling). ADO with other components of MDAC serves as a framework for client applications to access data stores. It removes the necessity to know implementation of database and lowers complexity of handling low level code requires for dealing with data.

ADO.NET

ADO.NET is an advanced database technology from Microsoft .NET Framework which provides communication between application system and database server. It is a component of the .NET Framework that is designed to work on disconnected model to access data from data store. Some of the .NET applications which are used to connect with database server are ASP.NET web applications, windows applications and console applications

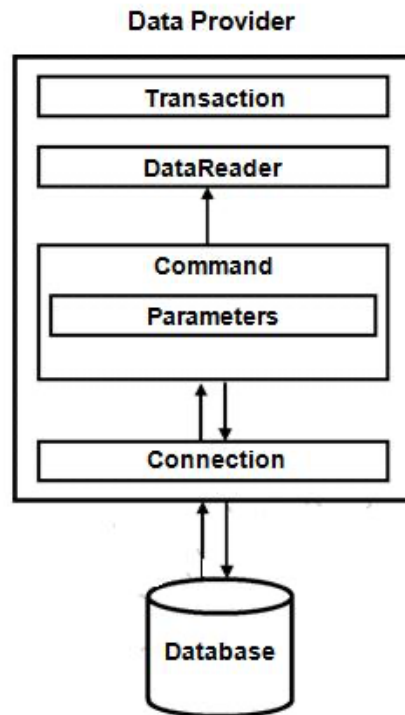
S.No.	ADO	ADO.NET
1.	It is based on COM (Component Object Modelling).	It is a CLR (Common Language Runtime) based library.
2.	It works only when data store is connected.	It does not need active connection to access data from data store.
3.	It has feature of locking.	It does not have feature of locking.
4.	It access and store data from data source by recordset object.	It access and store data from data source by dataset object.
5.	XML integration is not feasible in ADO.	XML integration is feasible in ADO.NET.
6.	In ADO, data is stored in binary form.	While in this, data is stored in XML.
7.	It allow us to create client side cursors only.	It give us the choice of using weather client side and server side cursors.
8.	It requires SQL JOINS and UNIONS to combine data from multiple tables in a single result table.	It uses DataRelational objects, for combining data from multiple tables without requiring JOINS and UNIONS.
9.	It supports sequential access of rows in a RecordSet.	It allows completely non-sequential data access in DataSet through collection based hierarchy.

ADO.NET Architecture

Connected Architecture

The architecture of ADO.net, in which **connection must be opened to access the data retrieved from database** is called as connected architecture.

Connected architecture was built on the classes' connection, command, datareader and transaction.



Connection : in connected architecture also the purpose of connection is to just establish a connection to database and it self will not transfer any data.

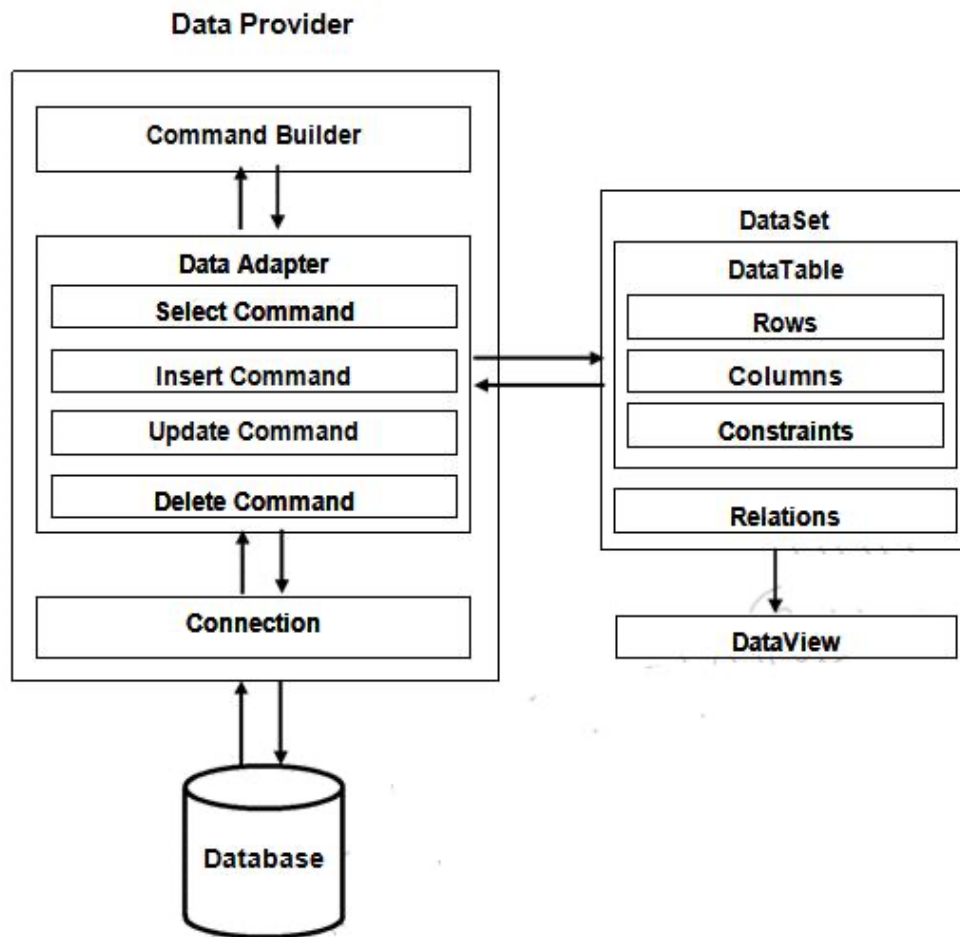
DataReader : DataReader is used to store the data retrieved by command object and make it available for .net application. Data in DataReader is read only and within the DataReader you can navigate only in forward direction and it also only one record at a time.

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

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.



Connection : Connection object is used to establish a connection to database and connection it self 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.

DataView : DataView is a view of table available in DataSet. It is used to find a record, sort the records and filter the records. By using dataview, you can also perform insert, update and delete as in case of a DataSet.

Shared and Database Specific Classes

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

1. Connection Class

In ADO.NET, we use connection classe to connect to the database. We can connect to a database using Connection class as follows:

Connection conn=new Connection("connection string");

2. Command Class

The Command class provides methods for storing and executing SQL statements and Stored Procedures.

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

- **ExecuteReader: Returns data to the client as rows.** This would typically be an SQL select statement or a Stored Procedure that contains one or more select statements. This method returns a DataReader object that can be used to fill a DataTable object or used directly for printing reports and so forth.
- **ExecuteNonQuery: Executes a command that changes the data in the database, such as an update, delete, or insert statement, or a Stored Procedure that contains one or more of these statements.** This method returns an integer that is the number of rows affected by the query.

- **ExecuteScalar:** This method only returns a single value. This kind of query returns a count of rows or a calculated value.
- **ExecuteXMLReader:** (SqlClient classes only) Obtains data from an SQL Server 2000 database using an XML stream. Returns an XML Reader object.

3. DataReader Class

The DataReader is used to retrieve data. It is used in conjunction with the Command class to execute an SQL Select statement and then access the returned rows.

Example of using Data Reader:

```
SqlDataReader myReader = myCommand.ExecuteReader();

while (myReader.Read())
    Console.WriteLine("\t{0}\t{1}", myReader.GetInt32(0), myReader
        .GetString(1));
myReader.Close();
```

4. DataAdapter Class

The DataAdapter is used to connect DataSets to databases. The DataAdapter is most useful when using data-bound controls in Windows Forms, but it can also be used to provide an easy way to manage the connection between your application and the underlying database tables, views and Stored Procedures.

5. DataSet Class

The DataSet is the heart of ADO.NET. The DataSet is essentially a collection of DataTable objects. In turn each object contains a collection of DataColumn and DataRow objects. The DataSet also contains a Relations collection that can be used to define relations among DataTable Objects.

Connect C# to MySQL

- First make sure you have downloaded and installed the **MySQL Connector/NET** from the [MySQL official website](#).
- Add reference **MySql.Data** in your project.

Note: If you are working with .Net Core project you can add Connector using following command:

dotnet add package MySql.Data

```
using MySql.Data.MySqlClient;
string constr = "SERVER=localhost; DATABASE=dbtest; UID=root;
                PASSWORD=";
MySqlConnection conn = new MySqlConnection(constr);
```

DataReader, DataAdapter, Dataset and Datatable :

DataReader is used to read the data from database and it is a read and forward only connection oriented architecture during fetch the data from database. DataReader will fetch the data very fast when compared with dataset. Generally we will use ExecuteReader object to bind data to dataReader.

//Example

```
SqlDataReader sdr = cmd.ExecuteReader();
```

DataReader

- Holds the connection open until you are finished (don't forget to close it!).
- Can typically only be iterated over once
- Is not as useful for updating back to the database

DataSet is a disconnected orient architecture that means there is no need of active connections during work with datasets and it is a collection of DataTables and relations between tables. It is used to hold multiple tables with data. You can select data form tables, create views based on table and ask child rows over relations. Also DataSet provides you with rich features like saving data as XML and loading XML data.

//Example

```
DataSet ds = new DataSet();  
adapter.Fill(ds);
```

DataAdapter will acts as a Bridge between DataSet and database. This dataadapter object is used to read the data from database and bind that data to dataset. Dataadapter is a disconnected oriented architecture.

//Example

```
SqlDataAdapter sda = new SqlDataAdapter(cmd);  
DataSet ds = new DataSet();  
da.Fill(ds);
```

DataAdapter

- Lets you close the connection as soon it's done loading data, and may even close it for you automatically
- All of the results are available in memory
- You can iterate over it as many times as you need, or even look up a specific record by index
- Has some built-in faculties for updating back to the database.

DataTable represents a single table in the database. It has rows and columns. There is no much difference between dataset and datatable, dataset is simply the collection of datatables.

//Example

```
DataTable dt = new DataTable();  
da.Fill(dt);
```

Difference between DataReader and DataAdapter:

1) A DataReader is an object returned from the ExecuteReader method of a DbCommand object. It is a forward-only cursor over the rows in the each result set. Using a DataReader, you can access each column of the result set, read all rows of the set, and advance to the next result set if there are more than one.

A DataAdapter is an object that contains four DbCommand objects: one each for SELECT, INSERT, DELETE and UPDATE commands. It mediates between these commands and a DataSet though the Fill and Update methods.

2) DataReader is a faster way to retrieve the records from the DB. DataReader reads the column. DataReader demands live connection but DataAdapter needs disconnected approach.

3) Data reader is an object through which you can read a sequential stream of data. it's a forward only data wherein you cannot go back to read previous data. data set and data adapter object help us to work in disconnected mode. data set is an in cache memory representation of tables. the data is filled from the data source to the data set thro' the data adapter. once the table in the dataset is modified, the changes are broadcast to the database back throw; the data adapter.

Creating a Table ADO.Net

```
using System;
using MySql.Data.MySqlClient;
using System.Data;

namespace MyProject
{
    class Program
    {
        MySqlConnection conn;
        MySqlCommand cmd;
        string sql="";
        public Program(){
            //creating connection
            string constr="SERVER=localhost; DATABASE=csit; UID=root;
                           PASSWORD="";
            conn=new MySqlConnection(constr);
            conn.Open();

            //creating table
            sql="CREATE TABLE IF NOT EXISTS employee(eid INT, name
                           VARCHAR(30))";
            cmd=new MySqlCommand(sql,conn);
            cmd.ExecuteNonQuery();
            Console.WriteLine("Table Created Successfully!");

            conn.Close();
        }
    }
}
```

```

        static void Main(string[] args)
        {
            new Program();
        }
    }
}

```

Output:

Table Created Successfully!

You can check table created or not in phpMyAdmin.

Inserting data using ADO.Net

```

using System;
using MySql.Data.MySqlClient;
using System.Data;

namespace MyProject
{
    class Program
    {
        MySqlConnection conn;
        MySqlCommand cmd;
        string sql="";
        public Program(){
            //creating connection
            string constr="SERVER=localhost; DATABASE=csit; UID=root;
                           PASSWORD=";
            conn=new MySqlConnection(constr);
            conn.Open();

            //inserting data
            sql="INSERT INTO employee(eid,name) VALUES (101,'Ram')";
            cmd=new MySqlCommand(sql,conn);
            cmd.ExecuteNonQuery();
            Console.WriteLine("Data Inserted Successfully!");
            conn.Close();
        }

        static void Main(string[] args)
        {
            new Program();
        }
    }
}

```

Output:

Data Inserted Successfully!

eid	name
101	Ram

Likewise, we just need to change SQL query to Update and Delete data.

Updating data using ADO.Net

```
//updating data
sql="UPDATE employee SET name='Raaju Poudel' WHERE eid=101";
cmd=new MySqlCommand(sql,conn);
cmd.ExecuteNonQuery();
Console.WriteLine("Data Updated Successfully!");
```

Deleting data using ADO.Net

```
//updating data
sql="DELETE FROM employee WHERE eid=101";
cmd=new MySqlCommand(sql,conn);
cmd.ExecuteNonQuery();
Console.WriteLine("Data Deleted Successfully!");
```

Selecting data using ADO.Net using DataSet

```
using System;
using MySql.Data.MySqlClient;
using System.Data;

namespace MyProject
{
    class Program
    {
        MySqlConnection conn;
        MySqlCommand cmd;
        string sql="";
        public Program(){
            //creating connection
            string constr="SERVER=localhost; DATABASE=csit; UID=root;
                                PASSWORD=";

            conn=new MySqlConnection(constr);
            conn.Open();

            //selecting data
            string sql="SELECT * FROM employee";
            cmd=new MySqlCommand(sql,conn);
            MySqlDataAdapter adapter=new MySqlDataAdapter(cmd);
            DataSet dt=new DataSet();
            adapter.Fill(dt);

            if(dt.Rows.Count!=0){
                for(int i=0;i<dt.Rows.Count;i++){
                    string sid=dt.Rows[i]["eid"].ToString();
                    string name=dt.Rows[i]["name"].ToString();
                    Console.WriteLine(sid+" "+name);
                }
            }
            conn.Close();
        }
    }
}
```



```

    }

    static void Main(string[] args)
    {
        new Program();
    }
}

```

Output:

```

101 Ram
102 Shyam

```

Working with Data Reader (Connected Architecture)

(Selecting Records)

```

using MySql.Data.MySqlClient;

public class Program{
    static void Main(string[] args){
        string str="SERVER=localhost; DATABASE=csit; UID=root;
                    PASSWORD="";

        MySqlConnection conn=new MySqlConnection(str);
        conn.Open();

        //selecting data
        string sql="SELECT * FROM student";
        MySqlCommand cmd=new MySqlCommand(sql,conn);
        //using data reader
        MySqlDataReader reader=cmd.ExecuteReader();
        while(reader.Read()){
            int sid=reader.GetInt32("sid"); //or we can use 0
            string name=reader.GetString(1);
            string address=reader.GetString("address");
            Console.WriteLine(sid+"\t"+name+"\t"+address);
        }
    }
}

```

Output:

```

dotnet watch 🚀 Started
101    Ram    Btm
102    Raaju Poudel    Btm
dotnet watch 🛑 Exited

```

Web Applications using ASP.NET

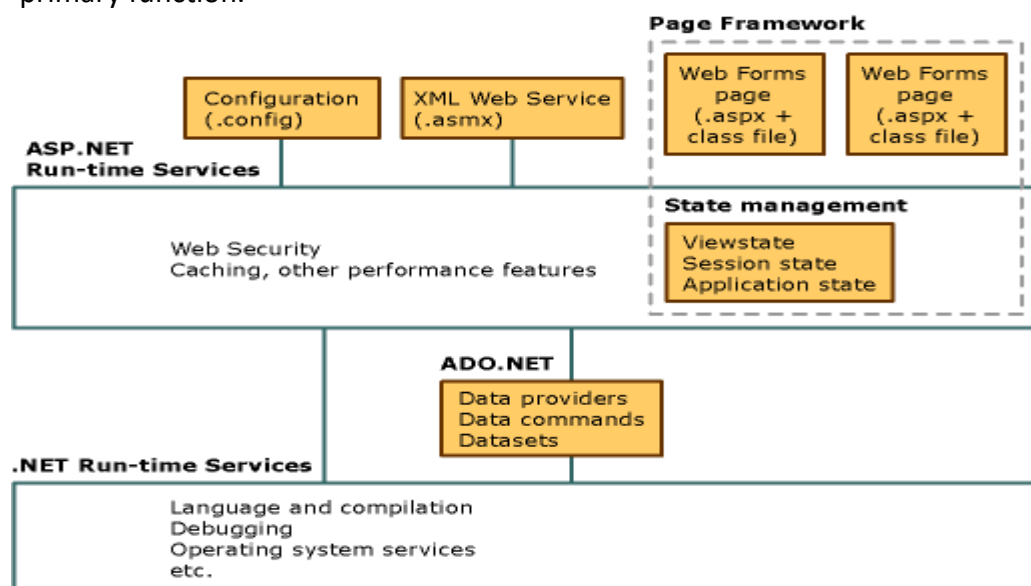
A Visual Studio Web application is built around ASP.NET. ASP.NET is a platform — including design-time objects and controls and a run-time execution context — for developing and running applications on a Web server.

ASP.NET Web applications run on a Web server configured with Microsoft Internet Information Services (IIS). However, you do not need to work directly with IIS. You can program IIS facilities using ASP.NET classes, and Visual Studio handles file management tasks such as creating IIS applications when needed and providing ways for you to deploy your Web applications to IIS.

Elements of ASP.NET Web Applications

Creating ASP.NET Web applications involves working with many of the same elements you use in any desktop or client-server application. These include:

- **Project management features** When creating an ASP.NET Web application, you need to keep track of the files you need, which ones need to be compiled, and which need to be deployed.
- **User interface** Your application typically presents information to users; in an ASP.NET Web application, the user interface is presented in Web Forms pages, which send output to a browser. Optionally, you can create output tailored for mobile devices or other Web appliances.
- **Components** Many applications include reusable elements containing code to perform specific tasks. In Web applications, you can create these components as XML Web services, which makes them callable across the Web from a Web application, another XML Web service, or a Windows Form, for example.
- **Data** Most applications require some form of data access. In ASP.NET Web applications, you can use ADO.NET, the data services that are part of the .NET Framework.
- **Security, performance, and other infrastructure features** As in any application, you must implement security to prevent unauthorized use, test and debug the application, tune its performance, and perform other tasks not directly related to the application's primary function.



Different Types of form controls in ASP.NET

Button Controls

ASP.NET provides three types of button control:

- **Button** : It displays text within a rectangular area.
- **Link Button** : It displays text that looks like a hyperlink.
- **Image Button** : It displays an image.

```
<asp:Button ID="Button1" runat="server" onclick="Button1_Click"
Text="Click" / >
```

Text Boxes and Labels

Text box controls are typically used to accept input from the user. A text box control can accept one or more lines of text depending upon the settings of the TextMode attribute.

Label controls provide an easy way to display text which can be changed from one execution of a page to the next. If you want to display text that does not change, you use the literal text.

```
<asp:TextBox ID="txtstate" runat="server" ></asp:TextBox>
```

Check Boxes and Radio Buttons

A check box displays a single option that the user can either check or uncheck and radio buttons present a group of options from which the user can select just one option.

To create a group of radio buttons, you specify the same name for the **GroupName** attribute of each radio button in the group. If more than one group is required in a single form, then specify a different group name for each group.

If you want check box or radio button to be selected when the form is initially displayed, set its Checked attribute to true. If the Checked attribute is set to true for multiple radio buttons in a group, then only the last one is considered as true.

```
<asp:CheckBox ID= "chkoption" runat= "Server">
</asp:CheckBox>
```

```
<asp:RadioButton ID= "rdboption" runat= "Server">
</asp: RadioButton>
```

List box Control

These control let a user choose from one or more items from the list. List boxes and drop-down lists contain one or more list items. These lists can be loaded either by code or by the **ListItemCollection** editor.

```
<asp:ListBox ID="ListBox1" runat="server">
</asp:ListBox>
```

HyperLink Control

The HyperLink control is like the HTML <a> element.

```
<asp:HyperLink ID="HyperLink1" runat="server">
    HyperLink
</asp:HyperLink>
```

Image Control

The image control is used for displaying images on the web page, or some alternative text, if the image is not available.

```
<asp:Image ID="Image1" ImageUrl="url" runat="server">
```

Drop down List Control

```
<asp:DropDownList ID="DropDownList1" runat="server"
</asp:DropDownList>
```

Launch another form on button click

```
protected void btnSelect_Click(object sender, EventArgs e)
{
    Response.Redirect("AnotherForm.aspx");
}
```

Example 1 – Creating a basic form in ASP.Net

Name:

Address:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="MyForm.aspx.cs" Inherits="WebApplication1.MyForm" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>This is my first application</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Label ID="lblName" runat="server" Text="Name:">
        </asp:Label>
        <asp:TextBox ID="txtName" runat="server"></asp:TextBox><br/>
        <asp:Label ID="lblAddress" runat="server" Text="Address:">
        </asp:Label>
        <asp:TextBox ID="txtAddress" runat="server"></asp:TextBox>
        <br/>
        <asp:Button ID="btnSubmit" runat="server" Text="Submit" />
    </form>
</body>
</html>
```

Example – 2 (Handling Events)

First Number

First Number

Result: 30

EventHandling.aspx

```
<form id="form1" runat="server">
    <div>
        <asp:Label ID="Label1" runat="server" Text="First Number">
            </asp:Label>
        <asp:TextBox ID="txtFirst" runat="server"></asp:TextBox>
        <br/><br/>
        <asp:Label ID="Label2" runat="server" Text="First Number">
            </asp:Label>
        <asp:TextBox ID="txtSecond" runat="server"></asp:TextBox>
        <br/><br/>
        <asp:Label ID="lblResult" runat="server" Text="Result:">
            </asp:Label> <br/><br/>
        <asp:Button ID="btnSubmit" runat="server" Text="Get Result"
            onClick="btnSubmit_Click"/>
    </div>
</form>
```

EventHandling.cs

```
protected void btnSubmit_Click(object sender, EventArgs e)
{
    int first = Convert.ToInt32(txtFirst.Text);
    int second = Convert.ToInt32(txtSecond.Text);
    int res = first + second;
    lblResult.Text = "Result: " + res;
}
```

Example 3 – Using Drop down list

Program

Selected Text: MCA Selected Value: 3

Dropdown.aspx

```
<form id="form1" runat="server">
    <div>
        <asp:Label ID="Label1" runat="server" Text="Program">
            </asp:Label>
        <asp:DropDownList ID="dropProgram" runat="server">
            </asp:DropDownList>
        <br/><br/>
        <asp:Label ID="lblSelected" runat="server" Text="Selected:">
            </asp:Label>
        <br/><br/>
        <asp:Button ID="btnSelect" runat="server" Text="Select"
            OnClick="btnSelect_Click" />
    </div>
</form>
```

Dropdown.cs

```
private void LoadData()
{
    List<ListItem> mylist=new List<ListItem>();
    mylist.Add(new ListItem("BCA","1"));
    mylist.Add(new ListItem("BBA","2"));
    mylist.Add(new ListItem("MCA","3"));
    mylist.Add(new ListItem("MBA","4"));
    dropProgram.Items.AddRange(mylist.ToArray());
}

protected void btnSelect_Click(object sender, EventArgs e)
{
    string text = dropProgram.SelectedItem.ToString();
    string value = dropProgram.SelectedValue;
    lblSelected.Text = "Selected Text: " + text + " Selected
        Value: " + value;
}
```

Example – 4 Using Radio button

Gender ☐ Male ☒ Female

Female Selected

Select

example.aspx

```
<form id="form1" runat="server">
    <div>
        <asp:Label ID="Label1" runat="server"
            Text="Gender"></asp:Label>
        <asp:RadioButton ID="radioMale" Text="Male" GroupName="gender"
            runat="server" />
        <asp:RadioButton ID="radioFemale" Text="Female"
            GroupName="gender" runat="server" />
    </div>
</form>
```

```

        <br/><br/>
        <asp:Label ID="lblSelected" runat="server" Text="Selected
            Radio:"> </asp:Label>
        <br/><br/>
        <asp:Button ID="btnSelect" runat="server" Text="Select"
            OnClick="btnSelect_Click" />
    </div>
</form>

```

example.cs

```

protected void btnSelect_Click(object sender, EventArgs e)
{
    if (radioMale.Checked)
        lblSelected.Text = "Male Selected";
    else
        lblSelected.Text = "Female Selected";
}

```

Example – 5 Using Check box

Gender ☐ Male ☒ Female

Female Selected

Select

example.aspx

```

<form id="form1" runat="server">
    <div>
        <asp:Label ID="Label1" runat="server"
            Text="Gender"></asp:Label>
        <asp:CheckBox ID="chkMale" Text="Male" GroupName="gender"
            runat="server" />
        <asp:CheckBox ID="chkFemale" Text="Female"
            GroupName="gender" runat="server" />
        <br/><br/>
        <asp:Label ID="lblSelected" runat="server" Text="Selected
            Radio:"> </asp:Label>
        <br/><br/>
        <asp:Button ID="btnSelect" runat="server" Text="Select"
            OnClick="btnSelect_Click" />
    </div>
</form>

```

example.cs

```

protected void btnSelect_Click(object sender, EventArgs e)
{
    if (chkMale.Checked)
        lblSelected.Text = "Male Selected";
    else
        lblSelected.Text = "Female Selected";
}

```

Event Handling

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. An **event handler** is responsible for handling events in C#.

An event handler, in C#, is a **method that contains the code that gets executed in response to a specific event that occurs in an application**. Event handlers are used in graphical user interface (GUI) applications to handle events such as button clicks and menu selections, raised by controls in the user interface.

The general syntax of an event is:

```
public void EventName (object sender, EventArgs e);
```

- **EventArgs e** is a parameter called e that contains the event data.
- **Object Sender** is a parameter called Sender that contains a reference to the control/object that raised the event.

Example:

```
protected void btn_Click (object sender, EventArgs e){  
    Button btn = sender as Button;  
    btn.Text = "clicked!";  
}
```

When Button is clicked, the btn_Click event handler will be fired. The "object sender" portion will be a reference to the button which was clicked.

Some of the events used in ASP.Net are:

Event	Attribute	Controls
Click	OnClick	Button, image button, link button, image map
Command	OnCommand	Button, image button, link button
TextChanged	OnTextChanged	Text box
SelectedIndexChanged	OnSelectedIndexChanged	Drop-down list, list box, radio button list, check box list.
CheckedChanged	OnCheckedChanged	Check box, radio button

Now you can give any example for event handling.

ASP.Net Example – 1

Creating Form and sending data in another page.

First Page

← → ↻ ⓘ 127.0.0.1:8080

Name:

Address:

Gender: ☒ Male ☐ Female

Course: ☐ Java ☒ C#

Program:

Second Page

← → ↻ ⓘ 127.0.0.1:8080/Second.aspx

Name: Raju Poudel
Address: Birtamod
Gender: Male
Program: BCA

First.aspx

```
<form runat="server">
    Name:
    <asp:TextBox runat="server" id="name"/>
    <br><br>
    Address:
    <asp:TextBox runat="server" id="address"/>
    <br><br>
    Gender:
    <asp:RadioButton runat="server" Text="Male"
        GroupName="gender" id="male"/>
    <asp:RadioButton runat="server" Text="Female"
        Checked="true" GroupName="gender" id="female"/>
    <br><br>
    Course:
    <asp:CheckBox runat="server" Text="Java" id="java"/>
    <asp:CheckBox runat="server" Text="C#" id="csharp"/>
    <br><br>
    Program:
    <asp:DropDownList runat="server" id="program">
```

```

        <asp:ListItem>BCA</asp:ListItem>
        <asp:ListItem>BBA</asp:ListItem>
        <asp:ListItem>BIM</asp:ListItem>
    </asp:DropDownList>
    <br><br>

    <asp:Button runat="server" Text="Submit"
        OnClick="Submit"/>
</form>

```

First.aspx.cs

```

public void Submit(object sender, EventArgs args)
{
    Second.name = name.Text;
    Second.address = address.Text;

    string gender = "";
    if (male.Checked)
        gender = "Male";
    else
        gender = "Female";

    Second.gender = gender;

    Second.program = program.Text;
    Response.Redirect("Second.aspx");
}

```

Second.aspx.cs

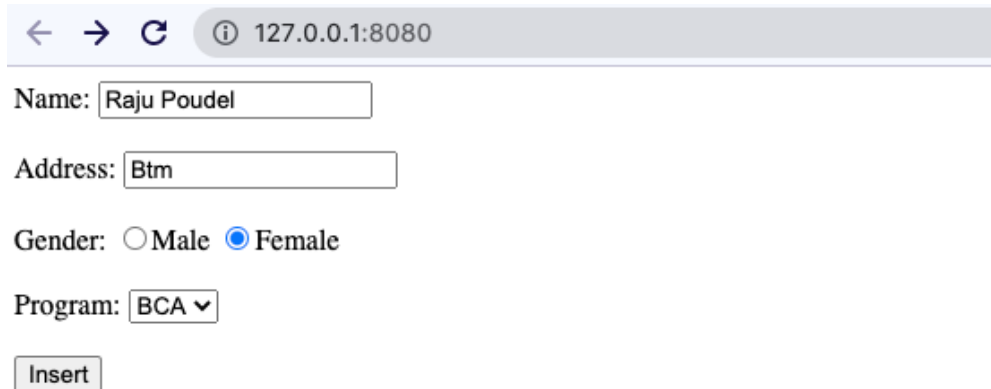
```

public partial class Second : System.Web.UI.Page
{
    public static string name, address, gender, program;

    public void Page_Load(object sender, EventArgs args)
    {
        Response.Write("Name: "+name);
        Response.Write("<br>Address: " + address);
        Response.Write("<br>Gender: " + gender);
        Response.Write("<br>Program: " + program);
    }
}

```

Inserting Data using ADO in ASP.Net



Name:

Address:

Gender: ☐ Male ☒ Female

Program:

Default.aspx

```
<form runat="server">
    Name:
    <asp:TextBox runat="server" id="name"/>
    <br><br>
    Address:
    <asp:TextBox runat="server" id="address"/>
    <br><br>
    Gender:
    <asp:RadioButton runat="server" Text="Male"
        GroupName="gender" id="male"/>
    <asp:RadioButton runat="server" Text="Female"
        Checked="true" GroupName="gender" id="female"/>
    <br><br>
    Program:
    <asp:DropDownList runat="server" id="program">
        <asp:ListItem>BCA</asp:ListItem>
        <asp:ListItem>BBA</asp:ListItem>
        <asp:ListItem>BIM</asp:ListItem>
    </asp:DropDownList>
    <br><br>
    <asp:Button runat="server" Text="Insert" OnClick="Insert"/>
</form>
```

Default.aspx.cs

```
public void Insert(object sender, EventArgs args)
{
    string nm, addr, gen, pro;
    //getting values
    nm = name.Text;
    addr = address.Text;

    if (male.Checked)
        gen = "Male";
```

```

else
    gen = "Female";

pro = program.Text;

//inserting in database
string constr = "SERVER=localhost; DATABASE=fifth;
    UID=root; PASSWORD=";
 MySqlConnection conn = new MySqlConnection(constr);
 conn.Open();
 string sql = "INSERT INTO student
    (name,address,gender,program)
    VALUES('"+nm+"','"+addr+"','"+gen+"','"+pro+"')";

 MySqlCommand cmd = new MySqlCommand(sql,conn);
 cmd.ExecuteNonQuery();
 Response.Write("Data Inserted Successfully!");
}

```

Complete CRUD Operation

Default.aspx

```

<form id="form1" runat="server">
    Student Id:
    <asp:TextBox runat="server" ID="sid"/>
    <br /><br />
    Student Name:
    <asp:TextBox runat="server" ID="name" />
    <br /><br />

    Gender:
    <asp:RadioButton runat="server" ID="male" GroupName="gender"
        Checked="true" Text="Male" />
    <asp:RadioButton runat="server" ID="female" GroupName="gender"
        Text="Female" />
    <br /><br />

    Course:
    <asp:DropDownList runat="server" ID="course">
        <asp:ListItem>BCA</asp:ListItem>
        <asp:ListItem>BBA</asp:ListItem>
        <asp:ListItem>BIM</asp:ListItem>
    </asp:DropDownList>
    <br /><br />

    <asp:Button runat="server" Text="Insert" OnClick="Insert"/>
    <asp:Button runat="server" Text="Delete" OnClick="Delete" />
    <asp:Button runat="server" Text="Select" OnClick="Select" />
    <br><br>

    <asp:DataGrid runat="server" ID="table" />
</form>

```

Default.aspx.cs

```
using System;
using MySql.Data.MySqlClient;
using System.Data;

namespace BCA
{
    public partial class WebForm1 : System.Web.UI.Page
    {
        MySqlConnection conn;
        protected void Page_Load(object sender, EventArgs e)
        {
            string constr = "SERVER=localhost;DATABASE=fifth;
                             UID=root;PASSWORD=";
            conn = new MySqlConnection(constr);
            conn.Open();
            Display();
        }

        protected void Insert(object sender, EventArgs e)
        {
            string nm, gen, cour;
            nm = name.Text;
            cour = course.Text;

            if (male.Checked)
                gen = "Male";
            else
                gen = "Female";

            string sql = "INSERT INTO student(name,gender,course)
                         VALUES('"+nm+"','"+gen+"','"+cour+"')";
            MySqlCommand cmd = new MySqlCommand(sql,conn);
            cmd.ExecuteNonQuery();

            Response.Write("Data Inserted Successfully!");
            Display();
        }

        public void Delete(object sender, EventArgs args)
        {
            int si;
            si = Convert.ToInt32(sid.Text);
            string sql = "DELETE FROM student WHERE sid='"+si+"'";
            MySqlCommand cmd = new MySqlCommand(sql,conn);
            cmd.ExecuteNonQuery();
            Response.Write("Data Deleted Successfully!");
            Display();
        }
    }
}
```

```

    }

    public void Select(object sender, EventArgs args)
    {
        Display();
    }

    public void Display()
    {
        string sql = "select * from student";
        MySqlCommand cmd = new MySqlCommand(sql, conn);

        MySqlDataAdapter adapter = new MySqlDataAdapter(cmd);
        DataTable dt = new DataTable();
        adapter.Fill(dt);

        table.DataSource = dt;
        table.DataBind();
    }
}

```