Unit - 4 Advanced C# [14 Hrs]

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.*

```
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](calling_method_name);
```

```
DelegateTest obj = new DelegateTest (MyMethod);
       // here,
       // " DelegateTest" is delegate name.
       // " obj" is instance_name
       // " MyMethod" is the calling method.
```

The following defines a delegate type called **Transformer**:

```
delegate int Transformer (int x);
```

Transformer is **compatible** with any method with an **int return type and a single int parameter**, such as this:

```
static int Square (int x) { return x * x; }
```

or

```
static int Square (int x) => x * x;
```

Assigning a method to a delegate variable creates a delegate instance:

```
Transformer t = new Transformer (Square);
```

Or

```
Transformer t = Square;
```

which can be invoked in the same way as a method:

```
int answer = t(3); // answer is 9
```

Example of simple delegate is shown below:

```
public delegate int MyDelegate(int x);
    class DelegateTest
    {
        static int MyMethod(int x)
        {
            return x * x;
        }

        static void Main(string[] agrs)
        {
            MyDelegate del = new MyDelegate(MyMethod);
            int res = del(5); //25
            Console.WriteLine("Result is : "+res);
            Console.ReadKey();
        }
    }
```

Output:

```
Result is : 25
```

Example 2

```
using System;

delegate int NumberChanger(int n);

   class TestDelegate {
      static int num = 10;

      public static int AddNum(int p) {
         num += p;
         return num;
      }
      public static int MultNum(int q) {
         num *= q;
         return num;
      }
      public static int getNum() {
         return num;
      }
      static void Main(string[] args) {
         //create delegate instances
         NumberChanger nc1 = new NumberChanger(AddNum);
         NumberChanger nc2 = new NumberChanger(MultNum);

         //calling the methods using the delegate objects
         nc1(25);
         Console.WriteLine("Value of Num: {0}", getNum());
         nc2(5);
         Console.WriteLine("Value of Num: {0}", getNum());
         Console.ReadKey();
      }
   }
```

Output:

```
Value of Num: 35
Value of Num: 175
```

Example 3

```
class Test
    {
        public delegate void addnum(int a, int b);
        public delegate void subnum(int a, int b);

        // method "sum"
        public void sum(int a, int b)
        {
            Console.WriteLine("(100 + 40) = {0}", a + b);
        }
```

```
    // method "subtract"
      public void subtract(int a, int b)
      {
          Console.WriteLine("(100 - 60) = {0}", a - b);
      }

      // Main Method
      public static void Main(String[] args)
      {
          addnum del_obj1 = new addnum(obj.sum);
          subnum del_obj2 = new subnum(obj.subtract);

          // pass the values to the methods by delegate object
          del_obj1(100, 40);
          del_obj2(100, 60);

          // These can be written as using
          // "Invoke" method
          // del_obj1.Invoke(100, 40);
          // del_obj2.Invoke(100, 60);
      }
}
```

Output:

```
(100 + 40) = 140
(100 - 60) = 40
```

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 multicasting delegates

```
using System;

delegate int NumberChanger(int n);
   class TestDelegate {
      static int num = 10;

      public static int AddNum(int p) {
         num += p;
         return num;
      }
      public static int MultNum(int q) {
         num *= q;
         return num;
      }
      public static int getNum() {
         return num;
      }
      static void Main(string[] args) {
         //create delegate instances
         NumberChanger nc;
         NumberChanger nc1 = new NumberChanger(AddNum);
         NumberChanger nc2 = new NumberChanger(MultNum);

         nc = nc1;
         nc += nc2;

         //calling multicast
         nc(5);
         Console.WriteLine("Value of Num: {0}", getNum());
         Console.ReadKey();
      }
   }
```

Output

```
Value of Num: 75
```

Delegates Mapping with Instance and Static Method

Created ClassA contains instance & static method,

```
class A
{
    public void InstanceMethod()
    {
        Console.WriteLine("InstanceMethod");
    }

    public static void StaticMethod()
    {
        Console.WriteLine("StaticMethod");
    }
}
```

Above class methods are used using delegate,

```
class Program
{
    //declaration of delegate
    delegate void Del();


    static void Main(string[] args)
    {
        A objA = new A();

        //Instance Method associated with delegate instance
        Del d = objA.InstanceMethod;
        d();



        //Static method associated with same delegate instance
        d = A.StaticMethod;
        d();

        Console.ReadLine();

    }
```

Output

Instance Method
Static Method

So using delegate, we can associate instance and static method under same delegate instance.

Delegates Vs Interfaces in C#

S.N.	DELEGATE	INTERFACE
1	It could be a method only.	It contains both methods and properties.
2	It can be applied to one method at a time.	If a class implements an interface, then it will implement all the methods related to that interface.
3	If a delegate available in your scope you can use it.	Interface is used when your class implements that interface, otherwise not.
4	Delegates can be implemented any number of times.	Interface can be implemented only one time.
5	It is used to handling events.	It is not used for handling events.
6	When you access the method using delegates you do not require any access to the object of the class where the method is defined.	When you access the method you need the object of the class which implemented an interface.
7	It does not support inheritance.	It supports inheritance.
8	It created at run time.	It created at compile time.
9	It can implement any method that provides the same signature with the given delegate.	If the method of interface implemented, then the same name and signature method override.
10	It can wrap any method whose signature is similar to the delegate and does not consider which from class it belongs.	A class can implement any number of interfaces, but can only override those methods which belongs to the interfaces.

Delegate Compatibility

Type compatibility

Delegate types are all incompatible with one another, even if their signatures are the same:

```
delegate void D1();
delegate void D2();
...

D1 d1 = Method1;
D2 d2 = d1;                          // Compile-time error
```

The following, however, is permitted:

```
D2 d2 = new D2 (d1);
```

Delegate instances are considered equal if they have the same method targets:

```
delegate void D();
...

D d1 = Method1;
D d2 = Method1;
Console.WriteLine (d1 == d2);         // True
```

Multicast delegates are considered equal if they reference the same methods *in the same order.*

Parameter compatibility

When you call a method, you can supply arguments that have more specific types than the parameters of that method. This is ordinary polymorphic behaviour. For exactly the same reason, a delegate can have more specific parameter types than its method target. This is called **contravariance**.

Here's an example:

```
delegate void StringAction (string s);

class Test
{
  static void Main()
  {
    StringAction sa = new StringAction (ActOnObject);
    sa ("hello");
  }

  static void ActOnObject (object o) => Console.WriteLine (o);   // hello
}
```

In this case, the String Action is invoked with an argument of type string. When the argument is then relayed to the target method, the argument gets implicitly up cast to an object.

Return type compatibility

If you call a method, you may get back a type that is more specific than what you asked for. This is ordinary polymorphic behaviour. For exactly the same reason, a delegate's target method may return a more specific type than described by the dele- gate. This is called **covariance**. For example:

```
delegate object ObjectRetriever();

class Test
{
  static void Main()
  {
    ObjectRetriever o = new ObjectRetriever (RetrieveString);
    object result = o();
    Console.WriteLine (result);      // hello
  }
  static string RetrieveString() => "hello";
}
```

Generic Delegate Types

A delegate type may contain generic type parameters. For example:

```
public delegate T Transformer (T arg);
```

With this definition, we can write a generalized Transform utility method that works on any type:

```
public class Util
{
  public static void Transform<T> (T[] values, Transformer<T> t)
  {
    for (int i = 0; i < values.Length; i++)
      values[i] = t (values[i]);
  }
}

class Test
{
  static void Main()
  {
    int[] values = { 1, 2, 3 };
    Util.Transform (values, Square);      // Hook in Square
    foreach (int i in values)
      Console.Write (i + "  ");           // 1   4   9
  }

  static int Square (int x) => x * x;
}
```

Func and Action Delegates

The Func and Action generic delegates were introduced in the .NET Framework version 3.5. Whenever we want to use delegates in our examples or applications, typically we use the following procedure:

- Define a custom delegate that matches the format of the method.
- Create an instance of a delegate and point it to a method.
- Invoke the method.

But, using these 2 Generics delegates we can simply eliminate the above procedure.

Since both the delegates are generic, you will need to specify the underlying types of each parameter as well while pointing it to a function. For for example Action<type,type,type……>

Action<>

- This Action<> generic delegate; points to a method that takes up to 16 Parameters and returns void.

Func<>

- The generic Func<> delegate is used when we want to point to a method that returns a value.
- This delegate can point to a method that takes up to 16 Parameters and returns a value.

- Always remember that the final parameter of Func<> is always the return value of the method. (For example, Func< int, int, string>, this version of the Func<> delegate will take 2 int parameters and returns a string value.)

Example is shown below

```
class MethodCollections
    {
        //Methods that takes parameters but returns nothing:

        public static void PrintText()
        {
            Console.WriteLine("Text Printed with the help of Action");
        }
        public static void PrintNumbers(int start, int target)
        {
            for (int i = start; i <= target; i++)
            {
                Console.Write(" {0}",i);
            }
            Console.WriteLine();
        }
        public static void Print(string message)
        {
            Console.WriteLine(message);
        }

        //Methods that takes parameters and returns a value:

        public static int Addition(int a, int b)
        {
            return a + b;
        }

        public static string DisplayAddition(int a, int b)
        {
           return string.Format("Addition of {0} and {1} is {2}",a,b,a+b);

        }

        public static string SHowCompleteName(string firstName, string las
                  tName)
        {
          return string.Format("Your Name is {0} {1}",firstName,lastName);

        }
        public static int ShowNumber()
        {
            Random r = new Random();
            return r.Next();
        }
    }
```

```
class Program
    {
        static void Main(string[] args)
        {
            Action printText = new Action(MethodCollections.PrintText);
            Action<string> print = new Action<string>(MethodCollections.Pr
                                                int);
            Action<int, int> printNumber = new Action<int, int>(MethodColl
                                                    ections.PrintNumbers);

            Func<int, int,int> add1 = new Func<int, int, int>(MethodCollec
                                                    tions.Addition);
            Func<int, int, string> add2 = new Func<int, int, string>(Metho
                                        dCollections.DisplayAddition);
            Func<string, string, string> completeName = new Func<string, s
                  tring, string>(MethodCollections.SHowCompleteName);
           Func<int> random = new Func<int>(MethodCollections.ShowNumber);

           Console.WriteLine("\n***************** Action<> Delegate Method
                                        ***************\n");
            printText();    //Parameter: 0 , Returns: nothing
            print("Abhishek");  //Parameter: 1 , Returns: nothing
            printNumber(5, 20); //Parameter: 2 , Returns: nothing
            Console.WriteLine();
            Console.WriteLine("***************** Func<> Delegate Methods **
                                        ***************\n");
            int addition = add1(2, 5);  //Parameter: 2 , Returns: int
            string addition2 = add2(5, 8);  //Parameter:2 ,Returns: string

            string name = completeName("Abhishek", "Yadav"); //Parameter:2
                                , Returns: string
           int randomNumbers = random();   ////Parameter: 0 , Returns: int
            Console.WriteLine("Addition: {0}",addition);
            Console.WriteLine(addition2);
            Console.WriteLine(name);
            Console.WriteLine("Random Number is: {0}",randomNumbers);

            Console.ReadLine();
        }
    }
```

```
****************** Action<> Delegate Methods ****************

Text Printed with the help of Action
Abhishek
 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

****************** Func<> Delegate Methods ******************

Addition: 7
Addition of 5 and 8 is 13
Your Name is Abhishek Yadav
Random Number is: 1848661255
```

Events

Events are user actions such as key press, clicks, mouse movements, etc., or some occurrence such as system generated notifications. Applications need to respond to events when they occur. For example, interrupts. Events are used for inter-process communication.

The class that sends or raises an event is called a Publisher and class that receives or handle the event is called "Subscriber".

Following are the key points about Event,

1. Event Handlers in C# return void and take two parameters.
2. The First parameter of Event - Source of Event means publishing object.
3. The Second parameter of Event - Object derived from EventArgs.
4. The publishers determines when an event is raised and the subscriber determines what action is taken in response.
5. An Event can have so many subscribers.
6. Events are basically used for the single user action like button click.
7. If an Event has multiple subscribers then event handlers are invoked synchronously.

Declaring Events

To declare an event inside a class, first of all, you must declare a delegate type for the even as:

```
public delegate string MyDelegate(string str);
```

then, declare the event using the **event** keyword –

```
event MyDelegate delg;
```

The preceding code defines a delegate named MyDelegate and an event named delg, which invokes the delegate when it is raised.

To declare an event inside a class, first a Delegate type for the Event must be declared like below:

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

Defining an event is a two-step process.

- First, you need to define a delegate type that will hold the list of methods to be called when the event is fired.
- Next, you declare an event using the event keyword.

To illustrate the event, we are creating a console application. In this iteration, we will define an event to add that is associated to a single delegate **DelEventHandler**.

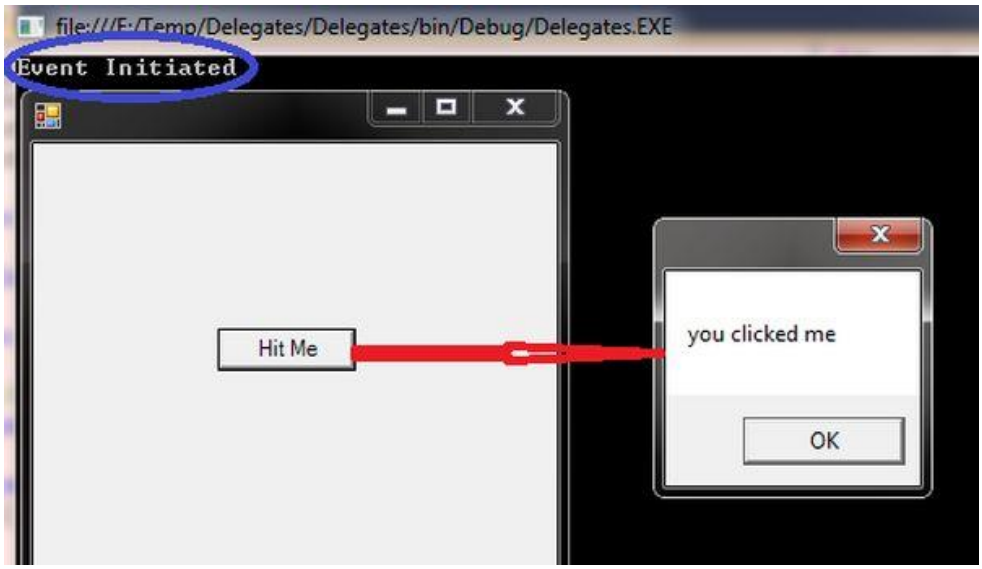
```
using System;
   public delegate void DelEventHandler();

    class Program
    {
        public static event DelEventHandler add;
```

```
        static void USA()
          {
              Console.WriteLine("USA");
          }

          static void India()
          {
              Console.WriteLine("India");
          }

          static void England()
          {
              Console.WriteLine("England");
          }


          static void Main(string[] args)
          {

              add += new DelEventHandler(USA);
              add += new DelEventHandler(India);
              add += new DelEventHandler(England);
              add.Invoke();

              Console.ReadLine();
          }

      }
```

Implementing event in a button click

```
using System;
using System.Drawing;
using System.Windows.Forms;

    //custom delegate
    public delegate void DelEventHandler();

    class Program :Form
    {
        //custom event
        public event DelEventHandler add;

        public Program()
        {
            // design a button over form
            Button btn = new Button();
            btn.Parent = this;
            btn.Text = "Hit Me";
            btn.Location = new Point(100,100);

            //Event handler is assigned to
```

```
            // the button click event
            btn.Click += new EventHandler(onClcik);
            add += new DelEventHandler(Initiate);

            //invoke the event
            add();
        }
        //call when event is fired
        public void Initiate()
        {
            Console.WriteLine("Event Initiated");
        }

        //call when button clicked
        public void onClcik(object sender, EventArgs e)
        {
            MessageBox.Show("You clicked me");
        }
        static void Main(string[] args)
        {
            Application.Run(new Program());

            Console.ReadLine();
        }
}
```



Can we use Events without Delegate?

No, Events use Delegates internally. Events are encapsulation over Delegates. There is already defined Delegate "EventHandler" that can be used like below:

public event EventHandler MyEvents;

So, it also used Delegate Internally.

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..
};
```

Example:

```
using System;

class GFG {

    public delegate void petanim(string pet);

    // Main method
    static public void Main()
    {

        // An anonymous method with one parameter
        petanim p = delegate(string mypet)
        {
            Console.WriteLine("My favorite pet is: {0}",
                                                mypet);
        };
        p("Dog");
    }
}
```

Output:

My favorite pet is: Dog

You can also use an anonymous method as an event handler.

```
MyButton.Click += delegate(Object obj, EventArgs ev)
{
    System.Windows.Forms.MessageBox.Show("Complete without
                 error...!!");
}
```

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 '=>' 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 => 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 => { statements };

Basic example of lambda expression:

```
class LambdaTest
    {
        static int test1() => 5;
        static int test2(int x) => x + 10;

        static void Main(string[] args)
        {
            int x=test1();
            int res = test2(x);
            Console.WriteLine("Result is: "+res);
        }
    }
```

Output:

```
Result is: 15
```

Unlike an expression lambda, a statement lambda can contain multiple statements separated by semicolons. It is used with delegates.

```
delegate void ModifyInt(int input);

ModifyInt addOneAndTellMe = x =>
{
    int result = x + 1;
    Console.WriteLine(result);
};
```

Exception Handling

A try statement specifies a code block subject to error-handling or clean-up code. The try block must be followed by a catch block, a finally block, or both. The catch block executes when an error occurs in the try block. The finally block executes after execution leaves the try block (or if present, the catch block), to per- form clean-up code, whether or not an error occurred.

A catch block has access to an Exception object that contains information about the error. You use a catch block to either compensate for the error or re throw the exception. You re throw an exception if you merely want to log the problem, or if you want to re throw a new, higher-level exception type.

A finally block adds determinism to your program: the CLR endeavours to always execute it. It's useful for clean-up tasks such as closing network connections.

A try statement looks like this:

```
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
}
```

Consider the following program:

```
class Test
{
  static int Calc (int x) => 10 / x;

  static void Main()
  {
    int y = Calc (0);
    Console.WriteLine (y);
  }
}
```

Because x is zero, the runtime throws a DivideByZeroException, and our program terminates. We can prevent this by catching the exception as follows:

```
class Test
{
  static int Calc (int x) => 10 / x;

  static void Main()
  {
    try
    {
      int y = Calc (0);

      Console.WriteLine (y);
    }
    catch (DivideByZeroException ex)
    {
      Console.WriteLine ("x cannot be zero");
    }
    Console.WriteLine ("program completed");
  }
}

OUTPUT:
x cannot be zero
program completed
```

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:

```
class Test
{
  static void Main (string[] args)
  {
    try
    {
      byte b = byte.Parse (args[0]);
      Console.WriteLine (b);
    }
    catch (IndexOutOfRangeException ex)
    {
      Console.WriteLine ("Please provide at least one argument");
    }
    catch (FormatException ex)
    {
      Console.WriteLine ("That's not a number!");
    }
    catch (OverflowException ex)
    {
      Console.WriteLine ("You've given me more than a byte!");
    }
  }
}
```

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

Throwing Exceptions

Exceptions can be thrown either by the runtime or in user code. In this example, Display throws a System.ArgumentNullException:

```
class Test
{
  static void Display (string name)
  {
    if (name == null)
      throw new ArgumentNullException (nameof (name));

    Console.WriteLine (name);
  }

  static void Main()
  {

     try { Display (null); }
     catch (ArgumentNullException ex)
     {
       Console.WriteLine ("Caught the exception");
     }
   }
}
```

Re-throwing an exception

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

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

Common Exception Types

System.ArgumentException
Thrown when a function is called with a bogus argument. This generally indi- cates a program bug.

System.ArgumentNullException
Subclass of ArgumentException that's thrown when a function argument is (unexpectedly) null.

System.ArgumentOutOfRangeException
Subclass of ArgumentException that's thrown when a (usually numeric) argu- ment is too big or too small. For example, this is thrown when passing a nega- tive number into a function that accepts only positive values.

System.InvalidOperationException
Thrown when the state of an object is unsuitable for a method to successfully execute, regardless of any particular argument values. Examples include read- ing an unopened file or getting the next element from an enumerator where the underlying list has been modified partway through the iteration.

System.NotSupportedException
Thrown to indicate that a particular functionality is not supported. A good example is calling the Add method on a collection for which IsReadOnly returns true.

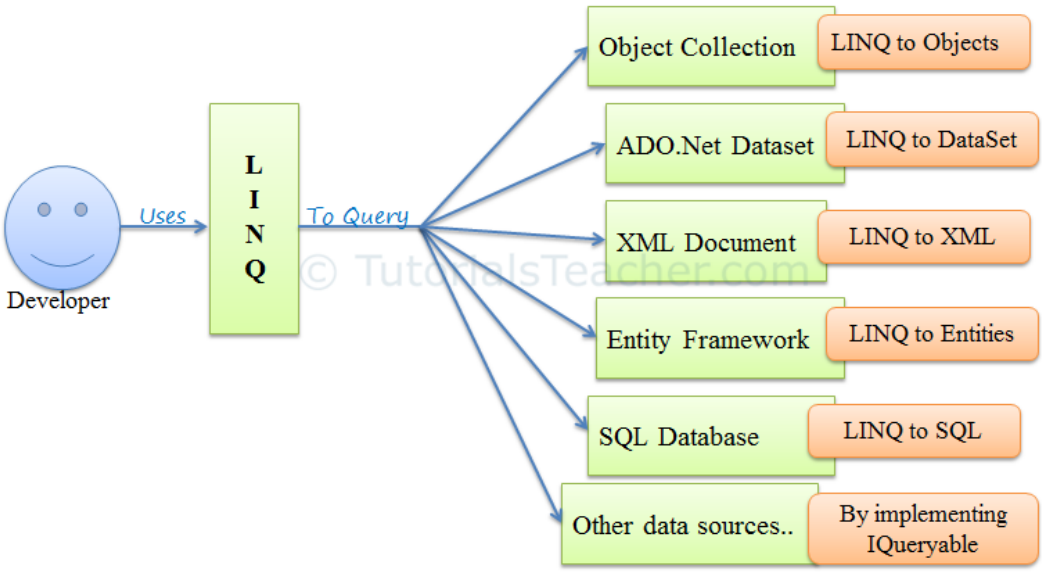
System.NotImplementedException
Thrown to indicate that a function has not yet been implemented.

System.ObjectDisposedException
Thrown when the object upon which the function is called has been disposed.

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 uses object-oriented approach on the result set and not to worry about transforming different formats of results into objects.

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

Example: LINQ Query to Array

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

Example: LINQ Query to List

```
// string collection
List<string> stringList = new List<string>() {
    "C# Tutorials",
    "VB.NET Tutorials",
    "Learn C++",
    "MVC Tutorials" ,
    "Java"
};

var result = from s in stringList
             where s.Contains("Tutorials")
             select s;

foreach(string value in result)
    Console.Write(value + " ");
```

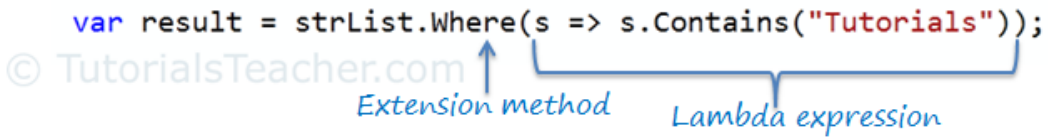
LINQ Method

The following is a sample LINQ method syntax query that returns a collection of strings which contains a word "Tutorials". **We use lambda expression for this purpose**.

Example: LINQ Method Syntax in C#

```
// string collection
List<string> stringList = new List<string>() {
    "C# Tutorials",
    "VB.NET Tutorials",
    "Learn C++",
    "MVC Tutorials" ,
    "Java"
};

// LINQ Query Syntax
var result = stringList.Where(s => s.Contains("Tutorials"));
```

The following figure illustrates the structure of LINQ method syntax.



Use of lambda Expression to LINQ

Example 1

```
using System;
using System.Collections.Generic;
using System.Linq;
public class demo
{
    public static void Main()
    {
        List<int> list = new List<int>() { 1, 2, 3, 4, 5, 6 };
        List<int> evenNumbers = list.FindAll(x => (x % 2) == 0);
```

```
        foreach (var num in evenNumbers)
        {
            Console.Write("{0} ", num);
        }
        Console.WriteLine();
        Console.Read();

    }
}
```

Output:

2 4 6

Example 2

```
using System;
using System.Collections.Generic;
using System.Linq;
class Dog
{
    public string Name { get; set; }
    public int Age { get; set; }
}
class demo{
    static void Main()
    {
        List<Dog> dogs = new List<Dog>() {
            new Dog { Name = "Rex", Age = 4 },
            new Dog { Name = "Sean", Age = 0 },
            new Dog { Name = "Stacy", Age = 3 }
         };
         var names = dogs.Select(x => x.Name);
         foreach (var name in names)
         {
             Console.WriteLine(name);

         }
         Console.Read();
    }
}
```

Output:

Rex
Sean
Stacy

Sorting using a lambda expression

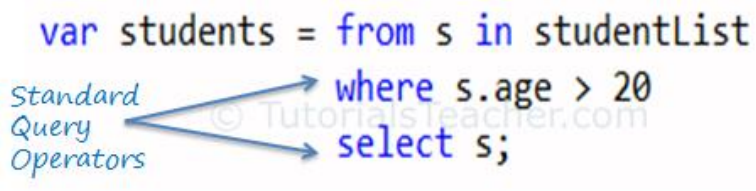
```
var sortedDogs = dogs.OrderByDescending(x => x.Age);
foreach (var dog in sortedDogs)
{
  Console.WriteLine("Dog {0} is {1} years old.", dog.Name, dog.Age
);
}
```

Output:

```
Dog Rex is 4 years old.
Dog Stacy is 3 years old.
Dog Sean is 0 years old.
```

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



Some examples of LINQ Query

Where Condition – Example 1

```
using System;
using System.Collections.Generic;
using System.Linq;
namespace BCA
{
    class LinqTest
    {
        static void Main(string[] args)
        {
            List<string> names = new List<string>() {
                  "Ram","Shyam","Hari","Gita"};
            //single condition
            //var result = names.Where(s => s.Contains("Ram"));

            //Multiple Condition
            var result = names.Where(s=>s.Contains("Ram") ||
                              s.Contains("Gita"));
            foreach (string val in result)
            {
                Console.WriteLine(val);
            }

            Console.ReadLine();
        }
    }
}
```

Output:

Ram
Gita

Where Condition – Example 2 (with object list)

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

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

    class LinqTest
    {
        static void Main(string[] args)
```

```
        {
            List<Student> mylist = new List<Student>(){
            new Student(1,"Ram","Btm"),
            new Student(2, "Hari", "Ktm"),
            new Student(3,"Shyam","Btm"),
            new Student(4, "Gita", "Ktm")
            };

            //var result = mylist.Where(s=>s.address.Contains("Btm"));
            var result = mylist.Where(s => s.address.Equals("Btm") &&
                                s.sid.Equals(1));
            Console.WriteLine("Sid\tName\tAddress");
            foreach (var res in result)
            {

Console.WriteLine(res.sid+"\t"+res.name+"\t"+res.address);
            }
            Console.ReadLine();
        }
    }
```

Output:

Sid	Name	Address
1	Ram	Btm

Joining multiple lists – (join, concat, union)

```
class LinqTest
    {
        static void Main(string[] args)
        {
            List<string> names = new List<string>() {
                      "Ram","Shyam","Hari"};
            List<string> address = new List<string>()
                      {"Btm","Ktm","Btm" };
            /*using join
            var result = names.Join(address,
                      str1 => str1,
                      str2 => str2,
                      (str1, str2) => str1);*/
            /*using concat
            var result = names.Concat(address);*/

            //using union
            var result = names.Union(address);
            foreach (var res in result)
            {
                Console.WriteLine(res);
            }

            Console.ReadLine();
        }    }
```

Using aggregate functions – Example 1

```
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();
        }
    }
```

Using aggregate functions – Example 2 (Object List)

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

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

    class LinqTest
    {
        static void Main(string[] args)
        {
            List<Student> mylist = new List<Student>(){
            new Student(1,"Ram","Btm"),
            new Student(2, "Hari", "Ktm"),
            new Student(3,"Shyam","Btm"),
            new Student(4, "Gita", "Ktm")
            };

            int maxId = mylist.Max(s=>s.sid);
            int count = mylist.Count();
            Console.WriteLine("Max Id="+maxId);
```

```
            Console.WriteLine("Total Students="+count);

            Console.ReadLine();
        }
    }
```

Output

```
Max Id=4
Total Students=4
```

Using Order By

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

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

    class LinqTest
    {
        static void Main(string[] args)
        {
            List<Student> mylist = new List<Student>(){
            new Student(1,"Ram","Btm"),
            new Student(2, "Hari", "Ktm"),
            new Student(3,"Shyam","Btm"),
            new Student(4, "Gita", "Ktm")
            };

            //var result = mylist.OrderBy(s=>s.name);
            //var result = mylist.OrderByDescending(s => s.name);

            //select name and address of student order by name in
                ascending where address is Ktm
            var result = mylist.Where(s=>s.address.Equals("Ktm")).
                          OrderBy(s=>s.name);

            Console.WriteLine("Name\tAddress");
            foreach (var res in result)
            {
                Console.WriteLine(res.name + "\t" + res.address);
            }
```

```
            Console.ReadLine();
        }
    }
```

Output:

Name	Address
Gita	Ktm
Hari	Ktm

Using Group By

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

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

    class LinqTest
    {
        static void Main(string[] args)
        {
            List<Student> mylist = new List<Student>(){
            new Student(1,"Ram","Btm"),
            new Student(2, "Hari", "Ktm"),
            new Student(3,"Shyam","Btm"),
            new Student(4, "Gita", "Ktm")
            };

            //select records group by address
            var groupResult = mylist.GroupBy(s=>s.address);
            foreach (var result in groupResult)
            {
                Console.WriteLine("Group Key: " + result.Key);
                      //Each group has key
                Console.WriteLine("Sid\tName\tAddress");
                foreach (var res in result)
                {

                      Console.WriteLine(res.sid+"\t"+res.name+"\t"
                      + res.address);
                }
```

```
            }
            Console.ReadLine();
        }
}
```

Output:

Group Key: Btm

Sid	Name	Address
1	Ram	Btm
3	Shyam	Btm

Group Key: Ktm

Sid	Name	Address
2	Hari	Ktm
4	Gita	Ktm

Working with Databases

Comparison between ADO and ADO.NET:

ADO is a Microsoft technology. It stands for ActiveX Data Objects. It is a Microsoft Active-X component. ADO is automatically installed with Microsoft IIS. It is a programming interface to access data in a database

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.

ADO : ActiveX Data Objects **and ADO.Net are two different ways to access database in Microsoft.**

ADO	**ADO.Net**
ADO is base on **COM** : **Component Object Modelling** based.	ADO.Net is based on **CLR** : **Common Language Runtime** based.
ADO stores data in **binary** format.	ADO.Net stores data in **XML** format i.e. parsing of data.
ADO can't be integrated with XML because ADO have limited access of XML.	ADO.Net can be integrated with XML as having robust support of XML.
In ADO, data is provided by **RecordSet**.	In ADO.Net data is provided by **DataSet** or **DataAdapter**.
ADO is **connection** oriented means it requires continuous active connection.	ADO.Net is **disconnected**, does not need continuous connection.
ADO gives rows as single table view, it scans sequentially the rows using **MoveNext** method.	ADO.Net gives rows as collections so you can access any record and also can go through a table via loop.
In ADO, You can create only **Client** side cursor.	In ADO.Net, You can create both **Client & Server** side cursor.
Using a single connection instance, ADO can not handle multiple transactions.	Using a single connection instance, ADO.Net can handle multiple transactions.

Working with Connection, Command:

Working with Connection:

Process of creating connection:

For SQL Server

Note: Sql Server must be installed

```
String conn_str;
SqlConnection connection;
conn_str = "Data Source=DESKTOP-EG4ORHN\SQLEXPRESS; Initial Catalog=billing;
            User ID=sa;Password=24518300";
connection = New SqlConnection(conn_str);
```

(Here, DESKTOP-EG4ORHN\SQLEXPRESS refers to a data source and billing refers to database name)

Working with Command:

```
SqlConnection connection;
 SqlCommand command;
 String conn_str="Data Source=Raazu\\SQLEXPRESS; Initial
            Catalog=billing; User ID=sa;Password=24518300";
 connection = new SqlConnection(conn_str);
 connection.Open();


String sql = "insert into tblCustomer(name,address) values(“Raaju”,
                    “Birtamode”)";
      command = new SqlCommand(sql, connection);
      command.ExecuteNonQuery();
      connection.Close();
```

For MS-Access

Note: Access Database Engine must be installed

```
OleDbConnection conn;
OleDbCommand command;
  string constr = "Provider=Microsoft.ACE.OLEDB.12.0;Data
        Source=C:\\Users\\Raazu\\Documents\\Visual Studio
        2012\\Projects\\DatabaseTest\\testdb.accdb";
  conn = new OleDbConnection(constr);
  conn.Open();

  string sql = "INSERT INTO tblStudent(name,address)
                    VALUES('Ram','Btm')";
  command = new OleDbCommand(sql,conn);
  command.ExecuteNonQuery();
  Console.WriteLine("Data Inserted Successfully !");
```

DataReader, DataAdaper, 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();
  da.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.

Complete Example – CRUD operation (MS-Access):

```
using System;
using System.Data;
using System.Data.OleDb;
namespace DatabaseTest
{
    class Program
    {
        OleDbConnection conn;
        OleDbCommand command;
        void CreateConnection()
        {
            string constr = "Provider=Microsoft.ACE.OLEDB.12.0;Data
                  Source=C:\\Users\\Raazu\\Documents\\Visual Studio
                  2012\\Projects\\DatabaseTest\\testdb.accdb";
            conn = new OleDbConnection(constr);
            conn.Open();
        }

        void InsertUpdateDelete(string sql)
        {
            command = new OleDbCommand(sql, conn);
            command.ExecuteNonQuery();
            Console.WriteLine("Operation Performed Successfully !");
        }

        void SelectRecords(string sql)
        {
            command = new OleDbCommand(sql, conn);
            OleDbDataAdapter adapter = new OleDbDataAdapter(command);
```

```
    DataTable dt = new DataTable();
    adapter.Fill(dt);
    if (dt.Rows.Count != 0)
    {
        Console.WriteLine("Sid\t Name\t Address");
        for (int i = 0; i < dt.Rows.Count;i++)
        {
            string sid = dt.Rows[i]["sid"].ToString();
            string name = dt.Rows[i]["name"].ToString();
            string address = dt.Rows[i]["address"].ToString();
            Console.WriteLine(sid+"\t"+name+"\t"+address);
        }
    }
}

static void Main(string[] args)
{
    Program obj = new Program();
    try
    {
        obj.CreateConnection();
        x: Console.WriteLine("1.Insert\t 2.Update\t 3.Delete\t
               4.Select");
        Console.WriteLine("Enter your choice: ");
        int n = Convert.ToInt32(Console.ReadLine());
        string sql="",nm = "", add = "";
        int id=0;
        switch (n)
        {
            case 1:
                Console.WriteLine("Ënter Name of Student: ");
                nm = Console.ReadLine();
                Console.WriteLine("Ënter Address of Student: ");
                add = Console.ReadLine();
                sql = "INSERT INTO tblStudent (name,address)
                       VALUES('"+nm+"','"+add+"')";
                obj.InsertUpdateDelete(sql);
                break;
            case 2:
                Console.WriteLine("Enter id to be updated");
                id = Convert.ToInt32(Console.ReadLine());
                Console.WriteLine("Ënter Name of Student: ");
                nm = Console.ReadLine();
                Console.WriteLine("Ënter Address of Student: ");
                add = Console.ReadLine();
                sql = "UPDATE tblStudent SET name='"+nm+"',
                      address='"+add+"' WHERE sid="+id;
                obj.InsertUpdateDelete(sql);
                break;
            case 3:
                Console.WriteLine("Enter id to be deleted");
                id = Convert.ToInt32(Console.ReadLine());
                sql = "DELETE FROM tblStudent WHERE sid="+id;
                obj.InsertUpdateDelete(sql);
```

```
                    break;
                case 4:
                    sql = "SELECT * FROM tblStudent";
                    obj.SelectRecords(sql);
                    break;
                default:
                    Console.WriteLine("Wrong Choice");
                    break;
            }
            goto x;
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex);
            Console.WriteLine("Connection Failed !");
        }


        Console.ReadKey();
    }
}
```

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.

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

Complete Program for CRUD operation

```
using MySql.Data.MySqlClient;
using System;
using System.Data;
namespace DatabaseTest
{
    class Program
    {
        MySqlConnection conn;
        MySqlCommand command;
        void CreateConnection()
        {
            string constr = "SERVER=localhost; DATABASE=dbtest;
                    UID=root; PASSWORD=;";
            conn = new MySqlConnection(constr);
            conn.Open();
        }
```

```
void InsertUpdateDelete(string sql)
  {
      command = new MySqlCommand(sql, conn);
      command.ExecuteNonQuery();
      Console.WriteLine("Operation Performed Successfully !");
  }

  void SelectRecords(string sql)
  {
      command = new MySqlCommand(sql, conn);
      MySqlDataAdapter adapter = new MySqlDataAdapter(command);
      DataTable dt = new DataTable();
      adapter.Fill(dt);
      if (dt.Rows.Count != 0)
      {
          Console.WriteLine("Sid\t Name\t Address");
          for (int i = 0; i < dt.Rows.Count;i++)
          {
              string sid = dt.Rows[i]["sid"].ToString();
              string name = dt.Rows[i]["name"].ToString();
              string address = dt.Rows[i]["address"].ToString();
              Console.WriteLine(sid+"\t"+name+"\t"+address);
          }
      }
  }

  static void Main(string[] args)
  {
      Program obj = new Program();
      try
      {
          obj.CreateConnection();
          x: Console.WriteLine("1.Insert\t 2.Update\t 3.Delete\t
                                    4.Select");
          Console.WriteLine("Enter your choice: ");
          int n = Convert.ToInt32(Console.ReadLine());
          string sql="",nm = "", add = "";
          int id=0;
          switch (n)
          {
              case 1:
                  Console.WriteLine("Ënter Name of Student: ");
                  nm = Console.ReadLine();
                  Console.WriteLine("Ënter Address of Student: ");
                  add = Console.ReadLine();
                  sql = "INSERT INTO tblStudent (name,address)
                              VALUES('"+nm+"','"+add+"')";
                  obj.InsertUpdateDelete(sql);
                  break;

              case 2:
                  Console.WriteLine("Enter id to be updated");
                  id = Convert.ToInt32(Console.ReadLine());
                  Console.WriteLine("Ënter Name of Student: ");
```

```
                        nm = Console.ReadLine();
                        Console.WriteLine("Ënter Address of Student: ");
                        add = Console.ReadLine();
                        sql = "UPDATE tblStudent SET name='"+nm+"',
                                    address='"+add+"' WHERE sid="+id;
                        obj.InsertUpdateDelete(sql);
                        break;

                    case 3:
                        Console.WriteLine("Enter id to be deleted");
                        id = Convert.ToInt32(Console.ReadLine());
                        sql = "DELETE FROM tblStudent WHERE sid="+id;
                        obj.InsertUpdateDelete(sql);
                        break;

                    case 4:
                        sql = "SELECT * FROM tblStudent";
                        obj.SelectRecords(sql);
                        break;


                    default:
                        Console.WriteLine("Wrong Choice");
                        break;
                }
                goto x;
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex);
                Console.WriteLine("Connection Failed !");
            }


            Console.ReadKey();
        }
    }
}
```

Complete CRUD operation for SQL Server

```
using System;
using System.Data;
using System.Data.SqlClient;
namespace DatabaseTest
{
    class Program
    {
        SqlConnection conn;
        SqlCommand command;
        void CreateConnection()
        {
            string constr="Data Source=Raazu\\SQLEXPRESS; Initial
                  Catalog=dbtest; User ID=sa;Password=24518300";
```

```
        conn = new SqlConnection(constr);
        conn.Open();
    }

    void InsertUpdateDelete(string sql)
    {
        command = new SqlCommand(sql, conn);
        command.ExecuteNonQuery();
        Console.WriteLine("Operation Performed Successfully !");
    }

    void SelectRecords(string sql)
    {
        command = new SqlCommand(sql, conn);
        SqlDataAdapter adapter = new SqlDataAdapter(command);
        DataTable dt = new DataTable();
        adapter.Fill(dt);
        if (dt.Rows.Count != 0)
        {
            Console.WriteLine("Sid\t Name\t Address");
            for (int i = 0; i < dt.Rows.Count;i++)
            {
                string sid = dt.Rows[i]["sid"].ToString();
                string name = dt.Rows[i]["name"].ToString();
                string address = dt.Rows[i]["address"].ToString();
                Console.WriteLine(sid+"\t"+name+"\t"+address);
            }
        }
    }

    static void Main(string[] args)
    {
        Program obj = new Program();
        try
        {
            obj.CreateConnection();
            x: Console.WriteLine("1.Insert\t 2.Update\t 3.Delete\t
                                    4.Select");
            Console.WriteLine("Enter your choice: ");
            int n = Convert.ToInt32(Console.ReadLine());
            string sql="",nm = "", add = "";
            int id=0;
            switch (n)
            {
                case 1:
                    Console.WriteLine("Ënter Name of Student: ");
                    nm = Console.ReadLine();
                    Console.WriteLine("Ënter Address of Student: ");
                    add = Console.ReadLine();
                    sql = "INSERT INTO tblStudent (name,address)
                                VALUES('"+nm+"','"+add+"')";
                    obj.InsertUpdateDelete(sql);
                    break;
```

```
                    case 2:
                        Console.WriteLine("Enter id to be updated");
                        id = Convert.ToInt32(Console.ReadLine());
                        Console.WriteLine("Ënter Name of Student: ");
                        nm = Console.ReadLine();
                        Console.WriteLine("Ënter Address of Student: ");
                        add = Console.ReadLine();
                        sql = "UPDATE tblStudent SET name='"+nm+"',
                                    address='"+add+"' WHERE sid="+id;
                        obj.InsertUpdateDelete(sql);
                        break;

                    case 3:
                        Console.WriteLine("Enter id to be deleted");
                        id = Convert.ToInt32(Console.ReadLine());
                        sql = "DELETE FROM tblStudent WHERE sid="+id;
                        obj.InsertUpdateDelete(sql);
                        break;

                    case 4:
                        sql = "SELECT * FROM tblStudent";
                        obj.SelectRecords(sql);
                        break;


                    default:
                        Console.WriteLine("Wrong Choice");
                        break;
                }
                goto x;
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex);
                Console.WriteLine("Connection Failed !");
            }


            Console.ReadKey();
        }
    }
}
```

Writing Windows Form Applications

Introduction to Win Forms:

Windows Forms (WinForms) is a graphical (GUI) class library included as a part of Microsoft .NET Framework, providing a platform to write rich client applications for desktop, laptop, and tablet PCs.

A Windows Forms application is an event-driven application supported by Microsoft's .NET Framework. Unlike a batch program, it spends most of its time simply waiting for the user to do something, such as fill in a text box or click a button.

All visual elements in the Windows Forms class library derive from the Control class. This provides a minimal functionality of a user interface element such as location, size, color, font, text, as well as common events like click and drag/drop.

Basic Controls:

The following table lists some of the commonly used controls:

S.N.	Widget & Description
1	**Forms** The container for all the controls that make up the user interface.
2	**TextBox** It represents a Windows text box control.
3	**Label** It represents a standard Windows label.
4	**Button** It represents a Windows button control.
5	**ListBox** It represents a Windows control to display a list of items.
6	**ComboBox** It represents a Windows combo box control.
7	**RadioButton**

	It enables the user to select a single option from a group of choices when paired with other RadioButton controls.
8	**CheckBox** It represents a Windows CheckBox.
9	**PictureBox** It represents a Windows picture box control for displaying an image.
10	**ProgressBar** It represents a Windows progress bar control.
11	**ScrollBar** It Implements the basic functionality of a scroll bar control.
12	**DateTimePicker** It represents a Windows control that allows the user to select a date and a time and to display the date and time with a specified format.
13	**TreeView** It displays a hierarchical collection of labeled items, each represented by a TreeNode.
14	**ListView** It represents a Windows list view control, which displays a collection of items that can be displayed using one of four different views.

For more information, refer to the practical exercises.

Web Applications using ASP.NET

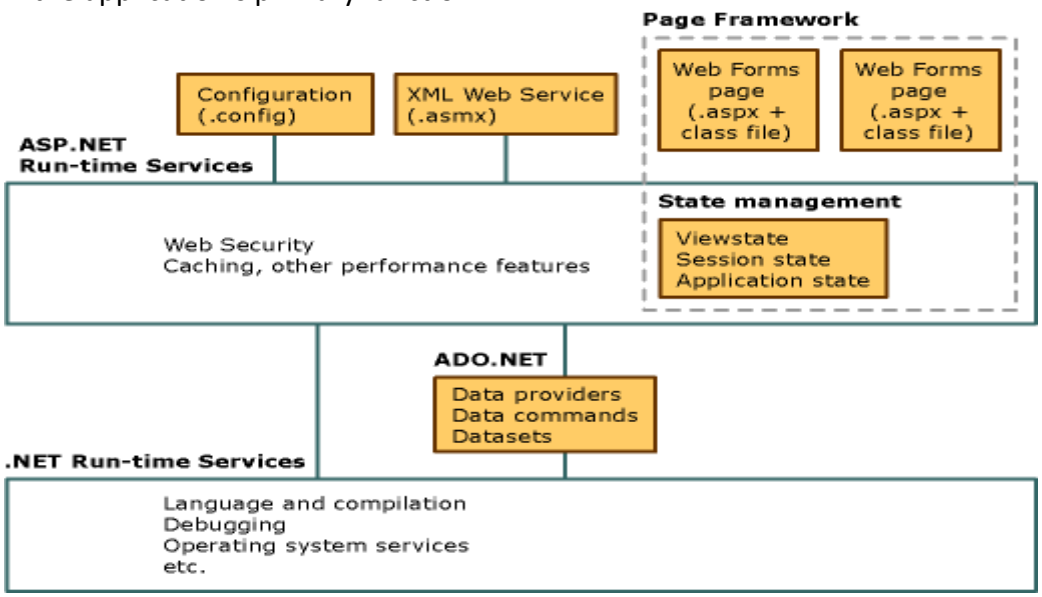
A Visual Studio Web application is built around ASP.NET. ASP.NET is a platform — including design-time objects and controls and a run-time execution context — for developing and running applications on a Web server.

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>
```

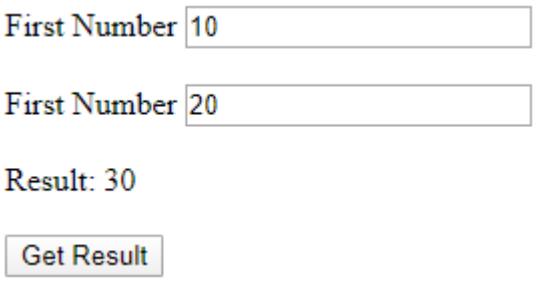
Launch another form on button click

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

Example 1 – Creating a basic form in ASP.Net


```
<%@ 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)



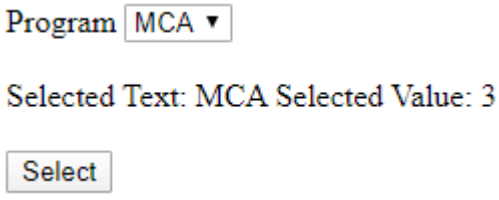
EventHandling.aspx

```
<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



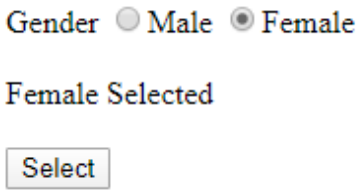
Dropdown.aspx

```
<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



example.aspx

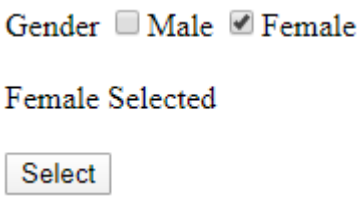
```
<form id="form1" runat="server">
    <div>
        <asp:Label ID="Label1" runat="server" Text="Gender"></asp:Label>
        <asp:RadioButton ID="radioMale" Text="Male" GroupName="gender"
                  runat="server" />
        <asp:RadioButton ID="radioFemale" Text="Female"
                 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



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";
        }
```

Validation Controls in ASP.NET

An important aspect of creating ASP.NET Web pages for user input is to be able to check that the information users enter is valid. ASP.NET provides a set of validation controls that provide an easy-to-use but powerful way to check for errors and, if necessary, display messages to the user.

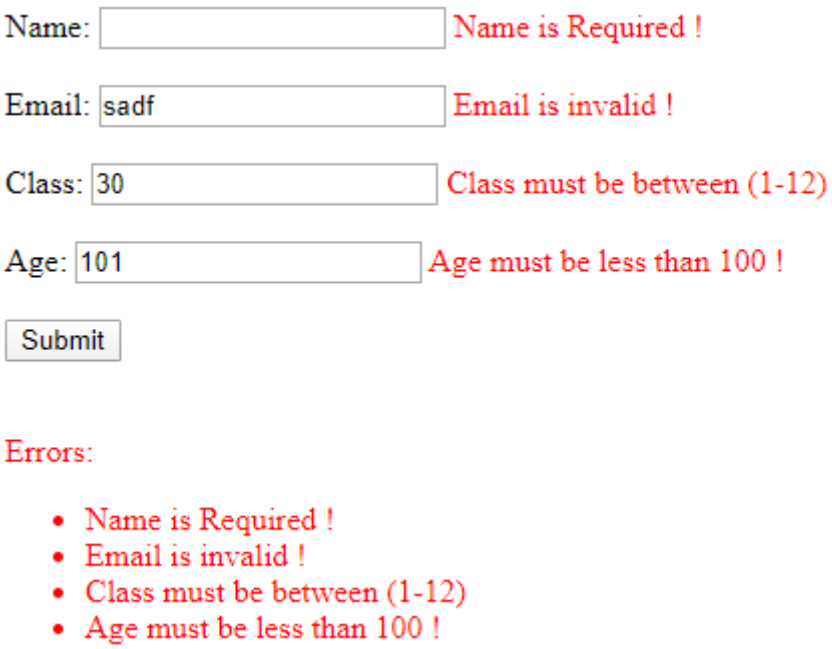
There are six types of validation controls in ASP.NET

- RequiredFieldValidation Control
- CompareValidator Control
- RangeValidator Control
- RegularExpressionValidator Control
- CustomValidator Control
- ValidationSummary

The below table describes the controls and their work:

Validation Control	Description
RequiredFieldValidation	Makes an input control a required field
CompareValidator	Compares the value of one input control to the value of another input control or to a fixed value
RangeValidator	Checks that the user enters a value that falls between two values
RegularExpressionValidator	Ensures that the value of an input control matches a specified pattern
CustomValidator	Allows you to write a method to handle the validation of the value entered
ValidationSummary	Displays a report of all validation errors occurred in a Web page

Example of Validation Controls



```
<form id="form1" runat="server">
    <div>
        <asp:Label ID="Label1" runat="server" Text="Name:"></asp:Label>
        <asp:TextBox ID="txtName" runat="server"></asp:TextBox>
        <asp:RequiredFieldValidator ID="validator1" runat="server"
            ForeColor="Red" ErrorMessage="Name is Required !"
            ControlToValidate="txtName"> </asp:RequiredFieldValidator>
        <br/><br/>

        <asp:Label ID="Label2" runat="server" Text="Email:"></asp:Label>
        <asp:TextBox ID="txtEmail" runat="server"></asp:TextBox>
        <asp:RegularExpressionValidator ID="validator2" runat="server"
            ForeColor="Red" ControlToValidate="txtEmail"
                  ErrorMessage="Email is invalid !"
         ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-
                                    .]\w+)*">
            </asp:RegularExpressionValidator>
        <br/><br/>

        <asp:Label ID="Label3" runat="server" Text="Class:"></asp:Label>
        <asp:TextBox ID="txtClass" runat="server"></asp:TextBox>
        <asp:RangeValidator ID="validator3"
               runat="server" ControlToValidate="txtClass"
                        ForeColor="Red"
               ErrorMessage="Class must be between (1-12)"
                             MaximumValue="12"
               MinimumValue="1" Type="Integer">
            </asp:RangeValidator>
         <br/><br/>

        <asp:Label ID="Label4" runat="server" Text="Age:"></asp:Label>
        <asp:TextBox ID="txtAge" runat="server"></asp:TextBox>
        <asp:CompareValidator ID="validator4" runat="server"
                  ValueToCompare="100"   ControlToValidate="txtAge"
                  ErrorMessage="Age must be less than 100 !"
                ForeColor="Red" Operator="LessThan" Type="Integer">
        </asp:CompareValidator>
        <br/><br/>

        <asp:Button ID="btnSubmit" runat="server" Text="Submit" />
        </div>
        <br/><br/>

      <asp:ValidationSummary ID="validator5" runat="server"
            ForeColor="Red" DisplayMode ="BulletList"
            ShowSummary ="true" HeaderText="Errors:" />
    </form>
```