# **­DOT NET**

## **OOPS CONCEPTS:**

Object Oriented Programming (OOP) is a programming model where programs are organized around objects and data rather than action and logic.   
  
OOP allows decomposition of a problem into a number of entities called objects and then builds data and functions around these objects.

1. The software is divided into a number of small units called objects. The data and functions are built around these objects.
2. The data of the objects can be accessed only by the functions associated with that object.
3. The functions of one object can access the functions of another object.

OOP has the following important features.

1. **Abstraction**
2. **Encapsulation**
3. **Inheritance**
4. **Polymorphism**

### **CLASS**

A class is the core of any modern Object Oriented Programming language such as C#. In OOP languages it is mandatory to create a class for representing data. **A class is a blueprint of an object that contains variables for storing data and functions to perform operations on the data.**

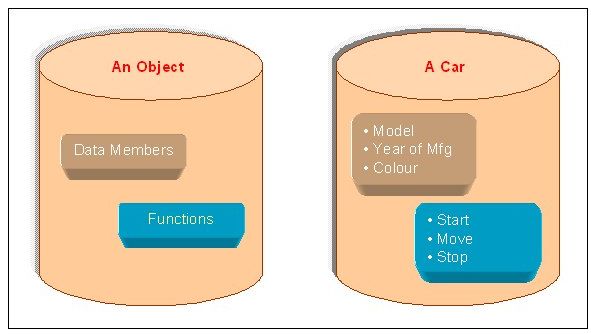
A class will not occupy any memory space and hence it is only a logical representation of data.

To create a class, you simply use the keyword "class" followed by the class name:

class Employee  
{  
   
}

### **OBJECT**

Objects are the basic run-time entities of an object oriented system. They may represent a person, a place or any item that the program must handle.   
  
"An object is a software bundle of related variable and methods."  
   
"An object is an instance of a class"



A class will not occupy any memory space. Hence to work with the data represented by the class you must create a variable for the class, that is called an object.

When an object is created using the new operator, memory is allocated for the class in the heap, the object is called an instance and its starting address will be stored in the object in stack memory.

When an object is created without the new operator, memory will not be allocated in the heap, in other words an instance will not be created and the object in the stack contains the value **null**.

When an object contains null, then it is not possible to access the members of the class using that object.  
   
class Employee  
{  
   
}

Syntax to create an object of class Employee:  
   
Employee objEmp = new Employee();

### **ABSTRACTION**

Abstraction is "**To represent the essential feature without representing the background details**."   
Abstraction lets you focus on what the object does instead of how it does it.   
Abstraction provides you a generalized view of your classes or objects by providing relevant information.  
Abstraction is the process of hiding the working style of an object, and showing the information of an object in an understandable manner.  
   
**Real-world Example of Abstraction**

Suppose you have an object Mobile Phone.  
   
Suppose you have 3 mobile phones as in the following:   
   
Nokia 1400 (Features: Calling, SMS)  
Nokia 2700 (Features: Calling, SMS, FM Radio, MP3, Camera)  
Black Berry (Features:Calling, SMS, FM Radio, MP3, Camera, Video Recording, Reading E-mails)  
   
Abstract information (necessary and common information) for the object "Mobile Phone" is that it makes a call to any number and can send SMS.  
   
So that, for a mobile phone object you will have the abstract class as in the following:

   abstract class MobilePhone

    {

        public void Calling();

        public void SendSMS();

    }

    public class Nokia1400 : MobilePhone

    {

    }

    public class Nokia2700 : MobilePhone

    {

        public void FMRadio();

        public void MP3();

        public void Camera();

    }

    public class BlackBerry : MobilePhone

    {

        public void FMRadio();

        public void MP3();

        public void Camera();

        public void Recording();

        public void ReadAndSendEmails();

    }  
   
Abstraction means putting all the variables and methods in a class that are necessary.

For example: Abstract class and abstract method. Abstraction is a common thing.

Abstraction is a process of hiding the implementation details and displaying the essential features.

**Example1**: A Laptop consists of many things such as processor, motherboard, RAM, keyboard, LCD screen, wireless antenna, web camera, usb ports, battery, speakers etc. To use it, you don't need to know how internally LCD screens, keyboard, web camera, battery, wireless antenna, speaker’s works.  You just need to know how to operate the laptop by switching it on. Think about if you would have to call to the engineer who knows all internal details of the laptop before operating it. This would have highly expensive as well as not easy to use everywhere by everyone.

So here the Laptop is an object that is designed to hide its complexity.

How to abstract: - By using **Access Specifiers**

**.Net has five access Specifiers**

**Public** -- Accessible outside the class through object reference.

**Private** -- Accessible inside the class only through member functions.

**Protected** -- Just like private but Accessible in derived classes also through member functions.

**Internal**-- Visible inside the assembly. Accessible through objects.

**Protected Internal** -- Visible inside the assembly through objects and in derived classes outside the assembly through member functions.

Let’s try to understand by a practical example:-

public class Class1

    {

        int  i;                                         //No Access specifier means private

        public  int j;                                 // Public

        protected int k;                          //Protected data

        internal int m;                         // Internal means visible inside assembly

        protected internal int n;         //inside assembly as well as to derived classes outside assembly

        static int x;                                 // This is also private

        public static int y;                       //Static means shared across objects

        [DllImport("MyDll.dll")]

        public static extern int MyFoo();       //extern means declared in this assembly defined in some other assembly

        public void myFoo2()

        {

            //Within a class if you create an object of same class then you can access all data members through object reference even private data too

            Class1 obj = new Class1();

            obj.i =10;   //Error can’t access private data through object.But here it is accessible.:)

            obj.j =10;

            obj.k=10;

            obj.m=10;

            obj.n=10;

       //     obj.s =10;  //Errror Static data can be accessed by class names only

            Class1.x = 10;

         //   obj.y = 10; //Errror Static data can be accessed by class names only

            Class1.y = 10;

        }

    }

Now lets try to copy the same code inside Main method and try to compile

[STAThread]

        static void Main()

        {

           //Access specifiers comes into picture only when you create object of class outside the class

            Class1 obj = new Class1();

       //     obj.i =10; //Error can’t access private data through object.

            obj.j =10;

      //      obj.k=10;     //Error can’t access protected data through object.

            obj.m=10;

            obj.n=10;

       //     obj.s =10;  //Errror Static data can be accessed by class names only

            Class1.x = 10;  //Error can’t access private data outside class

         //   obj.y = 10; //Errror Static data can be accessed by class names only

            Class1.y = 10;

        }

What if Main is inside another assembly

[STAThread]

        static void Main()

        {

           //Access specifiers comes into picture only when you create object of class outside the class

            Class1 obj = new Class1();

       //     obj.i =10; //Error can’t access private data through object.

            obj.j =10;

      //      obj.k=10;     //Error can’t access protected data through object.

     //     obj.m=10; // Error can’t access internal data outside assembly

    //      obj.n=10; // Error can’t access internal data outside assembly

       //     obj.s =10;  //Errror Static data can be accessed by class names only

            Class1.x = 10;  //Error can’t access private data outside class

         //   obj.y = 10; //Errror Static data can be accessed by class names only

            Class1.y = 10;

        }

In object-oriented software, complexity is managed by using **abstraction**.

**Abstraction** is a process that involves identifying the critical behavior of an object and eliminating irrelevant and complex details.

### **ENCAPSULATION**

Wrapping up a data member and a method together into a single unit (in other words class) is called Encapsulation. Encapsulation is like enclosing in a capsule. That is enclosing the related operations and data related to an object into that object.  
   
Encapsulation is like your bag in which you can keep your pen, book etcetera. It means this is the property of encapsulating members and functions.  
   
    class Bag  
    {  
        book;  
        pen;  
        ReadBook();  
    }  
   
Encapsulation means hiding the internal details of an object, in other words how an object does something. Encapsulation prevents clients from seeing its inside view, where the behaviour of the abstraction is implemented. Encapsulation is a technique used to protect the information in an object from another object. Hide the data for security such as making the variables private, and expose the property to access the private data that will be public.

So, when you access the property you can validate the data and set it.  
   
**Example 1**

    class Demo

    {

        private int \_mark;

        public int Mark

        {

            get { return \_mark; }

            set { if (\_mark > 0) \_mark = value; else \_mark = 0; }

        }

    }  
   
**Real-world Example of Encapsulation**

Let's use as an example Mobile Phones and Mobile Phone Manufacturers.  
Suppose you are a Mobile Phone Manufacturer and you have designed and developed a Mobile Phone design (a class). Now by using machinery you are manufacturing Mobile Phones (objects) for selling, when you sell your Mobile Phone the user only learns how to use the Mobile Phone but not how the Mobile Phone works.  
   
This means that you are creating the class with functions and by with objects (capsules) of which you are making available the functionality of your class by that object and without the interference in the original class.  
   
**Example 2**

**TV operation**

It is encapsulated with a cover and we can operate it with a remote and there is no need to open the TV to change the channel.   
Here everything is private except the remote, so that anyone can access the remote to operate and change the things in the TV.

### **INHERITANCE**

Inheritance is a process of deriving the new class from already existing class

**C#** is a complete object oriented programming language. Inheritance is one of the primary concepts of object-oriented programming. It allows you to reuse existing code. Through effective use of inheritance, you can save lot of time in your programming and also reduce errors, which in turn will increase the quality of work and productivity. A simple example to understand inheritance in C#.

    public class ParentClass

    {

        public ParentClass()

        {

            Console.WriteLine("Parent Constructor.");

        }

        public void print()

        {

            Console.WriteLine("I'm a Parent Class.");

        }

    }

    public class ChildClass : ParentClass

    {

        public ChildClass()

        {

            Console.WriteLine("Child Constructor.");

        }

        public static void Main()

        {

            ChildClass child = new ChildClass();

            child.print();

        }

    }  
   
**Output**

    Parent Constructor.  
    Child Constructor.  
    I'm a Parent Class.

### **POLYMORPHISM**

Polymorphism means one name, many forms. One function behaves in different forms.

In other words, "Many forms of a single object is called Polymorphism."

When a message can be processed in different ways is called polymorphism. Polymorphism means many forms.

Polymorphism is one of the fundamental concepts of OOP.

**Polymorphism provides following features:**

* It allows you to invoke methods of derived class through base class reference during runtime.
* It has the ability for classes to provide different implementations of methods that are called through the same name.
* **Polymorphism is of two types:**
* Compile time polymorphism/Overloading
* Runtime polymorphism/Overriding

**The Differences between Abstraction and Encapsulation**

|  |  |
| --- | --- |
| **Abstraction** | **Encapsulation** |
| 1.  Abstraction solves the problem at the design level. | 1. Encapsulation solves the problem in the implementation level. |
| 2. Abstraction hides unwanted data and provides relevant data. | 2. Encapsulation means hiding the code and data into a single unit to protect the data from the outside world. |
| 3. Abstraction lets you focus on what the object does instead of how it does it | 3. Encapsulation means hiding the internal details or mechanics of how an object does something. |
| 4. Abstraction: Outer layout, used in terms of design. For example: An external of a Mobile Phone, like it has a display screen and keypad buttons to dial a number. | 4.Encapsulation- Inner layout, used in terms of implementation.  For example: the internal details of a Mobile Phone, how the keypad button and display screen are connected with each other using circuits. |

 The easier way to understand abstraction and encapsulation is as follows.

**Real-world Example**

**Use an example of a Mobile Phone**

You have a Mobile Phone, you can dial a number using keypad buttons. You don't even know how these are working internally. This is called Abstraction. You only have the information that is necessary to dial a number. But not internal working of the mobile.  
   
But how does the Mobile Phone work internally? How are the keypad buttons connected with internal circuit? That is called Encapsulation.

Summary

"Encapsulation is accomplished using classes. Keeping data and methods that access that data into a single unit."

"Abstraction is accomplished using an Interface. Just giving the abstract information about what it can do without specifying the details."

"Information/Data hiding is accomplished using modifiers by keeping the instance variables private or protected."

#### **Compile Time Polymorphism**

Compile time polymorphism is method and operators overloading. It is also called early binding.

In method overloading method performs the different task at the different input parameters.

#### **Runtime Polymorphism**

Runtime time polymorphism is done using inheritance and virtual functions. Method overriding is called runtime polymorphism. It is also called late binding.

When **overriding** a method, you change the behavior of the method for the derived class. **Overloading** a method simply involves having another method with the same prototype.

**Caution:**Don't confused method overloading with method overriding, they are different, unrelated concepts. But they sound similar.

Method overloading has nothing to do with inheritance or virtual methods.

**Following are examples of methods having different overloads:**

void area(int side);

void area(int l, int b);

void area(float radius);

**Practical example of Method Overloading (Compile Time Polymorphism)**

using System;

namespace method\_overloading

{

    class Program

    {

        public class Print

        {

            public void display(string name)

            {

                Console.WriteLine ("Your name is : " + name);

            }

            public void display(int age, float marks)

            {

                Console.WriteLine ("Your age is : " + age);

                Console.WriteLine ("Your marks are :" + marks);

            }

        }

        static void Main(string[] args)

        {

            Print obj = new Print ();

            obj.display ("George");

            obj.display (34, 76.50f);

            Console.ReadLine ();

        }

    }

}

**Note:**In the code if you observe display method is called two times. Display method will work according to the number of parameters and type of parameters.

**When and why to use method overloading**

Use method overloading in situation where you want a class to be able to do something, but there is more than one possibility for what information is supplied to the method that carries out the task.

You should consider overloading a method when you for some reason need a couple of methods that take different parameters, but conceptually do the same thing.

**Method overloading showing many forms.**

using System;

namespace method\_overloading\_polymorphism

{

    Class Program

    {

        Public class Shape

        {

            Public void Area (float r)

            {

                float a = (float)3.14 \* r;

                // here we have used function overload with 1 parameter.

                Console.WriteLine ("Area of a circle: {0}",a);

            }

            Public void Area(float l, float b)

            {

                float x = (float)l\* b;

                // here we have used function overload with 2 parameters.

                Console.WriteLine ("Area of a rectangle: {0}",x);

            }

            public void Area(float a, float b, float c)

            {

                float s = (float)(a\*b\*c)/2;

                // here we have used function overload with 3 parameters.

                Console.WriteLine ("Area of a circle: {0}", s);

            }

        }

        Static void Main (string[] args)

        {

            Shape ob = new Shape ();

            ob.Area(2.0f);

            ob.Area(20.0f,30.0f);

            ob.Area(2.0f,3.0f,4.0f);

            Console.ReadLine ();

        }

    }

}

**Things to keep in mind while method overloading**

If you use overload for method, there are couple of restrictions that the compiler imposes.

The rule is that overloads must be different in their signature, which means the name and the number and type of parameters.

There is no limit to how many overload of a method you can have. You simply declare them in a class, just as if they were different methods that happened to have the same name.

**Method Overriding:**

Whereas **Overriding** means changing the functionality of a method without changing the signature. We can override a function in base class by creating a similar function in derived class. This is done by usingvirtual/override keywords.  
  
Base class method has to be marked with virtual keyword and we can override it in derived class usingoverride keyword.  
  
Derived class method will completely overrides base class method i.e. when we refer base class object created by casting derived class object a method in derived class will be called.

**Example**

// Base class  
public class BaseClass  
{  
public virtual void Method1()  
{  
Console.Write("Base Class Method");  
}  
}  
// Derived class  
public class DerivedClass : BaseClass  
{  
public override void Method1()  
{  
Console.Write("Derived Class Method");  
}  
}  
// Using base and derived class  
public class Sample  
{  
public void TestMethod()  
{  
// calling the overriden method  
DerivedClass objDC = new DerivedClass();   
objDC.Method1();  
 // calling the baesd class method  
BaseClass objBC = (BaseClass)objDC;   
objDC.Method1();  
}  
}

**OUTPUT**

Derived Class Method

Derived Class Method

### **CONSTRUCTORS AND DESTRUCTORS**

Classes have complicated internal structures, including data and functions, object initialization and cleanup for classes is much more complicated than it is for simple data structures. Constructors and destructors are special member functions of classes that are used to construct and destroy class objects. Construction may involve memory allocation and initialization for objects. Destruction may involve cleanup and deallocation of memory for objects.

* Constructors and destructors do not have return types nor can they return values.
* References and pointers cannot be used on constructors and destructors because their addresses cannot be taken.
* Constructors cannot be declared with the keyword virtual.
* Constructors and destructors cannot be declared const, or volatile.
* Unions cannot contain class objects that have constructors or destructors.

Constructors and destructors obey the same access rules as member functions. For example, if you declare a constructor with protected access, only derived classes and friends can use it to create class objects.

The compiler automatically calls constructors when defining class objects and calls destructors when class objects go out of scope. A constructor does not allocate memory for the class object it’s this pointer refers to, but may allocate storage for more objects than its class object refers to. If memory allocation is required for objects, constructors can explicitly call the new operator. During cleanup, a destructor may release objects allocated by the corresponding constructor. To release objects, use the delete operator.

**Example of Constructor**

class C

{

       private int x;

       private int y;

       public C (int i, int j)

       {

                 x = i;

                 y = j;

       }

       public void display ()

       {

               Console.WriteLine(x + "i+" + y);

       }

}

**Example of Destructor**

class D

{

        public D ()

        {

            // constructor

        }

        ~D ()

        {

           // Destructor

        }

}

## **Difference between Abstract class and Interface (please refer two villains of every interview)**

1. An Abstract class doesn't provide full abstraction but an interface does provide full abstraction; i.e. both a declaration and a definition is given in an abstract class but not so in an interface.
2. Using Abstract we can not achieve multiple inheritance but using an Interface we can achieve multiple inheritance.
3. We can not declare a member field in an Interface.
4. We can not use any access modifier i.e. public , private , protected , internal etc. because within an interface by default everything is public.
5. An Interface member cannot be defined using the keyword static, virtual, abstract or sealed.

### **Abstract Class**

The [**abstract**](https://msdn.microsoft.com/en-in/library/sf985hc5.aspx) keyword enables you to create classes and [class](https://msdn.microsoft.com/en-in/library/0b0thckt.aspx) members that are incomplete and must be implemented in a derived class. An abstract class cannot be instantiated. The purpose of an abstract class is to provide a common definition of a base class that multiple derived classes can share. For example, a class library may define an abstract class that is used as a parameter to many of its functions, and require programmers using that library to provide their own implementation of the class by creating a derived class.

Abstract classes may also define abstract methods. This is accomplished by adding the keyword **abstract** before the return type of the method

The purpose of abstract class is to provide default functionality to its sub classes.  
  
When a method is declared as abstract in the base class then every derived class of that class must provide its own definition for that method.  
  
An abstract class can also contain methods with complete implementation, besides abstract methods.  
  
When a class contains at least one abstract method, then the class must be declared as **abstract class**.   
  
It is **mandatory** to override abstract method in the derived class.  
  
When a class is declared as abstract class, then it is not possible to create an instance for that class. But it can be used as a parameter in a method.  
  
  
The following example creates three classes shape, circle and rectangle where circle and rectangle are inherited from the class shape and overrides the methods Area() and Circumference() that are declared as abstract in Shape class and as Shape class contains abstract methods it is declared as abstract class.

using System;  
  
namespace ProgramCall  
{  
  
    //Abstract class  
    abstract class Shape1  
    {  
  
        protected float R, L, B;  
  
        //Abstract methods can have only declarations  
        public abstract float Area();  
        public abstract float Circumference();  
  
    }  
  
  
    class Rectangle1 : Shape1  
    {  
        public void GetLB()  
        {  
            Console.Write("Enter  Length  :  ");  
  
            L = float.Parse(Console.ReadLine());   
  
            Console.Write("Enter Breadth : ");  
  
            B = float.Parse(Console.ReadLine());  
        }  
  
          
        public override float Area()  
        {  
            return L \* B;  
        }  
  
        public override float Circumference()  
        {  
            return 2 \* (L + B);  
        }  
  
    }  
  
  
    class Circle1 : Shape1  
    {  
  
        public void GetRadius()  
        {  
  
            Console.Write("Enter  Radius  :  ");  
            R = float.Parse(Console.ReadLine());  
        }  
  
        public override float Area()  
        {  
            return 3.14F \* R \* R;  
        }  
        public override float Circumference()  
        {  
            return 2 \* 3.14F \* R;  
  
        }  
    }  
    class MainClass  
    {  
        public static void Calculate(Shape1 S)  
        {  
  
            Console.WriteLine("Area : {0}", S.Area());  
            Console.WriteLine("Circumference : {0}", S.Circumference());  
  
        }  
        static void Main()  
        {  
  
            Rectangle1 R = new Rectangle1();   
            R.GetLB();  
            Calculate(R);  
  
            Console.WriteLine();  
  
            Circle1 C = new Circle1();   
            C.GetRadius();  
            Calculate(C);  
  
            Console.Read();  
  
        }  
    }  
}

**Output**

Enter  Length  :  10  
Enter Breadth : 12  
Area : 120  
Circumference : 44

Enter  Radius  :  5  
Area : 78.5  
Circumference : 31.4

### **Interface**

An interface contains only the signatures of [methods](https://msdn.microsoft.com/en-in/library/ms173114.aspx), [properties](https://msdn.microsoft.com/en-in/library/x9fsa0sw.aspx), [events](https://msdn.microsoft.com/en-in/library/awbftdfh.aspx) or [indexers](https://msdn.microsoft.com/en-in/library/6x16t2tx.aspx). A class or struct that implements the interface must implement the members of the interface that are specified in the interface definition. In the following example, class ImplementationClass must implement a method named SampleMethod that has no parameters and returns **void**.

**Example**

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System;

namespace InterfaceApplication

{

public interface ITransactions

{

// interface members

void showTransaction();

double getAmount();

}

public class Transaction : ITransactions

{

private string tCode;

private string date;

private double amount;

public Transaction()

{

tCode = " ";

date = " ";

amount = 0.0;

}

public Transaction(string c, string d, double a)

{

tCode = c;

date = d;

amount = a;

}

public double getAmount()

{

return amount;

}

public void showTransaction()

{

Console.WriteLine("Transaction: {0}", tCode);

Console.WriteLine("Date: {0}", date);

Console.WriteLine("Amount: {0}", getAmount());

}

}

class Tester

{

static void Main(string[] args)

{

Transaction t1 = new Transaction("001", "8/10/2012", 78900.00);

Transaction t2 = new Transaction("002", "9/10/2012", 451900.00);

t1.showTransaction();

t2.showTransaction();

Console.ReadKey();

}

}

}

OUTPUT

Transaction: 001

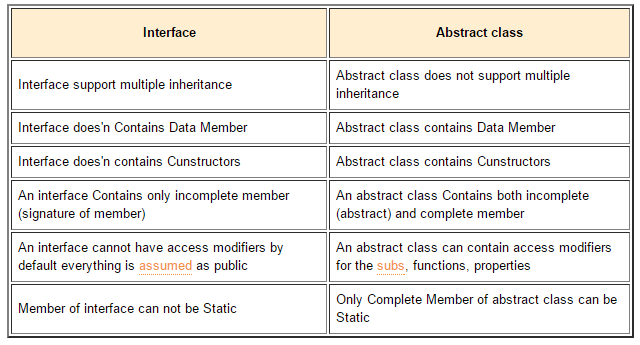
Date: 8/10/2012

Amount: 78900

Transaction: 002

Date: 9/10/2012

Amount: 451900



|  |  |  |
| --- | --- | --- |
| **Feature** | **Interface** | **Abstract class** |
| Multiple inheritance | A class may inherit several interfaces. | A class may inherit only one abstract class. |
| Default implementation | An interface cannot provide any code, just the signature. | An abstract class can provide complete, default code and/or just the details that have to be overridden. |
| Access Modfiers | An interface cannot have access modifiers for the subs, functions, properties etc everything is assumed as public | An abstract class can contain access modifiers for the subs, functions, properties |
| Core VS Peripheral | Interfaces are used to define the peripheral abilities of a class. In other words both Human and Vehicle can inherit from a IMovable interface. | An abstract class defines the core identity of a class and there it is used for objects of the same type. |
| Homogeneity | If various implementations only share method signatures then it is better to use Interfaces. | If various implementations are of the same kind and use common behaviour or status then abstract class is better to use. |
| Speed | Requires more time to find the actual method in the corresponding classes. | Fast |
| Adding functionality (Versioning) | If we add a new method to an Interface then we have to track down all the implementations of the interface and define implementation for the new method. | If we add a new method to an abstract class then we have the option of providing default implementation and therefore all the existing code might work properly. |
| Fields and Constants | No fields can be defined in interfaces | An abstract class can have fields and constrants defined |

## CONSTANT , READONLY , STATIC

### **Constant**

Constant fields or local variables must be assigned a value at the time of declaration and after that they cannot be modified. By default constant are static, hence you cannot define a constant type as static.

1. **public const int X = 10;**

A const field is a compile-time constant. A constant field or local variable can be initialized with a constant expression which must be fully evaluated at compile time.

1. **void Calculate(int Z)**
2. **{**
3. **const int X = 10, X1 = 50;**
4. **const int Y = X + X1; *//no error, since its evaluated a compile time***
5. **const int Y1 = X + Z; *//gives error, since its evaluated at run time***
6. **}**

You can apply const keyword to built-in value types (byte, short, int, long, char, float, double, decimal, bool), enum, a string literal, or a reference type which can be assigned with a value null.

1. **const MyClass obj1 = null;*//no error, since its evaluated a compile time***
2. **const MyClass obj2 = new MyClass();*//gives error, since its evaluated at run time***

Constants can be marked as public, private, protected, internal, or protected internal access modifiers.

Use the const modifier when you sure that the value a field or local variable would not be changed.

### **Read-Only**

A readonly field can be initialized either at the time of declaration or with in the constructor of same class. Therefore, readonly fields can be used for run-time constants.

1. **class MyClass**
2. **{**
3. **readonly int X = 10; *// initialized at the time of declaration***
4. **readonly int X1;**
6. **public MyClass(int x1)**
7. **{**
8. **X1 = x1; *// initialized at run time***
9. **}**
10. **}**

Explicitly, you can specify a readonly field as static since, like constant by default it is not static. Readonly keyword can be apply to value type and reference type (which initialized by using the new keyword) both. Also, delegate and event could not be readonly.

Use the readonly modifier when you want to make a field constant at run time.

The ***static keyword*** is used to specify a static member, which means static members are common to all the objects and they do not tied to a specific object. This keyword can be used with classes, fields, methods, properties, operators, events, and constructors, but it cannot be used with indexers, destructors, or types other than classes.

1. **class MyClass**
2. **{**
3. **static int X = 10;**
4. **int Y = 20;**
5. **public static void Show()**
6. **{**
7. **Console.WriteLine(X);**
8. **Console.WriteLine(Y); *//error, since you can access only static members***
9. **}**
10. **}**

**Key Points on Static Keyword**

1. If the static keyword is applied to a class, all the members of the class must be static.
2. Static methods can only access static members of same class. Static properties are used to get or set the value of static fields of a class.
3. Static constructor can't be parameterized. Access modifiers can not be applied on Static constructor, it is always a public default constructor which is used to initialize static fields of the class.

**What is the usage of static constructor?**

A static constructor is used to initialize any static data, or to perform a particular action that needs performed once only.   
It is called automatically before the first instance is created or any static members are referenced.

## **DELEGATES**

A delegate in C# is similar to a function pointer in C or C++. Using a delegate allows the programmer to encapsulate a reference to a method inside a delegate object. The delegate object can then be passed to code which can call the referenced method, without having to know at compile time which method will be invoked. Unlike function pointers in C or C++, delegates are object-oriented, type-safe, and secure.

A delegate declaration defines a type that encapsulates a method with a particular set of arguments and return type. For static methods, a delegate object encapsulates the method to be called. For instance methods, a delegate object encapsulates both an instance and a method on the instance. If you have a delegate object and an appropriate set of arguments, you can invoke the delegate with the arguments.

An interesting and useful property of a delegate is that it does not know or care about the class of the object that it references. Any object will do; all that matters is that the method's argument types and return type match the delegate's. This makes delegates perfectly suited for "anonymous" invocation.

### **Declaring Delegates**

Delegate declaration determines the methods that can be referenced by the delegate. A delegate can refer to a method, which has the same signature as that of the delegate.

For example, consider a delegate:

public delegate int MyDelegate (string s);

The preceding delegate can be used to reference any method that has a singlestring parameter and returns an int type variable.

Syntax for delegate declaration is:

delegate <return type> <delegate-name> <parameter list>

### Instantiating Delegates

Once a delegate type is declared, a delegate object must be created with thenew 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:

public delegate void printString(string s);

...

printString ps1 = new printString(WriteToScreen);

printString ps2 = new printString(WriteToFile);

Following example demonstrates declaration, instantiation, and use of a delegate that can be used to reference methods that take an integer parameter and returns an integer value.

using System;

delegate int NumberChanger(int n);

namespace DelegateAppl

{

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();

}

}

}

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

Value of Num: 35

Value of Num: 175

### **Multicasting of a Delegate**

Delegate objects can be composed using the "+" operator. A composed delegate calls the two delegates it was composed from. Only delegates of the same type can be composed. The "-" operator can be used to remove a component delegate from a composed delegate.

Using this property of delegates you can create an invocation list of methods that will be called when a delegate is invoked. This is called multicasting of a delegate. The following program demonstrates multicasting of a delegate:

using System;

delegate int NumberChanger(int n);

namespace DelegateAppl

{

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();

}

}

}

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

Value of Num: 75

### **Using Delegates**

The following example demonstrates the use of delegate. The delegateprintString can be used to reference method that takes a string as input and returns nothing.

We use this delegate to call two methods, the first prints the string to the console, and the second one prints it to a file:

using System;

using System.IO;

namespace DelegateAppl

{

class PrintString

{

static FileStream fs;

static StreamWriter sw;

// delegate declaration

public delegate void printString(string s);

// this method prints to the console

public static void WriteToScreen(string str)

{

Console.WriteLine("The String is: {0}", str);

}

//this method prints to a file

public static void WriteToFile(string s)

{

fs = new FileStream("c:\\message.txt",

FileMode.Append, FileAccess.Write);

sw = new StreamWriter(fs);

sw.WriteLine(s);

sw.Flush();

sw.Close();

fs.Close();

}

// this method takes the delegate as parameter and uses it to

// call the methods as required

public static void sendString(printString ps)

{

ps("Hello World");

}

static void Main(string[] args)

{

printString ps1 = new printString(WriteToScreen);

printString ps2 = new printString(WriteToFile);

sendString(ps1);

sendString(ps2);

Console.ReadKey();

}

}

}

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

The String is: Hello World

## **Anonymous Method**

Anonymous methods provide a technique to pass a code block as a delegate parameter. Anonymous methods are the methods without a name, just the body.You need not specify the return type in an anonymous method; it is inferred from the return statement inside the method body.

### **Example**

The following example demonstrates the concept:

using System;

delegate void NumberChanger(int n);

namespace DelegateAppl

{

class TestDelegate

{

static int num = 10;

public static void AddNum(int p)

{

num += p;

Console.WriteLine("Named Method: {0}", num);

}

public static void MultNum(int q)

{

num \*= q;

Console.WriteLine("Named Method: {0}", num);

}

public static int getNum()

{

return num;

}

static void Main(string[] args)

{

//create delegate instances using anonymous method

NumberChanger nc = delegate(int x)

{

Console.WriteLine("Anonymous Method: {0}", x);

};

//calling the delegate using the anonymous method

nc(10);

//instantiating the delegate using the named methods

nc = new NumberChanger(AddNum);

//calling the delegate using the named methods

nc(5);

//instantiating the delegate using another named methods

nc = new NumberChanger(MultNum);

//calling the delegate using the named methods

nc(2);

Console.ReadKey();

}

}

}

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

Anonymous Method: 10

Named Method: 15

Named Method: 30

## EXTENSION METHODS

**What are extension methods?**  
  
Extension methods enable you to add methods to existing types without creating a new derived type, recompiling, or otherwise modifying the original type.   
  
An extension method is a special kind of static method, but they are called as if they were instance methods on the extended type.  
  
**How to use extension methods?**  
  
An extension method is a static method of a static class, where the "this" modifier is applied to the first parameter. The type of the first parameter will be the type that is extended.  
  
Extension methods are only in scope when you explicitly import the namespace into your source code with a using directive.

**Example**

using System;  
using System.Text;  
  
namespace ClassLibExtMethod  
{  
    public class Class1  
    {  
        public string Display()  
        {  
            return ("I m in Display");  
        }  
  
        public string Print()  
        {  
            return ("I m in Print");  
        }  
    }  
}

using System;  
using System.Text;  
using ClassLibExtMethod;  
  
namespace ExtensionMethod1  
{  
    public static class XX  
    {  
         public static void NewMethod(this Class1 ob)  
        {  
            Console.WriteLine("Hello I m extended method");  
        }  
    }  
  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            Class1 ob = new Class1();  
            ob.Display();  
            ob.Print();  
            ob.NewMethod();  
            Console.ReadKey();  
        }  
    }  
}

## **REF, OUT KEYWORD**

The keywords ref and out are used to pass arguments within a method or function. Both indicate that an argument / parameter is passed by reference. By default parameters are passed to a method by value. By using these keywords (ref and out) we can pass a parameter by reference.

**Ref Keyword**  
The ref keyword passes arguments by reference. It means any changes made to this argument in the method will be reflected in that variable when control returns to the calling method.  
  
**Example code**

public static string GetNextName(ref int id)  
{  
    string returnText = "Next-" + id.ToString();  
    id += 1;  
    return returnText;  
}  
static void Main(string[] args)  
{  
    int i = 1;  
    Console.WriteLine("Previous value of integer i:" + i.ToString());  
    string test = GetNextName(ref i);  
    Console.WriteLine("Current value of integer i:" + i.ToString());  
}

**Output**

Ref Output  
**Out Keyword**  
The out keyword passes arguments by reference. This is very similar to the ref keyword.  
  
**Example Code**

public static string GetNextNameByOut(out int id)

{  
    id = 1;  
    string returnText = "Next-" + id.ToString();  
    return returnText;   
}

static void Main(string[] args)

{  
    int i = 0;  
    Console.WriteLine("Previous value of integer i:" + i.ToString());  
    string test = GetNextNameByOut(out i);  
    Console.WriteLine("Current value of integer i:" + i.ToString());  
}

**Output**

Out  
**Ref Vs Out**

|  |  |
| --- | --- |
| **Ref** | **Out** |
| The parameter or argument must be initialized first before it is passed to ref. | It is not compulsory to initialize a parameter or argument before it is passed to an out. |
| It is not required to assign or initialize the value of a parameter (which is passed by ref) before returning to the calling method. | A called method is required to assign or initialize a value of a parameter (which is passed to an out) before returning to the calling method. |
| Passing a parameter value by Ref is useful when the called method is also needed to modify the pass parameter. | **Declaring a parameter to an out method is useful when multiple values need to be returned from a function or method.** |
| It is not compulsory to initialize a parameter value before using it in a calling method. | A parameter value must be initialized within the calling method before its use. |
| When we use REF, data can be passed bi-directionally. | When we use OUT data is passed only in a unidirectional way (from the called method to the caller method). |
| Both ref and out are treated differently at run time and they are treated the same at compile time. | |
| Properties are not variables, therefore it cannot be passed as an out or ref parameter. | |

**Ref / Out keyword and method Overloading**Both ref and out are treated differently at run time and they are treated the same at compile time, so methods cannot be overloaded if one method takes an argument as ref and the other takes an argument as an out.  
  
**Example code**

public static string GetNextName(ref int id)

{  
    string returnText = "Next-" + id.ToString();  
    id += 1;  
    return returnText;  
}

public static string GetNextName(out int id)

{  
    id = 1;  
    string returnText = "Next-" + id.ToString();  
    return returnText;  
}

**Output** when the code is compiled:

compile the code  
  
However, method overloading can be possible when one method takes a ref or out argument and the other takes the same argument without ref or out.  
  
**Example Code**

public static string GetNextName(int id)

{  
    string returnText = "Next-" + id.ToString();  
    id += 1;  
    return returnText;  
}  
public static string GetNextName(ref int id)  
{  
    string returnText = "Next-" + id.ToString();  
    id += 1;  
    return returnText;  
}

## **GRBAGE COLLECTION**



The Garbage collection is very important technique in the .Net framework to free the unused managed code objects in the memory and free the space to the process. I will explain about the basics of the Garbage collection in this article.  
 **Garbage Collection in .Net framework**  
The garbage collection (GC) is new feature in Microsoft .net framework. When we have a class that represents an object in the runtime that allocates a memory space in the heap memory. All the behavior of that objects can be done in the allotted memory in the heap. Once the activities related to that object is get finished then it will be there as unused space in the memory.  
  
The earlier releases of Microsoft products have used a method like once the process of that object get finished then it will be cleared from the memory. For instance Visual Basic, An object get finishes that work then there we have to define a "nothing" to that object. So, it clears the memory space to the processors.  
  
Microsoft was planning to introduce a method that should automate the cleaning of unused memory space in the heap after the life time of that object. Eventually they have introduced a new technique "Garbage collection". It is very important part in the .Net framework. Now it handles this object clear in the memory implicitly. It overcomes the existing explicit unused memory space clearance.  
 **Garbage Collection**  
The heap memory is divided into number of generations. Normally it is three generations. The Generation 0 is for short live objects, Generation 1 is for medium live objects which are moved from Generation 0. Generation 3 is mostly stable objects.   
  
When an object is created then it will allocate the memory space which will be higher. It will be in the Generation 0 and the memory allocation will be continuous without any space between the generations of garbage collectors.  
 **How it works**  
Implicit Garbage Collection should be handled by the .Net framework. When object is created then it will be placed in the Generation 0. The garbage collection uses an algorithm which checks the objects in the generation, the objects life time get over then it will be removed from the memory. The two kinds of objects. One is Live Objects and Dead Objects. The Garbage collection algorithm collects all unused objects that are dead objects in the generation. If the live objects running for long time then based on that life time it will be moved to next generation.  
  
The object cleaning in the generation will not take place exactly after the life time over of the particular objects. It takes own time to implement the sweeping algorithm to free the spaces to the process.  
 **Exception Handling**  
The Garbage collection has designed such a way that it can be implicitly handling to collect the free spaces in memory. But as I said it takes own time to uses the algorithm to collect unused objects in the memory.  
  
If we want to forces to collect unused objects or explicitly release particular object from the momory.The code allows us to clear the object from the heap immediately.  
 **When it happens**  
The garbage collector periodically checks the heap memory to reclaim the objects when the object has no valid references in the memory.  
  
When an object is created then it will allocate the memory in the heap then it checks the available space for the newly created objects, if the available space is not adequate to allot the space then it automatically garbage collect the unused objects. If all are valid referenced objects then it gets additional space from the processor.   
  
If the object has reference with managed code objects then it will not free the memory space. However it cannot control the reference with unmanaged code objects, when application forces to collect the unused objects. But it can be achieved to write the explicit coding to avoid managed objects reference with unmanaged objects.   
 **Example code to know more about Garbage Collection**  
The Microsoft framework System namespace have the GC class, which exposes more method and property about garbage collection.  
 **MaxGeneration**  
This property in the GC class returns the total number of generations.

|  |
| --- |
| using System;  class GCExample1  {      public static void Main(string[] args)      {          try          {              Console.WriteLine("GC Maximum Generations:" + GC.MaxGeneration);          }          catch (Exception oEx)          {              Console.WriteLine("Error:" + oEx.Message);          }      }  } |

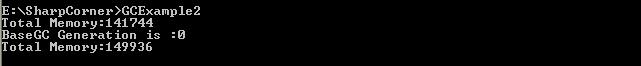
MaxGeneration property will return the highest generation in the garbage collection. It will be counted as total number of generations in the GC class which starts from 0.Here it has returned 2 as maxGeneration. That means totally three generations in the Garbage Collection. They are Generation 0, Generation 1 and Generation 2.

**GetTotalMemory and GetGeneration**



|  |
| --- |
| using System;  class BaseGC  {      public void Display()      {          Console.WriteLine("Example Method");      }  }    class GCExample2  {      public static void Main(string[] args)      {          try          {              Console.WriteLine("Total Memory:" + GC.GetTotalMemory(false));              BaseGC oBaseGC = new BaseGC();              Console.WriteLine("BaseGC Generation is :" + GC.GetGeneration(oBaseGC));              Console.WriteLine("Total Memory:" + GC.GetTotalMemory(false));          }          catch (Exception oEx)          {              Console.WriteLine("Error:" + oEx.Message);          }      }} |

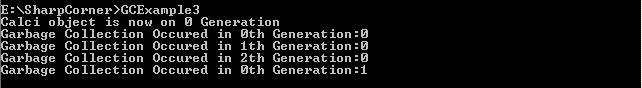
Here GetTotalMemory shows the total number of memory occupied by the various resources. Here I have added one more managed code objects in the heap memory. After adding, the size of the memory has increased.   
  
The GetGeneration method will find out the particular managed object in the which generation. Here it shows the Object oBaseGC in the 0th generation.



**CollectionCount and Collect**

|  |
| --- |
| using System;  class Calci  {      public int Add(int a, int b)      {          return (a + b);      }      public int Sub(int a, int b)      {          return (a - b);      }      public int Multi(int a, int b)      {          return (a \* b);      }      public int Divide(int a, int b)      {          return (a / b);      }  }    class GCExample3  {      public static void Main(string[] args)      {          Calci oCalci = new Calci();          Console.WriteLine("Calci object is now on " + GC.GetGeneration(oCalci) + " Generation");          Console.WriteLine("Garbage Collection Occured in 0th Generation:" +GC.CollectionCount(0));          Console.WriteLine("Garbage Collection Occured in 1th Generation:" +GC.CollectionCount(1));          Console.WriteLine("Garbage Collection Occured in 2th Generation:" +GC.CollectionCount(2));          GC.Collect(0);          Console.WriteLine("Garbage Collection Occured in 0th Generation:" +GC.CollectionCount(0));      }  } |

The CollectionCount helps us to find out the generation wise garbage collection occurred. As we know there are totally three generations in the garbage collector. Here I have passed argument as one for know the first generation. Initially it was 0. Then through the code I have collected the unused objects in the 0th generation. Again I have checked the CollectionCount in the 0thgeneration. Now it says 1.  
  
The Collect method used to collect the unreferenced objects in the heap memory. It will clear the object and reclaim the memory space.



## **C# GENERICS**

Generics allow you to delay the specification of the data type of programming elements in a class or a method, until it is actually used in the program. In other words, generics allow you to write a class or method that can work with any data type.

You write the specifications for the class or the method, with substitute parameters for data types. When the compiler encounters a constructor for the class or a function call for the method, it generates code to handle the specific data type. A simple example would help understanding the concept:

using System;

using System.Collections.Generic;

namespace GenericApplication

{

public class MyGenericArray<T>

{

private T[] array;

public MyGenericArray(int size)

{

array = new T[size + 1];

}

public T getItem(int index)

{

return array[index];

}

public void setItem(int index, T value)

{

array[index] = value;

}

}

class Tester

{

static void Main(string[] args)

{

//declaring an int array

MyGenericArray<int> intArray = new MyGenericArray<int>(5);

//setting values

for (int c = 0; c < 5; c++)

{

intArray.setItem(c, c\*5);

}

//retrieving the values

for (int c = 0; c < 5; c++)

{

Console.Write(intArray.getItem(c) + " ");

}

Console.WriteLine();

//declaring a character array

MyGenericArray<char> charArray = new MyGenericArray<char>(5);

//setting values

for (int c = 0; c < 5; c++)

{

charArray.setItem(c, (char)(c+97));

}

//retrieving the values

for (int c = 0; c< 5; c++)

{

Console.Write(charArray.getItem(c) + " ");

}

Console.WriteLine();

Console.ReadKey();

}

}

}

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

0 5 10 15 20

a b c d e

### **Features of Generics**

Generics is a technique that enriches your programs in the following ways:

* It helps you to maximize code reuse, type safety, and performance.
* You can create generic collection classes. The .NET Framework class library contains several new generic collection classes in the*System.Collections.Generic* namespace. You may use these generic collection classes instead of the collection classes in the*System.Collections* namespace.
* You can create your own generic interfaces, classes, methods, events, and delegates.
* You may create generic classes constrained to enable access to methods on particular data types.
* You may get information on the types used in a generic data type at run-time by means of reflection.

### **Generic Methods**

In the previous example, we have used a generic class; we can declare a generic method with a type parameter. The following program illustrates the concept:

using System;

using System.Collections.Generic;

namespace GenericMethodAppl

{

class Program

{

static void Swap<T>(ref T lhs, ref T rhs)

{

T temp;

temp = lhs;

lhs = rhs;

rhs = temp;

}

static void Main(string[] args)

{

int a, b;

char c, d;

a = 10;

b = 20;

c = 'I';

d = 'V';

//display values before swap:

Console.WriteLine("Int values before calling swap:");

Console.WriteLine("a = {0}, b = {1}", a, b);

Console.WriteLine("Char values before calling swap:");

Console.WriteLine("c = {0}, d = {1}", c, d);

//call swap

Swap<int>(ref a, ref b);

Swap<char>(ref c, ref d);

//display values after swap:

Console.WriteLine("Int values after calling swap:");

Console.WriteLine("a = {0}, b = {1}", a, b);

Console.WriteLine("Char values after calling swap:");

Console.WriteLine("c = {0}, d = {1}", c, d);

Console.ReadKey();

}

}

}

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

Int values before calling swap:

a = 10, b = 20

Char values before calling swap:

c = I, d = V

Int values after calling swap:

a = 20, b = 10

Char values after calling swap:

c = V, d = I

### **Generic Delegates**

You can define a generic delegate with type parameters. For example:

delegate T NumberChanger<T>(T n);

The following example shows use of this delegate:

using System;

using System.Collections.Generic;

delegate T NumberChanger<T>(T n);

namespace GenericDelegateAppl

{

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<int> nc1 = new NumberChanger<int>(AddNum);

NumberChanger<int> nc2 = new NumberChanger<int>(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();

}

}

}

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

Value of Num: 35

Value of Num: 175

## **STATIC METHODS VIRTUAL METHODS**

### **Virtual Methods**

A virtual method is a method that can be redefined in derived classes. A virtual method has an implementation in a base class as well as derived the class. It is used when a method's basic functionality is the same but sometimes more functionality is needed in the derived class. A virtual method is created in the base class that can be overriden in the derived class. We create a virtual method in the base class using the virtual keyword and that method is overriden in the derived class using the override keyword.

When a method is declared as a virtual method in a base class then that method can be defined in a base class and it is optional for the derived class to override that method. The overriding method also provides more than one form for a method. Hence it is also an example for polymorphism.

When a method is declared as a virtual method in a base class and that method has the same definition in a derived class then there is no need to override it in the derived class. But when a virtual method has a different definition in the base class and the derived class then there is a need to override it in the derived class.

When a virtual method is invoked, the run-time type of the object is checked for an overriding member. The overriding member in the most derived class is called, which might be the original member, if no derived class has overridden the member.

**Virtual Method**

1. By default, methods are non-virtual. We can't override a non-virtual method.
2. We can't use the virtual modifier with the static, abstract, private or override modifiers.

**Example1**

**Difference between virtual and non-virtual methods.**

We have two classes; one is a "Vehicle" class and another is a "Cart" class. The "Vehicle" class is the base class that has two methods; one is a virtual method "Speed()" and another is a non-virtual method "Average()". So the base class virtual method "Speed()" is overriden in the sub class. We have one more class "Program" (the execution class) that has an entry point where we create an instance of sub class "Cart" and that instance is assigned to the base class "Vehicle" type. When we call virtual and non-virtual methods by both class's instance then according to the run type the instance virtual method implementation is invoked; in other words both class's instances invoke the subclass override method and the non-virtual method invoked is determined based on the instance of the class.

using System;

namespace VirtualExample  
{     
    class Vehicle  
    {     
       public double distance=0.0;  
       public double hour =0.0;  
       public double fuel =0.0;

       public Vehicle(double distance, double hour, double fuel)  
       {  
           this.distance = distance;  
           this.hour = hour;  
           this.fuel = fuel;  
       }

       public void Average()  
       {  
           double average = 0.0;  
           average = distance / fuel;  
           Console.WriteLine("Vehicle Average is {0:0.00}", average);  
       }

       public virtual void Speed()  
       {  
           double speed = 0.0;  
           speed = distance / hour;  
           Console.WriteLine("Vehicle Speed is {0:0.00}", speed);  
       }  
    }

    class Car : Vehicle  
    {  
        public Car(double distance, double hour, double fuel)  
            : base(distance, hour, fuel)  
        {  
        }  
      public void Average()  
        {  
            double average = 0.0;  
            average = distance / fuel;  
            Console.WriteLine("Car Average is {0:0.00}", average);  
        }

        public override void Speed()  
        {  
            double speed = 0.0;             
            speed = distance / hour;  
            Console.WriteLine("Car Speed is {0:0.00}", speed);  
        }  
    }

    class Program  
   {  
        static void Main(string[] args)  
        {  
             double distance,hour,fuel=0.0;  
             Console.WriteLine("Enter the Distance");  
             distance = Double.Parse(Console.ReadLine());  
             Console.WriteLine("Enter the Hours");  
             hour = Double.Parse(Console.ReadLine());  
             Console.WriteLine("Enter the Fuel");  
             fuel = Double.Parse(Console.ReadLine());  
             Car objCar = new Car(distance,hour,fuel);  
             Vehicle objVeh = objCar;  
             objCar.Average();  
             objVeh.Average();  
             objCar.Speed();  
             objVeh.Speed();  
            Console.Read();  
        }         
    }  
}



**Example2**

**Invoked Virtual Method that override and not override in derived classes.**

We have three classes "Shape","Rectangle" and "Circle". Class "Shape" is a base class. Both "Rectangle" and "Circle" are derived classes from the base class. Base class "Shape" has virtual method "Area()". Virtual method "Area()" is overriden in the derived class "Rectangle" but not overriden in the derived class "Circle". When the virtual methods are overriden in a derived class and that derived class uses an instance then invokes a derived class overriden method. When a virtual method is not overriden in a derived class and uses that derived class instance then invokes base class virtual method.

using System;

namespace VirtualExample  
{     
    class Shape  
    {     
       public double length=0.0;  
       public double width =0.0;  
       public double radius =0.0;

       public Shape(double length, double width)  
       {  
           this.length = length;  
           this.width = width;            
       }

       public Shape(double radius)  
       {  
           this.radius = radius;  
       }

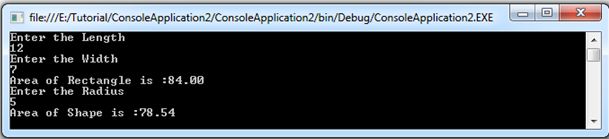
       public  virtual void Area()  
       {            
           double area = 0.0;  
           area = Math.PI \* Math.Pow(radius, 2);  
           Console.WriteLine("Area of Shape is :{0:0.00} ", area);  
       }  
    }

    class Rectangle  : Shape  
    {

        public Rectangle(double length, double width): base(length, width)  
        {  
        }

        public override void Area()  
        {  
            double area = 0.0;  
            area = length \* width;  
            Console.WriteLine("Area of Rectangle is :{0:0.00} ", area);  
        }  
    }  
     class Circle : Shape  
    {  
        public Circle(double radius)  
            : base(radius)  
        {  
        }  
    }

    class Program  
    {  
        static void Main(string[] args)  
        {  
             double length,width,radius=0.0;  
             Console.WriteLine("Enter the Length");  
             length = Double.Parse(Console.ReadLine());  
             Console.WriteLine("Enter the Width");  
             width = Double.Parse(Console.ReadLine());  
             Rectangle objRectangle = new Rectangle(length, width);  
              objRectangle.Area();  
             Console.WriteLine("Enter the Radius");  
             radius = Double.Parse(Console.ReadLine());  
             Circle objCircle = new Circle(radius);  
             objCircle.Area();  
            Console.Read();  
        }         
    }  
}



## **SEALED CLASSES**

Sealed class is used to define the inheritance level of a class.

The sealed modifier is used to prevent derivation from a class. An error occurs if a sealed class is specified as the base class of another class.

**Some points to remember:**     
  
1.  A class, which restricts inheritance for security reason is declared, sealed class.  
2.  Sealed class is the last class in the hierarchy.  
3.  Sealed class can be a derived class but can't be a base class.  
4.  A sealed class cannot also be an abstract class. Because abstract class has to provide functionality and here we are   
     restricting it to inherit.  
  
***Practical demonstration of sealed class***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace sealed\_class

{

    class Program

    {

        public sealed class BaseClass

        {

            public void Display()

        {

            Console.WriteLine("This is a sealed class which can;t be further inherited");

        }

    }

        public class Derived : BaseClass

        {

            // this Derived class can;t inherit BaseClass because it is sealed

        }

        static void Main(string[] args)

        {

            BaseClass obj = new BaseClass();

            obj.Display();

            Console.ReadLine();

        }

    }

}

### Sealed Methods

Sealed method is used to define the overriding level of a virtual method.

Sealed keyword is always used with override keyword.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace sealed\_method

{

    class Program

    {

        public class BaseClass

        {

            public virtual void Display()

            {

                Console.WriteLine("Virtual method");

            }

        }

       public class DerivedClass : BaseClass

        {

            // Now the display method have been sealed and can;t be overridden

            public override sealed void Display()

            {

                Console.WriteLine("Sealed method");

            }

        }

       //public class ThirdClass : DerivedClass

       //{

       //    public override void Display()

       //    {

       //        Console.WriteLine("Here we try again to override display method which is not possible and will give error");

       //    }

       //}

        static void Main(string[] args)

        {

            DerivedClass ob1 = new DerivedClass();

            ob1.Display();

            Console.ReadLine();

        }

    }

}

## **ALL ABOUT COLLECTIONS**

Collection classes are specialized classes for data storage and retrieval. These classes provide support for stacks, queues, lists, and hash tables. Most collection classes implement the same interfaces.

Collection classes serve various purposes, such as allocating memory dynamically to elements and accessing a list of items on the basis of an index etc. These classes create collections of objects of the Object class, which is the base class for all data types in C#.

|  |  |
| --- | --- |
| **Class** | **Description and Useage** |
| [**ArrayList**](https://www.tutorialspoint.com/csharp/csharp_arraylist.htm) | It represents ordered collection of an object that can be **indexed** individually.  It is basically an alternative to an array. However, unlike array you can add and remove items from a list at a specified position using an **index** and the array resizes itself automatically. It also allows dynamic memory allocation, adding, searching and sorting items in the list. |
| [**Hashtable**](https://www.tutorialspoint.com/csharp/csharp_hashtable.htm) | It uses a **key** to access the elements in the collection.  A hash table is used when you need to access elements by using key, and you can identify a useful key value. Each item in the hash table has a **key/value** pair. The key is used to access the items in the collection. |
| [**SortedList**](https://www.tutorialspoint.com/csharp/csharp_sortedlist.htm) | It uses a **key** as well as an **index** to access the items in a list.  A sorted list is a combination of an array and a hash table. It contains a list of items that can be accessed using a key or an index. If you access items using an index, it is an ArrayList, and if you access items using a key , it is a Hashtable. The collection of items is always sorted by the key value. |
| [**Stack**](https://www.tutorialspoint.com/csharp/csharp_stack.htm) | It represents a **last-in, first out** collection of object.  It is used when you need a last-in, first-out access of items. When you add an item in the list, it is called **pushing** the item and when you remove it, it is called **popping** the item. |
| [**Queue**](https://www.tutorialspoint.com/csharp/csharp_queue.htm) | It represents a **first-in, first out** collection of object.  It is used when you need a first-in, first-out access of items. When you add an item in the list, it is called**enqueue** and when you remove an item, it is called**deque**. |
| [**BitArray**](https://www.tutorialspoint.com/csharp/csharp_bitarray.htm) | It represents an array of the **binary representation** using the values 1 and 0.  It is used when you need to store the bits but do not know the number of bits in advance. You can access items from the BitArray collection by using an **integer index**, which starts from zero. |

For many applications, you want to create and manage groups of related objects. There are two ways to group objects: by creating arrays of objects, and by creating collections of objects.

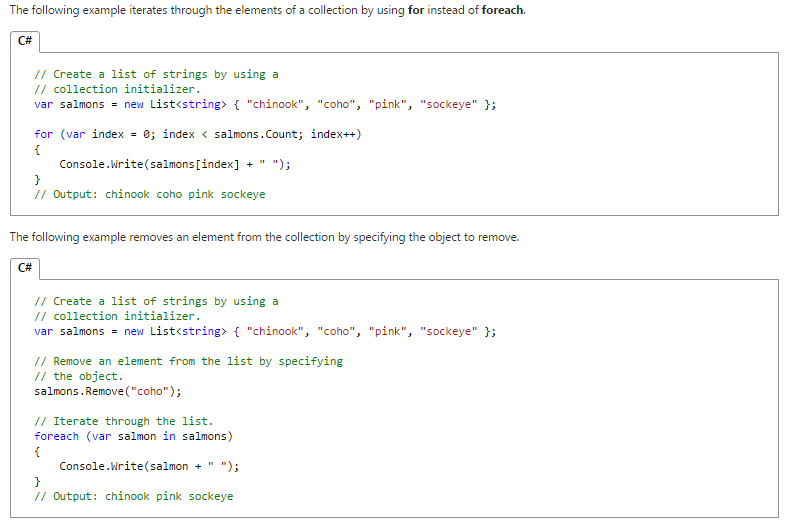
Arrays are most useful for creating and working with a fixed number of strongly-typed objects. For information about arrays, see [Arrays (C# Programming Guide)](https://msdn.microsoft.com/en-us/library/9b9dty7d.aspx).

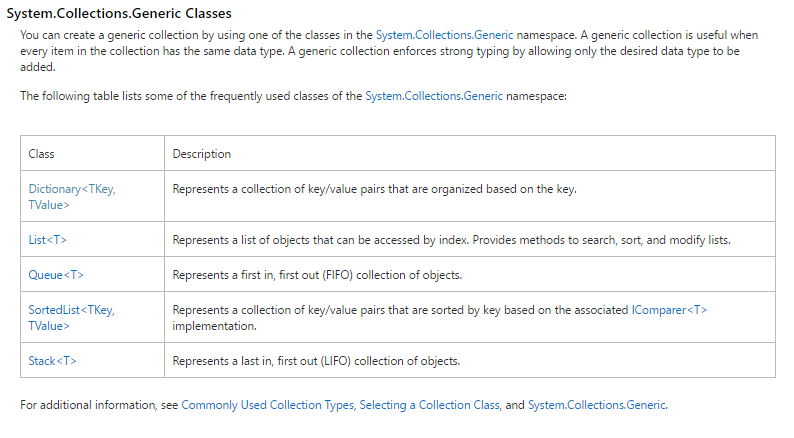
Collections provide a more flexible way to work with groups of objects. Unlike arrays, the group of objects you work with can grow and shrink dynamically as the needs of the application change. For some collections, you can assign a key to any object that you put into the collection so that you can quickly retrieve the object by using the key.

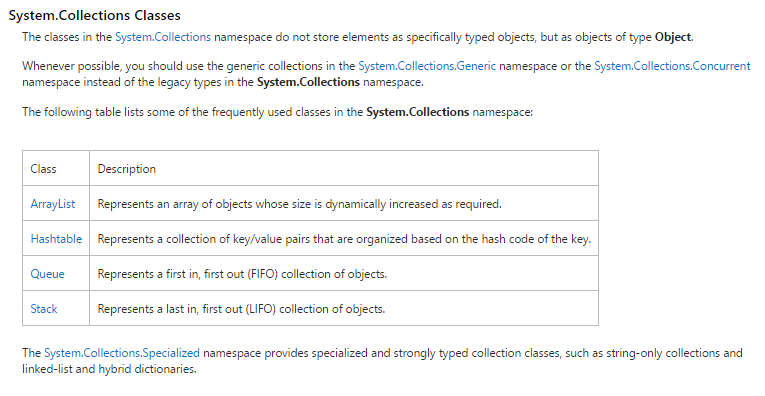
A collection is a class, so you must declare an instance of the class before you can add elements to that collection.

If your collection contains elements of only one data type, you can use one of the classes in the [System.Collections.Generic](https://msdn.microsoft.com/en-us/library/system.collections.generic.aspx) namespace. A generic collection enforces type safety so that no other data type can be added to it. When you retrieve an element from a generic collection, you do not have to determine its data type or convert it.









## **MICROSOFT .NET ASSEMBLY**

Execution of Dotnet framework?

The managed execution process includes the following steps, which are discussed in detail later in this topic:

1. [Choosing a compiler](https://msdn.microsoft.com/en-us/library/k5532s8a(v=vs.110).aspx#choosing_a_compiler).

To obtain the benefits provided by the common language runtime, you must use one or more language compilers that target the runtime.

1. [Compiling your code to MSIL](https://msdn.microsoft.com/en-us/library/k5532s8a(v=vs.110).aspx#compiling_to_msil).

Compiling translates your source code into Microsoft intermediate language (MSIL) and generates the required metadata.

1. [Compiling MSIL to native code](https://msdn.microsoft.com/en-us/library/k5532s8a(v=vs.110).aspx#compiling_msil_to_native_code).

At execution time, a just-in-time (JIT) compiler translates the MSIL into native code. During this compilation, code must pass a verification process that examines the MSIL and metadata to find out whether the code can be determined to be type safe.

1. [Running code](https://msdn.microsoft.com/en-us/library/k5532s8a(v=vs.110).aspx#running_code).

The common language runtime provides the infrastructure that enables execution to take place and services that can be used during execution.

**24). Difference between Dotnet and Dotnet framework?**

.net platform supports programming languages that are .net compatible. It is the platform using which we can build and develop the applications.   
  
.net framework is the engine inside the .net platform which actually compiles and produces the executable code.   
.net framework contains CLR(Common Language Runtime) and FCL(Framework Class Library) using which it produces the platform independent codes.

#### **What is the .NET Framework?**

The Microsoft .NET Framework is a platform for building, deploying, and running Web Services and applications. It provides a highly productive, standards-based, multi-language environment for integrating existing investments with next-generation applications and services as well as the agility to solve the challenges of deployment and operation of Internet-scale applications. The .NET Framework consists of three main parts: the common language runtime, a hierarchical set of unified class libraries, and a componentized version of Active Server Pages called ASP.NET.

#### **What is the common type system (CTS)?**

The common type system is a rich type system, built into the common language runtime, that supports the types and operations found in most programming languages. The common type system supports the complete implementation of a wide range of programming languages.

**What is the CTS, and how does it relate to the CLS?**

**CTS = Common Type System**. This is the full range of types that the .NET

runtime understands. Not all .NET languages support all the types in the CTS.

**CLS = Common Language Specification**. This is a subset of the CTS which all

.NET languages are expected to support. The idea is that any program which

uses CLS-compliant types can interoperate with any .NET program written in

any language. This interop is very fine-grained - for example a VB.NET class   can inherit from a C# class.

**IL/MSIL/CIL**- IL code is a CPU independent partially compiled code. It’s partially compiled because we do not know in what kind of environment .NET code will run and on runtime IL Code will compile to machine code using the environmental properties (CPU, OS, machine configuration, etc).

**ILDASM**- This is a tool provided by Visual Studio to view IL code. To run ILDASM, we have to select option “Visual Studio Command Prompt” from “Visual Studio Tools” and type ildasm. It will open the ildasm tool where we can open any *exe/dll.ildasm* tool read the assembly by reflection and it is showing us various properties, methods which our assembly has. Here, we can see IL code of any method/property by clicking on that.

**CLR**- CLR is the heart of the .NET framework and it does 4 primary important things:

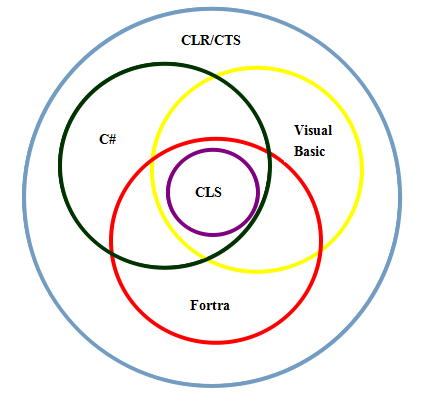
1. Garbage collection
2. CAS (Code Access Security)
3. CV (Code Verification)
4. IL to Native translation

**CTS** - CTS ensures that data types defined in two different languages get compiled to a common data type. This is useful because there may be situations when we want code in one language to be called in other language.

We can see a practical demonstration of CTS by creating the same application in C# and VB.NET and then compare the IL code of both applications. Here, the datatype of both IL code is same.

**CLS** - CLS is a subset of CTS. CLS is a set of rules or guidelines. When any programming language adheres to these set of rules, it can be consumed by any .NET language.CTS.

**JIT**- JIT compiles the IL code to Machine code just before execution and then saves this transaction in memory.



#### **What is the Common Language Specification (CLS)?**

The Common Language Specification is a set of constructs and constraints that serves as a guide for library writers and compiler writers. It allows libraries to be fully usable from any language supporting the CLS, and for those languages to integrate with each other. The Common Language Specification is a subset of the common type system. The Common Language Specification is also important to application developers who are writing code that will be used by other developers. When developers design publicly accessible APIs following the rules of the CLS, those APIs are easily used from all other programming languages that target the common language runtime.

#### **What is the Microsoft Intermediate Language (MSIL)?**

MSIL is the CPU-independent instruction set into which .NET Framework programs are compiled. It contains instructions for loading, storing, initializing, and calling methods on objects.

Combined with metadata and the common type system, MSIL allows for true cross-language integration.

Prior to execution, MSIL is converted to machine code. It is not interpreted.

It is a CPU independent partially compiled code. Partial/half compiled means this code is not yet compiled to machine/CPU specific instructions.

**IL code compilation:**

We do not know in what kind of [environment.NET](http://environment.net/) code will run. In other words we do not know what can be the end operating system, CPU configuration, machine configuration, security configuration etc. So the IL code is half compiled and on runtime this code is compiled to machine specific using the environmental properties (CPU, OS, machine configuration etc).

**Viewing IL code:**

Yes by using ILDASM simple tool we can view a IL code of a DLL or EXE. In order to view IL code using ILDASM, go to visual studio command prompt and run “ILDASM.EXE”. Once ILDASM is running you view the IL code.

What is COM, COM+ and DCOM ?

COM (Component Object Model) A standard that is used to for

communication between OS and the softwares. COM is used to create

reusable software components

COM+ : COM+ is an extension of Component Object Model (COM). COM+ is

both an OOP architecture and a set of operating system services.

DCOM an extension of the Component Object Model (COM) that allows COM

components to communicate across network boundaries. Traditional COM

components can only perform interprocess communication across process

boundaries on the same machine. DCOM uses the RPC mechanism to

transparently send and receive information between COM components (i.e.,

clients and servers) on the same network.

**25). What is GAC?**

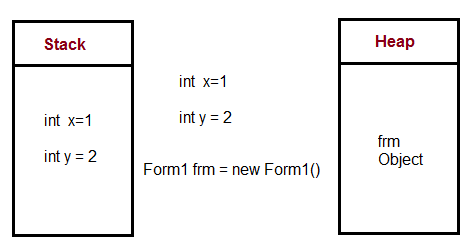
The Global Assembly Cache (GAC) is a folder in Windows directory to store the .NET assemblies that are specifically designated to be shared by all applications executed on a system.

**26).  What is the difference between stack and heap memory?**

Stack is used for static memory allocation and Heap for dynamic memory allocation, both stored in the computer's RAM .

Variables allocated on the stack are stored directly to the memory and access to this memory is very fast, and it's allocation is dealt with when the program is compiled. When a function or a method calls another function which in turns calls another function etc., the execution of all those functions remains suspended until the very last function returns its value. The stack is always reserved in a LIFO order, the most recently reserved block is always the next block to be freed. This makes it really simple to keep track of the stack, freeing a block from the stack is nothing more than adjusting one pointer.

Variables allocated on the heap have their memory allocated at run time and accessing this memory is a bit slower, but the heap size is only limited by the size of virtual memory . Element of the heap have no dependencies with each other and can always be accessed randomly at any time. You can allocate a block at any time and free it at any time. This makes it much more complex to keep track of which parts of the heap are allocated or free at any given time.



You can use the stack if you know exactly how much data you need to allocate before compile time and it is not too big. You can use heap if you don't know exactly how much data you will need at runtime or if you need to allocate a lot of data.

In a multi-threaded situation each thread will have its own completely independent stack but they will share the heap. Stack is thread specific and Heap is application specific. The stack is important to consider in exception handling and thread executions.

**27). What is the difference between synchronous and Asynchronous process?**

Asynchronous refers to processes that do not depend on each other's outcome, and can therefore occur on different threads simultaneously. The opposite is synchronous. Synchronous processes wait for one to complete before the next begins. For those Group Policy settings for which both types of processes are available as options, you choose between the faster asynchronous or the safer, more predictable synchronous processing.

**28). What is threading?**

Threads: Threads are often called lightweight processes. However they are not process.es A Thread is a small set of executable instructions, which can be used to isolate a task from a process. Multiple threads are efficient way to obtain parallelism of hardware and give interactive user interaction to your applications.

C# Thread:. . Net Framework has thread-associated classes in System.Threading namespace. The following steps demonstrate how to create a thread in C#.

Step 1. Create a System.Threading.Thread object.

Creating an object to System.Threading.Thread creates a managed thread in .Net environment. The Thread class has only one constructor, which takes a ThreadStart delegate as parameter. The ThreadStart delegate is wrap around the callback method, which will be called when we start the thread.

Step 2: Create the call back function

This method will be a starting point for our new thread. It may be an instance function of a class or a static function. Incase of instance function, we should create an object of the class, before we create the ThreadStartdelegate. For static functions we can directly use the function name to instantiate the delegate. The callback function should have void as both return type and parameter. Because the ThreadStart delegate function is declared like this. (For more information on delegate see MSDN for “Delegates”).

Step 3: Starting the Thread.

We can start the newly created thread using the Thread’s Start method. This is an asynchronous method, which requests the operating system to start the current thread.

For Example:

Hide   Shrink    Copy Code



*// This is the Call back function for thread.*

Public static void MyCallbackFunction()

{

while (true)

{

System.Console.WriteLine(“ Hey!, My Thread Function Running”);

………

}

}

public static void Main(String []args)

{

*// Create an object for Thread*

Thread MyThread = new Thread(new ThreadStart

(MyCallbackFunction));

MyThread.Start()

……

}

Killing a Thread:  
  
We can kill a thread by calling the Abort method of the thread. Calling the Abort method causes the current thread to exit by throwing the ThreadAbortException.

Hide   Copy Code

MyThread.Abort();

Suspend and Resuming Thread:

We can suspend the execution of a thread and once again start its execution from another thread using the Thread object’s Suspend and Resume methods.

Hide   Copy Code

MyThread.Suspend() *// causes suspend the Thread Execution.*

MyThread.Resume() *// causes the suspended Thread to resume its execution.*

Thread State:

A Thread can be in one the following state.

|  |  |
| --- | --- |
| Thread State | Description |
| Unstarted | Thread is Created within the common language run time but not Started still. |
| Running | After a Thread calls Start method |
| WaitSleepJoin | After a Thread calls its wait or Sleep or Join method. |
| Suspended | Thread Responds to a Suspend method call. |
| Stopped | The Thread is Stopped, either normally or Aborted. |

We can check the current state of a thread using the Thread’s ThreadState property.

Thread Priorty:

The Thread class’s ThreadPriority property is used to set the priority of the Thread. A Thread may have one of the following values as its Priority:  Lowest, BelowNormal, Normal, AboveNormal, Highest. The default property of a thread is Normal.

**29). What is the difference between process and thread?**

Process:

* An executing instance of a program is called a process.
* Some operating systems use the term ‘task‘ to refer to a program that is being executed.
* A process is always stored in the main memory also termed as the primary memory or random access memory.
* Therefore, a process is termed as an active entity. It disappears if the machine is rebooted.
* Several process may be associated with a same program.
* On a multiprocessor system, multiple processes can be executed in parallel.
* On a uni-processor system, though true parallelism is not achieved, a process scheduling algorithm is applied and the processor is scheduled to execute each process one at a time yielding an illusion of concurrency.
* Example: Executing multiple instances of the ‘Calculator’ program. Each of the instances are termed as a process.

Thread:

* A thread is a subset of the process.
* It is termed as a ‘lightweight process’, since it is similar to a real process but executes within the context of a process and shares the same resources allotted to the process by the kernel.
* Usually, a process has only one thread of control – one set of machine instructions executing at a time.
* A process may also be made up of multiple threads of execution that execute instructions concurrently.
* Multiple threads of control can exploit the true parallelism possible on multiprocessor systems.
* On a uni-processor system, a thread scheduling algorithm is applied and the processor is scheduled to run each thread one at a time.
* All the threads running within a process share the same address space, file descriptors, stack and other process related attributes.
* Since the threads of a process share the same memory, synchronizing the access to the shared data withing the process gains unprecedented importance.

**30). What is reflection?**

Reflection objects are used for obtaining type information at runtime. The classes that give access to the metadata of a running program are in theSystem.Reflection namespace.

The System.Reflection namespace contains classes that allow you to obtain information about the application and to dynamically add types, values, and objects to the application.

#### **Applications of Reflection**

Reflection has the following applications:

* It allows view attribute information at runtime.
* It allows examining various types in an assembly and instantiate these types.
* It allows late binding to methods and properties
* It allows creating new types at runtime and then performs some tasks using those types.

# **ASP.NET**

1. **What is ASP.NET**

ASP.NET is an open source web framework for building modern web applications and services. With ASP.NET you can quickly create web sites based on HTML, CSS and JavaScript

ASP.NET is an open-source server-side web application framework designed for web development to produce dynamic web pages. It was developed by Microsoft to allow programmers to build dynamic web sites, web applications and web services

1. **What is View State**

**Background**  
A web application is stateless. That means that a new instance of a page is created every time when we make a request to the server to get the page and after the round trip our page has been lost immediately. It only happens because of one server, all the controls of the Web Page is created and after the round trip the server destroys all the instances. So to retain the values of the controls we use state management techniques.

View State is the method to preserve the Value of the Page and Controls between round trips. It is a Page-Level State Management technique. View State is turned on by default and normally serializes the data in every control on the page regardless of whether it is actually used during a post-back.

1. **What is State Management in ASP.NET**



1. **Postback**

PostBack is the name given to the process of submitting an ASP.NET page to the server for processing. PostBack is done if certain credentials of the page are to be checked against some sources (such as verification of username and password using database). This is something that a client machine is not able to accomplish and thus these details have to be 'posted back' to the server.  
  
***What is AutoPostBack Property in ASP.NET:***  
  
If we create a web Page, which consists of one or more Web Controls that are configured to use AutoPostBack (Every Web controls will have their own AutoPostBack property), the ASP.Net adds a special JavaScipt function to the rendered HTML Page. This function is named \_doPostBack() . When Called, it triggers a PostBack, sending data back to the web Server.  
ASP.NET also adds two additional hidden input fields that are used to pass information back to the server. This information consists of ID of the Control that raised the event and any additional information if needed.

To work with the ASP.Net Web Controls events, we need a solid understanding of the web page life cycle. The following actions will be taken place when a user changes a control that has the AutoPostBack property set to true :

1. On the client side, the JavaScript \_doPostBack function is invoked, and the page is resubmitted to the server.
2. ASP.NET re-creates the Page object using the .aspx file.
3. ASP.NET retrieves state information from the hidden view state field and updates the controls accordingly.
4. The Page.Load event is fired.
5. The appropriate change event is fired for the control. (If more than one control has been changed, the order of change events is undetermined.)
6. The Page.PreRender event fires, and the page is rendered (transformed from a set of objects to an HTML page).
7. Finally, the Page.Unload event is fired.
8. The new page is sent to the client.
9. Page Life Cycle of ASP.NET



# **WPF**

# **ADO.NET**

Dataset,

data table

SQlreader sqladapter sql connection

Executereader

Executesclar

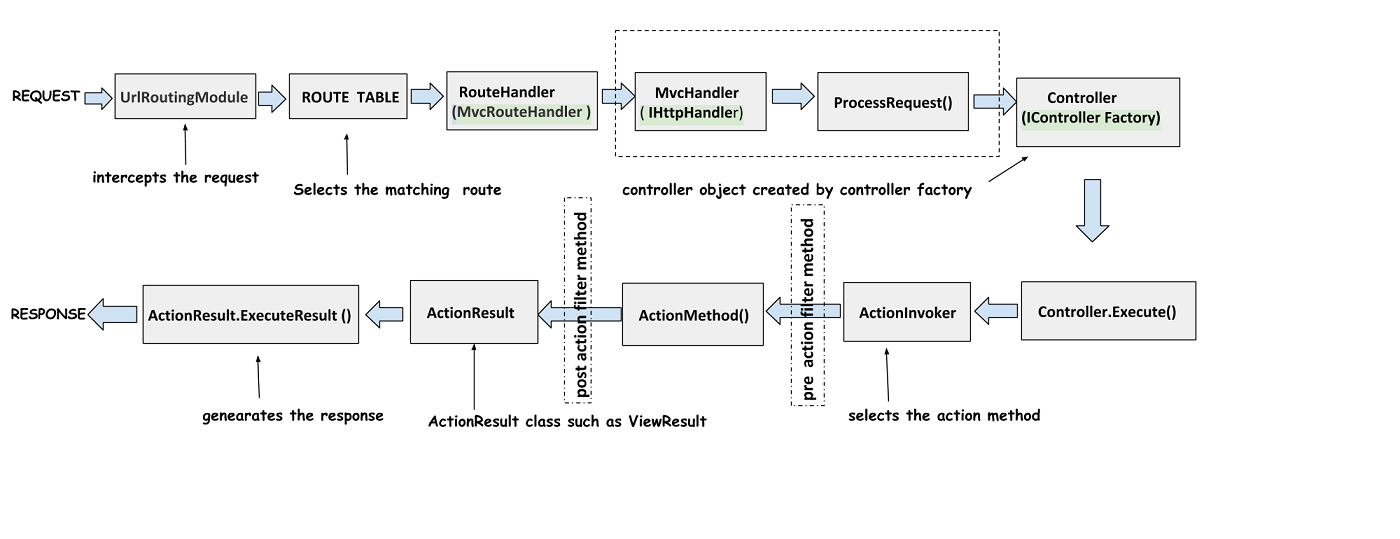
Executenonquery

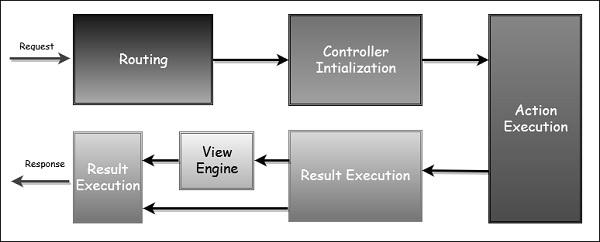
Dataadapter

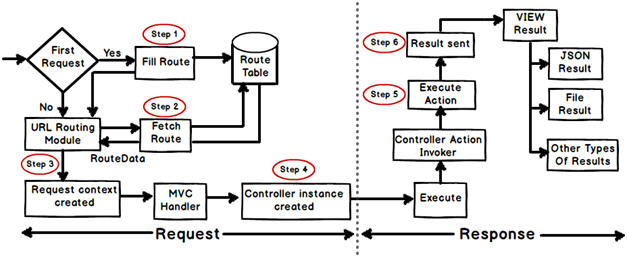
SQL injection

# **MVC**

**MVC Application Life Cycle**







## **What is MVC (Model View Controller)?**

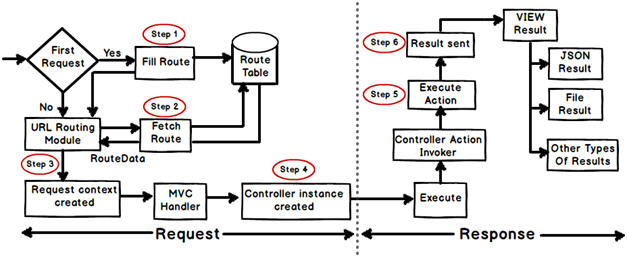
MVC is an architectural pattern which separates the representation and user interaction. It’s divided into three broader sections, Model, View, and Controller. Below is how each one of them handles the task.

* The View is responsible for the look and feel.
* Model represents the real world object and provides data to the View.
* The Controller is responsible for taking the end user request and loading the appropriate Model and View.

***Figure: MVC (Model view controller)***

## **Explain MVC application life cycle?**

There are six broader events which occur in MVC application life cycle below diagrams summarize it.

  
Image Courtesy: - <http://www.dotnetinterviewquestions.in/article_explain-mvc-application-life-cycle_210.html>

Any web application has two main execution steps first understanding the request and depending on the type of the request sending out appropriate response. MVC application life cycle is not different it has two main phases first creating the request object and second sending our response to the browser.

**Creating the request object: -**The request object creation has four major steps. Below is the detail explanation of the same.

**Step 1 Fill route: -** MVC requests are mapped to route tables which in turn specify which controller and action to be invoked. So if the request is the first request the first thing is to fill the route table with routes collection. This filling of route table happens in the global.asax file.

**Step 2 Fetch route: -** Depending on the URL sent “UrlRoutingModule” searches the route table to create “RouteData” object which has the details of which controller and action to invoke.

**Step 3 Request context created: -** The “RouteData” object is used to create the “RequestContext” object.

**Step 4 Controller instance created: -** This request object is sent to “MvcHandler” instance to create the controller class instance. Once the controller class object is created it calls the “Execute” method of the controller class.

**Creating Response object: -** This phase has two steps executing the action and finally sending the response as a result to the view.

## **Is MVC suitable for both Windows and Web applications?**

The MVC architecture is suited for a web application than Windows. For Window applications, MVP, i.e., “Model View Presenter” is more applicable. If you are using WPF and Silverlight, MVVM is more suitable due to bindings.

## **What are the benefits of using MVC?**

There are two big benefits of MVC:

* Separation of concerns is achieved as we are moving the code-behind to a separate class file. By moving the binding code to a separate class file we can reuse the code to a great extent.
* Automated UI testing is possible because now the behind code (UI interaction code) has moved to a simple .NET class. This gives us opportunity to write unit tests and automate manual testing.

## **Is MVC different from a three layered architecture?**

MVC is an evolution of a three layered traditional architecture. Many components of the three layered architecture are part of MVC. So below is how the mapping goes:

|  |  |  |
| --- | --- | --- |
| **Functionality** | **Three layered / tiered architecture** | **Model view controller architecture** |
| Look and Feel | User interface | View |
| UI logic | User interface | Controller |
| Business logic /validations | Middle layer | Model |
| Request is first sent to | User interface | Controller |
| Accessing data | Data access layer | Data Access Layer |

***Figure: Three layered architecture***

## **What is the latest version of MVC?**

MVC 6 is the latest version which is also termed as ASP VNEXT.

## **What is the difference between each version of MVC 2, 3 , 4, 5 and 6?**

**MVC 6**

ASP.NET MVC and Web API has been merged in to one.

Dependency injection is inbuilt and part of MVC.

Side by side - deploy the runtime and framework with your application

Everything packaged with NuGet, Including the .NET runtime itself.

New JSON based project structure.

No need to recompile for every change. Just hit save and refresh the browser.

Compilation done with the new Roslyn real-time compiler.

vNext is Open Source via the .NET Foundation and is taking public contributions.

vNext (and Rosyln) also runs on Mono, on both Mac and Linux today.

**MVC 5**

One ASP.NET

Attribute based routing

Asp.Net Identity

Bootstrap in the MVC template

Authentication Filters

Filter overrides

**MVC 4**

ASP.NET Web API

Refreshed and modernized default project templates

New mobile project template

Many new features to support mobile apps

Enhanced support for asynchronous methods

**MVC 3**

Razor

Readymade project templates

HTML 5 enabled templates

Support for Multiple View Engines

JavaScript and Ajax

Model Validation Improvements

**MVC 2**

Client-Side Validation

Templated Helpers

Areas

Asynchronous Controllers

Html.ValidationSummary Helper Method

DefaultValueAttribute in Action-Method Parameters

Binding Binary Data with Model Binders

DataAnnotations Attributes

Model-Validator Providers

New RequireHttpsAttribute Action Filter

Templated Helpers

Display Model-Level Errors

## **What are HTML helpers in MVC?**

HTML helpers help you to render HTML controls in the view. For instance if you want to display a HTML textbox on the view , below is the HTML helper code.

Hide   Copy Code

<%= Html.TextBox("LastName") %>

For checkbox below is the HTML helper code. In this way we have HTML helper methods for every HTML control that exists.

Hide   Copy Code

<%= Html.CheckBox("Married") %>

## **What is the difference between “HTML.TextBox” vs “HTML.TextBoxFor”?**

Both of them provide the same HTML output, “HTML.TextBoxFor” is strongly typed while “HTML.TextBox” isn’t. Below is a simple HTML code which just creates a simple textbox with “CustomerCode” as name.

Hide   Copy Code

Html.TextBox("CustomerCode")

Below is “Html.TextBoxFor” code which creates HTML textbox using the property name ‘CustomerCode” from object “m”.

Hide   Copy Code

Html.TextBoxFor(m => m.CustomerCode)

In the same way we have for other HTML controls like for checkbox we have “Html.CheckBox” and “Html.CheckBoxFor”.

## **What is routing in MVC?**

Routing helps you to define a URL structure and map the URL with the controller.

For instance let’s say we want that when a user types “http://localhost/View/ViewCustomer/”, it goes to the “Customer” Controller and invokes the DisplayCustomer action. This is defined by adding an entry in to theroutes collection using the maproute function. Below is the underlined code which shows how the URL structure and mapping with controller and action is defined.

Hide   Copy Code

routes.MapRoute(

"View", *// Route name*

"View/ViewCustomer/{id}", *// URL with parameters*

new { controller = "Customer", action = "DisplayCustomer",

id = UrlParameter.Optional }); *// Parameter defaults*

## **Where is the route mapping code written?**

The route mapping code is written in "RouteConfig.cs" file and registered using "global.asax" application start event.

## **Can we map multiple URL’s to the same action?**

Yes, you can, you just need to make two entries with different key names and specify the same controller and action.

## **Explain attribute based routing in MVC?**

This is a feature introduced in MVC 5. By using the "Route" attribute we can define the URL structure. For example in the below code we have decorated the "GotoAbout" action with the route attribute. The route attribute says that the "GotoAbout" can be invoked using the URL structure "Users/about".

Hide   Copy Code

public class HomeController : Controller

{

[Route("Users/about")]

public ActionResult GotoAbout()

{

return View();

}

}

## **What is the advantage of defining route structures in the code?**

Most of the time developers code in the action methods. Developers can see the URL structure right upfront rather than going to the “routeconfig.cs” and see the lengthy codes. For instance in the below code the developer can see right upfront that the “GotoAbout” action can be invoked by four different URL structure.

This is much user friendly as compared to scrolling through the “routeconfig.cs” file and going through the length line of code to figure out which URL structure is mapped to which action.

Hide   Copy Code

public class HomeController : Controller

{

[Route("Users/about")]

[Route("Users/WhoareWe")]

[Route("Users/OurTeam")]

[Route("Users/aboutCompany")]

public ActionResult GotoAbout()

{

return View();

}

}

## **How can we navigate from one view to another using a hyperlink?**

By using the ActionLink method as shown in the below code. The below code will create a simple URL which helps to navigate to the “Home” controller and invoke the GotoHome action.

Hide   Copy Code

<%= Html.ActionLink("Home","Gotohome") %>

## **How can we restrict MVC actions to be invoked only by GET or POST?**

We can decorate the MVC action with the HttpGet or HttpPost attribute to restrict the type of HTTP calls. For instance you can see in the below code snippet the DisplayCustomer action can only be invoked by HttpGet. If we try to make HTTP POST on DisplayCustomer, it will throw an error.

Hide   Copy Code

[HttpGet]

public ViewResult DisplayCustomer(int id)

{

Customer objCustomer = Customers[id];

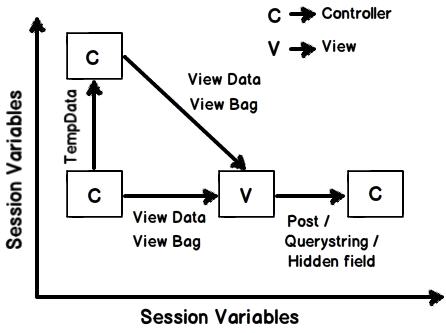
return View("DisplayCustomer",objCustomer);

}

## **How can we maintain sessions in MVC?**

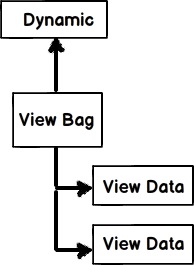
Sessions can be maintained in MVC by three ways: tempdata, viewdata, and viewbag.

## **What is the difference between tempdata, viewdata, and viewbag?**



***Figure: Difference between tempdata, viewdata, and viewbag***

* **Temp data** - Helps to maintain data when you move from one controller to another controller or from one action to another action. In other words when you redirect, tempdata helps to maintain data between those redirects. It internally uses session variables.
* **View data** - Helps to maintain data when you move from controller to view.
* **View Bag** - It’s a dynamic wrapper around view data. When you use Viewbag type, casting is not required. It uses the dynamic keyword internally.



***Figure: dynamic keyword***

* **Session variables -** By using session variables we can maintain data from any entity to any entity.
* **Hidden fields and HTML controls -** Helps to maintain data from UI to controller only. So you can send data from HTML controls or hidden fields to the controller using POST or GET HTTP methods.

Below is a summary table which shows the different mechanisms for persistence.

| **Maintains data between** | **ViewData/ViewBag** | **TempData** | **Hidden fields** | **Session** |
| --- | --- | --- | --- | --- |
| **Controller to Controller** | No | Yes | No | Yes |
| **Controller to View** | Yes | No | No | Yes |
| **View to Controller** | No | No | Yes | Yes |

## **What is difference between TempData and ViewData ?**

“TempData” maintains data for the complete request while “ViewData” maintains data only from Controller to the view.

## **Does “TempData” preserve data in the next request also?**

“TempData” is available through out for the current request and in the subsequent request it’s available depending on whether “TempData” is read or not.

So if “TempData” is once read it will not be available in the subsequent request.

## **What is the use of Keep and Peek in “TempData”?**

Once “TempData” is read in the current request it’s not available in the subsequent request. If we want “TempData” to be read and also available in the subsequent request then after reading we need to call “Keep” method as shown in the code below.

Hide   Copy Code

@TempData[&ldquo;MyData&rdquo;];

TempData.Keep(&ldquo;MyData&rdquo;);

The more shortcut way of achieving the same is by using “Peek”. This function helps to read as well advices MVC to maintain “TempData” for the subsequent request.

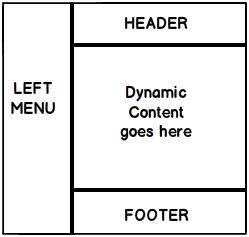
Hide   Copy Code

string str = TempData.Peek("Td").ToString();

If you want to read more in detail you can read from this detailed blog on [MVC Peek and Keep](http://www.codeproject.com/Articles/818493/MVC-Tempdata-Peek-and-Keep-confusion).

## **What are partial views in MVC?**

Partial view is a reusable view (like a user control) which can be embedded inside other view. For example let’s say all your pages of your site have a standard structure with left menu, header, and footer as shown in the image below.

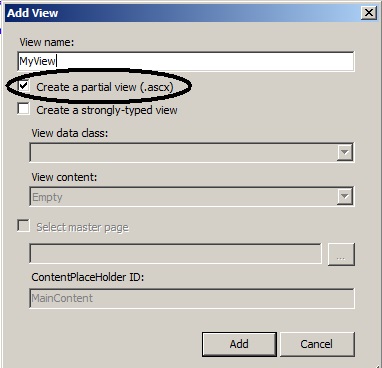


***Figure: Partial views in MVC***

For every page you would like to reuse the left menu, header, and footer controls. So you can go and create partial views for each of these items and then you call that partial view in the main view.

## **How did you create a partial view and consume it?**

When you add a view to your project you need to check the “Create partial view” check box.



F***igure: Create partial view***

Once the partial view is created you can then call the partial view in the main view using theHtml.RenderPartial method as shown in the below code snippet:

Hide   Copy Code

<body>

<div>

<% Html.RenderPartial("MyView"); %>

</div>

</body>

## **How can we do validations in MVC?**

One of the easiest ways of doing validation in MVC is by using data annotations. Data annotations are nothing but attributes which can be applied on model properties. For example, in the below code snippet we have a simple Customer class with a property customercode.

This CustomerCode property is tagged with a Required data annotation attribute. In other words if this model is not provided customer code, it will not accept it.

Hide   Copy Code

public class Customer

{

[Required(ErrorMessage="Customer code is required")]

public string CustomerCode

{

set;

get;

}

}

In order to display the validation error message we need to use the ValidateMessageFor method which belongs to the Html helper class.

Hide   Copy Code

<% using (Html.BeginForm("PostCustomer", "Home", FormMethod.Post))

{ %>

<%=Html.TextBoxFor(m => m.CustomerCode)%>

<%=Html.ValidationMessageFor(m => m.CustomerCode)%>

<input type="submit" value="Submit customer data" />

<%}%>

Later in the controller we can check if the model is proper or not by using the ModelState.IsValid property and accordingly we can take actions.

Hide   Copy Code

public ActionResult PostCustomer(Customer obj)

{

if (ModelState.IsValid)

{

obj.Save();

return View("Thanks");

}

else

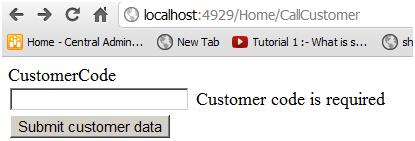
{

return View("Customer");

}

}

Below is a simple view of how the error message is displayed on the view.



***Figure: Validations in MVC***

## **Can we display all errors in one go?**

Yes, we can; use the ValidationSummary method from the Html helper class.

Hide   Copy Code

<%= Html.ValidationSummary() %>

What are the other data annotation attributes for validation in MVC?

If you want to check string length, you can use StringLength.

Hide   Copy Code

[StringLength(160)]

public string FirstName { get; set; }

In case you want to use a regular expression, you can use the RegularExpression attribute.

Hide   Copy Code

[RegularExpression(@"[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}")]public string Email { get; set; }

If you want to check whether the numbers are in range, you can use the Range attribute.

Hide   Copy Code

[Range(10,25)]public int Age { get; set; }

Sometimes you would like to compare the value of one field with another field, we can use the Compareattribute.

Hide   Copy Code

public string Password { get; set; }[Compare("Password")]public string ConfirmPass { get; set; }

In case you want to get a particular error message , you can use the Errors collection.

Hide   Copy Code

var ErrMessage = ModelState["Email"].Errors[0].ErrorMessage;

If you have created the model object yourself you can explicitly call TryUpdateModel in your controller to check if the object is valid or not.

Hide   Copy Code

TryUpdateModel(NewCustomer);

In case you want add errors in the controller you can use the AddModelError function.

Hide   Copy Code

ModelState.AddModelError("FirstName", "This is my server-side error.");

## **How can we enable data annotation validation on client side?**

It’s a two-step process: first reference the necessary jQuery files.

Hide   Copy Code

<script src="<%= Url.Content("~/Scripts/jquery-1.5.1.js") %>" type="text/javascript"></script>

The second step is to call the EnableClientValidation method.

Hide   Copy Code

<% Html.EnableClientValidation(); %>

## **What is Razor in MVC?**

It’s a light weight view engine. Till MVC we had only one view type, i.e., ASPX. Razor was introduced in MVC 3.

## **Why Razor when we already have ASPX?**

Razor is clean, lightweight, and syntaxes are easy as compared to ASPX. For example, in ASPX to display simple time, we need to write:

Hide   Copy Code

<%=DateTime.Now%>

In Razor, it’s just one line of code:

Hide   Copy Code

@DateTime.Now

## **So which is a better fit, Razor or ASPX?**

As per Microsoft, Razor is more preferred because it’s light weight and has simple syntaxes.

## **How can you do authentication and authorization in MVC?**

You can use Windows or Forms authentication for MVC.

## **How to implement Windows authentication for MVC?**

For Windows authentication you need to modify the web.config file and set the authentication mode to Windows.

Hide   Copy Code

<authentication mode="Windows"/>

<authorization>

<deny users="?"/>

</authorization>

Then in the controller or on the action, you can use the Authorize attribute which specifies which users have access to these controllers and actions. Below is the code snippet for that. Now only the users specified in the controller and action can access it.

Hide   Copy Code

[Authorize(Users= @"WIN-3LI600MWLQN\Administrator")]

public class StartController : Controller

{

*//*

*// GET: /Start/*

[Authorize(Users = @"WIN-3LI600MWLQN\Administrator")]

public ActionResult Index()

{

return View("MyView");

}

}

## **How do you implement Forms authentication in MVC?**

Forms authentication is implemented the same way as in ASP.NET. The first step is to set the authentication mode equal to Forms. The loginUrl points to a controller here rather than a page.

Hide   Copy Code

<authentication mode="Forms">

<forms loginUrl="~/Home/Login" timeout="2880"/>

</authentication>

We also need to create a controller where we will check if the user is proper or not. If the user is proper we will set the cookie value.

Hide   Copy Code

public ActionResult Login()

{

if ((Request.Form["txtUserName"] == "Shiv") &&

(Request.Form["txtPassword"] == "Shiv@123"))

{

FormsAuthentication.SetAuthCookie("Shiv",true);

return View("About");

}

else

{

return View("Index");

}

}

All the other actions need to be attributed with the Authorize attribute so that any unauthorized user making a call to these controllers will be redirected to the controller (in this case the controller is “Login”) which will do the authentication.

Hide   Copy Code

[Authorize]

PublicActionResult Default()

{

return View();

}

[Authorize]

publicActionResult About()

{

return View();

}

## **How to implement AJAX in MVC?**

You can implement AJAX in two ways in MVC:

* AJAX libraries
* jQuery

Below is a simple sample of how to implement AJAX by using the “AJAX” helper library. In the below code you can see we have a simple form which is created by using the Ajax.BeginForm syntax. This form calls a controller action called getCustomer. So now the submit action click will be an asynchronous AJAX call.

Hide   Copy Code

<script language="javascript">

function OnSuccess(data1)

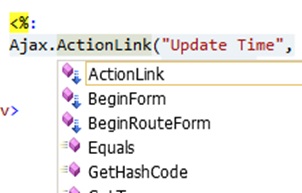
{

*// Do something here*

}

</script>

In case you want to make AJAX calls on hyperlink clicks, you can use the Ajax.ActionLink function as shown in the below code.



***Figure: Implement AJAX in MVC***

So if you want to create an AJAX asynchronous hyperlink by name GetDate which calls the GetDate function in the controller, below is the code for that. Once the controller responds, this data is displayed in the HTML DIVtag named DateDiv.

Hide   Copy Code

<span id="DateDiv" />

<%:

Ajax.ActionLink("Get Date","GetDate",

new AjaxOptions {UpdateTargetId = "DateDiv" })

%>

Below is the controller code. You can see how the GetDate function has a pause of 10 seconds.

Hide   Copy Code

public class Default1Controller : Controller

{

public string GetDate()

{

Thread.Sleep(10000);

return DateTime.Now.ToString();

}

}

The second way of making an AJAX call in MVC is by using jQuery. In the below code you can see we are making an AJAX POST call to a URL /MyAjax/getCustomer. This is done by using $.post. All this logic is put into a function called GetData and you can make a call to the GetData function on a button or a hyperlink click event as you want.

Hide   Copy Code

function GetData()

{

var url = "/MyAjax/getCustomer";

$.post(url, function (data)

{

$("#txtCustomerCode").val(data.CustomerCode);

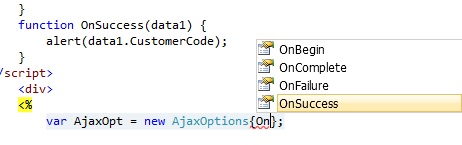
$("#txtCustomerName").val(data.CustomerName);

}

)

}

## **What kind of events can be tracked in AJAX?**



***Figure: Tracked in AJAX***

## **What is the difference between ActionResult and ViewResult?**

* ActionResult is an abstract class while ViewResult derives from the ActionResult class.ActionResult has several derived classes like ViewResult, JsonResult, FileStreamResult, and so on.
* ActionResult can be used to exploit polymorphism and dynamism. So if you are returning different types of views dynamically, ActionResult is the best thing. For example in the below code snippet, you can see we have a simple action called DynamicView. Depending on the flag (IsHtmlView) it will either return a ViewResult or JsonResult.

Hide   Copy Code

public ActionResult DynamicView()

{

if (IsHtmlView)

return View(); *// returns simple ViewResult*

else

return Json(); *// returns JsonResult view*

}

## **What are the different types of results in MVC?**

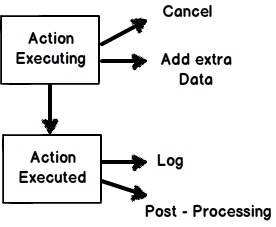
**Note**: It’s difficult to remember all the 12 types. But some important ones you can remember for the interview are ActionResult, ViewResult, and JsonResult. Below is a detailed list for your interest:

There 12 kinds of results in MVC, at the top is the ActionResult class which is a base class that can have 11 subtypes as listed below:

1. ViewResult - Renders a specified view to the response stream
2. PartialViewResult - Renders a specified partial view to the response stream
3. EmptyResult - An empty response is returned
4. RedirectResult - Performs an HTTP redirection to a specified URL
5. RedirectToRouteResult - Performs an HTTP redirection to a URL that is determined by the routing engine, based on given route data
6. JsonResult - Serializes a given ViewData object to JSON format
7. JavaScriptResult - Returns a piece of JavaScript code that can be executed on the client
8. ContentResult - Writes content to the response stream without requiring a view
9. FileContentResult - Returns a file to the client
10. FileStreamResult - Returns a file to the client, which is provided by a Stream
11. FilePathResult - Returns a file to the client

## **What are ActionFilters in MVC?**

ActionFilters help you to perform logic while an MVC action is executing or after an MVC action has executed.



***Figure: ActionFilters in MVC***

Action filters are useful in the following scenarios:

1. Implement post-processing logic before the action happens.
2. Cancel a current execution.
3. Inspect the returned value.
4. Provide extra data to the action.

You can create action filters by two ways:

* Inline action filter.
* Creating an ActionFilter attribute.

To create an inline action attribute we need to implement the IActionFilter interface. The IActionFilterinterface has two methods: OnActionExecuted and OnActionExecuting. We can implement pre-processing logic or cancellation logic in these methods.

Hide   Copy Code

public class Default1Controller : Controller , IActionFilter

{

public ActionResult Index(Customer obj)

{

return View(obj);

}

void IActionFilter.OnActionExecuted(ActionExecutedContext filterContext)

{

Trace.WriteLine("Action Executed");

}

void IActionFilter.OnActionExecuting(ActionExecutingContext filterContext)

{

Trace.WriteLine("Action is executing");

}

}

The problem with the inline action attribute is that it cannot be reused across controllers. So we can convert the inline action filter to an action filter attribute. To create an action filter attribute we need to inherit fromActionFilterAttribute and implement the IActionFilter interface as shown in the below code.

Hide   Copy Code

public class MyActionAttribute : ActionFilterAttribute , IActionFilter

{

void IActionFilter.OnActionExecuted(ActionExecutedContext filterContext)

{

Trace.WriteLine("Action Executed");

}

void IActionFilter.OnActionExecuting(ActionExecutingContext filterContext)

{

Trace.WriteLine("Action executing");

}

}

Later we can decorate the controllers on which we want the action attribute to execute. You can see in the below code I have decorated the Default1Controller with the MyActionAttribute class which was created in the previous code.

Hide   Copy Code

[MyActionAttribute]

public class Default1Controller : Controller

{

public ActionResult Index(Customer obj)

{

return View(obj);

}

}

## **What are the different types of action filters?**

1. Authorization filters
2. Action filters
3. Result filters
4. Exception filters

## **If we have multiple filters, what’s the sequence for execution?**

1. Authorization filters
2. Action filters
3. Response filters
4. Exception filters

## **Can we create our custom view engine using MVC?**

Yes, we can create our own custom view engine in MVC. To create our own custom view engine we need to follow three steps:

Let’ say we want to create a custom view engine where in the user can type a command like “<DateTime>” and it should display the current date and time.

**Step 1**: We need to create a class which implements the IView interface. In this class we should write the logic of how the view will be rendered in the render function. Below is a simple code snippet for that.

Hide   Copy Code

public class MyCustomView : IView

{

private string \_FolderPath; *// Define where our views are stored*

public string FolderPath

{

get { return \_FolderPath; }

set { \_FolderPath = value; }

}

public void Render(ViewContext viewContext, System.IO.TextWriter writer)

{

*// Parsing logic <dateTime>*

*// read the view file*

string strFileData = File.ReadAllText(\_FolderPath);

*// we need to and replace <datetime> datetime.now value*

string strFinal = strFileData.Replace("<DateTime>", DateTime.Now.ToString());

*// this replaced data has to sent for display*

writer.Write(strFinal);

}

}

**Step 2**: We need to create a class which inherits from VirtualPathProviderViewEngine and in this class we need to provide the folder path and the extension of the view name. For instance, for Razor the extension is “cshtml”; for aspx, the view extension is “.aspx”, so in the same way for our custom view, we need to provide an extension. Below is how the code looks like. You can see the ViewLocationFormats is set to the Views folder and the extension is “.myview”.

Hide   Copy Code

public class MyViewEngineProvider : VirtualPathProviderViewEngine

{

*// We will create the object of Mycustome view*

public MyViewEngineProvider() *// constructor*

{

*// Define the location of the View file*

this.ViewLocationFormats = new string[] { "~/Views/{1}/{0}.myview",

"~/Views/Shared/{0}.myview" }; *//location and extension of our views*

}

protected override IView CreateView(

ControllerContext controllerContext, string viewPath, string masterPath)

{

var physicalpath = controllerContext.HttpContext.Server.MapPath(viewPath);

MyCustomView obj = new MyCustomView(); *// Custom view engine class*

obj.FolderPath = physicalpath; *// set the path where the views will be stored*

return obj; *// returned this view paresing*

*// logic so that it can be registered in the view engine collection*

}

protected override IView CreatePartialView(ControllerContext controllerContext, string partialPath)

{

var physicalpath = controllerContext.HttpContext.Server.MapPath(partialPath);

MyCustomView obj = new MyCustomView(); *// Custom view engine class*

obj.FolderPath = physicalpath; *// set the path where the views will be stored*

return obj;

*// returned this view paresing logic*

*// so that it can be registered in the view engine collection*

}

}

**Step 3**: We need to register the view in the custom view collection. The best place to register the custom view engine in the ViewEngines collection is the global.asax file. Below is the code snippet for that.

Hide   Copy Code

protected void Application\_Start()

{

*// Step3 :- register this object in the view engine collection*

ViewEngines.Engines.Add(new MyViewEngineProvider());

&hellip;..

}

Below is a simple output of the custom view written using the commands defined at the top.

***Figure: Custom view engine using MVC***

If you invoke this view, you should see the following output:

## **How to send result back in JSON format in MVC**

In MVC, we have the JsonResult class by which we can return back data in JSON format. Below is a simple sample code which returns back a Customer object in JSON format using JsonResult.

Hide   Copy Code

public JsonResult getCustomer()

{

Customer obj = new Customer();

obj.CustomerCode = "1001";

obj.CustomerName = "Shiv";

return Json(obj,JsonRequestBehavior.AllowGet);

}

Below is the JSON output of the above code if you invoke the action via the browser.

## **What is WebAPI?**

HTTP is the most used protocol. For the past many years, browser was the most preferred client by which we consumed data exposed over HTTP. But as years passed by, client variety started spreading out. We had demand to consume data on HTTP from clients like mobile, JavaScript, Windows applications, etc.

For satisfying the broad range of clients REST was the proposed approach. You can read more about REST from the WCF chapter.

WebAPI is the technology by which you can expose data over HTTP following REST principles.

## **But WCF SOAP also does the same thing, so how does WebAPI differ?**

|  | **SOAP** | **WEB API** |
| --- | --- | --- |
| **Size** | Heavy weight because of complicated WSDL structure. | Light weight, only the necessary information is transferred. |
| **Protocol** | Independent of protocols. | Only for HTTP protocol |
| **Formats** | To parse SOAP message, the client needs to understand WSDL format. Writing custom code for parsing WSDL is a heavy duty task. If your client is smart enough to create proxy objects like how we have in .NET (add reference) then SOAP is easier to consume and call. | Output of WebAPI are simple string messages, JSON, simple XML format, etc. So writing parsing logic for that is very easy. |
| **Principles** | SOAP follows WS-\* specification. | WebAPI follows REST principles. (Please refer to REST in WCF chapter.) |

## **With WCF you can implement REST, so why WebAPI?**

WCF was brought into implement SOA, the intention was never to implement REST. WebAPI is built from scratch and the only goal is to create HTTP services using REST. Due to the one point focus for creating REST service, WebAPI is more preferred.

## How to implement WebAPI in MVC

Below are the steps to implement WebAPI:

**Step 1**: Create the project using the WebAPI template.

***Figure: Implement WebAPI in MVC***

**Step 2**: Once you have created the project you will notice that the controller now inherits from ApiControllerand you can now implement POST, GET, PUT, and DELETE methods of the HTTP protocol.

Hide   Copy Code

public class ValuesController : ApiController

{

*// GET api/values*

public IEnumerable<string> Get()

{

return new string[] { "value1", "value2" };

}

*// GET api/values/5*

public string Get(int id)

{

return "value";

}

*// POST api/values*

public void Post([FromBody]string value)

{

}

*// PUT api/values/5*

public void Put(int id, [FromBody]string value)

{

}

*// DELETE api/values/5*

public void Delete(int id)

{

}

}

**Step 3**: If you make an HTTP GET call you should get the below results:

***Figure: HTTP***

## **How can we detect that an MVC controller is called by POST or GET?**

To detect if the call on the controller is a POST action or a GET action we can use the Request.HttpMethodproperty as shown in the below code snippet.

Hide   Copy Code

public ActionResult SomeAction()

{

if (Request.HttpMethod == "POST")

{

return View("SomePage");

}

else

{

return View("SomeOtherPage");

}

}

## **What is bundling and minification in MVC?**

Bundling and minification helps us improve request load times of a page thus increasing performance.

## **How does bundling increase performance?**

Web projects always need CSS and script files. Bundling helps us combine multiple JavaScript and CSS files in to a single entity thus minimizing multiple requests in to a single request.

For example consider the below web request to a page . This page consumes two JavaScript files Javascript1.jsand Javascript2.js. So when this is page is requested it makes three request calls:

* One for the Index page.
* Two requests for the other two JavaScript files: Javascript1.js and Javascript2.js.

The below scenario can become worse if we have a lot of JavaScript files resulting in multiple requests, thus decreasing performance. If we can somehow combine all the JS files into a single bundle and request them as a single unit that would result in increased performance (see the next figure which has a single request).

## **So how do we implement bundling in MVC?**

Open BundleConfig.cs from the App\_Start folder.

In BundleConfig.cs, add the JS files you want bundle into a single entity in to the bundles collection. In the below code we are combining all the javascript JS files which exist in the Scripts folder as a single unit in to the bundle collection.

Hide   Copy Code

bundles.Add(new ScriptBundle("~/Scripts/MyScripts").Include(

"~/Scripts/\*.js"));

Below is how your BundleConfig.cs file will look like:

Hide   Copy Code

public class BundleConfig

{

public static void RegisterBundles(BundleCollection bundles)

{

bundles.Add(new ScriptBundle("~/Scripts/MyScripts").Include(

"~/Scripts/\*.js"));

BundleTable.EnableOptimizations = true;

}

}

Once you have combined your scripts into one single unit we then to include all the JS files into the view using the below code. The below code needs to be put in the ASPX or Razor view.

Hide   Copy Code

<%= Scripts.Render("~/Scripts/MyScripts") %>

If you now see your page requests you would see that script request is combined into one request.

## **How can you test bundling in debug mode?**

If you are in a debug mode you need to set EnableOptimizations to true in the bundleconfig.cs file or else you will not see the bundling effect in the page requests.

Hide   Copy Code

BundleTable.EnableOptimizations = true;

## **Explain minification and how to implement it**

Minification reduces the size of script and CSS files by removing blank spaces , comments etc. For example below is a simple javascript code with comments.

Hide   Copy Code

*// This is test*

var x = 0;

x = x + 1;

x = x \* 2;

After implementing minification the JavaScript code looks like below. You can see how whitespaces and comments are removed to minimize file size, thus increasing performance.

Hide   Copy Code

var x=0;x=x+1;x=x\*2;

## **How do we implement minification?**

When you implement bundling, minification is implemented by itself. In other words the steps to implement bundling and minification are the same.

## **Explain Areas in MVC?**

Areas help you to group functionalities in to independent modules thus making your project more organized. For example in the below MVC project we have four controller classes and as time passes by if more controller classes are added it will be difficult to manage. In bigger projects you will end up with 100’s of controller classes making life hell for maintenance.

If we can group controller classes in to logical section like “Invoicing” and “Accounting” that would make life easier and that’s what “Area” are meant to.

You can add an area by right clicking on the MVC solution and clicking on “Area” menu as shown in the below figure.

In the below image we have two “Areas” created “Account” and “Invoicing” and in that I have put the respective controllers. You can see how the project is looking more organized as compared to the previous state.

## **Explain the concept of View Model in MVC?**

A view model is a simple class which represents data to be displayed on the view.

For example below is a simple customermodel object with “CustomerName” and “Amount” property.

Hide   Copy Code

CustomerViewModel obj = new CustomerViewModel();

obj.Customer.CustomerName = "Shiv";

obj.Customer.Amount = 1000;

But when this “Customer” model object is displayed on the MVC view it looks something as shown in the below figure. It has “CustomerName” , “Amount” plus “**Customer Buying Level**” fields on the view / screen. “Customer buying Level” is a color indication which indicates how aggressive the customer is buying.

“Customer buying level” color depends on the value of the “Amount property. If the amount is greater than 2000 then color is red , if amount is greater than 1500 then color is orange or else the color is yellow.

In other words “Customer buying level” is an extra property which is calculated on the basis of amount.

So the Customer viewmodel class has three properties

* “TxtCustomerName” textbox takes data from “CustomerName” property as it is.
* “TxtAmount” textbox takes data from “Amount” property of model as it is.
* “CustomerBuyingLevelColor” displays color value depending on the “Amount “ value.

|  |  |
| --- | --- |
| **Customer Model** | **Customer ViewModel** |
| CustomerName | TxtCustomerName |
| Amount | TxtAmount |
|  | CustomerBuyingLevelColor |

## **What kind of logic view model class will have?**

As the name says view model this class has the gel code or connection code which connects the view and the model.

So the view model class can have following kind of logics:-

* **Color transformation logic: -**For example you have a “Grade” property in model and you would like your UI to display “red” color for high level grade, “yellow” color for low level grade and “green” color of ok grade.
* **Data format transformation logic :-**Your model has a property “Status” with “Married” and “Unmarried” value. In the UI you would like to display it as a checkbox which is checked if “married” and unchecked if “unmarried”.
* **Aggregation logic: -**You have two differentCustomer and Address model classes and you have view which displays both “Customer” and “Address” data on one go.
* **Structure downsizing: -**You have “Customer” model with “customerCode” and “CustomerName” and you want to display just “CustomerName”. So you can create a wrapper around model and expose the necessary properties.

## **How can we use two ( multiple) models with a single view?**

Let us first try to understand what the interviewer is asking. When we bind a model with a view we use the model dropdown as shown in the below figure. In the below figure we can only select one model.

But what if we want to bind “Customer” as well as “Order” class to the view.

For that we need to create a view model which aggregates both the classes as shown in the below code. And then bind that view model with the view.

Hide   Copy Code

public class CustOrderVM

{

public Customer cust = new Customer();

public Order Ord = new Order();

}

In the view we can refer both the model using the view model as shown in the below code.

Hide   Copy Code

<%= model.cust.Name %>

<%= model.Ord.Number %>

## **Explain the need of display mode in MVC?**

Display mode displays views depending on the device the user has logged in with. So we can create different views for different devices and display mode will handle the rest.

For example we can create a view “Home.aspx” which will render for the desktop computers and[Home.Mobile.aspx](http://www.codeproject.com/Articles/556995/Home.Mobile.aspx) for mobile devices. Now when an end user sends a request to the MVC application, display mode checks the “user agent” headers and renders the appropriate view to the device accordingly.

## **Explain MVC model binders?**

Model binder maps HTML form elements to the model. It acts like a bridge between HTML UI and MVC model. Many times HTML UI names are different than the model property names. So in the binder we can write the mapping logic between the UI and the model.

## **Explain the concept of MVC Scaffolding?**

Hide   Copy Code

Note :- Do not get scared with the word. Its actually a very simple thing.

Scaffolding is a technique in which the MVC template helps to auto-generate CRUD code. CRUD stands for create, read, update and delete.

So to generate code using scaffolding technique we need to select one of the types of templates (leave the empty one).

For instance if you choose “using Entity framework” template the following code is generated.

It creates controller code, view and also table structure as shown in the below figure.

## **What does scaffolding use internally to connect to database?**

It uses Entity framework internally.

## **How can we do exception handling in MVC?**

In the controller you can override the “OnException” event and set the “Result” to the view name which you want to invoke when error occurs. In the below code you can see we have set the “Result” to a view named as “Error”.

We have also set the exception so that it can be displayed inside the view.

Hide   Copy Code

public class HomeController : Controller

{

protected override void OnException(ExceptionContext filterContext)

{

Exception ex = filterContext.Exception;

filterContext.ExceptionHandled = true;

var model = new HandleErrorInfo(filterContext.Exception, "Controller","Action");

filterContext.Result = new ViewResult()

{

ViewName = "Error",

ViewData = new ViewDataDictionary(model)

};

}

}

To display the above error in view we can use the below code

Hide   Copy Code

@Model.Exception;

## **How can you handle multiple Submit buttons pointing to multiple actions in a single MVC view?**

Let us elaborate on what the interviewer wants to ask because the above question is just a single liner and is not clear about what the interviewer wants.

Take a scenario where you have a view with two submit buttons as shown in the below code.

Hide   Copy Code

<form action="Action1" method=post>

<input type=&rdquo;submit&rdquo; name=&rdquo;Submit1&rdquo;/>

<input type=&rdquo;submit&rdquo; name=&rdquo;Submit2&rdquo;>

</form>

In the above code when the end user clicks on any of the submit buttons it will make a HTTP POST to “Action1”.

The question from the interviewer is:-

“What if we have want that on “Submit1” button click it should invoke “Action1” and on the “Submit2” button click it should invoke “Action2”.”

Now that we have understood the question let us answer the question in a detailed manner. There are two approaches to solve the above problem one is the normal HTML way and the other is the “Ajax” way.

In the HTML way we need to create two forms and place the “Submit” button inside each of the forms. And every form’s action will point to different / respective actions. You can see the below code the first form is posting to “Action1” and the second form will post to “Action2” depending on which “Submit” button is clicked.

Hide   Copy Code

<form action="Action1" method=post>

<input type=&rdquo;submit&rdquo; name=&rdquo;Submit1&rdquo;/>

</form>

In case the interviewer complains that the above approach is not AJAX this is where the second approach comes in. In the Ajax way we can create two different functions “Fun1” and “Fun1” , see the below code. These function will make Ajax calls by using JQUERY or any other framework. Each of these functions are binded with the “Submit” button’s “OnClick” events.

Hide   Copy Code

<Script language="javascript">

function Fun1()

{

$.post(&ldquo;/Action1&rdquo;,null,CallBack1);

}

function Fun2()

{

$.post(&ldquo;/Action2&rdquo;,null,CallBack2);

}

</Script>

## **What is CSRF attack and how can we prevent the same in MVC?**

CSRF stands for Cross site request forgery. So if you see the dictonary meaning of forgery: -

*“It’s an act of copying or imitating things like signature on a cheque, official documents to deceive the authority source for financial gains.”*

So when it comes to website this forgery is termed as CSRF (Cross Site Request Forgery).

CSRF is a method of attacking a website where the attacker imitates a.k.a forges as a trusted source and sends data to the site. Genuine site processes the information innocently thinking that data is coming from a trusted source.

For example conside the below screen of a online bank. End user’s uses this screen to transfer money.

Below is a forged site created by an attacker which looks a game site from outside, but internally it hits the bank site for money transfer.

The internal HTML of the forged site has those hidden fields which have the account number and amount to do money transfer.

Hide   Copy Code

<div>

Win 1000000 US$

<form action="http://localhost:23936/Genuine/Transfer" method=post>

<input type=hidden name="amount" value="10000" />

<input type=hidden name="account" value="3002" />

<input type=submit value="Play the ultimate game" />

</form>

</div>

Now let’s say the user has logged in to the genuine bank site and the attacker sent this forged game link to his email. The end user thinking that it’s a game site clicks on the “Play the Ultimate Game” button and internally the malicious code does the money transfer process.

So a proper solution to this issue can be solved by using tokens: -

* + End user browses to the screen of the money transfer. Before the screen is served server injects a secret token inside the HTML screen in form a hidden field.
  + Now hence forth when the end user sends request back he has to always send the secret token. This token is validated on the server.

Implementing token is a two-step process in MVC: -

First apply “ValidateAntiForgeryToken” attribute on the action.

Hide   Copy Code

[ValidateAntiForgeryToken]

public ActionResult Transfer()

{

// password sending logic will be here

return Content(Request.Form["amount"] +

" has been transferred to account "

+ Request.Form["account"]);

}

Second in the HTML UI screen call “@Html.AntiForgeryToken()” to generate the token.

Hide   Copy Code

<div>

Transfer money

<form action="Transfer" method=post>

Enter Amount

<input type="text" name="amount" value="" />

Enter Account number

@Html.AntiForgeryToken()

<input type=submit value="transfer money" />

</form>

</div>

So now henceforth when any untrusted source send a request to the server it would give the below forgery error.

If you do a view source of the HTML you would find the below verification token hidden field with the secret key.

Hide   Copy Code

<input name="\_\_RequestVerificationToken" type="hidden" value="7iUdhsDNpEwiZFTYrH5kp/q7jL0sZz+CSBh8mb2ebwvxMJ3eYmUZXp+uofko6eiPD0fmC7Q0o4SXeGgRpxFp0i+Hx3fgVlVybgCYpyhFw5IRyYhNqi9KyH0se0hBPRu/9kYwEXXnVGB9ggdXCVPcIud/gUzjWVCvU1QxGA9dKPA=" />

Please do read this blog which has detailed steps of how model binders can be created using “IModelBinder” interface: - [Explain MVC model Binders?](http://computerauthor.blogspot.in/2014/06/explain-mvc-model-binders-aspnet-mvc_1385.html)

Download an e-learning copy of MVC interview Q&A from the top of this article for your preparation.

For technical training related to various topics including ASP.NET, Design Patterns, WCF, MVC, BI, WPF contact SukeshMarla@gmail.com or visit [www.sukesh-marla.com](http://www.sukesh-marla.com/)

# **ENTITY FRAMEWORK**

# **LINQ**

# **HTML, CSS, JAVASCRIPT**

# **ALL ABOUT SQL**

## Procedures

**Stored Procedure**: Stored Procedure in **SQL** Server can be defined as the set of logical group of **SQL** statements which are grouped to perform a specific task. There are many benefits of using a stored procedure. The main benefit of using a stored procedure is that it increases the performance of the database.

**What is a database transaction?**

Database transaction takes database from one consistent state to another. At the end of the transaction the system must be in the prior state if the transaction fails or the status of the system should reflect the successful completion if the transaction goes through.

**What is a Database Lock ?**

Database lock tells a transaction, if the data item in questions is currently being used by other transactions.

**What are the type of locks ?**

**1. Shared Lock**

When a shared lock is applied on data item, other transactions can only read the item, but can't write into it.

**2. Exclusive Lock**

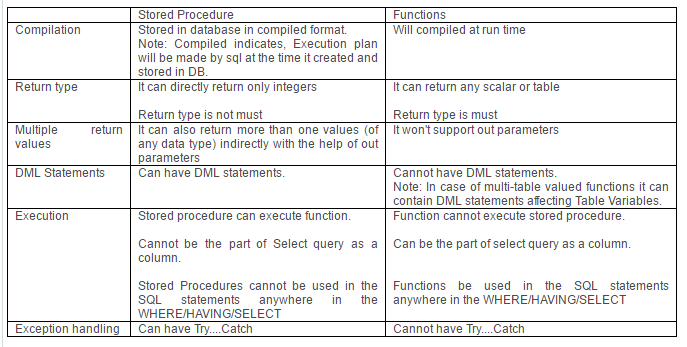
When an exclusive lock is applied on data item, other transactions can't read or write into the data item

## Functions

### **Limitations and Restrictions**

* User-defined functions cannot be used to perform actions that modify the database state.
* User-defined functions cannot contain an OUTPUT INTO clause that has a table as its target.
* User-defined functions can not return multiple result sets. Use a stored procedure if you need to return multiple result sets.
* Error handling is restricted in a user-defined function. A UDF does not support TRY…CATCH, @ERROR or RAISERROR.
* User-defined functions cannot call a stored procedure, but can call an extended stored procedure.
* User-defined functions cannot make use of dynamic SQL or temp tables. Table variables are allowed.
* SET statements are not allowed in a user-defined function.

## Difference Between Procedure and Functions



**Difference 10:**

**Function cannot have the transaction statements.**

The transaction statement cannot be used in the function. Normally we won't do any DML operations in the function.

**Stored procedure can use transaction statements.**

The transaction statement can be used inside the stored procedures.

## Cusrors

A cursor is a temporary work area created in the system memory when a SQL statement is executed. A cursor contains information on a select statement and the rows of data accessed by it.It is a database object that is used to retrieve data from a result set one row at a time. Cursor is used when we need to update records in a database table in a row by row manner. Cursors can hold more than one row at a time, but it can process only one row at a time.The current set that is being held is called as ‘Active Set’.

There are many advantages of using cursors in SQL following are some of the few

* Using Cursor we can perform row by row processing so we can perform row wise validation or operations on each row.
* Cursors can provide the first few rows before the whole result set is assembled. Without using cursors, the entire result set must be delivered before any rows are displayed by the application. So using cursor, better response time is achieved.
* If we make updates, without using cursors in our application then we must send separate SQL statements to the database server to apply the changes. This can cause the possibility of concurrency problems if the result set has changed since it was queried by the client. In turn, this raises the possibility of lost updates. So using cursor, better concurrency Control can be achieved.
* Cursors can be faster than a while loop but at the cost of more overhead.



## Triggers

SQL Triggers are a set of statements similar to stored procedures which will be executed in response to certain operations that happen in DB. These statements can’t be called explicitly. The only means by which these statements will be executed is the action for which they are written. It is a DB object bound to the table. In short these statements will cell called when a DDL or DML statements are executed. DDL statements such as SQL-CREATE, ALTER, DROP and DML Statements such as INSERT, UPDATE, and DELETE are all the source for execution for the triggers. In addition, once a trigger is in place, application programmers and end users cannot circumvent it. When a trigger is activated, the control shifts from the application program to the database manager. The operating system executes your coded trigger program to perform the actions you designed. The application waits until the trigger ends and then gains control again.



## DDL Statements

## DML Statements

## DCL Statements

## Joins

In order to avoid data duplication, data is stored in related tables. **Join** keyword is used to fetch data from related tables. "Join" return rows when there is at least one match in both table. Type of joins are

**Right Join**

Return all rows from the right table, even if there are no matches in the left table.  
**Outer Join**

The FULL OUTER JOIN keyword returns all rows from the left table (table1) and from the right table (tabl,e2).

The FULL OUTER JOIN keyword combines the result of both LEFT and RIGHT joins.  
**Left Join**

Return all rows from the left table, even if there are no matches in the right table.  
**Full Join**

Return rows when there is a match in one of the tables.

**Self Join**

Self-join is query used to **join a table to itself**. Aliases should be used for the same table comparison.

## Constraints

**What is a primary key?**

A primary key is a column whose values **uniquely identify every row** in a table. Primary key values can never be reused. If a row is deleted from the table, its primary key may not be assigned to any new rows in the future. To define a field as primary key, following conditions had to be met :

1. No two rows can have the same primary key value.

2. Every row must have a primary key value.

3. The primary key field cannot be null.

4. Value in a primary key column can never be modified or updated, if any foreign key refers to that primary key.

**What is a Composite Key ?**

A Composite primary key is a type of candidate key, which represents a set of columns whose values uniquely identify every row in a table.  
**For example -** if "Employee\_ID" and "Employee Name" in a table is combined to uniquely identify a row its called a Composite Key.

**What is a Foreign Key ?**

When a "one" table's primary key field is added to a related "many" table in order to create the common field which relates the two tables, it is called a foreign key in the "many" table.  
  
For example, the salary of an employee is stored in salary table. The relation is established via foreign key column “Employee\_ID\_Ref” which refers “Employee\_ID” field in the Employee table.

**18. What is a Unique Key ?**

Unique key is same as primary with the difference being the existence of null. Unique key field allows one value as NULL value.

## Where Clause

The WHERE clause is used to extract only those records that fulfill a specified criterion.

**SQL WHERE Syntax**

SELECT column\_name,column\_name  
FROM table\_name  
WHERE column\_name operator value;

## Group By

The GROUP BY statement is used in conjunction with the aggregate functions to group the result-set by one or more columns.

**SQL GROUP BY Syntax**

SELECT column\_name, aggregate\_function(column\_name)  
FROM table\_name  
WHERE column\_name operator value  
GROUP BY column\_name;

**SELECT Shippers.ShipperName,COUNT(Orders.OrderID) AS NumberOfOrders FROM Orders  
LEFT JOIN Shippers  
ON Orders.ShipperID=Shippers.ShipperID  
GROUP BY ShipperName;**

## Order By

The ORDER BY key word is used to sort the result-set by one or more columns.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in a descending order, you can use the DESC keyword.

**SYNTAX**

SELECT column\_name,column\_name  
FROM table\_name  
ORDER BY column\_nameASC|DESC,column\_name ASC|DESC;

## Having Clause

The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

**SQL HAVING Syntax**

SELECT column\_name, aggregate\_function(column\_name)  
FROM table\_name  
WHERE column\_name operator value  
GROUP BY column\_name  
HAVING aggregate\_function(column\_name) operator value;

**SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders FROM (Orders  
INNER JOIN Employees  
ON Orders.EmployeeID=Employees.EmployeeID)  
GROUP BY LastName  
HAVING COUNT(Orders.OrderID) > 10;**

## Views

A view is nothing more than a SQL statement that is stored in the database with an associated name. A view is actually a composition of a table in the form of a predefined SQL query.

Views, which are kind of virtual tables, allow users to do the following:

* Structure data in a way that users or classes of users find natural or intuitive.
* Restrict access to the data such that a user can see and (sometimes) modify exactly what they need and no more.
* Summarize data from various tables which can be used to generate reports.

**What are the advantages and disadvantages of views in a database?**

**Advantages**:

1. Views don't store data in a physical location.

2. The view can be used to hide some of the columns from the table.

3. Views can provide Access Restriction, since data insertion, update and deletion is not possible with the view.

**Disadvantages**:

1. When a table is dropped, associated view become irrelevant.

2. Since the view is created when a query requesting data from view is triggered, its a bit slow.

3. When views are created for large tables, it occupies more memory.

## Indexes

A **clustered index** reorders the way records in the table are physically stored. There can be only one clustered index per table. It makes data retrieval faster.  
A **non clustered index** does not alter the way it was stored but creates a completely separate object within the table. As a result insert and update command will be faster.

The CREATE INDEX statement is used to create indexes in tables.

Indexes allow the database application to find data fast; without reading the whole table.

**Indexes**

An index can be created in a table to find data more quickly and efficiently.

The users cannot see the indexes, they are just used to speed up searches/queries.

**Note:** Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So you should only create indexes on columns (and tables) that will be frequently searched against.

**SQL CREATE INDEX Syntax**

Creates an index on a table. Duplicate values are allowed:

CREATE INDEX index\_name  
ON table\_name (column\_name)

**SQL CREATE UNIQUE INDEX Syntax**

Creates a unique index on a table. Duplicate values are not allowed:

CREATE UNIQUE INDEX index\_name  
ON table\_name (column\_name)

**Note:** The syntax for creating indexes varies amongst different databases. Therefore: Check the syntax for creating indexes in your database.

**CREATE INDEX Example**

The SQL statement below creates an index named "PIndex" on the "LastName" column in the "Persons" table:

CREATE INDEX PIndex  
ON Persons (LastName)

If you want to create an index on a combination of columns, you can list the column names within the parentheses, separated by commas:

CREATE INDEX PIndex  
ON Persons (LastName, FirstName)

### **Clustered Index (CI)**

A clustered index is something that reorganizes the way records in the table are physically stored. Therefore a table can have only one clustered index. The leaf nodes of a clustered index contain the data pages, by which I mean the key-value pair in the clustered index has the index key and the actual data value. Also remember, a clustered index will be created on a table by default the moment a primary key is created on the table. A clustered index is something like your train ticket B4/24, you know that you need to board coach B4 and sit on seat number 24. So this index physically leads you to your actual seat.

### **Non-Clustered Index (NCI)**

A non-clustered index is a special type of index in which the logical order of the index does not match the physical stored order of the rows on disk. The leaf node of a non-clustered index does not consist of the data pages but a pointer to it. That goes to say that a non-clustered index can’t survive on its own - it needs a base to live on. A non-clustered index uses a clustered index (if defined) or the heap to build itself.

When a non-clustered index uses the heap, the leaf node (or the pointer) is a physical location of the data. When it uses a clustered index, the leaf node (or the pointer) is the clustered index key value and this key value in tu rn points to the actual data.

## Transcation In SQl

A transaction is a unit of work that is performed against a database. Transactions are units or sequences of work accomplished in a logical order, whether in a manual fashion by a user or automatically by some sort of a database program.

A transaction is the propagation of one or more changes to the database. For example, if you are creating a record or updating a record or deleting a record from the table, then you are performing transaction on the table. It is important to control transactions to ensure data integrity and to handle database errors.

Practically, you will club many SQL queries into a group and you will execute all of them together as a part of a transaction.

### **Properties of Transactions:**

Transactions have the following four standard properties, usually referred to by the acronym ACID:

* **Atomicity:** ensures that all operations within the work unit are completed successfully; otherwise, the transaction is aborted at the point of failure, and previous operations are rolled back to their former state.
* **Consistency:** ensures that the database properly changes states upon a successfully committed transaction.
* **Isolation:** enables transactions to operate independently of and transparent to each other.
* **Durability:** ensures that the result or effect of a committed transaction persists in case of a system failure.

### **Transaction Control:**

There are following commands used to control transactions:

* **COMMIT:** to save the changes.
* **ROLLBACK:** to rollback the changes.
* **SAVEPOINT:** creates points within groups of transactions in which to ROLLBACK
* **SET TRANSACTION:** Places a name on a transaction.

DECLARE @intErrorCode INT

BEGIN TRAN

UPDATE Authors

SET Phone = '415 354-9866'

WHERE au\_id = '724-80-9391'

SELECT @intErrorCode = @@ERROR

IF (@intErrorCode <> 0) GOTO PROBLEM

UPDATE Publishers

SET city = 'Calcutta', country = 'India'

WHERE pub\_id = '9999'

SELECT @intErrorCode = @@ERROR

IF (@intErrorCode <> 0) GOTO PROBLEM

COMMIT TRAN

PROBLEM:

IF (@intErrorCode <> 0) BEGIN

PRINT 'Unexpected error occurred!'

ROLLBACK TRAN

END

Before the real processing starts, the BEGIN TRAN statement notifies **SQL** Server to treat all of the following actions as a single **transaction**. It is followed by two UPDATE statements. If no errors occur during the updates, all changes are committed to the database when **SQL** Server processes the COMMIT TRAN statement, and finally the stored procedure finishes. If an error occurs during the updates, it is detected by if statements and execution is continued from the PROBLEM label. After displaying a message to the user, **SQL** Server rolls back any changes that occurred during processing. Note: Be sure to match BEGIN TRAN with either COMMIT or ROLLBACK.

## Try Catch

BEGIN **TRY**

Print ' I am level 1 '

BEGIN **TRY**

Print ' I am level 2 '

SELECT 1/0;

END **TRY**

BEGIN **CATCH**

SELECT

ERROR\_NUMBER() AS ErrorNumber,

ERROR\_SEVERITY() AS ErrorSeverity,

ERROR\_STATE() as ErrorState,

ERROR\_PROCEDURE() as ErrorProcedure,

ERROR\_LINE() as ErrorLine,

ERROR\_MESSAGE() as ErrorMessage;

Print ' I am going out level 2 '

END **CATCH**

Print ' I am level 1 again '

END **TRY**

BEGIN **CATCH**

*-- Execute the error retrieval routine.*

SELECT

ERROR\_NUMBER() AS ErrorNumber,

ERROR\_SEVERITY() AS ErrorSeverity,

ERROR\_STATE() as ErrorState,

ERROR\_PROCEDURE() as ErrorProcedure,

ERROR\_LINE() as ErrorLine,

ERROR\_MESSAGE() as ErrorMessage;

END **CATCH**

# **PROJECT MANAGEMENT**