Contents

What is a class ?

A class is the generic definition of what an object is. A Class describes all the attributes of the object, as well as the methods that implement the behavior of the member object. In other words, class is a template of an object. The Setter and Getter methods are used to store and fetch data from the variable.

What is an Object?

An object is an instance of a class. It contains real values instead of variables.

Abstraction:

Abstraction is a process of hiding the implementation details and displaying the essential features.
How to abstract: - By using **Access Specifiers**

What are the Access Modifiers in C# ?

Different Access Modifier are - Public, Private, Protected, Internal, Protected Internal

Public – When a method or attribute is defined as Public, it can be accessed from any code in the project. For example, in the above Class "Employee" getName() and setName() are public.

Private - When a method or attribute is defined as Private, It can be accessed by any code within the containing class only. For example, in the above Class "Employee" attributes name and salary can be accessed within the Class Employee Only. If an attribute or class is defined without access modifiers, it's default access modifier will be private.

Protected - When attribute and methods are defined as protected, it can be accessed by any method in the inherited classes and any method within the same class. The protected access modifier cannot be applied to classes and interfaces. Methods and fields in a interface can't be declared protected.

Internal – If an attribute or method is defined as Internal, access is restricted to classes within the current project assembly.

Protected Internal – If an attribute or method is defined as Protected Internal, access is restricted to classes within the current project assembly and types derived from the containing class.

Explain Static Members in C# ?

If an attribute's value had to be same across all the instances of the same class, the static keyword is used. For example, if the Minimum salary should be set for all employees in the employee class, use the following code.

private static double MinSalary = 30000;

To access a private or public attribute or method in a class, at first an object of the class should be created. Then by using the object instance of that class, attributes or methods can be accessed. To

access a static variable, we don't want to create an instance of the class containing the static variable. We can directly refer that static variable as shown below.

```
double var = Employee.MinSalary ;
```

What is Reference Type in C# ?

Let us explain this with the help of an example. In the code given below,

```
Employee emp1;
Employee emp2 = new Employee();
emp1 = emp2;
```

Here emp2 has an object instance of Employee Class. But emp1 object is set as emp2. What this means is that the object emp2 is referred in emp1, rather than copying emp2 instance into emp1. When a change is made in emp2 object, corresponding changes can be seen in emp1 object.

Define Property in C#

Properties are a type of class member, that are exposed to the outside world as a pair of Methods. For example, for the static field Minsalary, we will Create a property as shown below.

```
private double minimumSalary;
public static double MinSalary
{
 get
 {
  return minimumSalary;
 }
 set
 {
  minimumSalary = value;
 }
}
```

So when we execute the following lines code
```
double minSal = Employee.MinSalary;
```
get Method will get triggered and value in minimumSalary field will be returned. When we execute,
```
Employee. MinSalary = 3000;
```

set Method will get triggered and value will be stored in minimumSalary field.

Explain Overloading in C#

When methods are created with the same name, but with different signature its called overloading. For example, WriteLine method in console class is an example of overloading. In the first instance, it takes one variable. In the second instance, “WriteLine” method takes two variable. **Instead of creating multiply different method of doing the same things, we can create one function by using different numbers or types of parameters.**

```
Console.WriteLine(x);
Console.WriteLine("The message is {0}", Message);
```

Different types of overloading in C# are

Constructor overloading

In Constructor overloading, n number of constructors can be created for the same class. But the signatures of each constructor should vary. For example

```
public class Employee
{
 public Employee()
 { }
 public Employee(String Name)
 { }
}
```

Function overloading

In Function overloading, n number of functions can be created for the same class. But the signatures of each function should vary. For example

```
public class Employee
{
 public void Employee()
 { }
 public void Employee(String Name)
 { }
}
```

Operator overloading

We had seen function overloading in the previous example. For operator Overloading, we will have a look at the example given below. We had defined a class rectangle with two operator overloading methods.

```
class Rectangle
{
 private int Height;
 private int Width;

 public Rectangle(int w,int h)
 {
   Width=w;
   Height=h;
 }
 public static bool operator >(Rectangle a,Rectangle b)
 {
```

```
  return a.Height > b.Height ;
 }
 public static bool operator <(Rectangle a,Rectangle b)
 {
  return a.Height < b.Height ;
 }
}
```

Let us call the operator overloaded functions from the method given below. When first if condition is triggered, the first overloaded function in the rectangle class will be triggered. When second if condition is triggered, the second overloaded function in the rectangle class will be triggered.

```
public static void Main()
{
Rectangle obj1 =new Rectangle();
Rectangle obj2 =new Rectangle();

 if(obj1 > obj2)
 {
  Console.WriteLine("Rectangle1 is greater than Rectangle2");
 }

 if(obj1 < obj2)
 {
  Console.WriteLine("Rectangle1 is less than Rectangle2");
 }
}
```

What is Polymorphism

Polymorphism means many forms (**ability to take more than one form**). In Polymorphism poly means "multiple" and morph means "forms" so polymorphism means many forms.

1) It allows you to invoke methods of derived class through base class reference during runtime.

2) It has the ability for classes to provide different implementations of methods that are called through the same name.

In polymorphism we will declare methods with same name and different parameters in same class or methods with same name and same parameters in different classes. Polymorphism has ability to provide different implementation of methods that are implemented with same name.

In Polymorphism we have 2 different types those are

- **Overloading** (Called as Early Binding or Compile Time Polymorphism or static binding)

- **Overriding** (Called as Late Binding or Run Time Polymorphism or dynamic binding)

Overloading

Overloading means we will declare methods with same name but different signatures because of this we will perform different tasks with same method name. This overloading also called as **compile time polymorphism** or **early binding**.

Method Overloading or compile time polymorphism means same method names with different signatures (different parameters)

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.

Overriding

Overriding also called as **run time polymorphism** or **late binding** or **dynamic polymorphism**. Method overriding or run time polymorphism means same method names with same signatures.

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 using virtual/override keywords.

Base class method has to be marked with virtual keyword and we can override it in derived class using override 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.

Many times you will have dependencies in sub-classes, or may have to implement things differently.

```
class BasicCarModel
{
    public virtual void Accelerate(double speed)
    {
         RotateFrontWheels(speed);
    }
}

class LuxuryCarModel : BasicCarModel
{
    public override void Accelerate(double speed)
    {
        RotateAllWheels(speed);
    }
}
```

What is Data Encapsulation

Encapsulation is a process of binding the data members and member functions into a single unit.

Data Encapsulation is defined as the process of hiding the important fields from the end user. In the above example, we had used getters and setters to set value for MinSalary. The idea behind this is that, private field "minimumSalary" is an important part of our classes. So if we give a third party code to have complete control over the field without any validation, it can adversely affect the functionality. This is inline with the OOPS Concept that an external user should know about the what an object does. How it does it, should be decided by the program. So if a user set a negative value for MinSalary, we can put a validation in the set method to avoid negative values as shown below

```
set

{

 if(value > 0)

 {

        minSalary = value;

 }

}
```

Explain Inheritance in C#

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

In object-oriented programming (OOP), inheritance is a way to reuse code of existing objects. In inheritance, there will be two classes - base class and derived classes. A class can inherit attributes and methods from existing class called base class or parent class. The class which inherits from a base class is called derived classes or child class. For more clarity on this topic, let us have a look at 2 classes shown below. Here Class Car is Base Class and Class Ford is derived class.

```
class Car
{
 public Car()
 {
  Console.WriteLine("Base Class Car");
 }

 public void DriveType()
 {
  Console.WriteLine("Right Hand Drive");
 }
}
```

```
class Ford : Car
{
 public Ford()
 {
  Console.WriteLine("Derived Class Ford");
 }

 public void Price()
 {
  Console.WriteLine("Ford Price : 100K $");
 }
}
```

When we execute following lines of code ,

```
Ford CarFord = new Ford();
CarFord.DriveType();
CarFord.Price();
```

Output Generated is as given below.

```
Base Class Car
Derived Class Ford
Right Hand Drive
Ford Price : 100K $
```

What this means is that, all the methods and attributes of Base Class car are available in Derived Class Ford. When an object of class Ford is created, constructors of the Base and Derived class get invoked. Even though there is no method called DriveType() in Class Ford, we are able to invoke the method because of inheriting Base Class methods to derived class.

Can Multiple Inheritance implemented in C# ?

In C#, derived classes can inherit from one base class only. If you want to inherit from multiple base classes, use interface. (if A and B have the same method and C extends both, which method does it take?

Explain the use of Virtual Keyword in C#

When we want to give permission to a derived class to override a method in base class, Virtual keyword is used. For example. lets us look at the classes Car and Ford as shown below.

```
class Car
{
 public Car()
 {
  Console.WriteLine("Base Class Car");
 }
 public virtual void DriveType()
 {
  Console.WriteLine("Right Hand Drive");
 }
```

```
}

class Ford : Car
{
 public Ford()
 {
  Console.WriteLine("Derived Class Ford");
 }
 public void Price()
 {
  Console.WriteLine("Ford Price : 100K $");
 }
 public override void DriveType()
 {
  Console.WriteLine("Right Hand ");
 }
}
```

When following lines of code get executed

```
Car CarFord = new Car();
CarFord.DriveType();
CarFord = new Ford();
CarFord.DriveType();
```

Output is as given below.

```
Base Class Car
Right Hand Drive
Base Class Car
Derived Class Ford
Right Hand
```

Difference Between Overloading and Overriding

Overloading

Overloading is when you have multiple methods in the same scope, with the same name but different signatures.

```
//Overloading
public class test
{
   public void getStuff(int id)
   {}
   public void getStuff(string name)
   {}
}
```

Overriding

Overriding is a principle that allows you to change the functionality of a method in a child class.

```
//Overriding
public class test
{
      public virtual getStuff(int id)
      {
         //Get stuff default location
      }
}

public class test2 : test
{
      public override getStuff(int id)
      {
         //base.getStuff(id);
         //or - Get stuff new location
      }
}
```

What is Method Hiding in C# ?

If the derived class doesn't want to use methods in the base class, derived class can implement the same method in a derived class with same signature. For example, in the classes given below, DriveType() is implemented in the derived class with same signature. This is called Method Hiding.

```
class Car
{
 public void DriveType()
 {
  Console.WriteLine("Right Hand Drive");
 }
}

class Ford : Car
{
 public void DriveType()
 {
  Console.WriteLine("Right Hand ");
 }
}
```

What is Abstract Class in C#?

If we don't want a class to be instantiated, define the class as abstract. An abstract class can have abstract and non abstract classes. If a method is defined as abstract, it must be implemented in derived class.

Abstract class is a base class or a parent class. Abstract classes can have empty abstract methods or it can have implemented methods which can be overridden by child classes.

For example, in the classes given below, method DriveType is defined as abstract.

```
abstract class Car
{
 public Car()
 {
  Console.WriteLine("Base Class Car");
 }
 public abstract void DriveType();
}

class Ford : Car
{
 public void DriveType()
 {
  Console.WriteLine("Right Hand ");
 }
}
```

Method DriveType get implemented in derived class.

What is Sealed Classes in c#

If a class is defined as Sealed, it cannot be inherited in derived class. Example of a sealed class is given below.

```
public sealed class Car
{
 public Car()
 {
  Console.WriteLine("Base Class Car");
 }

 public void DriveType()
 {
  Console.WriteLine("Right Hand ");
 }
}
```

What is abstract class

If we don't want a class to be instantiated, define the class as abstract. An abstract class can have abstract and non abstract classes. If a method is defined as abstract, it must be implemented in derived class.

Abstract class is a base class or a parent class. Abstract classes can have empty abstract methods or it can have implemented methods which can be overridden by child classes.

What are interfaces

Interface is a contract class with empty methods , properties and functions. Any class which implements the interface has to compulsory implement all the empty methods , functions and properties of the interface.

What's the difference between abstract class and interface

There are many differences, below are some key two differences :-

- Abstract class are base class or parent class while interfaces are contracts.
- Abstract class can have some implemented methods and functions while interfaces methods and functions are completely empty.
- Abstract classes are inherited while interfaces are implemented.
- Abstract classes are used when we want to increase reusability in inheritance while interfaces are used to force a contract.

Can we create an object of abstract class or interface

No we can not.

In what scenarios will you use an abstract class and in what scenarios will you use a interface

If you want to increase reusability in inheritance then abstract classes are good. If you want implement or force some methods across classes must be for uniformity you can use a interface. So to increase reusability via inheritance use abstract class as it is nothing but a base class and to force methods use interfaces.

What is a Constructor in C# ?

Constructor is a special method that get invoked/called automatically, whenever an object of a given class gets instantiated. In our class car, constructor is defined as shown below

```
public Car()
{
 Console.WriteLine("Base Class Car");
}
```

When ever an instance of class car is created from the same class or its derived class(Except Few Scenarios), Constructor get called and sequence of code written in the constructor get executed.

```
interface Breaks
{
 void BreakType();
}

interface Wheels
```

```
{
 void WheelType();
}

class Ford : Breaks, Wheels
{
 public Ford()
 {
  Console.WriteLine("Derived Class Ford");
 }
 public void Price()
 {
  Console.WriteLine("Ford Price : 100K $");
 }
 public void BreakType()
 {
  Console.WriteLine("Power Break");
 }
 public void WheelType()
 {
  Console.WriteLine("Bridgestone");
 }
}
```

What is a Destructor in C# ?

Destructor is a special method that get invoked/called automatically whenever an object of a given class gets destroyed. Main idea behind using destructor is to free the memory used by the object.

Array and ArrayList

Arrays

Arrays are strongly typed collection of same datatype and these arrays are fixed length that cannot be changed during runtime. Generally in arrays we will store values with index basis that will start with zero. If we want to access values from arrays we need to pass index values.

Arraylists

Array lists are not strongly type collection. It will store values of different datatypes or same datatype. Array list size will increase or decrease dynamically it can take any size of values from any data type. These Array lists will be accessible with "**System.Collections**" namespace

Difference between Array and ArrayList

Arrays	ArrayLists
These are strong type collection and allow to store fixed length	Array Lists are not strong type collection and size will increase or decrease dynamically

In arrays we can store only one datatype either int, string, char etc…	In arraylist we can store all the datatype values
Arrays belong to System.Array namespace	Arraylist belongs to System.Collection namespaces

Most of the time, we tend to choose array lists rather than arrays since we have no idea how big it is going to turn out. Arrays are ideal when you know how many items you are going to put in it.

Array are sequence of homogeneous data while ArrayList is sequence of heterogenous data. That's why we have to typecast every data in ArrayLists.

Arrays are multidimensional but ArrayList is always single-dimensional.

Arrays are strongly typed, and work well as parameters. If you know the length of your collection and it is fixed, you should use an array.ArrayLists are not strongly typed, every Insertion or Retrial will need a cast to get back to your original type.

ArrayList vs List

You'll be able to use the LINQ extension methods directly with `List<object>`, but not with `ArrayList`. `List<T>` is a generic class. It supports storing values of a specific type without casting to or from `object` (which would have incurred boxing/unboxing overhead when `T` is a value type in the `ArrayList` case). `ArrayList` simply stores `object` references. As a generic collection, it implements the generic `IEnumerable<T>` interface and can be used easily in LINQ (without requiring any `Cast` or `OfType` call).

`ArrayList` belongs to the days that C# didn't have generics. It's deprecated in favor of `List<T>`. You shouldn't use `ArrayList` in new code that targets .NET >= 2.0 unless you have to interface with an old API that uses it.

Array vs LinkedList

(1) The size of the arrays is fixed: So we must know the upper limit on the number of elements in advance. Also, generally, the allocated memory is equal to the upper limit irrespective of the usage, and in practical uses, upper limit is rarely reached.

(2) Inserting a new element in an array of elements is expensive, because room has to be created for the new elements and to create room existing elements have to shifted.

For example, suppose we maintain a sorted list of IDs in an array id[].

id[] = [1000, 1010, 1050, 2000, 2040, …..].

And if we want to insert a new ID 1005, then to maintain the sorted order, we have to move all the elements after 1000 (excluding 1000).

Deletion is also expensive with arrays until unless some special techniques are used. For example, to delete 1010 in id[], everything after 1010 has to be moved.

So Linked list provides following two advantages over arrays
1) Dynamic size
2) Ease of insertion/deletion

Linked lists have following drawbacks:
1) Random access is not allowed. We have to access elements sequentially starting from the first node. So we cannot do binary search with linked lists.
2) Extra memory space for a pointer is required with each element of the list.
3) Arrays have better cache locality that can make a pretty big difference in performance.

Java and .Net Compare

1).Net has Multilanguage support. While java has based on java language only.

2).Net requires the support of .Net Framework which is mostly based on Windows platform while Java requires JVM which could be commonly used in difference platform(Linux).

3)From the developer view, .Net is easier to configure or set up the environment than Java. .Net is much better in building UIs while Java need more work. .Net has Lambdas expression and LINQ which supports a bunch of functional programming.

.Net
1) If the application is Windows platform specific then use .Net
2) .Net is Language independent, so if the team has multiple skill expertise C#, VB.NET , C++ , developers can still work on the same project with different skill set.
3) MS technologies provides RAD (rapid application development) to deviler project faster, customers always prefer faster delivery.
4) Debugging is very effort-less therefore, can fix the bugs quicker.
5) deployment is very easy and simple
6) Ajax implementation is simple & easy

.Java on the other hand
1) Platform independent
2) Open source
3) Dependant on the 3rd party tools to develop applications

What is HTTPhandler?

HttpHandler is a low level request and response API which is made to service incoming Http request. Every incoming Http request recieved by ASP.NET is ultimately processed by a instance of a class that implements HttpHandler.

Which method is used to enforce garbage collection in .NET?

System.GC.Collect ( ) method.

What is the difference between dispose() and finalize ()?

Although Dispose and Finalize both methods are used by CLR to perform garbage collection of runtime objects of .NET applications but there is a difference between them.

The Finalize method is called automatically by the runtime while the Dispose method is called by the programmer.

What is Garbage collection?

Garbage collection is used to prevent memory leaks during execution of programs. There is a low-priority process name as garbage collector manages the allocation and deallocation a memory for applications. It also checks for the unreferenced variables and objects. If there is ny object which is no further used by application the Garbage collector frees up the memory from that object. The garbage collectoion attempts to reclaim memory occupied by objects that are no longer in use by the program.

How would C/C++ Manage memory?

New(allocate) and delete(deallocate) operators are provided by C++ for runtime memory management. They are used for dynamic allocation and freeing of memory while a program is running.

The new operator allocates memory and returns a pointer to the start of it. The delete operator frees memory previously allocated using new. The general form of using them is:

p_var = new type;
delete p_var;
new allocates memory on the heap. If there is insufficient memory, then new will fail and a bad_alloc exception will be generated. The program should handle this exception

What are shallow and deep copy?

A shallow copy just copies the values of the data as they are. Even if there is a pointer that points to dynamically allocated memory, the pointer in the copy will point to the same dynamically allocated object.

A deep copy creates a copy of the dynamically allocated objects too. You would need to use a copy constructor and overload an assignment operator for this

What is a default constructor?

Default constructor is the constructor with no arguments or all the arguments has default values.

Virtual/Override/New

Virtual Keyword

`Virtual` keyword is used for generating a virtual path for its derived classes on implementing method overriding. `Virtual` keyword is used within a set with `override` keyword. It is used as:

Hide Copy Code

```
// Base Class
    class A
    {
        public virtual void show()
        {
            Console.WriteLine("Hello: Base Class!");
            Console.ReadLine();
        }
    }
```

Override Keyword

`Override` keyword is used in the derived class of the base class in order to override the base class method. `Override` keyword is used with `virtual` keyword, as:

Hide Copy Code

```
// Base Class
    class A
    {
        public virtual void show()
        {
            Console.WriteLine("Hello: Base Class!");
            Console.ReadLine();
        }
    }

// Derived Class

    class B : A
    {
        public override void show()
        {
            Console.WriteLine("Hello: Derived Class!");
            Console.ReadLine();
        }
```

```
    }
```

New Keyword

`New` keyword is also used in polymorphism concept, but in the case of method overloading So what does overloading means, in simple words we can say procedure of hiding your base class through your derived class.

It is implemented as:

Hide Copy Code

```
class A
    {
        public void show()
        {
            Console.WriteLine("Hello: Base Class!");
            Console.ReadLine();
        }
    }

    class B : A
    {
        public new void show()
        {
            Console.WriteLine("Hello: Derived Class!");
            Console.ReadLine();
        }
    }
```

Virtual & Override

- Used in polymorphism implementation
- Includes same method name and same params'
- Used in method overriding concept
- It is also called run time polymorphism
- Causes late binding

New

- It is also used in polymorphism concept
- Includes same method name and different params'
- Used in method overloading concept
- It is compile time polymorphism
- Cause early binding