Object Oriented Programming

What is class?

A class is the primary building block of object oriented programming.

```
//Stucture of class
public class Students
{
  public string firstName { get; set; }
  public string lastName { get; set; }
}
```

What is an object?

Object is an instance of a class.

```
Students student = new Students();
student.PrintFullName();
```

What are methods?

Methods are block of code that contain series of statement.

```
//Method print FullName
public void PrintFullName()
{
  Console.WriteLine("Enter first name");
  firstName = Convert.ToString(Console.ReadLine());

  Console.WriteLine("Enter last name");
  lastName = Convert.ToString(Console.ReadLine());

  Console.WriteLine("Your FullName:" + firstName+" "+lastName);
}
```

What is extension method?

An extension method enables us to add methods to existing types without creating a new derived type, recompiling, or modify the original types.

```
using System;
using System.Linq;
namespace InterviewQuestions
{
    class Demo
    {
        static void Main()
        {
           string strName=”farhan”;
            string result = strName.ChangeFirstLetterCase();

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

```
public class StringHelper
{
    public static string ChangeFirstLetterCase(string inputString)
    {
        if (inputString.Length > 0)
        {
            char[] charArray = inputString.ToCharArray();
            charArray[0] = char.IsUpper(charArray[0]) ?
                char.ToLower(charArray[0]) : char.ToUpper(charArray[0]);
            return new string(charArray);
        }

        return inputString;
    }
}
```

What is anonymous method?

An anonymous method is inline unnamed method in the code. It is created using the delegate keyword and doesn't required name and return type.

```
public delegate void Print(int value);

static void Main(string[] args)
{
    Print print = delegate(int val) {
        Console.WriteLine("Inside Anonymous method. Value: {0}", val);
    };

    print(100);
}
```

What is method overloading?

A method with same name but different signature is called method overloading.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace InterviewQuestions
{
    class MethodOverloading
    {
        static void Main()
        {
            Demo demo = new Demo();
            int Total=demo.Add(10, 20);
            int intTotal=demo.Add(10, 20, 30);
            double DecimalTotal=demo.Add(10.5, 15.5, 20.5);

            Console.WriteLine("SUM:" + Total);
            Console.WriteLine("SUM:" + intTotal);
            Console.WriteLine("SUM:" + DecimalTotal);
        }
    }
```

```
    public class Demo
    {
        public int Add(int FirstNumber, int SecondNumber)
        {
            return FirstNumber + SecondNumber;
        }

        public int Add(int FirstNumber, int SecondNumber, int ThirdNumber)
        {
            return FirstNumber + SecondNumber+ThirdNumber;
        }

        public double Add(double FirstNumber, double SecondNumber, double ThirdNumber)
        {
            return FirstNumber + SecondNumber + ThirdNumber;
        }
    }
}
```

What is method overriding?

A method with same name and same parameters is called method overriding.

```
using System;
namespace MethodOverriding_Demo
{
    public class Customer
    {
        public string firstName;
        public string lastName;

        public virtual void CustomerFullName()
        {
            Console.WriteLine("Full Name:"+firstName+ " " +lastName);
        }
    }

    public class PlatinumCustomer : Customer
    {
        public override void CustomerFullName()
        {
            base.CustomerFullName();
        }
    }

    public class GoldCustomer : Customer
    {
        public override void CustomerFullName()
        {
            base.CustomerFullName();
        }
    }

    public class SilverCustomer : Customer
    {
        public override void CustomerFullName()
        {
            base.CustomerFullName();
        }
    }
}
```

```
using System;
namespace MethodOverriding_Demo
{
    class Program
    {
        static void Main(string[] args)
        {
            PlatinumCustomer pc = new PlatinumCustomer();
            pc.firstName = "Farhan";
            pc.lastName = "Ahmed";
            pc.CustomerFullName();
            GoldCustomer gc = new GoldCustomer();
            gc.firstName = "Abdul";
            gc.lastName = "Jabbar";
            gc.CustomerFullName();
            SilverCustomer sc = new SilverCustomer();
            sc.firstName = "Abdul";
            sc.lastName = "Vaheed";
            sc.CustomerFullName();
            Console.ReadLine();
        }
    }
}
```

What is the different between Method Overriding and Method Hiding?

Method Overriding: A method with same name and same parameters is called method overriding.

```
using System;
namespace IntroductionToOOP
{
    public class BaseClass
    {
        public virtual void Print()
        {
            Console.WriteLine("I am base class");
        }
    }

    public class DrivedClass:BaseClass
    {
        public override void Print()
        {
            Console.WriteLine("I am drived class");
        }
    }
}
using System;
namespace IntroductionToOOP
{
    class Program
    {
        static void Main(string[] args)
        {
            var B = new DrivedClass();
            B.Print();
        }
    }
}
```

Method Hiding: A method with same name and same parameters hide the method of base class by using new key word is called method hiding.

```
using System;

namespace IntroductionToOOP
{
    public class BaseClass
    {
        public void Print()
        {
            Console.WriteLine("I am base class");
        }
    }

    public class DrivedClass:BaseClass
    {
        public new void Print()
        {
            Console.WriteLine("I am drived class");
        }
    }
}

using System;
namespace IntroductionToOOP
{
    class Program
    {
        static void Main(string[] args)
        {
            var B = new BaseClass();
            B.Print();
            var D = new DrivedClass();
            D.Print();
        }
    }
}
```

What are the access modifiers in c#?

1. **Public**: Access is not restricted.
2. **Private**: Access is limited to the containing type.
3. **Internal**: Access is limited to the current assembly.
4. **Protect**: Access is limited to the containing class or types derived from the containing class.
5. **Protected Internal**: Access is limited to the current assembly or types derived from the containing class.

What are access modifiers used for?

Access Modifiers are used to control the accessibility of types and members with in the types.

What is object oriented programming?

Object Oriented Programming is a technique to develop logical modules, such as classes that contains properties, fields and events. Object Oriented Programming provides many concepts such as abstraction, encapsulation, inheritance, and polymorphism.

There are four main pillars of object oriented programming.

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

Why do we need OOPs in programming language?

1. OOP provides a clear modular structure for programs.
2. OOP makes it easy to maintain and modify existing code.
3. OOP provides a good framework for code libraries where supplied software components can be easily adapted and modified by the programmer.
4. Reuse of code through inheritance.
5. Effective problem solving.
6. Flexibility through polymorphism

What is abstraction?

Abstraction: Abstraction is a process of hiding irrelevant information and showing only relevant information to the user.

What is encapsulation?

Encapsulation: Encapsulation is an ability to hide data and behavior that are not necessary to its user by using access modifier.

What is inheritance?

Inheritance: Inheritance is the ability to create new classes based on an existing class.

What is polymorphism?

Polymorphism: Polymorphism means that you can have multiple classes that can be used interchangeably, even though each class implements the same properties or methods in different ways.

What are the type of inheritance?

There are four types of Inheritance in Object oriented programming.

1. **Single Inheritance**: It contains one base class and one derived class.
2. **Hierarchical Inheritance**: It contains one base class and multiple derived classes of same base class.
3. **Multilevel Inheritance**: It contains a class derived from another derived class.
4. **Multiple Inheritance**: It contains several base class and a derived class.

Single Inheritance

```
using System;
namespace Inheritance_Demo
{
    public class Manager
    {
        public void Print()
        {
            Console.WriteLine("I am manager class");
        }

    }

    public class TeamLead : Manager
    {
        public new void Print()
        {
            Console.WriteLine("I am Team Lead class drived from manager class");
        }
    }
```

Hierarchical Inheritance

```
using System;
namespace Inheritance_Demo
{
    public class Manager
    {
        public void Print()
        {
            Console.WriteLine("I am manager class");
        }
    }

    public class TeamLead : Manager
    {
        public new void Print()
        {
            Console.WriteLine("I am Team Lead class drived from manager class");
        }
    }

    public class Employee : Manager
    {
```

```
        public new void Print()
        {
            Console.WriteLine("I am employee class drived from manager class");
        }
    }
}
```

Multilevel Inheritance

```
using System;
namespace Inheritance_Demo
{
    public class Manager
    {
        public void Print()
        {
            Console.WriteLine("I am manager class");
        }
    }

    public class TeamLead : Manager
    {
        public new void Print()
        {
            Console.WriteLine("I am Team Lead class drived from manager class");
        }
    }

    public class Employee : TeamLead
    {
        public new void Print()
        {
            Console.WriteLine("I am employee class drived from manager class");
        }
    }
}
```

Multiple Inheritance

Multiple inheritance is not allow or not possible in c# class but It can be achieved through interface**.**

```
using System;
namespace Multiple_Inheritace_Demo
{
    public class Doctor
    {
        public void doctor()
        {
            Console.WriteLine("I am doctor class");
        }
    }
}

using System;
namespace Multiple_Inheritace_Demo
{
    interface IAccountant
    {
        void accountant();
```

```
    }
}

using System;
namespace Multiple_Inheritace_Demo
{
    interface IEngineer
    {
        void engineer();
    }
}

using System;
namespace Multiple_Inheritace_Demo
{
    class Professional : Doctor, IAccountant,IEngineer
    {
        public void accountant()
        {
            Console.WriteLine("I am an accountant interface");
        }

        public void engineer()
        {
            Console.WriteLine("I am an engineer interface");
        }
    }
}

using System;
namespace Multiple_Inheritace_Demo
{
    class Program
    {
        static void Main(string[] args)
        {
            Professional professional = new Professional();
            professional.doctor();
            professional.accountant();
            professional.engineer();
            Console.ReadLine();
        }
    }
}
```

What are the types of polymorphism?

There are basically two types of polymorphism in c#.

1. Static or compile time polymorphism
2. Dynamic or runtime polymorphism

Static or compile time polymorphism

Static polymorphism is also called as compile time polymorphism. In static polymorphism methods are overloading with same name but different signatures or parameters. It is also called method overloading.

```
class MethodOverloading
    {
        public int Add(int firstNumber, int secondNumber)
        {
            return firstNumber + secondNumber;
        }

        public double Add(double firstNumber, double secondNumber, double thirdNumber)
        {
            return firstNumber + secondNumber +thirdNumber;
        }
    }

class Program
    {
        static void Main(string[] args)
        {
            MethodOverloading MO = new MethodOverloading();
            int Total= MO.Add(10, 20);
            double SecondTotal= MO.Add(10, 20, 30);

            Console.WriteLine("Total of first method:"+ Total);
            Console.WriteLine("Total of second method:"+ SecondTotal);
        }
    }
```

Dynamic or runtime polymorphism

Dynamic polymorphism is also called runtime polymorphism. In dynamic polymorphism methods have same name and same signature but different in implementation. It is also called method overriding.

```
using System;
namespace MethodOverriding_Demo
{
    public class Customer
    {
        public string firstName;
        public string lastName;

        public virtual void CustomerFullName()
        {
            Console.WriteLine("Full Name:"+firstName+ " " +lastName);
        }
    }

    public class PlatinumCustomer : Customer
    {
        public override void CustomerFullName()
        {
            base.CustomerFullName();
        }
    }

    public class GoldCustomer : Customer
    {
        public override void CustomerFullName()
```

```
        {
            base.CustomerFullName();
        }
    }

    public class SilverCustomer : Customer
    {
        public override void CustomerFullName()
        {
            base.CustomerFullName();
        }
    }
}

using System;
namespace MethodOverriding_Demo
{
    class Program
    {
        static void Main(string[] args)
        {
            PlatinumCustomer pc = new PlatinumCustomer();
            pc.firstName = "Farhan";
            pc.lastName = "Ahmed";
            pc.CustomerFullName();
            GoldCustomer gc = new GoldCustomer();
            gc.firstName = "Abdul";
            gc.lastName = "Jabbar";
            gc.CustomerFullName();
            SilverCustomer sc = new SilverCustomer();
            sc.firstName = "Abdul";
            sc.lastName = "Vaheed";
            sc.CustomerFullName();
            Console.ReadLine();
        }
    }
}
```