OOPS Question

Q1. What is Class And Object?

- ⇨ Class is a blueprint of object and object is a instance of an class

EX: Class Is a car and object is a model of car or color and price.

Q2. What is Inheritance?

- ⇨ It is process of inheriting the properties and behaviors of existing class into new class
- ⇨ Syntax:

```
Class <Parent_class>
{


}

Class < child_class> : <Parent_class>
{


}
```

Q3. What is multiple inheritance?

- ⇨ In Multiple Inheritance, child class can have more than one parent class

```
Class A1
{

}

Class A2
{

}

Class B : A1,A2
{

}
```

Note: Multiple Inheritance is not possible in c#

Q3. What is Constructor?

- ⇨ Constructor is a method that has the same name as that of class name.
- ⇨ Constructor do not have return type. So they can not return values
- ⇨ Access modifier can be used with constructor
- ⇨ It invoked when objects get created

- ⇨ It Allocates the appropriate memory to objects
- ⇨ It initialize the member variable of a class
- ⇨ We can declare more than one constructor in a class
- ⇨ Constructor can be overloaded

Example:

```
Namespace constructor
{
    Class Program
    {
        Public program()
        {
            Console.WriteLine("Hello I am Constructor");
        }
        Static void Main ( string args[])
        {
            Program obj =,new Program();
            Console.ReadLine();
        }
}
```

Q4. What is datatype? How many types

- ⇨ A data type specifies the size and type of variable values.
- ⇨ There are two types of datatypes: primitive datatype and non primitive datatype
- ⇨ Primitive Datatype: Primitive datatype is pre-defined by the programming language
- ⇨ The size and the type of varable value are specified
- ⇨ Non Primitive data type: These data types are not actually defined by the programming language but are created by programmer
- ⇨ The size is not fixed.

Q5. What is accessmodifier?

- ⇨ **access modifier** specifies the visibility/ accessibility of class and its members
- ⇨ **access modifier provide restriction in class and its members**

There are total 6 access modifier in c#

1. Public
2. Private
3. Protected
4. Internal
5. Protected Internal
6. Private protected

Q6. What is Polymorphism?

- ⇨ Polymorphism means having many forms

For Example

+ operator is used for Addition of two integers

For Example

10 + 20 = 30

It is also used for joining Two strings

For Example:

“Spider” + “man” = “Spiderman”

Types of Polymorphism

There 2 types of polymorphism

1. Complite time polymorphism / Static polymorphism
2. Run time polymorphism / Dynamic polymorphism

- ⇨ Compile time polymorphism achieved using method overloading
- ⇨ Run time polumorphism achieved by using method overriding

Q7. What is Methodoverloading

- ⇨ According to methodoverloading class can have multiple methods having same name but different parameters.

Example:

```
Namespace Methodoverloading
{
    Class Program
    {
        Void Sum (int a, int b)
        {
            Console.WriteLine (a+b);
        }

        Void Sum (float a, float b)
        {
            Console.WriteLine (a+b);
        }

        Static void Main (string[] args)
        {
            Program obj = new Program();
            Obj.Sum(10, 20);
            Console.ReadLine();
        }
    }
}
```

Q8 What is Methodoverridding?

⇨ With Methodoverriding, we can override the method of parent class with same method of child class

```
Class A
{
        Void Print()
        {
        }
}

Class B : A
{
        Void Print()
        {
        }
}
```

Example

```
Namespace MethodOverriding
{
         Class Animal
        {
                Public void Eat()
                {
                        Console.WriteLine(" Animal is Eating ");
                }
        }

        Class Dog : Animal
        {
                Public void Eat()
                {
                        Console.WriteLine(" Dog is Eating ");

                }
        }
```

```
        Public static void Main (string[] args)
        {
                Dog tommy = nre Dog();
                Tommy.Eat();
                Console.ReadLine();
        }
    }
}
```

Output: Dog is Eating

Q9. What is Abstraction?

- ⇨ Data abstraction is the process of hiding certain details and showing only essential information to the user.
- ⇨ Abstration can be achieved with either abstraction classes or interfaces.

Example:

```
abstract class Animal

{

  public abstract void animalSound();

  public void sleep()

  {

    Console.WriteLine("Zzz");

  }

}
```

From the example above, it is not possible to create an object of the Animal class:

```
Animal myObj = new Animal(); // Will generate an error (Cannot create
an instance of the abstract class or interface 'Animal')
```

To access the abstract class, it must be inherited from another class.

```
using System;

namespace MyApplication
{
  // Abstract class
  abstract class Animal
  {
    // Abstract method (does not have a body)
    public abstract void animalSound();
    // Regular method
    public void sleep()
    {
      Console.WriteLine("Zzz");
    }
  }

  // Derived class (inherit from Animal)
  class Pig : Animal
  {
    public override void animalSound()
    {
      // The body of animalSound() is provided here
      Console.WriteLine("The pig says: wee wee");
    }
  }

  class Program
  {
    static void Main(string[] args)
    {
      Pig myPig = new Pig();  // Create a Pig object
```

```
    myPig.animalSound();
    myPig.sleep();
  }
 }
}
```

Output: The pig says: wee wee

Zzz

Q10. What is Interface?

⇨ An `interface` is a completely "**abstract class**", which can only contain abstract methods and properties (with empty bodies):

Example:

```
      // interface
interface Animal
{
  void animalSound(); // interface method (does not have a body)
  void run(); // interface method (does not have a body)
}
```

To access the interface methods, the interface must be "implemented" (kinda like inherited) by another class. To implement an interface, use the `:` symbol (just like with inheritance). The body of the interface method is provided by the "implement" class. Note that you do not have to use the `override` keyword when implementing an interface:

```
using System;

namespace MyApplication
{
 // Interface
 interface IAnimal
 {
  void animalSound(); // interface method (does not have a body)
 }

 // Pig "implements" the IAnimal interface
 class Pig : IAnimal
 {
  public void animalSound()
  {
```

```
    // The body of animalSound() is provided here
    Console.WriteLine("The pig says: wee wee");
  }
 }

 class Program
 {
  static void Main(string[] args)
  {
   Pig myPig = new Pig();  // Create a Pig object
   myPig.animalSound();
  }
 }
}
```

Output: The pig says: wee wee

Q11. Difference between Inheritance and interface?

BASIS FOR COMPARISON	INTERFACE	INHERITANCE
Basic	Derives new classes by using an existing class.	Employs class abstraction and multiple inheritance.
Provides	Abstraction and multiple inheritance.	Reusability
Members	Declared as constant.	Declared as constant and variable.
Class definition	Contain the abstract methods.	Contain the code for each of its methods.
Access specifiers used	Public	Public, private or protected.
Instantiation	Cannot be used to declare objects.	Class can be instantiated using objects.

⇨

Q12. What is MultipleInterface?

⇨ To implement multiple interfaces, separate them with a comma:

Example:

```
using System;


namespace MyApplication
{
  interface IFirstInterface
  {
    void myMethod(); // interface method
  }


  interface ISecondInterface
  {
    void myOtherMethod(); // interface method
  }


  // Implement multiple interfaces
  class DemoClass : IFirstInterface, ISecondInterface
  {
    public void myMethod()
    {
      Console.WriteLine("Some text..");
    }
    public void myOtherMethod()
    {
      Console.WriteLine("Some other text...");
    }
  }
```

```
 class Program
 {
   static void Main(string[] args)
   {
     DemoClass myObj = new DemoClass();
     myObj.myMethod();
     myObj.myOtherMethod();
   }
 }
}
```

Output:

Some text..

Some other text...

Q13. What is try…catch? What is finally?

- ⇨ The `try` statement allows you to define a block of code to be tested for errors while it is being executed.
- ⇨ The `catch` statement allows you to define a block of code to be executed, if an error occurs in the try block.
- ⇨ The `finally` statement lets you execute code, after `try...catch`, regardless of the result:

Example:

```
using System;


namespace MyApplication
{
  class Program
  {
    static void Main(string[] args)
    {
      try
      {
        int[] myNumbers = {1, 2, 3};
        Console.WriteLine(myNumbers[10]);
      }
      catch (Exception e)
      {
        Console.WriteLine("Something went wrong.");
      }
      finally
      {
        Console.WriteLine("The 'try catch' is finished.");
      }
    }
  }
```

}

Output: Something went wrong.

The 'try catch' is finished.

Q14. Can we inherite Abstract class?

⇨ Yes, An Abstract class can inherit from a concrete class(non-Abstract class) and can also inherit from the following- According to inheritance concept in C#, an Abstract class can inherit from only one class either it can be Abstract or Concrete class but it can inherit from multiple interface.

Q15. What is the types of error Handling Exception?

Below are the different ways -

- Try-Catch
- Override onException from Controller class
- HandleError Attribute
- CustomException by HandleErrorAttribute
- https://www.c-sharpcorner.com/article/exception-handling-in-asp-net-mvc/
-

Try-catch

```
1. public IActionResult Index()
2. {
3.     try
4.     {
5.         int i = 10;
6.         i = i / 0;
7.         return View();
8.     }
9.     catch (Exception Ex)
10.     {
11.         return View("Error");
12.     }
13. }
```

Override on Exception

```
1. public class HomeController : Controller
2. {
3.     protected overridevoid OnException(ExceptionContext fi
   lterContext)
4.     {
```

```
5.          base.OnException(filterContext);
6.      }
7.
8.      public ActionResult Index()
9.      {
10.           return View();
11.       }
12.  }
```