**Virtual Keyword in C#**

# Method Overriding is an OOPs concept closely knit with Inheritance. When a child class method overrides the parent class method of the same name, parameters and return type, it is termed as method overriding. A virtual keyword is an indication to the compiler that a method may be overridden in derived classes.

# Coming to the C# perspective, the virtual keyword is used to modify the declaration of any property, method or event to allow overriding in a derived class. In simple terms, the virtual keyword implements the method overriding concept in C#.

# **Syntax**

# Any property, method or event can be overridden by adding the virtual keyword in the base class and override keyword in the derived class.

# Add the virtual keyword in the base class declaration:

public class Base {

public virtual int abc { get; set; } // this property can be overridden

public virtual void Xyz() { } // this method can be overridden

# }

public class Derived : Base {

public override int abc { get; set; } // this overrides the base class property

public override void Xyz() { } // this overrides the base class method

}

# **Static Keyword in C#**

The static keyword is used to make a data item non-instantiable. It can be used with classes, methods, variables, constructors, operators etc. However, it cannot be used with destructors, indexers etc.

# **Static Class**

# A static class is non-instantiable i.e. a variable of the class cannot be created using the new keyword. So, the static class members have to be accessed using the class name itself.

# A static class is defined using the keyword static. It can only have static data members and static methods. If this rule is not followed, there is a compile time error.

# A program that demonstrates a static class is given as follows:

# Source Code: Program that demonstrates a static class in C#

## using System;

## namespace StaticClassDemo

## {

## static class SClass

## {

## public static int staticVar = 5;

## public static void staticMethod()

## {

## Console.WriteLine("Inside Static Method");

## }

## }

## class Test

## {

## static void Main(string[] args)

## {

## Console.WriteLine("Value of static variable: {0}", SClass.staticVar);

## SClass.staticMethod();

## }

## }

## }

# You cannot create an object of the static class; therefore the members of the static class can be accessed directly using a class name

# **C# Keywords sealed**

The “sealed” keyword in C# is used to limit a class’s or a member’s ability to inherit from another class. A class or member thereof that has the designation “sealed” cannot be inherited by another class or overridden by a derived class.

Classes, methods, and properties can all use the “sealed” keyword. A class that has the designation “sealed” cannot be inherited by any other class. Similar to this, any derived class cannot override a method or property that is declared as sealed. Let’s examine the “sealed” keyword’s use with some samples from the code.

sealed class Vehicle

{

    public void Drive()

    {

        Console.WriteLine("Driving the vehicle.");

    }

}

class Car : Vehicle // This will give an error since Vehicle is marked as sealed

{

}

# **constructor chaining in C#**

One of the main benefits of constructor chaining is that it reduces code duplication and improves readability. By calling another constructor, you can avoid repeating the same initialization logic in multiple constructors. This makes the code more concise, consistent, and easier to maintain.

Another benefit of constructor chaining is that it helps enforce the inheritance hierarchy and the principle of least privilege. By calling a base class constructor using

:base

 you can ensure that the base class properties and methods are properly initialized before the derived class. This prevents errors and inconsistencies in the object state. By calling a constructor of the same class using

:this

, you can limit the access to the private or protected members of the class. This enhances the encapsulation and security of the class.

**Basic OOPs Interview Questions**

**What is meant by the term OOPs?**

OOPs refers to Object-Oriented Programming. It is the programming paradigm that is defined using objects. Objects can be considered as real-world instances of entities like class, that have some characteristics and behaviors.

**What is the need for OOPs?**

There are many reasons why OOPs is mostly preferred, but the most important among them are:

* OOPs helps users to understand the software easily, although they don’t know the actual implementation.
* With OOPs, the readability, understandability, and maintainability of the code increase multifold.
* Even very big software can be easily written and managed easily using OOPs.

**What is meant by Structured Programming?**

* **Structured Programming** refers to the method of programming which consists of a completely structured control flow. Here structure refers to a block, which contains a set of rules, and has a definitive control flow, such as (if/then/else), (while and for), block structures, and subroutines.

**What are the main features of OOPs?**

OOPs or Object Oriented Programming mainly comprises of the below four features, and make sure you don't miss any of these:

* Inheritance
* Encapsulation
* Polymorphism
* Data Abstraction

**What is a class?**

* A class can be understood as a template or a blueprint, which contains some values, known as member data or member, and some set of rules, known as behaviors or functions. So when an object is created, it automatically takes the data and functions that are defined in the class.  
  Therefore the class is basically a template or blueprint for objects. Also one can create as many objects as they want based on a class.

**What is an object?**

* An object refers to the instance of the class, which contains the instance of the members and behaviors defined in the class template. In the real world, an object is an actual entity to which a user interacts, whereas class is just the blueprint for that object. So the objects consume space and have some characteristic behavior

**What is encapsulation?**

One can visualize Encapsulation as the method of putting everything that is required to do the job, inside a capsule and presenting that capsule to the user. What it means is that by Encapsulation, all the necessary data and methods are bind together and all the unnecessary details are hidden to the normal user. So Encapsulation is the process of binding data members and methods of a program together to do a specific job, without revealing unnecessary details.  
  
Encapsulation can also be defined in two different ways:  
  
1) **Data hiding:** Encapsulation is the process of hiding unwanted information, such as restricting access to any member of an object.  
  
2) **Data binding:** Encapsulation is the process of binding the data members and the methods together as a whole, as a class.

**What is Polymorphism?**

Polymorphism is composed of two words - “poly” which means “many”, and “morph” which means “shapes”. Therefore Polymorphism refers to something that has many shapes

In OOPs, Polymorphism refers to the process by which some code, data, method, or object behaves differently under different circumstances or contexts. Compile-time polymorphism and Run time polymorphism are the two types of polymorphisms in OOPs languages.

**What is Compile time Polymorphism and how is it different from Runtime Polymorphism?**

Compile Time Polymorphism: Compile time polymorphism, also known as Static Polymorphism, refers to the type of Polymorphism that happens at compile time. What it means is that the compiler decides what shape or value has to be taken by the entity in the picture.

Runtime Polymorphism: Runtime polymorphism, also known as Dynamic Polymorphism, refers to the type of Polymorphism that happens at the run time. What it means is it can't be decided by the compiler. Therefore what shape or value has to be taken depends upon the execution. Hence the name Runtime Polymorphism.

**What is meant by Inheritance?**

The term “inheritance” means “receiving some quality or behavior from a parent to an offspring.” In object-oriented programming, inheritance is the mechanism by which an object or class (referred to as a child) is created using the definition of another object or class (referred to as a parent). Inheritance not only helps to keep the implementation simpler but also helps to facilitate code reuse

**What is Abstraction?**

If you are a user, and you have a problem statement, you don't want to know how the components of the software work, or how it's made. You only want to know how the software solves your problem. Abstraction is the method of hiding unnecessary details from the necessary ones. It is one of the main features of OOPs.

**How much memory does a class occupy?**

Classes do not consume any memory. They are just a blueprint based on which objects are created. Now when objects are created, they actually initialize the class members and methods and therefore consume memory.

**Is it always necessary to create objects from class?**

No. An object is necessary to be created if the base class has non-static methods. But if the class has static methods, then objects don’t need to be created. You can call the class method directly in this case, using the class name.

**What is a constructor?**

Constructors are special methods whose name is the same as the class name. The constructors serve the special purpose of initializing the objects.  
For example, suppose there is a class with the name “MyClass”, then when you instantiate this class, you pass the syntax:  
MyClass myClassObject = new MyClass();

**What is a destructor?**

Contrary to constructors, which initialize objects and specify space for them, Destructors are also special methods. But destructors free up the resources and memory occupied by an object. Destructors are automatically called when an object is being destroyed.

**What is the difference between overloading and overriding?**

Overloading is a compile-time polymorphism feature in which an entity has multiple implementations with the same name. For example, Method overloading and Operator overloading.

Whereas Overriding is a runtime polymorphism feature in which an entity has the same name, but its implementation changes during execution.

**What is an abstract class?**

An abstract class is a special class containing abstract methods. The significance of abstract class is that the abstract methods inside it are not implemented and only declared. So as a result, when a subclass inherits the abstract class and needs to use its abstract methods, they need to define and implement them

**What is meant by Garbage Collection in OOPs world?**

Object-oriented programming revolves around entities like objects. Each object consumes memory and there can be multiple objects of a class. So if these objects and their memories are not handled properly, then it might lead to certain memory-related errors and the system might fail.

Garbage collection refers to this mechanism of handling the memory in the program. Through garbage collection, the unwanted memory is freed up by removing the objects that are no longer needed.