What are the four fundamental principles of .1 ?#Object-Oriented Programming (OOP) in C

Encapsulation: Encapsulation helps to wrap up the functions and data .1
together in a single unit. By privatizing the scope of the data members it
can be achieved. This particular feature makes the program
.inaccessible to the outside world
Inheritance: It allows one class to inherit the attributes and functions of .2
another. This helps to promote code reusability and maintainability by
allowing new classes (derived classes) to reuse the functionality of
.existing classes without modifying them
Polymorphism: It means the ability to take more than one form. This .3
property makes the same entities such as functions, and operators
.perform differently in different scenarios
Abstraction: It means providing only the essential details to the outside .4
world and hiding the internal details, i.e., hiding the background details
.or implementation

?What is an object .2

An object is a real-world entity having a particular behavior and a state. It can be physical or logical. An object is an instance of a class and memory is .allocated only when an object of the class is created

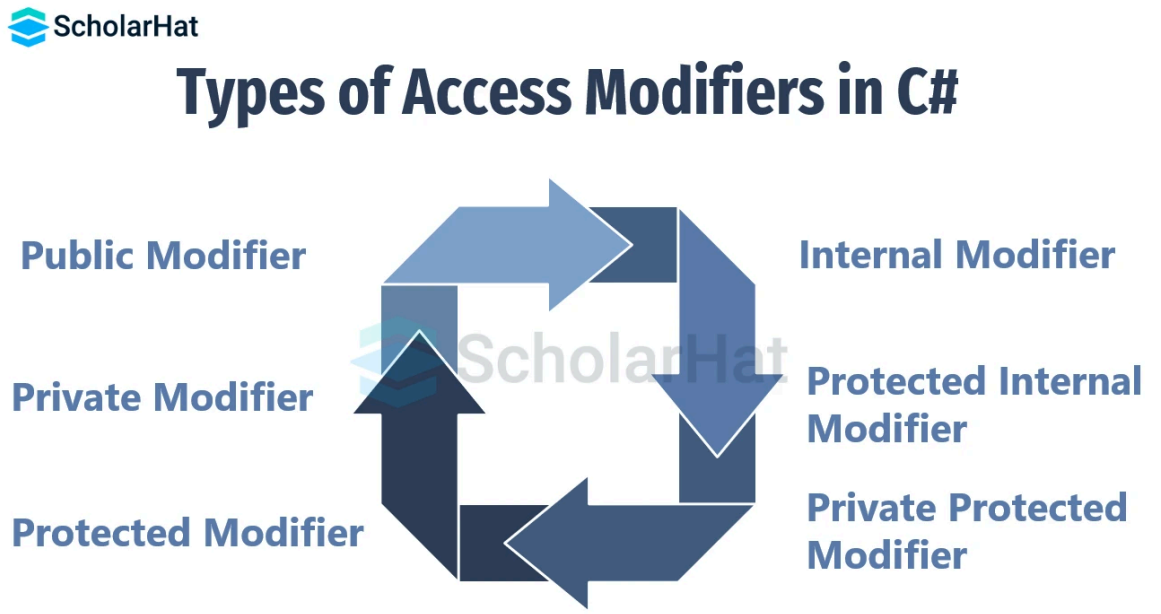
?#What are classes in C .2

The Class is a collection of objects. It is like a blueprint for an object. It is the foundational element of object-oriented programming. An instance of a class .is created for accessing its data types and member functions

Read More: Objects and Classes in C#: Examples and Differences

What are access modifiers in C#? What are their .4 ?types

Access modifiers in C# control the visibility and accessibility of classes, methods, and variables. They define who can access and interact with these members, providing security and encapsulation in object-oriented .programming, with options like public, private, protected, internal, and more



:In C#, there are 6 different types of Access Modifiers

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

.#Explain the types of Access Modifiers in C .5

public: In C#, the 'public' access modifier allows classes, methods, and .1 variables to be accessed from any part of the program, making them widely accessible. It promotes encapsulation and facilitates interaction between different components, enabling seamless communication in

object-oriented programming.

Example

```
} public class MyClass
;public int rollno
    2.    }
```

private: The access is available to the containing class or types derived .3 from the containing class within the current project.

Example

```
} class MyClass
;private int rollno
    4.    }
```

protected: It restricts member accessibility within the defining class and .5 its derived classes.

Example

```
} class MyClass
;protected int rollno
    6.    }
```

internal: It restricts the access of a member within its own assembly. .7

Example

```
} class MyClass
;internal int rollno
    8.    }
```

protected internal: Members declared as protected internal are .9 accessible within the same assembly and any derived classes, even if they are defined in a different assembly. This access modifier combines the functionality of both protected and internal.

Example

```
} class MyClass
;protected internal int rollno
    10.       }
```

private protected: The access is available to the containing class or.11 types derived from the containing class within the current project in an

assembly.

Example

```
} class MyClass
;protected internal int rollno
12.      }
```

#Read More: Access Modifiers in C

?#What is inheritance in C .6

Inheritance in object-oriented programming (OOP) allows one class to inherit the attributes and functions of another. The class that inherits the properties is known as the child class/derived class and the main class is called the parent class/base class. This means that the derived class can use all of the base class's members as well as add its own. Because it reduces the need to duplicate code for comparable classes, inheritance promotes code reusability .and maintainability

Inheritance is an is-a relationship. We can use inheritance only if there is an .is-a relationship between two classes

:In the above figure

A car is a Vehicle .1
A bus is a Vehicle .2
A truck is a Vehicle .3
A bike is a Vehicle .4

Here, the Car class can inherit from the Vehicle class, the bus class can .inherit from the Vehicle class, and so on

Read More: What is Inheritance in Java

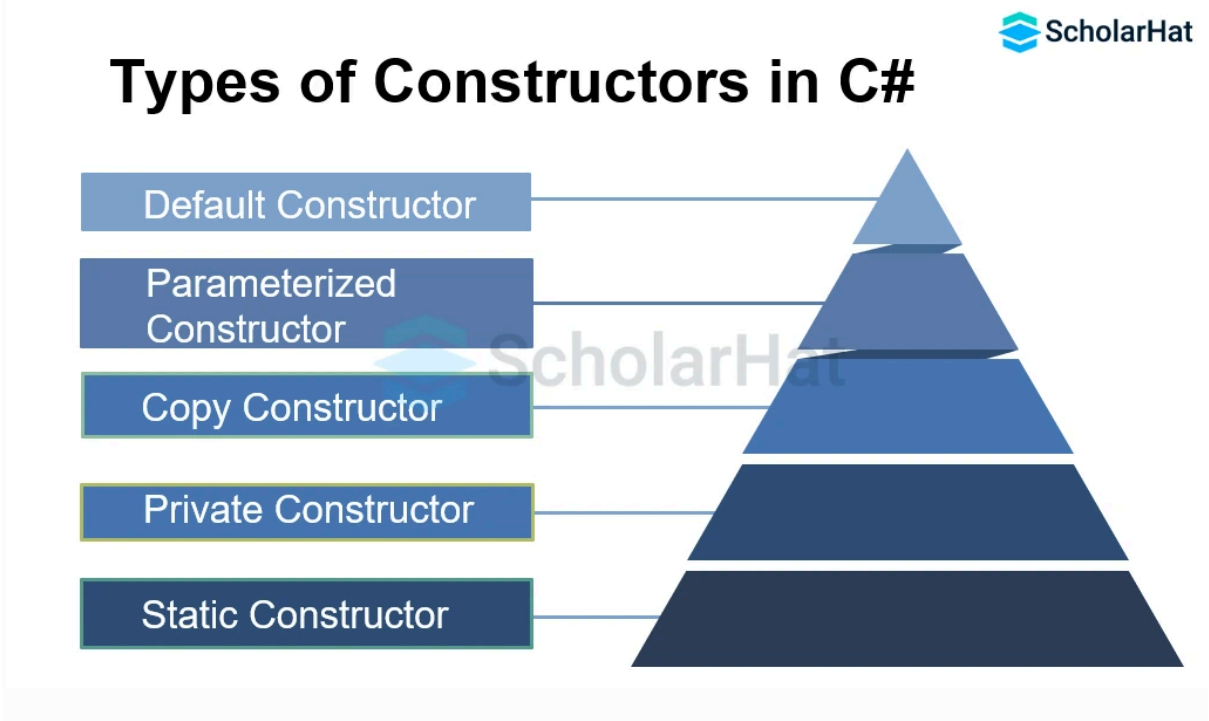
?#What is a constructor in C .7

A constructor is a special member function invoked automatically when an object of a class is created. It is generally used to initialize the data members of the new object. They are also used to run a default code when an object is created. The constructor has the same name as the class and does not return .any value

```
;()MyClass obj = new MyClass
```

In the above code, the method called after the “new” keyword - MyClass(), is the constructor of this class. This will help to instantiate the data members and .methods and assign them to the object obj

?#What are the types of constructors in C .8



Default Constructor: If we have not defined a constructor in our class, .1 then the C# will automatically create a default constructor with an empty code and no parameters. Here, all the numeric fields are initialized to 0, whereas string and object are initialized as null.

Example

```
;using System
```

```
} namespace Constructor

} class Program

;int x

} (static void Main(string[] args

call default constructor //
;()Program p = new Program

;(Console.WriteLine("Default value of x: " + p.x
;()Console.ReadLine

{
{
{
```

2.

3. Run Code >>

   Here, C# automatically creates a default constructor. The default constructor initializes any uninitialized variable with the default value.

   ## Output

```
Default value of a: 0
```

4.

.5 Parameterized Constructor: A constructor is considered parameterized if it accepts at least one parameter. Each instance of the class may have a different initialization value.

Example

```
.C# Program to illustrate calling of parameterized constructor //
;using System
} namespace ParameterizedConstructorExample

} class Scholar

.data members of the class //
;String name
;int id

parameterized constructor would //
initialized data members with //
the values of passed arguments //
.while object of that class created //
```

```
(Scholar(String name, int id
}
;this.name = name
;this.id = id
{

Main Method //
()public static void Main
}

This will invoke parameterized //
.constructor //
;(Scholar scholar1 = new Scholar("DNT", 1
+ Console.WriteLine("ScholarName = " + scholar1.name
;(and ScholarId = " + scholar1.id "
{
{
```

6. }
7. Run Code >>

   This C# code defines a Scholar class with a parameterized constructor. It creates an object scholar1, initializes its name and id, and then prints the values.

   ## Output

8. ScholarName = DNT and ScholarId = 1

.9 Private Constructor: A constructor is referred to as a private constructor if it was created with the private specifier. This class cannot be derived from by any other classes, nor can an instance of this class be created.

Example

```
;using System

} namespace Constructor

} class Program

private constructor //
} () private Program
;("Console.WriteLine("Private Constructor
{
{

} class Testing
```

```
} (static void Main(string[] args

call private constructor //
;()Program p = new Program
;()Console.ReadLine
{
{
{
```

10.

11. Run Code >>

Output

```
HelloWorld.cs(18,19): error CS0122: `Constructor.Program.Program()' is inaccessible
due to its protection level
(HelloWorld.cs(8,12): (Location of the symbol related to previous error
```

12.

13. Static Constructor: We use the static keyword to create a static constructor. Static constructors are designed to be used just once to initialize static class fields or data. We can have only one static constructor in a class. It cannot have any parameters or access modifiers.

Example

```
;using System

} namespace Constructor

} class Program

static constructor //
} ()static Program
;("Console.WriteLine("Static Constructor
{

parameterless constructor //
} ()Program
;("Console.WriteLine("Default Constructor
{

} (static void Main(string[] args

call parameterless constructor //
```

```
;()Program p1 = new Program

call parameterless constructor again //
;()Program p2 = new Program

;()Console.ReadLine
{
{
{
```

14.

15. Run Code >>

We cannot call a static constructor directly. However, when we call a regular constructor, the static constructor gets called automatically.

Output

```
Static Constructor
Default Constructor
Default Constructor
```

16.

17. Copy Constructor: We use a copy constructor to create an object by copying data from another object.

Example

```
;using System

} namespace Constructor

} class Car
;string brand

constructor //
} (Car (string theBrand
;brand = theBrand
{

copy constructor //
} (Car(Car c1
;brand = c1.brand
{

} (static void Main(string[] args
call constructor //
;("Car car1 = new Car("Safari
```

```
;(Console.WriteLine("Brand of car1: " + car1.brand

call the copy constructor //
;(Car car2 = new Car(car1
;(Console.WriteLine("Brand of car2: " + car2.brand

;()Console.ReadLine

{
{
{
```

18.

19. Run Code >>

Inside the copy constructor, we have assigned the value of the brand for the car1 object to the brand variable for the car2 object. Hence, both objects have the same value of the brand.

Output

```
Brand of car1: Safari
Brand of car2: Safari
```

20.

Read More: C# Constructor

.#Explain the concept of static members in C .9

Static members in C# are class-level members that belong to the class itself rather than to individual instances of the class. Static members are shared among all instances of a class, and they can be accessed directly using the .class name without creating an instance of the class

:Static members include

1. Static fields: Class-level variables that hold data shared among all .instances of the class
2. Static properties: Class-level properties that allow you to control the .access to static fields
3. Static methods: Class-level methods that can be called without creating an instance of the class. They can only access static members of the .class

Can you create an object of class with a private .10 ?#constructor in C

No, an object of a class having a private constructor can not be instantiated .from outside of the class

What is the difference between Struct and Class in .11 ?#C

Category	Struct	Class
Type	It is a value type	It is of reference type
Inherits from	It inherits from System.Value type	It inherits from System.Object type
Used for	Usually used for smaller amounts of data	Usually used for large amounts of data
Inherited to	It can not be inherited from other types	It can be inherited to other class
Abstract	It can not be abstract	It can be abstract type
New keyword	No need to create the object with the new keyword	Can not use an object of a class by using the new keyword

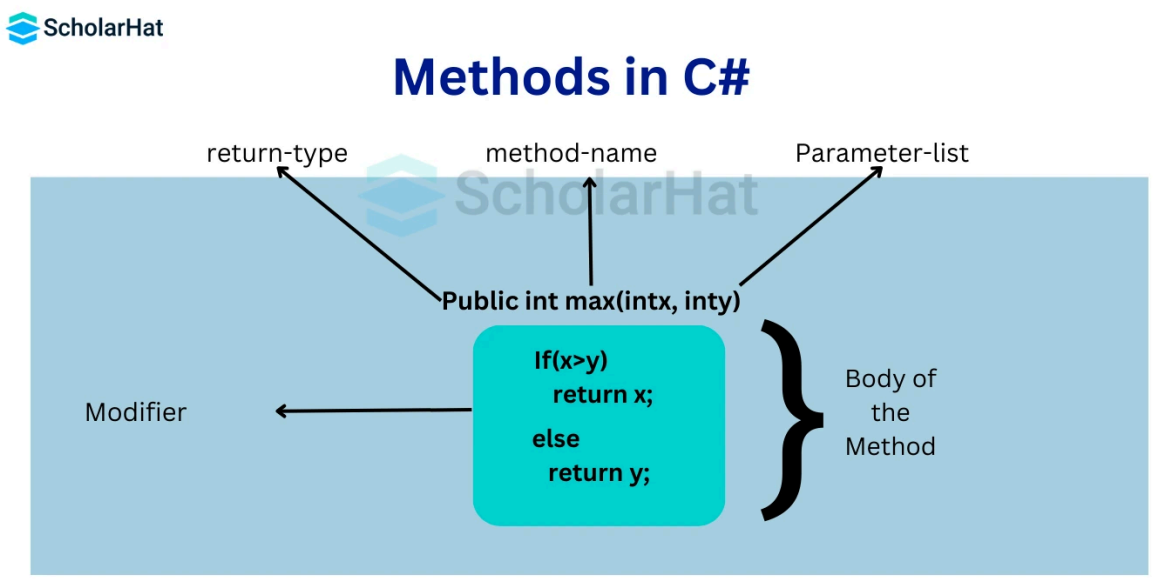
Default constructor	Do not have permission to create any default constructor	You can create a default constructor

In which conditions can we use static methods or .12 ?#properties in C

The member does not depend on the state of an object and can be .shared among all objects of the class
The member represents a utility function or a helper method that does .not require access to instance-specific data
You want to provide a global point of access for a certain functionality or .value that should be consistent across all instances of the class

.#Explain the structure of a method in C .13

A method in C# is a block of code within a class or a struct that performs a specific task or operation. Methods are defined with a method signature, which includes the method name, return type, parameters, and access modifiers. They encapsulate logic and can be called to execute their .functionality



Syntax

```
} ()access_modifier returnType methodName
method body //
{
```

,Here

access_modifier: Defines the visibility (public, private, etc.) of the .method
returnType: Specifies the data type the method returns or 'void' for no .return
.methodName: Unique identifier for the method
.(parameters: Input values the method accepts (optional
.{} body: contains the method's code logic enclosed in curly braces

#Read More: Methods in C

Can we create multiple objects from the same class? .14 ?How

.#Yes, we can create multiple objects from the same class in C

Example illustrating multiple object creation from the #same class in C

```
} namespace ClassObjects

} class Employee

;string department

} (static void Main(string[] args

create Employee object //
;()Employee sourav = new Employee

set department for sourav //
;"sourav.department = "SEO
;(Console.WriteLine("Sourav: " + sourav.department
```

```
create second object of Employee //
;()Employee sakshi = new Employee

set department for sakshi //
;"sakshi.department = "all rounder
;(Console.WriteLine("Sakshi: " + sakshi.department

;()Console.ReadLine
{
{
{
```

<< Run Code

In the above code, we have created two objects: sourav and sakshi from the Employee class. You can observe both the objects have their own version of .the department field with different values

Output

```
Sourav: SEO
Sakshi: all rounder
```

.#Explain method overloading in C .15

Method overloading in C# allows defining multiple methods with the same name in the same class but with different parameters. These methods perform similar tasks but can accept different types or numbers of parameters, enabling flexibility and enhancing code readability and reusability. Overloaded methods may have the same or different return types, but they must have .different parameters

Example

```
;using System

} namespace MethodOverload

} class Program

method with one parameter //
```

```
} (void display(int x
;(Console.WriteLine("Arguments: " + x
{

method with two parameters //
} (void display(int x, int y
;(Console.WriteLine("Arguments: " + x + " and " + y
{
} (static void Main(string[] args

;()Program p1 = new Program
;(p1.display(550
;(p1.display(550, 200
;()Console.ReadLine
{
{
<<(
```

.In the above code, we have overloaded the display() method

one method has one parameter .1
another has two parameters .2

?#What do you understand by variable scope in C .16

The scope of a variable refers to the part of the program where that variable is .accessible

:#There are three types of scope of a variable in C

.#Explain the types of variable scope in C .17

1. Class Level: It refers to the visibility and accessibility of variables and methods within a class. Class-level variables are known as fields and are declared outside of methods, constructors, and blocks of the class.

Example

```
using System;
namespace VariableScope
{
class Program
{
// class level variable
string str = "Class Level variable";

public void display()
{
Console.WriteLine(str);
}

static void Main(string[] args)
{
Program p = new Program();
p.display();

Console.ReadLine();
}
}
}
```

2.

3. Run Code >>

Output

```
Class Level variable
```

4.

5. Method Level: Variables declared within a method are accessible only within that method.

Example

```
using System;
namespace VariableScope
{
class Program
{

public void display()
{
string str = "inside method";
```

```
accessing method level variable //
;(Console.WriteLine(str
{

} (static void Main(string[] args
;()Program p = new Program
;()p.display

;()Console.ReadLine
{
{
{
```

6.

7. Run Code >>

   In the above code, we have created the str variable and accessed it within the same method display(). Hence, the code runs without any error.

   ## Output

```
inside method
```

8.

Block Level: It refers to the visibility and accessibility of variables within a specific block of code(for loop, while loop, if..else). .9

Example

```
;using System

} namespace VariableScope
} class Program
} ()public void display

} (++for(int i=0;i<=3;i
;(Console.WriteLine(i
{

{

} (static void Main(string[] args
;()Program p = new Program
;()p.display

;()Console.ReadLine
{
```

```
{
{
```

10.

11. Run Code >>

In the above program, we have initialized a block-level variable i inside the for loop. Therefore it's accessible only within the for loop block.

Output

```
0
1
2
```

12. 3

#Read More: Scope of Variables in C

?#What is this keyword in C .18

.In C#, this keyword refers to the current instance of a class

Example

```
;using System

} namespace ThisKeyword
} class This

;int number
} (This(int number
this.num refers to the instance field //
;this.number = number
;(Console.WriteLine("object of this: " + this
{

} (static void Main(string[] args

;(This t = new This(4
;(Console.WriteLine("object of t: " + t
;()Console.ReadLine
{
{
{
```

<< Run Code

In the above code, there's an object named t of the class This. We have printed the name of the object t and this keyword of the class. We can see the name of both t and this is the same. This is because this keyword refers to the .current instance of the class which is t

Output

```
object of this: ThisKeyword.This
object of t: ThisKeyword.This
```

?What is a destructor .19

A Destructor is automatically invoked when an object is finally .destroyed
The name of the Destructor is the same as the class and prefixed with a .(~) tilde
A Destructor is used to free the dynamically allocated memory and .release the resources

?Can “this” be used in a static method .20

A static method does not need the creation of an object. In other words, static methods do not belong to a particular object. Their existence is independent of .the creation of the object

Whereas, “this” keyword always points to a reference or instance of a class. As there is no object in the static method, therefore, “ this “ keyword can't be .used in a static method

Intermediate C# OOPs Interview Questions and Answers

.#Differentiate static and const in C .21

static	const
Declared using the static keyword	Declared using the const keyword. By default, a const is static and cannot be changed
Classes, constructors, methods, variables, properties, events, and operators can be static. The struct, indexers, enum, destructors, or .finalizers cannot be static	Only the class-level fields or variables can be constant
Static members can only be accessed within the static methods. The non-static methods cannot .access static members	The constant fields must be initialized at the time of declaration. Therefore, const variables are used for compile-time constants
The value of the static members can be modified using ClassName.StaticMemberName	Constant variables cannot be modified after the declaration
Static members can be accessed using ClassName.StaticMemberName, but .cannot be accessed using object	Const members can be accessed using ClassName.ConstVariableName, but cannot be accessed using the object

?#What are abstract classes in C .22

Abstract classes in C# are classes that cannot be instantiated, and they are meant to be inherited by other classes. They are used to provide a common

definition of a base class that can be shared by multiple derived classes. We .use the abstract keyword to create an abstract class

Abstract classes can have abstract and non-abstract methods and properties. .Derived classes must provide an implementation for these abstract members

```
} abstract class Program

abstract method //
;()public abstract void display1

non-abstract method //
} ()public void display2
;("Console.WriteLine("Non abstract method
{
{
```

#Read More: Abstract Class in C

.#Elaborate abstraction in C .23

Abstraction means providing only the essential details to the outside world and hiding the internal details, i.e., hiding the background details or implementation. Abstraction is a programming technique that depends on the .separation of the interface and implementation details of the program

.#The abstract classes are used to achieve abstraction in C

Example

```
;using System


public abstract class Shape
}
Abstract method to calculate area //
;()public abstract double CalculateArea

Concrete method //
()public void PrintDetails
```

```
}
;(".Console.WriteLine("This is a shape
{
{

Concrete class implementing the Shape abstract class //
public class Rectangle : Shape
}
{ ;public double Width { get; set
{ ;public double Height { get; set

Implementing the abstract method to calculate area //
()public override double CalculateArea
}
;return Width * Height
{
{

Interface example //
public interface IAnimal
}
Interface method //
;()void MakeSound
{

Concrete class implementing the IAnimal interface //
public class Dog : IAnimal
}
Implementing the interface method //
()public void MakeSound
}
;("!Console.WriteLine("Woof
{
{

Another concrete class implementing the IAnimal interface //
public class Cat : IAnimal
}
Implementing the interface method //
()public void MakeSound
}
;("!Console.WriteLine("Meow
{
{

class Program
```

```
}
(static void Main(string[] args
}
Using abstraction through abstract class //
;()Shape rectangle = new Rectangle
;()rectangle.PrintDetails
;Rectangle)rectangle).Width = 5))
;Rectangle)rectangle).Height = 3))
;(()Console.WriteLine("Area of rectangle: " + rectangle.CalculateArea

Using abstraction through interface //
;()IAnimal dog = new Dog
!dog.MakeSound(); // Outputs: Woof

;()IAnimal cat = new Cat
!cat.MakeSound(); // Outputs: Meow
{
{
```

<< Run Code

The Shape abstract class defines a common behavior CalculateArea() which must be implemented by concrete shapes. It also contains a .()concrete method PrintDetails

The Rectangle class inherits from Shape and provides a concrete .()implementation of CalculateArea

The IAnimal interface defines a common behavior MakeSound() that .must be implemented by classes representing different animals

The Dog and Cat classes implement the IAnimal interface with their .sound implementations

Output

```
.This is a shape
Area of rectangle: 15
!Woof
!Meow
```

Read More: C# Class Abstraction

What is the difference between Abstraction and .24 ?#Encapsulation in C

Encapsulation	Abstraction
Encapsulation is the process or method of containing the information in a single unit and providing this .single unit to the user	Abstraction is the process or method of gaining information
Main feature: Data hiding. It is a common practice to add data hiding in any real-world product to protect it from the external world. In OOPs, this is done through specific access modifiers	Main feature: reduce complexity, promote maintainability, and also provide a clear separation between the interface and its concrete implementation
problems are solved at the .implementation level	problems are solved at the design or interface level
It is a method to hide the data in a single entity or unit along with a method to protect information from outside	It is the method of hiding the unwanted information
encapsulation can be implemented using access modifiers i.e. private, protected, and public	We can implement abstraction using abstract classes and interfaces
implementation complexities are hidden using abstract classes and interfaces	the data is hidden using methods of getters and setters
the objects that result in encapsulation need not be abstracted	The objects that help to perform abstraction are encapsulated

Encapsulation hides data and the user cannot access the same directly	Abstraction provides access to specific parts of data
Encapsulation focus is on “How” it should be done	Abstraction focus is on “what” should be done

.#Throw light on the base keyword in C .25

The base keyword is used to access members of the base class from .within a derived class
Call a method on the base class that has been overridden by another .method
Specify which base-class constructor should be called when creating .instances of the derived class
.It is an error to use the base keyword from within a static method

?What is an interface .26

In C#, an interface is similar to an abstract class. However, unlike abstract classes, all methods of an interface are fully abstract. We need to use the .interface keyword to create an interface

Example

```
} interface IShapes

method without body //
;()void calculateArea
{
```

,In the above code

.IShapes is the name of the interface
By convention, the interface starts with I so that we can identify it just by .seeing its name
.We cannot use access modifiers inside an interface
.All members of an interface are public by default

.An interface doesn't allow fields

We cannot create objects of an interface. To use an interface, other classes .must implement it

.#Mention some advantages of interfaces in C .27

Multiple Inheritance-like behavior: C# does not support multiple inheritance for classes, but a class can implement multiple interfaces. This allows a class to inherit behavior from multiple sources, promoting .code reuse and flexibility in class design
Loose Coupling: Interfaces facilitate loose coupling between classes. By programming against interfaces rather than concrete classes, you can easily change the implementation details without affecting the client .code that relies on the interface
Code Extensibility: Interfaces allow you to add new functionality to existing classes without modifying their source code. You can create new interfaces and implement them in the existing classes, enhancing .the capabilities of those classes
Abstraction: Similar to abstract classes, interfaces help us to achieve .#abstraction in C

What differentiates an abstract class from an .28 ?#interface in C

Abstract Class	Interface
It contains both declaration .and implementation parts	It contains only the declaration of methods, properties, events, or indexers. Since C# 8, default implementations can also be .included in interfaces
Multiple inheritance is not achieved by abstract class	Multiple inheritance is achieved by .interface

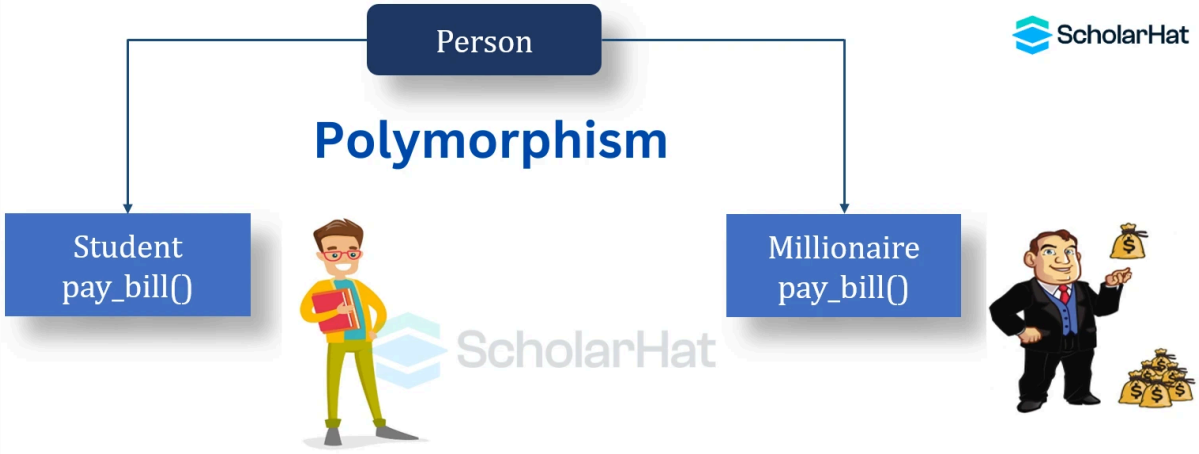
It contains constructor	It does not contain constructors
It can contain static members	It does not contain static members
It can contain different types of access modifiers like public, private, .protected, etc	It only contains a public access modifier because everything in the interface is public
The performance of an abstract class is fast	The performance of the interface is slow because it requires time to search actual method in the corresponding class
It is used to implement the core identity of class	It is used to implement the peripheral abilities of a class
A class can only use one abstract class	A class can use multiple interface
Abstract class can contain methods, fields, constants, .etc	An interface can only contain methods, .properties, indexers, and events
It can be fully, partially, or .not implemented	It should be fully implemented
To declare an abstract class, the abstract keyword is used	To declare an interface, the interface keyword is used

Are private class members inherited from the .29 ?derived class

Yes, the private members are also inherited in the derived class but we will not .be able to access them

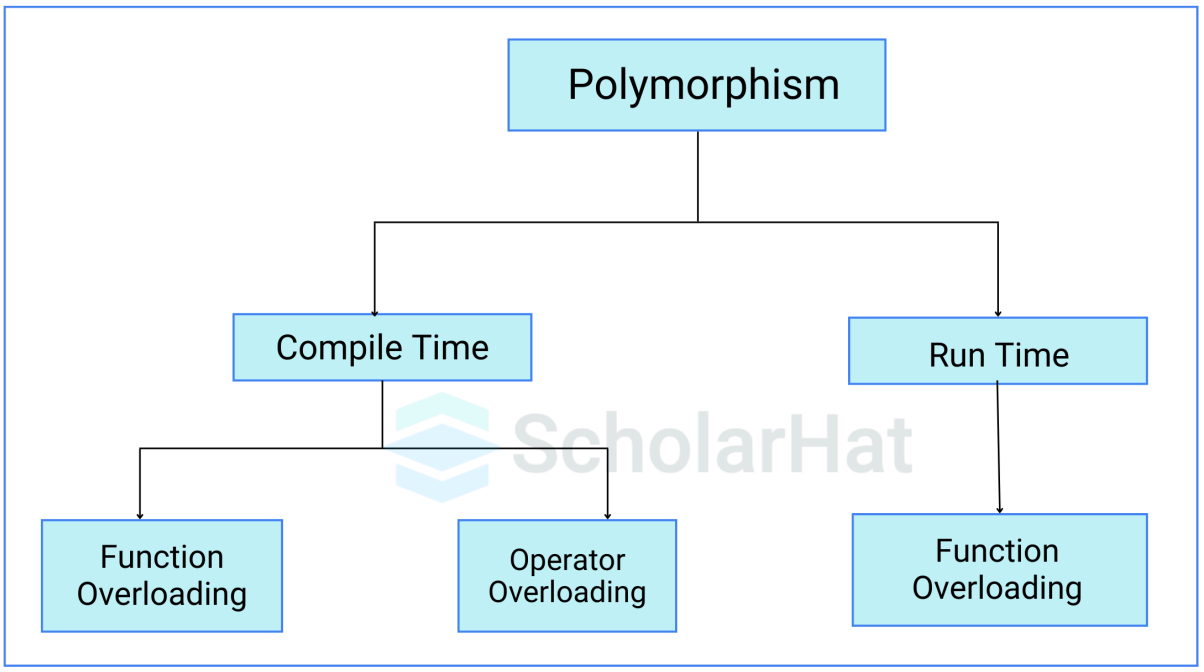
?What is polymorphism .30

Polymorphism is made up of two words, poly means more than one and morphs means forms. So, polymorphism means the ability to take more than one form. This property makes the same entities such as functions, and operators perform differently in different scenarios. You can perform the same .task in different ways



In the given figure, a person when a student pays his bills. After completing his studies, the same person becomes a millionaire and pays bills of various types. Here the person is the same and his functions are also the same. The .only difference is in the type of bills

?#What are the different types of polymorphism in C .31



:There are two types of polymorphism

1. Compile Time Polymorphism / Static Polymorphism: In this, the compiler at the compilation stage knows which functions to execute during the program execution. It matches the overloaded functions or operators with the number and type of parameters at the compile time.
:The compiler performs compile-time polymorphism in two ways
1. Function Overloading: If two or more functions have the same name but different numbers and types of parameters, they are known as overloaded functions.
2. Operator Overloading: It is function overloading, where different operator functions have the same symbol but different operands. And, depending on the operands, different operator functions are executed.
2. Runtime Polymorphism: In this, the compiler at the compilation stage does not know the functions to execute during the program execution. The function call is not resolved during compilation, but it is resolved in the run time based on the object type. This is also known as dynamic binding or late binding. This is achieved through function overriding.
When you redefine any base class method with the same signature i.e. same return type, number, and type of parameters in the derived class, it is known as function overriding.

#Explain the method overriding in C .32

Overriding means changing the functionality of a method without changing the .signature

You can override a method in the base class by creating a similar method in .the derived class. This can be done by using virtual/override keywords

1. virtual:allows the method to be overridden by the derived class
2. override:indicates the method is overriding the method from the base class

Example

```
;using System
class Polygon
}
method to render a shape //
()public virtual void render
}
;("...Console.WriteLine("Rendering Polygon
{
{

class Rectangle : Polygon
}
overriding render() method //
()public override void render
}
;("...Console.WriteLine("Rendering Rectangle
{
{
class myProgram
}
()public static void Main
}
obj1 is the object of Polygon class //
;()Polygon obj1 = new Polygon

calls render() method of Polygon Superclass //
;()obj1.render

here, obj1 is the object of derived class Rectangle //
;()obj1 = new Rectangle
```

```
calls render() method of derived class Rectangle //
;()obj1.render
{
{
```

<< Run Code

In the above example, we have created a superclass: Polygon, and a subclass: Rectangle. The Rectangle class is overriding the render() method of .the base class, Polygon

Output

```
...Rendering Polygon
...Rendering Rectangle
```

?#What is the use of a static constructor in C .33

A static constructor is a special constructor that gets called before the .first object of the class is created
It is used to initialize any static data or to perform a particular action that .needs to be performed once only
The time of execution of the static constructor is not known. But, it is definitely before the first object creation – maybe at the time of loading .assembly

?#How is exception handling implemented in C .34

Exception handling in C# is a mechanism to handle runtime errors and gracefully recover from them or display meaningful error messages to the .users. It can be implemented using try, catch, and finally blocks

1. try: The code that might throw an exception is placed inside the try block. If an exception occurs within the try block, the execution is .transferred to the appropriate catch block
2. catch: The catch block is used to handle the exception that occurred in the try block. You can have multiple catch blocks to handle different .types of exceptions
3. finally: This block contains code that must be executed regardless of whether an exception was thrown or not. It is typically used for cleanup .tasks, such as closing file handles or database connections

Example

```
;using System

class Program
}
(static void Main(string[] args
}
;int result

try
}
;int num1 = 10
;int num2 = 0

;result = num1 / num2
{
(catch (DivideByZeroException ex
}
;(".Console.WriteLine("Error: Division by zero is not allowed
;result = -1
{
(catch (Exception ex
}
;("{Console.WriteLine($"Error: {ex.Message
;result = -1
{
finally
}
Console.WriteLine("This line will be executed regardless of whether an exception
;(".was thrown or not
{

Console.WriteLine($"Result: {result}"); // Output: Result: -1
{
{
```

<< Run Code

In this example, the code inside the try block attempts to divide by zero, which will throw a DivideByZeroException. The catch block handles this exception and displays a meaningful error message, while the finally block executes .regardless of the exception

Output

```
.Error: Division by zero is not allowed
.This line will be executed regardless of whether an exception was thrown or not
Result: -1
```

#Read More: Errors and Exceptions Handling in C

?What is Implicit interface implementation .35

This is the most regular or obvious way to implement members of an interface. Here you do not specify the interface name of the members and implement .implicitly

Example

```
;using System

Define an interface //
public interface IShape
}
void Draw(); // Interface method
{

Define a class that implements the interface implicitly //
public class Circle : IShape
}
Implementation of the interface method //
()public void Draw
}
;("Console.WriteLine("Drawing a circle
{
{

class Program
}
(static void Main(string[] args
}
Create an instance of the Circle class //
;()Circle circle = new Circle

Call the Draw method through the interface reference //
```

```
circle.Draw(); // Output: Drawing a circle
{
{
```

<< Run Code

We define an IShape interface with a single method Draw(). The Circle class implicitly implements the IShape interface by providing an implementation for .the Draw() method

Output

```
Drawing a circle
```

Advanced OOPs in C# Interview Questions & Answers for Experienced

How do early and late binding impact an .36 application's performance, maintainability, and ?extensibility

	Early Binding	Late Binding
Perform ance	better performance since the method or property calls are resolved at compile time, and at runtime it does not need to perform any additional lookups or type .checks	performance overhead because at runtime it must perform additional work to determine the method or property to be called, including type checks, .lookups, etc

Maintainability	more maintainable code because the types and method signatures are known at compile time. This makes it easier to identify and fix issues during .development	less maintainable code because errors may not be detected until runtime, making it hard to identify and .fix issues
Extensibility	less flexible in terms of extensibility, as the types and method signatures must be known at compile .time	allows for greater extensibility because the types and method signatures do not need to be known at compile time, allowing more flexible implementations

What are the advantages and disadvantages of .37 ?using immutable objects in an application

Advantages of using Immutable Objects
Simplified reasoning about state: Immutable objects make it easier to reason about the state of an application, as their state cannot change .after they have been created
Thread-safety: Immutable objects are inherently thread-safe, as there is no risk of race conditions or other concurrency-related issues when .using them in multi-threaded environments
Hashing and caching: Immutable objects make excellent candidates for hashing and caching, as their state does not change. This can improve the performance of certain operations, such as dictionary lookups or .memoization

Disadvantages of using Immutable Objects
Object creation overhead: Creating immutable objects can introduce object creation overhead, as new instances must be created for every .state change
Complexity: In some cases, using immutable objects can introduce complexity to an application, as they may require additional classes or .patterns to manage state changes

What are the differences between value types and .38 ?reference types in C# OOP

Memory storage location: Value types are stored on the stack, while .reference types are stored on the heap
Copying behavior: Value types create a copy of the data when they are assigned or passed, while reference types use the same instance of the .data when they are assigned or passed
Equality comparison: Value types compare the data by value, while .reference types compare the data by reference
Immutability vs. mutability: Value types are immutable i.e. they cannot be changed after they are created, while reference types are mutable, .which means that they can be changed after they are created
Parameter passing: When passing value types as method parameters, they are passed by value, meaning a copy of the data is created. Reference types, on the other hand, are passed by reference, so changes made to the parameter within the method affect the original .object
Common use cases: Value types are used for simple and primitive data, such as numbers, booleans, and structs, while reference types are used .for complex and dynamic data, such as objects, arrays, and classes

Read More: Data Types in C# with Examples: Value and Reference Data Type

?When to use value types .39

:Value types are useful in situations when

The data is small and simple and does not need to be modified after it is .created
The data needs to be accessed quickly and directly, without any .indirection or overhead
The data needs to be isolated and independent and does not affect or .depend on other data

Examples of scenarios where value types are good

Calculating mathematical expressions

```
;int result = (a + b) * c
```

Storing logical values

```
;bool isValid = true
```

Defining custom data types that have a fixed size and structure

```
struct Rectangle
}
;int width
;int height
;{
```

?In C#, how can you achieve immutability in a class .40

Immutability is a property of an object that ensures its state cannot be changed after it has been created. In C#, you can achieve immutability in a class by making its fields read-only and initializing them during object .construction

#Example of Implementing an Immutable Class in C

```
public class Employee
}
{ ;public string FirstName { get
{ ;public string LastName { get

(public Employee(string firstName, string lastName
}
;FirstName = firstName
;LastName = lastName
{
{
```

In the above code, the Employee class has two read-only properties, FirstName and LastName, that are initialized in the constructor. Once an .instance of Employee is created, its state cannot be changed

In C#, how can you prevent a class from being .41 ?inherited further while allowing instantiation

In C#, you can prevent a class from being inherited further by marking it as sealed. A sealed class can still be instantiated, but it cannot be used as a .base class for other classes

Example

```
public sealed class MyClass
}
()public void MyMethod
}
... //
{
{

The following class definition would cause a compilation error //
public class MyDerivedClass : MyClass
}
... //
{
```

.By sealing a class, you restrict its extensibility in the class hierarchy

How does the C# support multiple inheritance in .42 ?OOP

While C# does not allow a class to inherit directly from multiple classes, it supports a form of multiple inheritance through interfaces. A class can implement multiple interfaces, providing a way to inherit behavior from multiple sources. By using multiple interfaces, a class can define its behavior based on .the contracts established in those interfaces

Example demonstrating the use of multiple interfaces in #C

```
;using System

namespace interface1
}
public interface intf1
}
;()void message
{

public interface intf2
}
;()void message
{

public class child: intf1, intf2
}
()public void message
}
;("Console.WriteLine("multiple inheritance using interface
{
{

class Program
}
(static void Main(string[] args
}
;()child c = new child
;()c.message
;()Console.ReadKey

{
{
{
```

<< Run Code

The child class implements both intf1 and intf2 interfaces, providing a single implementation of the message() method that satisfies the requirements of .both interfaces

Output

```
multiple inheritance using interface
```

?#Can an abstract class be sealed in C .43

No, an abstract class cannot be a sealed class because the sealed modifier prevents a class from being inherited and the abstract modifier requires a .class to be inherited

?#What is a sealed class in C .44

A sealed class restricts the inheritance feature of object-oriented programming. Once a class is defined as a sealed class, the class can not be inherited. A class restricting inheritance for security reasons is declared as a sealed class. The sealed class is the last in the hierarchy. It can be a derived class but cannot be a base class. It can not also be an abstract class. Because abstract class has to provide functionality and here you are .restricting it to inherit

```
;using System

public sealed class SealedClass
}
Property //
{ ;public string Property1 { get; set

Method //
()public void Method1
}
;("Console.WriteLine("Method1() called
{
{

This class cannot inherit from SealedClass because it's sealed //
public class DerivedClass : SealedClass {} // Uncommenting this line will result in a //
compilation error

class Program
}
(static void Main(string[] args
}
Create an instance of the sealed class //
;()SealedClass sealedObj = new SealedClass

Access properties and methods of the sealed class //
```

```
;"sealedObj.Property1 = "Value
;()sealedObj.Method1

;()Console.ReadLine
{
{
```

<< Run Code

The DerivedClass is an attempt to derive from SealedClass, but it's .commented out because a sealed class cannot be inherited from it

Output

```
Method1() called
```

?#What is constructor chaining in C .45

Constructor chaining is an approach where a constructor calls another .constructor in the same or base class

```
;using System

public class MyClass
}
;private int value1
;private string value2

Constructor with two parameters //
(public MyClass(int value1, string value2
}
;this.value1 = value1
;this.value2 = value2
{

Constructor with one parameter, chaining to the two-parameter constructor //
("public MyClass(int value1) : this(value1, "default
}

{

Constructor with no parameters, chaining to the one-parameter constructor //
(public MyClass() : this(0
```

```
}

{


()public void DisplayValues
}
;("{Console.WriteLine($"Value1: {value1}, Value2: {value2
{
{

class Program
}
(static void Main(string[] args
}
Create instances of MyClass using different constructors //
;("MyClass obj1 = new MyClass(10, "Hello
;(MyClass obj2 = new MyClass(20
;()MyClass obj3 = new MyClass

Display values of each instance //
;(":Console.WriteLine("Object 1
;()obj1.DisplayValues

;(":Console.WriteLine("\nObject 2
;()obj2.DisplayValues

;(":Console.WriteLine("\nObject 3
;()obj3.DisplayValues

;()Console.ReadLine
{
{
```

<< Run Code

MyClass defines three constructors: one with two parameters, one with one parameter, and one with no parameters. Each constructor chains to another .using the "this" keyword, passing appropriate arguments

Output

```
:Object 1
Value1: 10, Value2: Hello
```

```
:Object 2
Value1: 20, Value2: default

:Object 3
Value1: 0, Value2: default
```

?#What do you understand by properties in C .46

Properties are members that provide a flexible mechanism to read, write, or compute the values of private fields. By using properties we can access private fields. A property is a return type function/method with one parameter or without a parameter. These are always public data members. It uses .methods to access and assign values to private fields called accessors

#Read More: Properties in C

?#What is the use of the IDisposable interface in C .47

The primary use of the IDisposable interface is to clean up unmanaged resources i.e. database connections, sockets, etc. It defines a single method Dispose() responsible for releasing these resources. The Dispose() method is called explicitly by the consumer of an object or implicitly by the garbage .collector when the object is being finalized

How do you catch multiple exceptions at once in .48 ?#C

.#You can catch multiple exceptions using condition statements in C

Is it possible to achieve method extension using an .49 ?Interface

.Yes, it is possible to achieve method extension using an Interface

Read More: C# Extension Method

.#Explain partial class in C .50

A partial class is only used to split the definition of a class into two or more classes in the same source code file or more than one source file. You can create a class definition in multiple files but it will be compiled as one class at run time and also when you create an instance of this class. So, you can .access all the methods from all source files with the same object

Partial Class can be created in the same namespace and it is not allowed to create a partial class in a different namespace. You can use the “partial” keyword with all the class names that you want to bind together with the same .name of the class in the same namespace

```
;using System

File 1: MyClass_Part1.cs //
public partial class MyClass
}
()public void Method1
}
;("Console.WriteLine("Method1 from File 1
{
{

File 2: MyClass_Part2.cs //
public partial class MyClass
}
()public void Method2
}
;("Console.WriteLine("Method2 from File 2
{
{

Main Program //
class Program
}
(static void Main(string[] args
}
Create an instance of MyClass //
;()MyClass obj = new MyClass

Call methods from both parts of the partial class //
obj.Method1(); // Output: Method1 from File 1
obj.Method2(); // Output: Method2 from File 2
```

```
;()Console.ReadLine
{
{
```

<< Run Code

In the above code, we defined a partial class MyClass across two source files (MyClass_Part1.cs and MyClass_Part2.cs). Each source file contains a portion of the class definition, with different methods (Method1 in .(MyClass_Part1.cs and Method2 in MyClass_Part2.cs

Output

```
Method1 from File 1
Method2 from File 2
```