OOP   
LIST OUTLINE QUESTIONS  
**1. What is Object-Oriented Programming (OOP)?**

**2. What is a class?**

**3. What is an object?**

**4. What is inheritance?**

**5. Explain the different types of inheritance.**

**6. What is encapsulation?**

**7. What is polymorphism?**

**8. What is abstraction?**

**9. What is the difference between method overloading and method overriding?**

**10. What are abstract classes?**

**11. What is an interface?**

**12. What is the "this" keyword?**

**13. What is the "super" keyword?**

**14. What is method hiding?**

**15. What are access modifiers, and what are their purposes?**

**16. Explain the "final" keyword.**

**17. What is a constructor?**

**18. What is a destructor?**

**19. Explain the concept of static members**

**20. What is the "instanceof" operator used for?**

**21. What is a constructor chaining?**

**22. What is the difference between composition and inheritance**

**23. What is method visibility, and how is it controlled?**

**24. Explain the concept of dynamic method dispatch.**

**25. What is the difference between shallow copy and deep copy?**

**26. What are virtual functions (or virtual methods)? 27. What is the Liskov Substitution Principle (LSP)?**

**Bonus Coding Questions to implement what we've studied:**

**Create a class "BankAccount" with attributes account number and balance. Implement methods to deposit and withdraw funds, and a display method to show the account details.**

**Create a base class "Shape" with methods to calculate the area and perimeter (pure virtual). Implement derived classes "Rectangle" and "Circle" that inherit from "Shape" and provide their own area and perimeter calculations.**

**Create a class "Person" with a static member variable "count" that keeps track of the number of instances created.**

**Create a class "Employee" with attributes name and salary. Implement overloaded operators + and - to combine and compare employees based on their salaries.**

**Create a class "Time" with attributes hours and minutes. Implement the << operator to display time in the format "hh:mm".**

**1. What is Object-Oriented Programming (OOP)?**

OOP stands for Object-Oriented Programming.

Procedural Programing is about writing procedures or method that perform operations on the data, while object-oriented programming is about creating object that contain both data and method.

* OOP is faster and easier to execute
* OOP provides a clear structure for the programs
* OOP helps to keep the C# code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
* OOP makes it possible to create full reusable applications with less code and shorter development time

**2. What is a class?**

**A class is like a blueprint** that defines how objects will look and behave.  
It groups together **data (attributes)** and **functions (methods)** that work on that data.

**3. What is an object?**

An object is an instance of a class. It represents a real-world entity and can hold its own state (attributes) and behavior (methods) defined by the class.

**4. What is inheritance?**

**Inheritance** means one class can use the properties and methods of another class.  
It helps in **reusing code** and shows an **“is-a” relationship** (e.g., Dog is an Animal).

* **Derived Class** (child) - the class that inherits from another class
* **Base Class** (parent) - the class being inherited from

**5. Explain the different types of inheritance.**

* Single Inheritance (One class inherits from one parent class.)

Code:

using System;

// Parent class (Base class)

class Animal

{

public void Eat()

{

Console.WriteLine("This animal eats food.");

}

}

// Child class (Derived class) inherits from Animal

class Dog : Animal

{

public void Bark()

{

Console.WriteLine("The dog barks.");

}

}

class Program

{

static void Main(string[] args)

{

Dog d = new Dog();

d.Eat(); // Inherited from Animal

d.Bark(); // Own method

}

}

Output:

This animal eats food.

The dog barks.

* Multilevel Inheritance (A class inherits from another class, which itself inherits from another.)

using System;

// Base class

class Animal

{

public void Eat()

{

Console.WriteLine("This animal eats food.");

}

}

// Derived class 1 (inherits from Animal)

class Dog : Animal

{

public void Bark()

{

Console.WriteLine("The dog barks.");

}

}

// Derived class 2 (inherits from Dog)

class Puppy : Dog

{

public void Weep()

{

Console.WriteLine("The puppy is weeping.");

}

}

class Program

{

static void Main(string[] args)

{

Puppy p = new Puppy();

// Methods from all classes in the chain

p.Eat(); // From Animal

p.Bark(); // From Dog

p.Weep(); // From Puppy

}

}

Output:  
This animal eats food.

The dog barks.

The puppy is weeping.

* Hierarchical Inheritance (Many classes inherit from the same parent class.)

using System;

// Base class

class Animal

{

public void Eat()

{

Console.WriteLine("This animal eats food.");

}

}

// Derived class 1

class Dog : Animal

{

public void Bark()

{

Console.WriteLine("The dog barks.");

}

}

// Derived class 2

class Cat : Animal

{

public void Meow()

{

Console.WriteLine("The cat meows.");

}

}

class Program

{

static void Main(string[] args)

{

Dog d = new Dog();

d.Eat(); // From Animal

d.Bark(); // From Dog

Cat c = new Cat();

c.Eat(); // From Animal

c.Meow(); // From Cat

}

}

**Output:**

This animal eats food.

The dog barks.

This animal eats food.

The cat meows.

**6. What is encapsulation?**

**Encapsulation** means **wrapping data (fields/variables) and methods (functions)** into a single unit (class).  
In C#, it is mainly done using **access modifiers** (private, public, protected) so that data is **hidden** and accessed only through methods or properties.

Code:

using System;

class BankAccount

{

// Private field (data is hidden)

private double balance;

// Public method to set value (write access)

public void Deposit(double amount)

{

if (amount > 0)

{

balance += amount;

Console.WriteLine("Deposited: " + amount);

}

else

{

Console.WriteLine("Invalid deposit amount!");

}

}

// Public method to get value (read access)

public double GetBalance()

{

return balance;

}

}

class Program

{

static void Main(string[] args)

{

BankAccount acc = new BankAccount();

acc.Deposit(500); // Allowed via method

Console.WriteLine("Current Balance: " + acc.GetBalance());

// Not allowed: acc.balance = 1000; // Error (balance is private)

}

}

**Output:**

Deposited: 500

Current Balance: 500

### Encapsulation with Property

### using System;

### class Student

### {

### // private field

### private string name;

### // public property (getter & setter)

### public string Name

### {

### get { return name; } // getter

### set { name = value; } // setter

### }

### }

### class Program

### {

### static void Main(string[] args)

### {

### Student s = new Student();

### // Using setter

### s.Name = "Abrar";

### // Using getter

### Console.WriteLine("Student Name: " + s.Name);

### }

### }

### Output: Student Name: Abrar

**7. What is polymorphism?**

**Polymorphism** means **"many forms"**.  
 In OOP, it allows the **same method name** to behave **differently** depending on the object or how it is used.

Example in real life:

* A **person** can be a **teacher at school**, a **customer in a shop**, or a **player in a game** → one person, many roles.

### Types of Polymorphism in C#

1. **Compile-time Polymorphism (Method Overloading) →** Same method name but different parameters.
2. **Runtime Polymorphism (Method Overriding) →** Same method in parent and child, behavior depends on object.

Example 1 – Method Overloading (Compile-time Polymorphism)

using System;

class Calculator

{

public int Add(int a, int b)

{

return a + b;

}

public double Add(double a, double b)

{

return a + b;

}

}

class Program

{

static void Main(string[] args)

{

Calculator calc = new Calculator();

Console.WriteLine(calc.Add(2, 3)); // Calls int version

Console.WriteLine(calc.Add(2.5, 3.5)); // Calls double version

}

}

OUTPUT  
5

6

Example 2 – Method Overriding (Runtime Polymorphism)

using System;

class Animal

{

public virtual void Sound()

{

Console.WriteLine("Animal makes a sound");

}

}

class Dog : Animal

{

public override void Sound()

{

Console.WriteLine("Dog barks");

}

}

class Program

{

static void Main(string[] args)

{

Animal a = new Dog(); // Parent reference, Child object

a.Sound(); // Calls Dog's version (runtime polymorphism)

}

}

Output:

Dog barks

**8. What is abstraction?**

using System;

abstract class Animal

{

// Abstract method (no body)

public abstract void Sound();

// Normal method

public void Sleep()

{

Console.WriteLine("Animal is sleeping");

}

}

class Dog : Animal

{

public override void Sound()

{

Console.WriteLine("Dog barks");

}

}

class Program

{

static void Main(string[] args)

{

Animal a = new Dog();

a.Sound(); // Must be implemented by Dog

a.Sleep(); // From base class

}

}

**OUTPUT:**

Dog barks

Animal is sleeping

**9. What is the difference between method overloading and method overriding?**

Method overloading occurs when a class has multiple methods with the same name but different parameter lists. The appropriate method is determined at compile time based on the method signature. Method overriding occurs when a subclass provides a specific implementation for a method that is already defined in its superclass. The decision on which method to call is made at runtime based on the actual object type.

**10. What are abstract classes?**

An abstract class is a class that cannot be instantiated directly and may contain one or more abstract methods. Abstract methods are declared without implementation and must be implemented by subclasses.

**11. What is an interface?**

An interface is a blueprint that defines a set of methods that a class must implement. It provides a way to achieve multiple inheritance in languages that do not support it directly.

**12. What is the "this" keyword?**

The "this" keyword refers to the current object instance within a class. It is used to access instance variables or call instance methods of the class.

**13. What is the "super" keyword?**

The "super" keyword is used to call the superclass's constructor or refer to the superclass's methods or variables from a subclass.

**14. What is method hiding?**

Method hiding occurs when a subclass defines a static method with the same name and signature as a static method in the superclass. The subclass's method hides the superclass's method, and the method invoked depends on the reference type rather than the object type.

**15. What are access modifiers, and what are their purposes?**

Access modifiers define the visibility and accessibility of class members (attributes, methods, constructors). The main access modifiers are public, private, protected, and package-private/default. They control how members can be accessed from other classes and packages.

**16. Explain the "final" keyword.**

n OOP, the "final" keyword can be applied to a class, method, or variable. A final class cannot be subclassed, a final method cannot be overridden, and a final variable cannot be reassigned once initialized.

**17. What is a constructor?**

A constructor is a special method that is automatically called when an object of a class is created. It is used to initialize the object's state and perform setup tasks.

**18. What is a destructor?**

A destructor is a special method that is called when an object is destroyed or goes out of scope. It is used to release resources and perform cleanup operations.

**19. Explain the concept of static members**

Static members (attributes or methods) belong to the class rather than individual objects. They are shared among all instances of the class and can be accessed using the class name.

**20. What is the "instanceof" operator used for?**

The "instanceof" operato is used to test whether an object is an instance of a particular class or implements a specific interface. It checks the object's type at runtime and returns a boolean value indicating whether the object is an instance of the specified class or interface.

**21. What is a constructor chaining?**

Constructor chaining is the process of calling one constructor from another within the same class or between base and derived classes. It allows constructors to reuse code and perform common initialization tasks.

**22. What is the difference between composition and inheritance**

Composition represents a "has-a" relationship, where a class contains objects of other classes as its members. Inheritance represents an "is-a" relationship, where a subclass inherits properties and behaviors from a superclass.

**Bonus Coding Questions to implement what we've studied:**

**Create a class "BankAccount" with attributes account number and balance. Implement methods to deposit and withdraw funds, and a display method to show the account details.**

**using System;**

public class HelloWorld

{

public static void Main(string[] args)

{

BankAccount ac = new BankAccount(101, 5000);

ac.Display();

ac.Deposit(2000);

ac.Withdraw(3000);

ac.Display();

}

}

public class BankAccount

{

private int accountNumber;

private double balance;

public BankAccount(int accNum, double initialBalance)

{

accountNumber = accNum;

balance = initialBalance;

}

public void Deposit(double amount)

{

balance += amount;

Console.WriteLine("Deposited: " + amount);

}

public void Withdraw(double amount)

{

if (amount <= balance)

{

balance -= amount;

Console.WriteLine("Withdrawn: " + amount);

}

else

{

Console.WriteLine("Insufficient Balance!");

}

}

public void Display()

{

Console.WriteLine("\nAccount Number: " + accountNumber);

Console.WriteLine("Balance: " + balance);

}

}

**OUTPUT:**

Account Number: 101

Balance: 5000

Deposited: 2000

Withdrawn: 3000

Account Number: 101

Balance: 4000

**Create a base class "Shape" with methods to calculate the area and perimeter . Implement derived classes "Rectangle" and "Circle" that inherit from "Shape" and provide their own area and perimeter calculations.**

using System;

public class HelloWorld

{

public static void Main(string[] args)

{

Shape rect = new Rectangle(10, 5);

rect.Display();

Shape circle = new Circle(7);

circle.Display();

}

}

public abstract class Shape

{

public abstract double Area();

public abstract double Perimeter();

public void Display()

{

Console.WriteLine("Area : " + Area());

Console.WriteLine("Perimeter : " + Perimeter());

Console.WriteLine();

}

}

public class Rectangle : Shape

{

private double length;

private double width;

public Rectangle(double l, double w)

{

length = l;

width = w;

}

public override double Area()

{

return length \* width;

}

public override double Perimeter()

{

return 2 \* (length + width);

}

}

public class Circle : Shape

{

private double radius;

public Circle(double r)

{

radius = r;

}

public override double Area()

{

return Math.PI \* radius \* radius;

}

public override double Perimeter()

{

return 2 \* Math.PI \* radius;

}

}

OUTPUT:  
Area : 50

Perimeter : 30

Area : 153.9380400259

Perimeter : 43.9822971502571