1. Why do we need OOP?



1. What are the important pillars in OOP?



1. What is a Class and Object?
2. Abstraction VS Encapsulation?
3. Explain Inheritance?



1. Explain Virtual Keyword?
2. What is Overriding?



1. Explain Overloading?
2. Overloading VS Overriding?
3. What is Polymorphism?



1. Can Polymorphism work without Inheritance?
2. Explain Static VS Dynamic Polymorphism?
3. Explain Operator Overloading?
4. How to do Custom Operator Overloading?
5. Why do we need Abstract Classes?
6. Are Abstract methods Virtual?
7. Can we create an instance of Abstract Classes?
8. Is it compulsory to implement Abstract Methods?
9. Why simple base class cannot replace Abstract Classes?
10. Explain Interface and Why do we need it?
11. Can we write logic in Interface?
12. Can we define methods as private in Interface?
13. To change Interface what's the best practice?
14. Explain Multiple Inheritance in Interfaces?
15. Explain Interface Segregation Principle?
16. Can we create instance of Interface?
17. Can we do multiple inheritance of Interface 2

Questions with answers:

**1. What is Object-Oriented Programming (OOP)?**  
Answer: Object-Oriented Programming is a programming paradigm that organizes code into objects, which are instances of classes. It emphasizes concepts like inheritance, encapsulation, polymorphism, and abstraction to improve code reusability and maintainability.



**2. What is a class?**  
Answer: A class is a blueprint or template that defines the structure and behavior of objects. It serves as a user-defined data type and encapsulates data (attributes) and methods (functions) that operate on that data.



**3. What is an object?**  
Answer: 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?**  
Answer: Inheritance is a mechanism where one class (subclass or derived class) acquires properties and behaviors from another class (superclass or base class). It promotes code reusability and establishes an "is-a" relationship between classes.



**5. Explain the different types of inheritance.**  
Answer: The main types of inheritance are:  
Single Inheritance: A subclass inherits from a single superclass.  
Multiple Inheritance: A subclass inherits from multiple superclasses (supported by some languages like C++).  
Multilevel Inheritance: A class is derived from another class, which, in turn, is derived from a base class.  
Hierarchical Inheritance: Multiple subclasses inherit from a single superclass.



**6. What is encapsulation?**  
Answer: Encapsulation is the bundling of data (attributes) and methods (functions) that operate on the data within a single unit (i.e., the class). It restricts access to the data by providing public interfaces (public methods) to interact with the object's state. Encapsulation helps in data hiding and protecting the internal state of an object.



**7. What is polymorphism?**  
Answer: Polymorphism allows objects of different classes to be treated as objects of a common superclass. It enables a single interface to represent multiple types of objects. Polymorphism is achieved through method overloading (compile-time polymorphism) and method overriding (runtime polymorphism).



**8. What is abstraction?**  
Answer: Abstraction is the process of hiding the implementation details of an object and exposing only the relevant features or behavior. It allows developers to work with high-level concepts and ignore low-level implementation complexities.



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

**23. What is method visibility, and how is it controlled?**  
Answer: Method visibility refers to the accessibility of methods from other classes. It is controlled by access modifiers like "public," "private," "protected," and "package-private/default."

**24. Explain the concept of dynamic method dispatch.**  
Answer: Dynamic method dispatch is the mechanism that enables a method call on an object to be resolved at runtime based on the actual type of the object, rather than the reference type.

**25. What is the difference between shallow copy and deep copy?**  
Answer: Shallow copy creates a new object but copies only the references of the original object's members. Deep copy, on the other hand, creates a new object and copies all the content, including recursively copying nested objects, if any.

**26. What are virtual functions (or virtual methods)?**  
Answer: Virtual functions allow late binding or runtime polymorphism, enabling a subclass to provide its implementation for a method declared in a base class.

**27. What is the Liskov Substitution Principle (LSP)?**  
Answer: The Liskov Substitution Principle states that objects of a superclass should be replaceable with objects of its subclasses without affecting the correctness of the program. Subclasses must maintain the behavior expected by the superclass.

Bonus Coding Questions to implement what we've studied:

*Note: I code in C++ only, @aniket123456789 has contributed all the codes in Java*

**1. 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.**

Java:

public class BankAccount {

private String accountNumber;

private double balance;

public BankAccount(String accountNumber, double initialBalance) {

this.accountNumber = accountNumber;

this.balance = initialBalance;

}

public void deposit(double amount) {

balance += amount;

}

public void withdraw(double amount) {

if (balance >= amount) {

balance -= amount;

} else {

System.out.println("Insufficient balance!");

}

}

public void display() {

System.out.println("Account Number: " + accountNumber);

System.out.println("Balance: " + balance);

}

public static void main(String[] args) {

BankAccount account = new BankAccount("123456789", 1000);

account.deposit(500);

account.withdraw(200);

account.display();

}

}

**2. 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.**

Java:

abstract class Shape {

public abstract double area();

public abstract double perimeter();

}

class Rectangle extends Shape {

private double length;

private double width;

public Rectangle(double l, double w) {

length = l;

width = w;

}

@Override

public double area() {

return length \* width;

}

@Override

public double perimeter() {

return 2 \* (length + width);

}

}

class Circle extends Shape {

private double radius;

public Circle(double r) {

radius = r;

}

@Override

public double area() {

return Math.PI \* radius \* radius;

}

@Override

public double perimeter() {

return 2 \* Math.PI \* radius;

}

}

public class Main {

public static void main(String[] args) {

Shape shape1 = new Rectangle(5, 3);

Shape shape2 = new Circle(4);

System.out.println("Rectangle Area: " + shape1.area());

System.out.println("Rectangle Perimeter: " + shape1.perimeter());

System.out.println("Circle Area: " + shape2.area());

System.out.println("Circle Perimeter: " + shape2.perimeter());

}

}

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

Java:

public class Person {

private String name;

private static int count = 0;

public Person(String n) {

name = n;

count++;

}

public static int getCount() {

return count;

}

public String getName() {

return name;

}

public static void main(String[] args) {

Person person1 = new Person("Alice");

Person person2 = new Person("Bob");

System.out.println("Total Persons: " + Person.getCount());

System.out.println(person1.getName());

System.out.println(person2.getName());

}

}

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

Java:

public class Employee {

private String name;

private double salary;

public Employee(String n, double s) {

name = n;

salary = s;

}

public double getSalary() {

return salary;

}

public boolean lessThan(Employee other) {

return salary < other.salary;

}

public boolean greaterThan(Employee other) {

return salary > other.salary;

}

public Employee add(Employee other) {

return new Employee("Combined", salary + other.salary);

}

public Employee subtract(Employee other) {

return new Employee("Difference", salary - other.salary);

}

public static void main(String[] args) {

Employee emp1 = new Employee("Alice", 5000);

Employee emp2 = new Employee("Bob", 6000);

if (emp1.lessThan(emp2)) {

System.out.println("Bob has a higher salary than Alice.");

} else {

System.out.println("Alice has a higher salary than Bob.");

}

Employee combined = emp1.add(emp2);

System.out.println("Combined Salary: " + combined.getSalary());

Employee difference = emp1.subtract(emp2);

System.out.println("Salary Difference: " + difference.getSalary());

}

}

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

Java:

public class Time {

private int hours;

private int minutes;

public Time(int h, int m) {

hours = h;

minutes = m;

}

@Override

public String toString() {

return hours + ":" + (minutes < 10 ? "0" : "") + minutes;

}

public static void main(String[] args) {

Time time = new Time(14, 30);

System.out.println("Time: " + time);

}

}