

Last Updated: Sep 29, 2024

Medium

Java OOPs Interview Questions



Author
SHIVANGI MALL

Share 3 upvotes

Zomato Case study : How data analysis is used in a food delivery business

Join Masterclass for free

Speaker
Megna Roy



11 Dec, 2024 @ 07:00 PM

Introduction

Object-Oriented Programming (OOP) is a fundamental concept in Java and one of the most important topics for Java developers to master. Understanding OOP principles such as encapsulation, inheritance, polymorphism, and abstraction is crucial for solving complex software design problems. For candidates preparing for Java OOPs interview questions, it's essential to have a deep understanding of these concepts, their implementation in Java, and how to apply them to real-world scenarios.

So, let's get started and help you ace your interview with the curated list of Java OOps Interview Questions.



Basic Java OOPs Interview Questions

1. What is the difference between OOP and SOP?

Answer: The following are the differences between OOP and SOP:-

OOP	SOP
Stands for object-oriented Programming.	It means Structured Programming.
It depends upon the creation of an object in our program, which is a collection of data and functions.	It depends upon breaking down of program into smaller programs and functions.
The quality of programs is increased as well as the productivity of system analysis.	Increases the quality and development time of computer programs.
Function overloading is possible in OOP language.	Function overloading is not possible in OOP language.
OOPs principles such as abstraction, encapsulation, etc are followed.	There is no concepts of abstraction, encapsulation involved in Structured programming.

2. What is the definition of Object-Oriented Programming (OOP)?

Answer: Object-Oriented Programming (OOPs) is a programming style that emphasizes objects above functions and procedures. Classes are made up of individual items. OOPs incorporates real-world concepts such as **inheritance**, **polymorphism**, and **concealing** into programming. It also allows data and code to be linked together.

3. What are the benefits of using OOPs?

Answer: OOPs allow for greater clarity in programming, making it easier to solve complicated problems.

- Through inheritance, code can be reused, decreasing repetition.
- Encapsulation binds data and code together.
- Because OOPs enable data hiding, private information is kept hidden.
- To make it easier to tackle problems can be broken down into smaller chunks.
- Polymorphism allows the program to be more flexible by allowing entities to have numerous forms.

4. Name any seven of the most widely used OOP languages.

There are several OOP languages, but the following are the most popular:

- Python
- Java
- Go
- Dart
- C++
- C#
- Ruby

5. What are the four key characteristics of OOPs?

Answer: The following are the four [Characteristics of OOPS](#):

- Inheritance
- Encapsulation
- Polymorphism
- Data Abstraction

6. What are the drawbacks of using OOPs?

Answer: Some of the drawbacks of OOPs are:-

- Usually, it isn't appropriate for minor issues.
- Intensive testing is required.
- It takes a longer time to fix the problem.
- It requires proper planning.

7. What are the benefits and drawbacks of OOP?

Answer: Below is the some of the advantages and disadvantages of OOPs:-

Advantages of OOP

- It follows a bottom-up approach.
- It models the real world well.
- It makes code reusability possible.
- Using abstraction, the user is not exposed to unnecessary data.
- OOP requires designers to go through a long and extensive design phase, which yields a better design with fewer faults.
- Break down a large problem into smaller chunks.
- Programmers are able to achieve their objectives more quickly.
- Reduces the amount of complexity.
- Code may be easily redesigned and extended without affecting other features.

Disadvantages of OOP

- Proper planning is required.
- Program design is tricky.
- The programmer should be well skilled.
- Classes tend to be overly generalized.

8. What is an object?

Answer: An object is a real-world entity that is the fundamental unit of OOPs, such as a chair, a cat, or a dog. Objects have a variety of states, properties, and behaviors. Objects interact with each other through defined methods, enabling the modeling of complex systems by representing entities and their relationships within software applications.

9. What exactly is a class?

Answer: A class is an object's blueprint or template. It's a data type that's been specified by the user. Variables, constants, member functions, and other functionality are defined within a class. At runtime, it does not use any memory. It's worth noting that classes aren't considered data structures. It's a logical thing.

10. What are the differences between class and object?

Answer: Some of the differences between class and object are given below

Class	Object
Class is a logical entity.	Object is a real-world entity.
Class is conceptual.	Object is real.
Class binds data and methods together into a single unit.	Object is just like a variable of a class.
Class does not occupy space in the memory.	Object occupies space in the memory.

[Learn](#)[Contests & Events](#)[Interview prep](#)[Practice](#)[Resources](#)[Login](#)

A class can exist without any object.	Objects cannot exist without the class.
---------------------------------------	---

11. What is the difference between class and structure?

Answer: Some of the differences between class and structure are given below:-

Class	Structure
Class is a group of common object that shares common properties.	The structure is a collection of different data types.
Its members are private by default.	Its members are public by default.
The keyword class defines a class.	The keyword struct defines a structure.

[s Interview Que...](#)[ie difference bet...](#)[ie definition of ...](#)[the benefits of ...](#)[seven of the m...](#)[the four key cha...](#)[the drawbacks ...](#)[the benefits an...](#)[n object?](#)[ctly is a class?](#)[Live mast](#)

Am
by /
Dat
Am

13 Dec, 20

19+ registers

Why is a class...

re the differenc...

the difference ...

the definition o...

re various types...

re the limitation...

the difference ...

hybrid inherita...

hierarchical inh...

a superclass?

re the rules for c...

o you mean by ...

OOPs Interview...

the definition of ...

polymorphism?

method overridi...

es it mean whe...

the definition of ...

virtual functio...

a constructor?

e different types...

ch memory does...

Class	Structure
A class instance is referred to as an object.	A structure variable is an instance of a structure.

12. What is the definition of inheritance?

Answer: OOPs has a feature called inheritance, which allows classes to inherit common properties from other classes. If a class called vehicle exists, other classes like 'car,' 'bike,' and so on can inherit common characteristics from it. This characteristic assists in the removal of extraneous code, resulting in smaller overall code size.

13. What are various types of inheritance?

Answer: Various types of inheritance are:-

- Single inheritance
- Multiple inheritances
- Multilevel inheritance
- Hierarchical inheritance
- Hybrid inheritance

14. What are the limitations of Inheritance?

Answer: Yes, greater powers bring greater challenges. Inheritance is a great feature in OOPs, but it comes with some drawbacks. Inheritance takes longer to process since it must go through numerous classes in order to be implemented. Inheritance also involves two classes: a base class and a child class, which are extremely closely related. As a result, if changes are required, they may need to be made in both classes. Inheritance may be difficult to implement as well. As a result, if not implemented appropriately, this could result in unexpected errors or inaccurate outputs.

15. What is the difference between multiple and multilevel inheritance?

Answer: The following are the differences between multiple and multilevel inheritance:-

Multiple Inheritance	Multilevel Inheritance
When a class inherits from more than one base class, multiple inheritance is used.	Multilevel inheritance means a class inherits from another class which itself is a subclass of some other base class.
Example: A class defining a child inherits from two base classes, Mother and Father.	Example: A class describing a sports car will inherits from a base class Car which in turn inherits another class Vehicle.

16. What is hybrid inheritance?

Answer: The term "hybrid" refers to a combination of two or more words. Hybrid inheritance occurs when two or more forms of inheritance are combined. Multiple and multi-level inheritance are combined in hybrid inheritance.

17. What is hierarchical inheritance, and how does it work?

Answer: Hierarchical inheritance is the inheritance of several subclasses from a single base class. The vehicle class, for example, can have subclasses such as 'car,' 'bike,' and so on.

18. What is a superclass?

Answer: A superclass, often known as a base class, is a class that serves as a parent to other classes sharing common attributes and methods. Inheritance allows subclasses to inherit these shared characteristics, promoting code reuse and modular design. The Vehicle class, for example, is a superclass of the Car class.

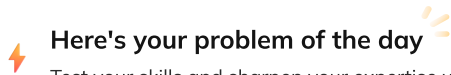
19. What are the rules for creating a constructor?

Answer: Following are some of the rules for creating a constructor:-

- It isn't allowed to have a return type.
- It must be named the same as the Class.
- It cannot be marked as static.
- It isn't possible to classify it as abstract.
- It's not possible to override it.
- It cannot be final.

20. What do you mean by abstraction?

Answer: The OOPS idea of abstraction is used to build the structure of real-world objects. Only the general states and behaviors are captured during this construction, leaving more precise states and behaviors to the implementers.



Test your skills and sharpen your expertise with today's problem

Skill covered: **Programming**

What does DISTINCT do in SQL?

☐ Returns all rows

☐ Filters duplicate rows

☐ Sorts the result

☐ Joins tables

Submit

[Choose another skill to practice](#)

Advanced Java OOPs Interview Questions

21. What is the definition of encapsulation?

Answer: Encapsulation is an OOPS concept for creating and defining an object's permissions and restrictions, as well as the variables and methods that make up the object. Making a class's member variables private and offering public getter and setter methods is a straightforward way to demonstrate the concept. Access level modifiers in Java are divided into four categories: public, protected, no modifier, and private.

22. What is polymorphism?

Answer: The appearance of something in multiple forms is known as polymorphism. Polymorphic reference variables, polymorphic methods, polymorphic return types, and polymorphic argument types are all supported by Java.

23. What is method overriding, and how does it work?

Answer: Method overriding is an OOPs feature that allows a child class or subclass to rewrite methods in the base or parent class. The overridden method has the same name as the original, as

well as the same signature, which refers to the arguments supplied and the return type.

24. What does it mean when an operator is overloaded?

Answer: Operator overloading refers to the usage of user-defined types to implement operators based on the arguments provided to them. This feature allows you to extend the functionality of operators beyond their built-in uses, enhancing the expressiveness and flexibility of your code while adhering to the principles of object-oriented programming.

25. What is the definition of an abstract class?

Answer: A class with abstract methods is known as an abstract class. Although these methods are declared, they are not defined. If these methods are to be utilized in a subclass, they must be specified in that subclass solely.

26. What are virtual functions?

Answer: Virtual functions are present in the parent class and are overridden by the subclass. These functions are used to achieve runtime polymorphism. They allow a base class to define a method as virtual, indicating that derived classes may override it. When an object of a derived class is accessed through a pointer or reference of the base class, the appropriate overridden function in the derived class is invoked based on the actual object type.

27. What is a constructor?

Answer: Constructor is a type of method that is used to initialize objects of a class and has the same name as the class. They play a crucial role in ensuring the proper instantiation of objects. They can enforce certain constraints or actions during object creation, contributing to the integrity and consistency of the class's instances within a program.

28. Name the different types of constructors.

Answer: The different types of constructors are:-

- Default constructor
- Parameterized constructor
- Copy constructor
- Static constructor
- Private constructor

29. How much memory does a class occupy?

Answer: No memory is used by classes. Class are nothing more than a blueprint from which objects are produced. When objects are created now, the class members and methods are really initialized, which consumes memory.

30. What is an exception?

Answer: An exception is a type of notice that causes a program's usual execution to be interrupted. Exceptions give an error a pattern and pass it on to the exception handler to be resolved. The program's state is saved as soon as an exception is raised.

31. What is the definition of exception handling?

Answer: Exception handling is a crucial notion in Object-Oriented Programming for dealing with errors. Errors can be thrown and caught, and an exception handler implements a centralized mechanism to resolve them.

32. How does Java support multiple inheritance without using various inheritance?

Java achieves multiple inheritance through interfaces. Allowing a class to implement various interfaces can inherit diverse sets of abstract methods and contracts, avoiding the complexities of traditional multiple inheritance. This approach promotes clear separation between interface and class inheritance, enhancing maintainability and adaptability in the codebase.

33. What is the purpose of the final keyword in Java, and how does it

impact OOP principles?

The final keyword restricts modifications on classes, methods, and variables. Applied to a class, it prevents inheritance, ensuring the class cannot be extended. When applied to a method, it prevents overriding. While final contributes to achieving immutability and safeguarding against undesired changes, excessive use may curtail the extensibility and flexibility intrinsic to OOP's objectives.

34. Explain the concept of encapsulation and provide an example in Java.

Encapsulation embodies an OOP principle that consolidates data (attributes) and methods (behavior) relevant to the data within a single entity (class). Controlled access to data is achieved through methods facilitating data protection and abstraction. For example, using private instance variables and public getter and setter methods in a Person class ensures disciplined access and manipulation of the person's attributes.

35. Explain the concept of composition in Java OOPs.

Composition entails the creation of complex objects by assembling simpler components or objects. This is achieved by incorporating instances of other classes as member variables within a class. This methodology fosters code reusability and modular design, allowing components to be substituted or added without disrupting the overall structure. For instance, a Car class could comprise instances of Engine, Wheels, and Seat classes, each handling distinct functionalities.

36. What is the role of the super keyword in Java, and how does it relate to inheritance?

The super keyword references the superclass of a derived class. It is frequently employed to invoke overridden superclass methods, facilitating an extension of behavior inherited from the superclass. This mechanism supports code reuse, permitting the utilization of existing functionalities while introducing or modifying behaviors specific to the subclass.

37. What is a singleton design pattern, and how does it ensure the creation of only one instance?

The singleton design pattern ensures the existence of only one instance of a class while offering a universal access point to that instance. This is accomplished by making the class constructor private and presenting a static method to access the single instance. The instance is created upon the first request, and subsequent requests retrieve the existing instance. This pattern often manages singular resources such as configurations or database connections.

38. How does Java implement runtime polymorphism, and why is it essential in OOP?

Java employs runtime polymorphism through method overriding and dynamic method dispatch. By overriding a method from a superclass in a subclass, the actual method called at runtime is based on the object's type. This dynamic method dispatch facilitates adaptable and extensible code, enabling diverse subclasses to offer specialized implementations while adhering to a shared interface.

39. How does Java support method overloading and method overriding, and what is the difference?

Java supports method overloading by enabling a class to define multiple methods sharing the same name but distinct parameter lists. The compiler distinguishes these methods based on parameter number or type. Method overriding, on the other hand, transpires when a subclass supplies a distinct implementation for a method already defined in its superclass. Overriding, marked by the @Override annotation, empowers dynamic method dispatch, a cornerstone of achieving runtime polymorphism.

40. Explain the concept of abstract classes and interfaces in Java, and when would you use each?

Abstract classes are inimitable and intended to be extended by subclasses. They encompass both abstract (unimplemented) and concrete (implemented) methods. Interfaces, conversely, outline contractual obligations for classes to fulfill by offering an assortment of abstract methods to be overridden. Abstract classes suit scenarios necessitating a shared foundation with default

implementations, whereas interfaces establish a uniform contract for multiple classes, fostering structured code organization and enabling various inheritances.

Java OOPs Coding Problem Questions

41. Write a Java program to demonstrate encapsulation by creating a Person class with private variables and public getter/setter methods.

Java ▲

```
class Person {
    private String name;
    private int age;

    // Getter and Setter methods
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}

public class Main {
    public static void main(String[] args) {
        Person person = new Person();
        person.setName("Rohit");
        person.setAge(30);

        System.out.println("Name: " + person.getName());
        System.out.println("Age: " + person.getAge());
    }
}
```

 You can also try this code with [Online Java Compiler](#)

[Run Code](#)

Output:

```
Name: Rohit
Age: 30
```

42. Write a Java program to demonstrate inheritance by creating a Vehicle superclass and a Car subclass that inherits from Vehicle.

Java ▲

```
class Vehicle {
    public void startEngine() {
        System.out.println("Engine started");
    }
}

class Car extends Vehicle {
    public void drive() {
        System.out.println("Car is driving");
    }
}
```



```

}

public class Main {
    public static void main(String[] args) {
        Car car = new Car();
        car.startEngine(); // Inherited method
        car.drive();
    }
}

```

 You can also try this code with [Online Java Compiler](#)

[Run Code](#)

Output:

```

Engine started
Car is driving

```

43. Write a Java program to demonstrate polymorphism by creating a Shape class and overriding its method in subclasses Circle and Rectangle.

Java



```

class Shape {
    public void draw() {
        System.out.println("Drawing a shape");
    }
}

class Circle extends Shape {
    @Override
    public void draw() {
        System.out.println("Drawing a circle");
    }
}

class Rectangle extends Shape {
    @Override
    public void draw() {
        System.out.println("Drawing a rectangle");
    }
}

public class Main {
    public static void main(String[] args) {
        Shape shape1 = new Circle();
        Shape shape2 = new Rectangle();

        shape1.draw(); // Calls Circle's draw method
        shape2.draw(); // Calls Rectangle's draw method
    }
}

```

 You can also try this code with [Online Java Compiler](#)

[Run Code](#)

Output:

```

Drawing a circle
Drawing a rectangle

```

44. Write a Java program to demonstrate abstraction by creating an abstract class Animal and implementing its abstract methods in Dog and Cat classes.

Java



```

abstract class Animal {
    public abstract void sound();
}


class Dog extends Animal {
    public void sound() {
        System.out.println("Dog barks");
    }
}

class Cat extends Animal {
    public void sound() {
        System.out.println("Cat meows");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal dog = new Dog();
        Animal cat = new Cat();

        dog.sound();
        cat.sound();
    }
}

```

 You can also try this code with [Online Java Compiler](#)

[Run Code](#)

Output:

```

Dog barks
Cat meows

```

45. Write a Java program to demonstrate method overloading by creating a Calculator class with multiple add methods.

Java



```

class Calculator {
    // Method to add two integers
    public int add(int a, int b) {
        return a + b;
    }

    // Method to add three integers
    public int add(int a, int b, int c) {
        return a + b + c;
    }

    // Method to add two doubles
    public double add(double a, double b) {
        return a + b;
    }
}

public class Main {
    public static void main(String[] args) {
        Calculator calc = new Calculator();

        System.out.println("Sum of 2 integers: " + calc.add(10, 20));
        System.out.println("Sum of 3 integers: " + calc.add(10, 20, 30));
        System.out.println("Sum of 2 doubles: " + calc.add(10.5, 20.5));
    }
}

```

 You can also try this code with [Online Java Compiler](#)

[Run Code](#)