Basic OOPs Interview Questions and Answers

1 What do you mean by OOPs?

Ans - OOPs stand for object-oriented programming. As the name suggests, this is a pattern of programming that emphasizes more on objects and classes rather than functions. It is not a programming language but a paradigm used by almost all new generations of programming languages like Java, C++, etc.

2 Name some of the programming languages which use the concept of OOPs?

Ans - Nowadays, almost all programming languages use object-oriented concepts.

3 What is the need for Object-oriented programming?

Ans - As we evolve, our problems also become more complex and the code. To solve this problem of writing complex codes, we need object-oriented programming. It allows reusability of code which shortens the code to a greater extent. The primary importance of OOPs is to bind together the data and the functions that operate on them with the help of its essential features.

4 What are the essential features of OOPs?

Ans - There are four main pillars of object-oriented programming

1. Encapsulation 2. Polymorphism 3. Inheritance 4. Data Abstraction

Encapsulation

Encapsulation name suggests, It is the wrapping up of data and methods (functions) that manipulates them in a single unit. It is one of the essential features of OOPs and is achieved with the help of access modifiers.

Polymorphism

Polymorphism in OOPs refers to the objects created that can take many forms in different instances. The Word polymorphism has been made from "poly," which means many, and "morphs," which means forms. It can also be understood as function overloading, as in it, any object or method gets to act differently on different instances.

Inheritance

The capability of a class to derive properties and characteristics from another class is called Inheritance. It is an actual world entity that gets implemented in OOPs.

There are three different modes of Inheritance.

1. Public mode 2. Protected mode 3. Private mode

These modes of Inheritance are used in OOPs according to the requirements.

Inheritance can be further divided into six types

1. Single Inheritance
2. Multiple Inheritance
3. Multilevel Inheritance
4. Hierarchical Inheritance
5. Hybrid Inheritance
6. Multipath Inheritance

Data abstraction

Data abstraction is the concept of object-oriented programming which shows only the necessary attributes of a code and hides all the unnecessary things. It helps in avoiding code duplication and thus increases reusability.

5 Discuss the advantages and disadvantages of OOPs?

Following are some of the advantages of OOPs

Advantages of OOPs	Disadvantages of OOPs
Decreases the complexity of the code	Requires more time to get executed
It allows code reusability rather than rebuilding the same thing again and again.	Programmers need to be highly proficient in handling objects
It enhances the security of the code with the help of data abstraction	It can get a little tricky sometimes
We can easily extend the code without disturbing other functionalities.	Not suitable for all types of problems
It helps in creating a flexible piece of code	
Easy handling and manipulation of code	
It follows a bottom-up approach	

6 What are classes and objects in OOPs?

Ans - A class in object-oriented programming is a user-defined data type that contains a type's variables and methods(functions). A class object can only access these data variables and member functions.

An object in object-oriented programming is nothing but an instance of a class used in accessing the data members and member functions of a class.

7 What are the differences between OOPs and structural programming?

Object-oriented programming	Structural programming
It is used to solve lengthily and complex problems	It is used to solve moderate or easy level problems
Follows a bottom-top approach	Follows a top-bottom approach
Offers reusability of code	Reusability of code is not allowed
It is more secure	It is less secure
It uses the concept of inheritance	It does not use the concept of inheritance
The code is more flexible	Generates less flexible code
Uses data abstraction to hide unnecessary information	Does not hide anything from the user

8. What are the differences between class and object?

Class	Object
Class is a template on which objects are declared.	An object is an instance of the class.
When a class is created, no memory is allocated	Whenever an object is created. Memory is allocated.
It is a logical entity.	It is an actual entity.
It has to be declared only once.	It can be declared multiple times as per the requirements.
A class can exist without an object.	An object cannot exist without a class.

Must Read Characteristics of OOPS

9. What are the differences between class and structure?

Ans -

Class	Structure
We declare a class using the ‘class’ keyword.	We declare a structure using the ‘struct’ keyword.
Members of the class are private by default.	Members of the structure are public by default.
Inheritance is supported.	Inheritance is not supported.
It is of reference type.	It is of a value type.
It can deal with both variables and functions.	It can only deal with variables.

10. Do class and object acquire any memory?

When we build a class, no memory is allocated to it as it is just a blueprint. But when an object of the class is created, memory is allocated accordingly.

We saw some of the fundamental questions the Interviewer can ask in an interview. In the next blog, we will look for some advanced questions on OOPs. Let us now summarize what we learned in this article.

11. Explain the four pillars of OOPs.

Ans - The four pillars of Object-Oriented Programming (OOPs) are:

1. **Encapsulation:** Bundling data (attributes) and methods (functions) that operate on that data within a single unit called a class. This promotes data protection and information hiding.
2. **Abstraction:** Focusing on the essential details of an object while hiding its underlying implementation. This allows users to interact with objects without worrying about the complexities of their internal workings.
3. **Inheritance:** Creating new classes (subclasses) that inherit properties and behaviors from existing classes (superclasses). This promotes code reusability and simplifies the creation of class hierarchies.
4. **Polymorphism:** The ability of objects of different classes to respond to the same method call in different ways. This allows for flexible and dynamic behavior in programs.

These principles work together to create well-structured, maintainable, and reusable code. Encapsulation protects data integrity, abstraction simplifies object interaction, inheritance promotes code reuse, and polymorphism enables flexible method behavior.

12. Describe the use cases for inheritance.

Ans - Inheritance is beneficial when you have a set of existing functionalities (parent class) that can be reused as a foundation for creating new classes (child classes) with specialized features. Here are some common use cases:

- **Creating Class Hierarchies:** Inheritance allows you to establish relationships between classes, where child classes inherit general attributes and methods from a parent class, and then add their own specific functionalities. For example, an Animal class can be a parent class for Dog and Cat child classes, inheriting common attributes like name and methods like eat(), while also having specific methods like bark() for dogs and purr() for cats.
- **Code Reusability:** By inheriting common functionalities, you avoid duplicating code and streamline development.
- **Extending Functionality:** Child classes can add new methods or override inherited methods to provide specialized behavior.

There are different types of inheritance to suit various needs:

- **Single Inheritance:** A child class inherits from one parent class.
- **Multilevel Inheritance:** A child class inherits from a parent class, which itself inherits from another parent class, creating a chain of inheritance.
- **Hierarchical Inheritance:** Multiple child classes inherit from a single parent class.

13. Differentiate between method overloading and overriding.

Method Overloading: Occurs within the same class when you define multiple methods with the same name but different numbers or types of parameters. The compiler determines which overloaded method to call based on the arguments provided at runtime.

Method Overriding: Involves redefining a method inherited from a parent class within a child class. The child class method has the same name, return type, and sometimes parameter list (though covariant parameter types are allowed) as the parent class method. Overriding allows for customizing inherited behavior in child classes.

Here's an example:

- Java

```
class Shape {
  public void area() {
    System.out.println("Area calculation for generic shape");
  }
}
class Square extends Shape {
  @Override // Not mandatory, but good practice to indicate overriding
  public void area(int side) { // Overloaded method with parameter
    System.out.println("Area of square: " + side * side);
  }
}
```

In this example, the Shape class has a general area() method. The Square class inherits this method but overrides it with a new area(int side) method specific to calculating the area of a square.

14. Explain the benefits of encapsulation.

Ans - Encapsulation offers several advantages:

- **Data Protection:** By encapsulating data (attributes) within a class and making them private, you control access to that data. Public methods within the class act as gatekeepers, ensuring data is modified only through authorized channels.
- **Information Hiding:** Encapsulation hides the internal implementation details of a class, exposing only essential functionalities through public methods. This simplifies code interaction and reduces the risk of unintended modifications.
- **Modularity:** Encapsulation promotes modularity by grouping related data and methods within a single unit (class). This makes code easier to understand, maintain, and reuse.
- **Improved Maintainability:** Changes made within a class are less likely to impact other parts of the codebase if data access is controlled through proper encapsulation.

15. Describe a scenario where polymorphism is useful.

Ans - Polymorphism allows objects of different classes to respond to the same method call in different ways. Suppose we have a scenario i.e., animal sounds with polymorphism. Imagine you're building a program for a zoo website that displays information and plays sounds for different animals. Here's how polymorphism can be useful:

- **Base Class Animal:** Create a base class Animal with attributes like name and type (e.g., mammal, bird). It can also have a generic makeSound() method that might print a message like "Animal sound".
- **Subclasses for Specific Animals:** Create subclasses like Dog, Cat, and Bird that inherit from Animal. Each subclass can override the makeSound() method to provide its specific sound (e.g., Dog.makeSound() prints "Woof!", Cat.makeSound() prints "Meow!").
- **Polymorphic Array:** Create an array of type Animal and add instances of Dog, Cat, and Bird to it. This allows you to treat all objects as animals initially.
- **Looping and Polymorphic Behavior:** Loop through the animal array and call the makeSound() method on each object. Here's the magic of polymorphism: even though the variable holding each object is of type Animal, the actual makeSound() method called will be the overridden version specific to the object's subclass (Dog, Cat, or Bird), resulting in the appropriate animal sound being played.

Advanced OOPs Interview Questions and Answers

16. What is polymorphism in OOPs and how is it implemented in programming languages?

Ans - Polymorphism is the ability of objects to be treated as instances of their parent class rather than their actual class. It allows one interface to be used for a general class of actions. In programming languages like Java, polymorphism is implemented through method overloading (compile-time polymorphism) and method overriding (runtime polymorphism).

17. Explain the difference between abstraction and encapsulation with examples.

Ans - Abstraction is the concept of hiding the complex implementation details and showing only the essential features of an object. Encapsulation is the mechanism of wrapping the data (variables) and code (methods) together as a single unit. For example, a car’s dashboard is an abstraction that hides the engine’s complexity, while encapsulation can be seen in how the engine components are hidden inside the engine block.

18. What is the Liskov Substitution Principle in OOPs?

Ans - The Liskov Substitution Principle states that objects of a superclass should be replaceable with objects of a subclass without affecting the functionality of the program. It ensures that a subclass can stand in for its superclass and function appropriately. This principle is crucial for maintaining the integrity of the OOPs hierarchy.

19. How does the Single Responsibility Principle (SRP) benefit software design?

Ans - The Single Responsibility Principle states that a class should have only one reason to change, meaning it should have only one job or responsibility. SRP benefits software design by making systems more modular, easier to maintain, and less prone to bugs. Each class has a clear and focused purpose, which simplifies debugging and future enhancements.

20. What is method overriding and when would you use it?

Ans - Method overriding occurs when a subclass provides a specific implementation of a method that is already defined in its superclass. It is used when a subclass needs to tailor a method's behavior to fit its specific requirements. For example, a **Bird** class might have a **fly** method, but a subclass **Penguin** would override this method to indicate that penguins cannot fly.

21. Can you explain the concept of inheritance and its types?

Ans - Inheritance is a mechanism in OOPs where one class (the subclass) inherits attributes and methods from another class (the superclass). There are different types of inheritance: single inheritance (one subclass inherits from one superclass), multiple inheritance (one subclass inherits from multiple superclasses, though not supported in some languages like Java), multilevel inheritance (a subclass inherits from another subclass), hierarchical inheritance (multiple subclasses inherit from one superclass), and hybrid inheritance (a combination of two or more types of inheritance).

22. What is an abstract class and how is it different from an interface?

Ans - An abstract class is a class that cannot be instantiated and is meant to be subclassed. It can have both abstract methods (without implementation) and concrete methods (with implementation). An interface is a contract that defines a set of methods without any implementation. In Java, a class can implement multiple interfaces but can inherit from only one abstract class.

23. Explain the concept of encapsulation with a real-world example.

Ans - Encapsulation is the bundling of data and methods that operate on the data within a single unit, typically a class. A real-world example of encapsulation is a capsule in medicine, which contains the drug (data) and provides a controlled way to take the medicine (methods). Similarly, a class in programming encapsulates data and provides methods to interact with that data.

24. How do constructors work in OOPs?

Ans - Constructors are special methods used to initialize objects. They have the same name as the class and do not have a return type. Constructors are called when an object is created. There are two types: default constructors (no arguments) and parameterized constructors (with arguments). They set the initial state of an object.

25. What is multiple inheritance and why is it not supported in some languages like Java?

Ans - Multiple inheritance is when a class can inherit from more than one superclass. It is not supported in languages like Java because it can lead to the "diamond problem," where a class inherits conflicting implementations from multiple superclasses. Instead, Java uses interfaces to achieve similar functionality without the complexity.

26. What is a destructor and how is it different from a constructor?

Ans - A destructor is a special method called when an object is destroyed or goes out of scope, used to free resources the object may have acquired during its lifetime. Unlike constructors, destructors are not explicitly called in most programming languages; they are invoked by the garbage collector (e.g., in C++ and Python).

27. How does method overloading differ from method overriding?

Ans - Method overloading occurs when multiple methods in the same class have the same name but different parameters (number, type, or order). It is a compile-time polymorphism. Method overriding occurs when a subclass provides a specific implementation for a method already defined in its superclass. It is a runtime polymorphism.

28. What is the Open/Closed Principle in OOPs?

Ans - The Open/Closed Principle states that software entities (classes, modules, functions) should be open for extension but closed for modification. This means you can add new functionality by extending existing components without altering their existing code, promoting flexibility and preventing bugs in the existing codebase.

29. Explain the concept of dynamic dispatch.

Ans - Dynamic dispatch is a mechanism by which a call to an overridden method is resolved at runtime rather than compile-time. It is used in polymorphism to ensure that the correct method implementation is called based on the actual object type, not the reference type.

30. What are design patterns and how do they relate to OOPs?

Ans - Design patterns are proven solutions to common software design problems. They provide a template for how to solve a problem in various contexts. Design patterns relate to OOPs as they leverage OOPs principles like encapsulation, inheritance, and polymorphism to provide flexible and reusable design solutions.

Frequently Asked Questions

What are some of the programming paradigms?

Following are some of the programming paradigms

1. Procedural programming
2. Imperative programming
3. Object-oriented programming
4. Functional programming

What are some of the languages that follow procedural programming paradigms?

The languages which use procedural programming paradigms are FORTRAN, C, ALGOL, COBOL, PASCAL, etc.

What are the different types of variables in OOPs?

There are generally three types of variables

1. Instance variable
2. Static variables
3. Local variables

What are the 7 concepts of OOPs?

Traditionally, OOPs is based on 4 pillars (Encapsulation, Abstraction, Inheritance, Polymorphism). Some may include additional concepts like classes, objects, and messages, but these can be considered foundational aspects related to the core pillars.

Intermediate Java OOPs Interview Questions

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.

Multiple Inheritance	Multilevel Inheritance
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.

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.

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.

Advanced Java OOPs Interview Questions

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.

Frequently Asked Questions

Why is Java known as an object-oriented programming language?

Due to the lack of global scope, Java is exclusively an object-oriented language. Everything is an object in java, and all program functions and data are contained within classes and objects. It has a large number of classes organized into packages, as well as a Java object model that is easy to extend.