Interview Questions - OOPS Java

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

- OOP is a programming paradigm that models real-world entities as objects.
- It focuses on objects and classes to design modular, reusable, and maintainable software.
- Example: A **car** modeled as an object with attributes (color, brand) and methods (accelerate, brake).

---

2. What are the four main principles of OOP?

- **Abstraction**: Hides complex implementation and shows only essential features (e.g., ATM interface).
- **Encapsulation**: Groups data and methods inside a class and restricts access using access modifiers (e.g., `private balance`).
- **Inheritance**: Reuses fields and methods from a parent class (e.g., `Dog` inherits from `Animal`).

- **Polymorphism**: Allows the same method to behave differently (e.g., `draw()` in `Shape` class used by `Circle` and `Square`).

---

3. What is a class in Java?

- A class is a blueprint to create objects, defining the data (attributes) and behaviors (methods).
- Example: A `Car` class with attributes like `color`, `model` and methods like `drive()`.

---

4. What is an object in Java?

- An object is an instance of a class that holds specific data values.
- Example: `Car myCar = new Car();` creates an object with individual properties.

---

5. What is encapsulation? Why is it important?

- **Encapsulation** bundles data (fields) and methods into one class and restricts access using access modifiers.
- Protects data from unauthorized access by making fields `private` and exposing them through `public` getters/setters.
- Example: A `BankAccount` class with a `private balance` field and `getBalance()` method to view it.
- **Prevents accidental modification** and maintains internal consistency.

---

6. What is abstraction? Provide an example.

- Abstraction hides the complexity of the implementation from the user and exposes only necessary features.
- Example: Using a **remote control** to operate a TV hides the technical process of signals being sent.

---

7. What is inheritance? How does it help?

- Inheritance allows a class (child) to reuse the properties of another class (parent).
- Helps to avoid code duplication and promotes reusability.

- Example: A `Dog` class inherits `Animal` properties like `eat()` and `sleep()`.

---

8. What is polymorphism in Java? Provide examples.

- **Compile-time polymorphism**: Achieved using **method overloading** (same method name, different parameters).
- **Runtime polymorphism**: Achieved using **method overriding** (same method in parent and child classes).
- Example: `add(int, int)` and `add(double, double)` are overloaded methods.

---

9. What is method overloading?

- Method overloading allows a class to define multiple methods with the same name but different parameters.
- Example: `add(int, int)` and `add(float, float)` in a Calculator class.

---

10. What is method overriding?

- When a subclass provides a specific implementation of a method already defined in its parent class.
- Example: `toString()` method in `Object` class overridden in `Person` class to display details.

---

11. What is the difference between overloading and overriding?

- Overloading happens within the same class; overriding occurs between parent and child classes.
- Overloading is resolved at compile-time; overriding is resolved at runtime.

---

12. What is the purpose of the `this` keyword?

- Refers to the current object of the class.
- Helps to avoid naming conflicts between class fields and parameters.
- Example: `this.name = name` sets the instance variable `name`.

---

13. What is the `super` keyword?

- Refers to the parent class object.
- Used to call parent class constructors and methods.
- Example: `super()` to invoke the parent constructor.

---

14. What is upcasting in Java?

- Assigning a child class object to a parent class reference.
- Example: `Animal a = new Dog();` allows treating a Dog object as an Animal.

---

15. What is downcasting in Java?

- Casting a parent class reference back to a child class type.
- Example: `Dog d = (Dog) a;` after upcasting.

---

16. What is the difference between interface and abstract class?

- **Interface**: Provides full abstraction with only method signatures.
- **Abstract class**: Can contain both abstract and concrete methods.
- Example: **PaymentProcessor** interface with implementations like **UPI** and **CreditCard**.

---

17. What is the Java Collections Framework?

- Provides classes and interfaces for storing and manipulating groups of objects.
- Example: `List`, `Set`, `Map`.

---

18. What is the difference between `List` and `Set`?

- **List** allows duplicates and maintains order (e.g., `ArrayList`).
- **Set** doesn't allow duplicates and may not maintain order (e.g., `HashSet`).

---

19. What is an Iterator?

- Used to traverse elements in a collection sequentially.
- Example: `Iterator<String> it = list.iterator();`

20. What is multithreading? Why is it useful?

- Multithreading allows concurrent execution of multiple threads to improve performance.
- Example: A web server handling multiple requests simultaneously.

21. How can we create a thread in Java?

- By extending `Thread` class or implementing `Runnable` interface.

22. What is synchronization?

- Ensures that only one thread accesses a shared resource at a time.
- Example: Synchronizing access to a shared bank account balance.

23. What is the Stream API in Java?

- Introduced in Java 8 to process collections in a functional style.
- Example: Using `filter()` to filter even numbers from a list.

24. What is garbage collection?

- Garbage collection automatically frees memory by destroying unused objects.

25. What is the difference between `throw` and `throws`?

- `throw`: Used to explicitly throw an exception.
- `throws`: Declares that a method may throw exceptions.

26. What is exception handling in Java?

- A mechanism to handle runtime errors gracefully using `try-catch` blocks.

27. What is the purpose of the `finally` block?

- Executes regardless of whether an exception occurs.
- Example: Closing a file in `finally` to avoid resource leakage.

28. What is file handling in Java?

- Reading and writing to files using classes like `FileReader` and `FileWriter`.

---

29. What is the difference between JVM, JRE, and JDK?

- **JVM**: Executes bytecode.
- **JRE**: Provides libraries to run Java programs.
- **JDK**: Contains tools for development and testing.

---

30. What is dynamic method dispatch?

- Resolves overridden method calls at runtime based on the actual object type.

---

31. What is coupling in OOP?

- Coupling refers to the degree of dependency between classes.
- **Loose coupling** is preferred for easier maintenance.

---

32. What is cohesion in OOP?

- Cohesion refers to how closely related a class's responsibilities are.
- Example: A `User` class managing only user-related logic.

---

33. What is narrowing conversion in Java?

- Converting a **larger data type** into a **smaller data type**.
- Example: `double d = 9.7; int i = (int) d;` (Explicit casting needed).
- May result in **loss of data** due to smaller memory capacity.

---

34. What is widening conversion in Java?

- Converting a **smaller data type** to a **larger data type** automatically (implicit).
- Example: `int i = 10; double d = i;` (No casting needed).
- No risk of data loss.

---

35. What is the Reflection API in Java?

- Allows a program to inspect or modify itself at runtime.
- Used to **dynamically load classes**, invoke methods, or access fields.
- Example: Reflection is used in **Java frameworks** like Spring to create objects at runtime.

---

36. What is the difference between Stack and Heap memory?

- **Stack**: Stores local variables and method calls. Faster but limited in size.
- **Heap**: Stores objects created dynamically using `new`. Slower but larger.
- Example: Variables inside a method go to the **stack**, while objects go to the **heap**.

---

37. What is coupling in OOP, and why is loose coupling preferred?

- **Coupling** measures how tightly two classes or modules depend on each other.
- **Loose coupling**: Classes have fewer dependencies, easier to maintain and extend.
- Example: Using **interfaces** promotes loose coupling between modules.

---

38. What is cohesion in OOP?

- Refers to how well the methods and data within a class are related to one specific responsibility.
- **High cohesion** is desired as it makes classes easier to maintain and understand.
- Example: A `Payment` class only handling payment logic ensures high cohesion.

---

39. What is the difference between checked and unchecked exceptions?

- **Checked exceptions**: Checked at compile-time (e.g., `IOException`).
- **Unchecked exceptions**: Occur at runtime (e.g., `NullPointerException`).

- Example: File handling code must handle `IOException`.

---

40. What is the purpose of the `throws` keyword in Java?

- Declares that a method **may throw exceptions**, forcing the caller to handle them.
- Example: `public void readFile() throws IOException { ... }`

---

41. How does Java achieve platform independence?

- Java code is compiled into **bytecode**, which can run on any platform with a JVM.
- Example: The same `.class` file can run on **Windows, macOS, or Linux**.

---

42. What are lambda expressions in Java?

- Lambda expressions provide a **concise way to represent functional interfaces**.
- Example: `(a, b) -> a + b` is a lambda that adds two numbers.
- Used in **Stream API** for operations like `filter()` and `map()`.

---

43. What is an inner class in Java?

- A class defined within another class.
- Example: A `Button` class having an inner class **EventListener** to handle events.

---

44. What is an anonymous class?

- A class without a name, defined at the point of use.
- Example: `Runnable r = new Runnable() { public void run() { ... } };`

---

45. What are functional interfaces?

- An interface with **only one abstract method**.
- Example: `Runnable` with `run()` method, used in **lambda expressions**.

---

46. What is serialization in Java?

- The process of **converting an object to a byte stream** to save or transfer it.
- Example: Saving a **user object** to a file using `ObjectOutputStream`.

---

47. What is deserialization?

- **Reconstructing an object** from a byte stream.
- Example: Reading an object from a file using `ObjectInputStream`.

---

48. What is the `transient` keyword?

- Marks a field to be **excluded from serialization**.
- Example: Password fields in a `User` class can be marked as `transient`.

---

49. What is the `volatile` keyword in Java?

- Ensures that **changes to a variable are visible to all threads** immediately.
- Example: Used in **multi-threading** to avoid caching issues.

---

50. What is a singleton class?

- A class that allows **only one instance** to exist.
- Example: **Database connection manager** in applications.

---

51. What is the difference between `==` and `equals()`?

- `==`: Compares **references** (memory addresses).
- `equals()`: Compares **content** (actual values).
- Example: `"abc" == new String("abc")` returns `false`, but `equals()` returns `true`.

---

52. What is the Stream API?

- Provides operations to process data in a **functional style** (e.g., `filter()`, `map()`).
- Example: Filtering names starting with "A" from a list using `list.stream().filter(...)`.

---

53. What is the difference between `ArrayList` and `LinkedList`?

- **ArrayList**: Fast for random access, slower for insertions/deletions.
- **LinkedList**: Faster insertions/deletions, slower for random access.
- Example: Use `ArrayList` for frequent reads, `LinkedList` for frequent updates.

---

54. What is file handling in Java?

- Reading and writing data to files using classes like `FileReader` and `FileWriter`.
- Example: Saving user input to a **log file**.

---

55. What is dynamic method dispatch?

- The process where **overridden methods are resolved at runtime** based on the object type.
- Example: A `Dog` object referenced by an `Animal` reference will call `Dog`'s `speak()`.

---

56. What is the difference between `final`, `finally`, and `finalize()`?

- **final**: Used to declare constants or prevent inheritance.
- **finally**: Ensures code runs even if an exception occurs.
- **finalize()**: Called by the garbage collector before an object is destroyed.

---

57. What is a marker interface?

- An interface without methods, used to mark classes for special behavior.
- Example: **Serializable** interface marks a class for serialization.

---

58. What are generics in Java?

- Allows creating **classes and methods that work with any data type**.
- Example: `List<Integer>` ensures only integers are added to the list.

---

59. What is the difference between shallow and deep copy?

- **Shallow copy**: Copies only references, not the actual objects.
- **Deep copy**: Copies the actual objects as well.
- Example: Cloning an object with nested objects requires a **deep copy**.

---

60. What is a ConcurrentHashMap?

- A thread-safe version of `HashMap` that allows **concurrent access**.
- Example: Used in multi-threaded applications for better performance.

---

61. What is immutability in Java?

- An **immutable object** cannot be modified after creation.
- Example: **String** objects in Java are immutable.

---

62. What is a constructor?

- A special method used to **initialize objects**.
- Example: `public Car(String model) { this.model = model; }`

---

63. What is the difference between a constructor and a method?

- Constructors initialize objects; methods perform operations.
- Constructors have the same name as the class; methods have any name.

---

64. What is a copy constructor?

- A constructor that **creates a new object by copying another object**.
- Example: `public Car(Car c) { this.model = c.model; }`

---

65. What is coupling and how to reduce it?

- Coupling is the dependency between modules.
- Use **interfaces** and **dependency injection** to reduce coupling.

---

66. What is `@Override` annotation?

- Indicates that a method **overrides** a superclass method.

67. What is `@FunctionalInterface`?

- Marks an interface with only one abstract method for lambda use.

68. What are streams and buffers in Java?

- Streams handle data sequentially; buffers improve efficiency with temporary storage.

69. What is `enum` in Java?

- Defines a set of constants. Example: `enum Day { MON, TUE, WED }`

70. What is the difference between HashMap and TreeMap?

- HashMap is unordered; TreeMap maintains sorted order.