**OOP in Java Viva Questions with Answers**

Some example questions are given below so that you can prepare well for the upcoming Viva. Please note that some of the given answers may not be absolute, and your answer may vary with respect to the context and the depth of the understanding you may have.

**Easy Questions**

1. **What is Object-Oriented Programming (OOP)?**
   * OOP is a programming paradigm based on the concept of "objects," which contain data (attributes) and behavior (methods). Key principles are encapsulation, inheritance, polymorphism, and abstraction.
2. **Define the four main principles of OOP.**
   * **Encapsulation:** Bundling data and methods within objects.
   * **Inheritance:** Reusing properties and behavior of a parent class.
   * **Polymorphism:** Using one interface to represent different implementations.
   * **Abstraction:** Hiding implementation details and exposing essential features.
3. **What is a class in Java?**
   * A class is a blueprint for creating objects, defining attributes and methods.
4. **What is an object in Java?**
   * An object is an instance of a class that represents a real-world entity.
5. **Explain the purpose of encapsulation in Java. How is it implemented?**
   * Encapsulation restricts access to object data using access modifiers and exposes controlled access via getters and setters.
6. **What is the significance of the this keyword?**
   * The this keyword refers to the current object instance and is used to resolve naming conflicts or call other constructors.
7. **Define constructor overloading. Provide an example.**
   * Constructor overloading allows multiple constructors with different parameters in the same class.
8. **Explain the purpose of the main method in Java.**
   * It is the entry point for any Java application: public static void main(String[] args).
9. **What is method overloading? How does it differ from method overriding?**
   * **Overloading:** Same method name, different parameter lists within the same class.
   * **Overriding:** Subclass redefines a method of its parent class with the same signature.
10. **What is inheritance in OOP? Provide an example.**
    * Inheritance allows a class to acquire properties and methods of another class using the extends keyword.
11. **Define polymorphism in Java and provide an example.**
    * Polymorphism allows methods to perform differently based on the object calling it (e.g., method overriding).
12. **What are access modifiers in Java?**
    * **Public:** Accessible everywhere.
    * **Private:** Accessible only within the class.
    * **Protected:** Accessible in the package and subclasses.
    * **Default:** Accessible in the package.
13. **Explain the final keyword and where it can be applied in Java.**
    * It can make variables constants, prevent method overriding, and stop inheritance.
14. **What are primitive data types in Java?**
    * **byte, short, int, long, float, double, char, boolean**.
15. **What is the purpose of wrapper classes?**
    * Wrapper classes convert primitives into objects (e.g., int to Integer).
16. **How does Java achieve platform independence?**
    * Java compiles code into bytecode, which can run on any platform with a JVM.
17. **What are packages in Java, and why are they used?**
    * Packages organize classes and interfaces, preventing name conflicts and improving modularity.
18. **What is the difference between a class variable and an instance variable?**
    * **Class variables:** Declared static, shared among all objects.
    * **Instance variables:** Unique to each object.
19. **What is a default constructor in Java?**
    * A constructor automatically provided by Java when no explicit constructor is defined.
20. **What is the difference between null, undefined, and an empty string in Java?**
    * **null:** Refers to the absence of an object.
    * **undefined:** Not a valid concept in Java.
    * **Empty string:** A string with no characters ("").
21. **What is the purpose of the static keyword?**
    * It defines class-level variables or methods, shared across all instances.
22. **What is a nested class in Java?**
    * A class defined within another class. It can be static or non-static.
23. **What is a Java interface?**
    * A blueprint for classes, containing abstract methods and constants.
24. **What is the void return type?**
    * Indicates a method does not return any value.
25. **Explain the difference between continue and break.**
    * **Break:** Exits a loop.
    * **Continue:** Skips the current iteration and moves to the next.
26. **What is type casting in Java?**
    * Converting one data type to another (e.g., int to double).
27. **What are local variables?**
    * Variables declared inside a method or block, accessible only within that scope.
28. **What is a final variable?**
    * A variable whose value cannot be changed once assigned.
29. **Explain the term "immutable object" in Java.**
    * An object whose state cannot be modified after creation (e.g., String).
30. **What is a lambda expression in Java?**
    * A concise way to implement functional interfaces using -> syntax.

**Medium Difficulty Questions**

1. **Explain abstraction in OOP and how it is achieved in Java.**
   * Abstraction hides implementation details using abstract classes and interfaces.
2. **What is the difference between an abstract class and an interface in Java?**
   * Abstract classes can have methods with implementations, interfaces cannot (prior to Java 8).
3. **How does Java handle multiple inheritance?**
   * Java avoids multiple inheritance with classes but allows it using interfaces.
4. **Describe the process of creating and using a thread in Java.**
   * Implement the Runnable interface or extend the Thread class.
5. **What is autoboxing and unboxing in Java? Provide an example.**
   * Autoboxing converts primitives to wrapper objects automatically. Unboxing does the reverse.
6. **Explain the difference between checked and unchecked exceptions.**
   * **Checked:** Must be handled at compile time (e.g., IOException).
   * **Unchecked:** Occurs at runtime (e.g., ArithmeticException).
7. **How does Java's garbage collection mechanism work?**
   * JVM automatically deallocates unused objects to free memory.
8. **What is the super keyword in Java? How is it used in inheritance?**
   * super calls the parent class's methods or constructors.
9. **What are Java's exception-handling mechanisms?**
   * Using try-catch blocks, throw statements, and finally blocks.
10. **How does the String pool in Java optimize memory?**
    * JVM stores string literals in a pool to avoid duplication and save memory.
11. **Differentiate between StringBuilder and StringBuffer.**
    * **StringBuilder:** Faster but not thread-safe.
    * **StringBuffer:** Thread-safe but slower.
12. **What is the purpose of lambda expressions? Provide a simple example.**
    * Used to implement functional programming and concise code for functional interfaces.
13. **Explain how the Scanner class is used for user input in Java.**
    * Scanner reads input via methods like nextLine() or nextInt().
14. **What is the significance of the transient keyword in Java?**
    * It marks variables not to be serialized.
15. **How do static and non-static methods differ in Java?**
    * **Static:** Called on the class.
    * **Non-static:** Called on an object.
16. **What is the purpose of constructor chaining?**
    * To call one constructor from another in the same class, improving code reuse.
17. **What are functional interfaces in Java?**
    * Interfaces with a single abstract method (e.g., Runnable).
18. **Explain the difference between throw and throws.**
    * **Throw:** Used to explicitly throw an exception.
    * **Throws:** Declares exceptions a method might throw.
19. **What are runtime exceptions?**
    * Exceptions occurring during program execution (e.g., NullPointerException).
20. **What is the role of the finally block?**
    * Executes code after try-catch, regardless of exceptions.

**Hard Questions**

1. **What are the advantages and limitations of using interfaces in Java?**
   * **Advantages:** Multiple inheritance, abstraction.
   * **Limitations:** No state management, complexity.
2. **How is runtime polymorphism implemented in Java? Provide an example.**
   * Achieved through method overriding and dynamic method dispatch.
3. **Explain how memory is allocated for objects and classes in Java.**
   * **Heap:** For objects.
   * **Stack:** For method calls and local variables.
4. **What is method hiding, and how does it differ from method overriding?**
   * **Hiding:** Applies to static methods.
   * **Overriding:** Applies to instance methods.
5. **Describe how Java ensures type safety at compile time.**
   * Through generics and static type checking.
6. **How do Java's annotations enhance programming?**
   * Provide metadata for code, e.g., @Override, @Deprecated.
7. **What is the Just-In-Time (JIT) compiler, and how does it improve Java's performance?**
   * Converts bytecode to machine code at runtime for faster execution.
8. **Explain the role of the Java Virtual Machine (JVM) in program execution.**
   * Executes bytecode, manages memory, and ensures security.
9. **What is a singleton design pattern, and how is it implemented in Java?**
   * Ensures only one instance of a class. Use private constructors and static methods.
10. **How does Java's Stream API simplify data processing?**
    * Enables functional-style operations on collections, e.g., filtering, mapping.
11. **Explain how Java achieves multithreading and thread synchronization.**
    * By synchronized blocks/methods and java.util.concurrent package.
12. **What are the key differences between ArrayList and LinkedList in Java?**
    * **ArrayList:** Fast for random access.
    * **LinkedList:** Efficient for insertions and deletions.
13. **How does Java handle concurrency in large-scale applications?**
    * Using thread pools, concurrent collections, and synchronization techniques.
14. **What are weak references in Java, and why are they used?**
    * They allow objects to be garbage collected when not strongly referenced.
15. **How does the ForkJoinPool framework work in Java?**
    * Splits tasks into smaller subtasks and merges results for parallelism.
16. **What is reflection in Java?**
    * Allows inspection and manipulation of classes, methods, and fields at runtime.
17. **Explain serialization and deserialization in Java.**
    * Serialization converts objects to byte streams; deserialization reverses the process.
18. **What is an enum in Java?**
    * A special data type for a fixed set of constants.
19. **How are inner classes used in Java?**
    * Inner classes are defined within another class, providing encapsulation and access to outer class members.
20. **What is the difference between parallel and sequential streams in Java's Stream API?**
    * Parallel streams process data concurrently; sequential streams process data in order.

In addition to the above questions, please expect questions on the following format too.

1. How would you design a class hierarchy for a transportation system that includes cars, bikes, and buses? What principles of OOP would you apply and why?
2. Given two objects of a class Student with fields name and grade, how would you compare them to check for equality of content? Why should hashCode and equals be overridden together?
3. If two threads access a shared resource in Java, how would you ensure thread safety? Explain the pros and cons of your approach.
4. How would you use composition to model a class Library that contains multiple Book objects, instead of using inheritance? Why is composition preferred in this case?
5. Design a singleton class for a logging system in Java. What are the potential issues with your design in a multithreaded environment, and how would you solve them?
6. Explain how Java handles memory management for objects. What happens when you create an object inside a method and the method finishes execution?
7. Given a large list of numbers, how would you filter and sort the list using Java Streams? Provide a brief explanation of your approach.
8. If you were to extend the functionality of an existing Java class without modifying its source code, how would you achieve that? What approach would you prefer—inheritance or a decorator pattern—and why?
9. How does Java handle method resolution at runtime when an overridden method is called on a parent reference pointing to a child object? What is this mechanism called?
10. Describe a scenario where using an interface is more beneficial than using an abstract class. How does this choice affect scalability and design?