## ◆ OOPs in Java – Questions & Answers

1. What are the 4 pillars of OOP?

Encapsulation, Abstraction, Inheritance, Polymorphism.

2. Explain Encapsulation with example.

Wrapping data and methods into a single unit (class). Example: private variables with public getters/setters.

3. Explain Abstraction with example.

Hiding implementation details and showing only necessary features. Example: Abstract class, Interface.

4. Difference between Abstraction and Encapsulation?

Abstraction ➔ Hides implementation (what to do).

Encapsulation ➔ Hides data (how data is accessed).

5. What is Inheritance? Give an example.

Mechanism where one class acquires properties of another. Example: class Dog extends Animal.

6. Can Java support multiple inheritance?

Not with classes (to avoid ambiguity), but possible with interfaces.

7. Difference between IS-A and HAS-A relationship in Java?

IS-A ➔ Inheritance (Dog IS-A Animal).

HAS-A ➔ Composition (Car HAS-A Engine).

8. What is Polymorphism? Types of Polymorphism?

Polymorphism = one entity behaving differently. Types: Compile-time (overloading), Runtime (overriding).

9. Difference between Method Overloading and Method Overriding?

Overloading ➔ Same method name, different parameters (compile-time).

Overriding ➔ Same method in parent & child, different implementation (runtime).

10. What is Dynamic (Runtime) Polymorphism?

Achieved by method overriding; resolved at runtime.

11. What is Static (Compile-time) Polymorphism?

Achieved by method overloading; resolved at compile time.

12. What is Constructor in Java?

Special method used to initialize objects.

13. Types of Constructors in Java?

Default, Parameterized, Copy constructor (manually created).

14. Difference between default constructor and parameterized constructor?

Default ➔ no parameters.

Parameterized ➔ accepts parameters to initialize values.

15. Can we overload constructors in Java?

Yes, by changing parameter lists.

16. What is the use of this keyword in constructors?

Refers to current object, used to call another constructor or resolve variable conflicts.

17. Difference between super and this keyword?

this ➔ current object.

super ➔ parent class object.

18. What is the role of getters and setters in Encapsulation?

They provide controlled access to private variables.

19. Difference between Abstract Class and Interface?

Abstract class ➔ can have both abstract + concrete methods.

Interface ➔ only abstract methods (Java 7), can have default/static (Java 8+).

20. Can interface have constructor? Why/Why not?

No, because interfaces cannot be instantiated.

21. Difference between final, finally, and finalize in Java?

final ➔ keyword (constants, methods, classes).

finally ➔ block (exception handling).

finalize() ➔ method (GC cleanup).

22. Can we override a static method in Java?

No, static methods belong to class not object (method hiding instead).

23. Can we override a private method in Java?

No, private methods are not inherited.

24. Can we achieve abstraction without abstract class?

Yes, using interfaces.

25. Why do we need OOPs instead of procedural programming?

Better modularity, reusability, scalability, and security.

## ◆ SOLID Principles – Questions & Answers

26. What is SOLID in Java?

A set of 5 design principles for writing maintainable, scalable code.

27. Explain each of the 5 SOLID principles.

S ➔ Single Responsibility Principle

O ➔ Open/Closed Principle

L ➔ Liskov Substitution Principle

I ➔ Interface Segregation Principle

D ➔ Dependency Inversion Principle

28. Explain Single Responsibility Principle (SRP) with example.

A class should have only one responsibility. Example: A UserService should not handle database and logging both.

29. What happens if a class violates SRP?

It becomes hard to maintain, test, and reuse.

30. Explain Open/Closed Principle (OCP) with example.

Classes should be open for extension but closed for modification. Example: Using interfaces instead of modifying existing classes.

31. How does Java achieve OCP in design?

Through inheritance, polymorphism, and interfaces.

32. Explain Liskov Substitution Principle (LSP) with example.

Subclasses should be replaceable with parent class without breaking functionality.
Example: A Square should not violate Rectangle behavior.

33. What will happen if we break LSP?

Unexpected behavior when subclass replaces superclass.

34. Explain Interface Segregation Principle (ISP).

Clients should not be forced to implement methods they don’t use.

35. Why is ISP important in real projects?

Prevents bulky interfaces and keeps code flexible.

36. Explain Dependency Inversion Principle (DIP).

High-level modules should depend on abstractions, not concrete implementations.

37. Difference between Dependency Injection and Dependency Inversion?

DIP ➔ Principle (design concept).

DI ➔ Implementation technique (Spring, Guice).

38. Which SOLID principle relates to Abstraction most?

Dependency Inversion Principle.

39. Which SOLID principle reduces tight coupling?

Dependency Inversion.

40. Real-life example of SRP and OCP in Java?

SRP ➔ Separate Logger class.

OCP ➔ Payment system extended for UPI without modifying existing card payment code.

41. Which principle is violated if class has too many responsibilities?

Single Responsibility Principle.

42. Which SOLID principle ensures extensibility without modifying old code?

Open/Closed Principle.

43. Which SOLID principle is most related to interfaces?

Interface Segregation Principle.

44. Which SOLID principle focuses on high-level modules not depending on low-level modules?

Dependency Inversion Principle.

45. Why are SOLID principles important in software design?

They improve readability, maintainability, testability, and scalability.

◆ Comparator & Comparable – Questions & Answers

46. What is Comparable in Java?

Interface used to define natural ordering of objects (compareTo).

47. What is Comparator in Java?

Interface used to define custom ordering of objects (compare).

48. Difference between Comparable and Comparator?

Comparable ➔ single natural order (inside class).

Comparator ➔ multiple custom orders (outside class).

49. When to use Comparable vs Comparator?

Comparable ➔ natural order (e.g., RollNo).

Comparator ➔ multiple sorting (e.g., by Name, by Age).

50. Can we sort objects without implementing Comparable?

Yes, using Comparator.

51. How do you implement Comparable in Java?

Implement Comparable<T> and override compareTo().

52. How do you implement Comparator in Java?

Implement Comparator<T> and override compare().

53. Can a class implement both Comparable and Comparator?

Yes, but usually we keep Comparable for natural order and external Comparator for custom order.

54. Can we define multiple Comparators for one class?

Yes, by creating multiple Comparator implementations.

55. What is the difference between compareTo() and compare() method?

compareTo() ➔ Comparable (single natural order).

compare() ➔ Comparator (custom order).

56. What happens if compareTo() returns 0?

Objects are considered equal.

57. What happens if compareTo() returns positive or negative?

Negative ➔ current object < other object.

Positive ➔ current object > other object.

58. Can Comparator be implemented using Lambda expressions?

Yes, from Java 8 onwards. Example: (a, b) -> a.getName().compareTo(b.getName()).

59. Give an example of sorting using Comparator and Lambda.

Collections.sort(list, (a, b) -> a.getAge() - b.getAge());

60. Which interface is part of java.lang and which is part of java.util?

Comparable ➔ java.lang.

Comparator ➔ java.util.

61. What will happen if we don’t implement Comparable in a class and try to sort it?

It throws ClassCastException.

62. Which is better: Comparable or Comparator? Why?

Comparator, because it allows multiple sorting strategies.

63. Can we reverse sorting order using Comparator?

Yes, by using Collections.reverseOrder() or Comparator.reversed().

64. Can we chain multiple Comparators in Java? (using thenComparing)

Yes, Example:
Comparator.comparing(Employee::getName).thenComparing(Employee::getAge).

65. Real-life example of Comparable and Comparator?

Comparable ➔ Sorting students by roll number.

Comparator ➔ Sorting students by name, marks, or age.

## 1. Interface in Java

### ✅ Definition:

An **interface** in Java is a **blueprint of a class** that contains only **abstract methods (before Java 8)** and constants.
It's used to achieve **100% abstraction** and **multiple inheritance**.

### ✅ Key Points:

- Methods in interface are **abstract by default** (till Java 7).
- From **Java 8 onwards**:
  - Can have **default methods** (with body).
  - Can have **static methods** (with body).
- Variables in interface are:
  - public static final by default (constants).
- A class **implements** an interface, not extends.

### ✅ Example:

```
interface Animal {
  void sound(); // abstract method
}


class Dog implements Animal {
  public void sound() {
    System.out.println("Bark");
  }
}
```

📌 **Interview Tip**:
Interface is best when you need to enforce a **contract** (rules) across multiple classes.

---

## 2. Abstraction in Java

### ✅ Definition:

**Abstraction** is the process of **hiding implementation details** and showing only **essential features**.

In Java, abstraction is achieved in two ways:

1. **Abstract Class** (0–100% abstraction).
2. **Interface** (100% abstraction until Java 7, now can have partial).

✅ **Abstract Class:**

- Declared using abstract keyword.
- Can have **abstract methods (no body)** + **normal methods (with body)**.
- Cannot be instantiated directly.

✅ **Example:**

```
abstract class Animal {
  abstract void sound(); // abstract method

  void sleep() { // concrete method
    System.out.println("Sleeping...");
  }
}

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

📌 **Interview Tip**:

- Use **abstract class** when you want to provide a **base class with some common code + abstract methods**.
- Use **interface** when you just need rules (no implementation sharing).

---

## 3. Exception Handling in Java

### ✅ Definition:

**Exception Handling** in Java is a mechanism to handle **runtime errors** so the program doesn’t crash and executes smoothly.

### ✅ Types of Exceptions:

1. **Checked Exceptions** → Checked at compile-time (e.g., IOException, SQLException).
2. **Unchecked Exceptions** → Occur at runtime, not checked at compile time (e.g., NullPointerException, ArithmeticException).
3. **Errors** → Serious issues not handled by program (e.g., OutOfMemoryError).

### ✅ Keywords Used:

- try → Code that may throw exception.
- catch → Handles the exception.
- finally → Always executes (cleanup code).
- throw → Used to explicitly throw an exception.
- throws → Declares exceptions in method signature.

### ✅ Example:

```
public class Example {
  public static void main(String[] args) {
    try {
      int a = 10 / 0; // risky code
    } catch (ArithmeticException e) {
      System.out.println("Cannot divide by zero!");
    } finally {
      System.out.println("Finally block always runs.");
    }
  }
}
```

📌 **Interview Tip**:

- **Checked vs Unchecked** is frequently asked.
- Always mention the **importance of finally block** (used for closing resources like DB connection, file, etc.).

---

✅ **Quick Interview Answer Recap**:

- **Interface** → Blueprint, multiple inheritance, 100% abstraction (till Java 7).
- **Abstraction** → Hiding implementation, achieved using abstract class & interface.
- **Exception Handling** → Mechanism to handle runtime errors using try-catch-finally, throw, throws.

## Interface & Abstraction – Java Interview Q&A

Q1. ⭐ What is an interface in Java?

An interface is a contract that defines abstract methods without implementation.

Q2. ⭐ Can an interface have variables?

Yes, but they are public static final by default (constants).

Q3. ⭐ What is the default access modifier for interface methods?

public abstract.

Q4. ⭐ Can we create objects of an interface?

No, but we can create reference variables of it.

Q5. ⭐ Difference between abstract class and interface?

Abstract class can have both concrete and abstract methods, interface only abstract (till Java 7).

Q6. ⭐ From Java 8 onwards, what extra can interfaces have?

default and static methods.

Q7. ⭐ Can interfaces have constructors?

No, because they cannot be instantiated.

Q8. Can one class implement multiple interfaces?

Yes, multiple inheritance is achieved using interfaces.

Q9. ⭐ Can an interface extend another interface?

Yes, interfaces can extend multiple interfaces.

Q10. Can an abstract class implement an interface?

Yes, but it doesn't need to implement all methods.

Q11. ⭐ What happens if a class does not implement all methods of an interface?

The class must be declared abstract.

Q12. ⭐ Why use interfaces?

To achieve 100% abstraction and multiple inheritance.

Q13. Difference between abstract method and interface method?

Abstract method can exist in abstract class, interface methods are always abstract (till Java 7).

Q14. ⭐ Can we use private methods in interface?

Yes, from Java 9 onwards (only for helper methods inside interface).

Q15. What is marker interface?

An interface with no methods (e.g., Serializable).

Q16. ⭐ Example of real-life interface usage?

List, Set, Map in Collections Framework.

Q17. ⭐ Can interface extend a class?

No, it can only extend another interface.

Q18. Difference between multiple inheritance in classes and interfaces?

Multiple inheritance with classes causes ambiguity, interfaces solve it using implementation in child class.

Q19. ⭐ When to use abstract class vs interface?

Use abstract class when you want partial abstraction, interface when you need full abstraction or multiple inheritance.

Q20. ⭐ Can an interface have nested interfaces?

Yes, interfaces can contain nested interfaces.

## 5. Collections

1. ⭐ Difference between List, Set, Map?

→ List = ordered, duplicates; Set = unique; Map = key-value.

2. What is HashMap?

→ Stores key-value pairs, allows null key.

3. Difference between HashMap and Hashtable?

→ HashMap is not synchronized, Hashtable is.

4. ⭐ Difference between HashMap and LinkedHashMap?

→ LinkedHashMap maintains insertion order.

5. Difference between HashSet and TreeSet?

→ HashSet = unordered, TreeSet = sorted.

6. What is ArrayList?

→ Dynamic array, ordered.

7. ⭐ Difference between ArrayList and LinkedList?

→ ArrayList = fast random access, LinkedList = fast insert/delete.

8. What is ConcurrentHashMap?

→ Thread-safe map without blocking.

9. Difference between fail-fast and fail-safe iterators?

→ Fail-fast throws ConcurrentModificationException, fail-safe doesn’t.

10. ⭐ What is Comparator vs Comparable?

→ Comparable = natural order, Comparator = custom order.

11. Which collection is synchronized?

→ Vector, Hashtable.

12. Difference between Stack and Queue?

→ Stack = LIFO, Queue = FIFO.

13. ⭐ What is PriorityQueue?

→ Queue with natural/custom ordering.

14. What is IdentityHashMap?

→ Compares keys with == instead of equals().

15. Difference between HashMap and WeakHashMap?

→ WeakHashMap allows GC of keys.

16. ⭐ Which is faster: HashMap or TreeMap?

→ HashMap (O(1)) vs TreeMap (O(log n)).

17. What is EnumSet?

→ High-performance Set for enums.

18. What is CopyOnWriteArrayList?

→ Thread-safe ArrayList alternative.

19. ⭐ When to use LinkedHashSet?

→ When uniqueness + insertion order needed.

20. Which collection allows duplicates and null?

→ List.

## 6. Multithreading

1. ⭐ What is multithreading?

Running multiple threads concurrently.

2. Ways to create thread?

Extend Thread or implement Runnable.

3. ⭐ Difference between process and thread?

Process = independent, thread = lightweight sub-process.

4. What is synchronization?

Controlling access to shared resources.

5. ⭐ Difference between synchronized method and block?

Method locks entire object, block locks part.

6. What is deadlock?

Two threads waiting on each other’s resources.

7. ⭐ Difference between wait() and sleep()?

wait releases lock, sleep doesn’t.

8. What is ThreadLocal?

Provides thread-specific variable.

9. What is daemon thread?

Background thread that ends when JVM ends.

10. ⭐ Executor framework?

High-level API for managing thread pools.

## 7. Exception Handling

1. ⭐ Difference between checked and unchecked exception?

Checked = compile-time, unchecked = runtime.

2. What is finally block?

Always executes, even if exception occurs.

3. ⭐ Can finally block be skipped?

Yes, on System.exit().

4. What is throw vs throws?

throw = used to throw, throws = method declaration.

5. ⭐ Difference between error and exception?

Error = unrecoverable, Exception = recoverable.

6. Custom exception?

By extending Exception or RuntimeException.

7. ⭐ Multiple catch blocks?

Yes, ordered from specific to general.

## 8. JVM & Memory Management

1. ⭐ What is JVM?

Java Virtual Machine, runs bytecode.

2. Difference between JDK, JRE, JVM?

JDK = development kit, JRE = runtime, JVM = virtual machine.

3. ⭐ JVM memory areas?

Heap, Stack, Metaspace, PC Register, Native Method Stack.

4. What is garbage collection?

Automatic memory cleanup of unused objects.

5. ⭐ How to request garbage collection?

Using System.gc().

6. What is stack vs heap?

Stack = method calls/variables, Heap = objects.

7. ⭐ ClassLoader?

Loads classes into memory (Bootstrap, Extension, Application).

## 9. Java 8 Features

1. ⭐ Key Java 8 features?

Lambda, Streams, Optional, Default methods, Date/Time API.

2. What is lambda expression?

Short form of anonymous function.

3. ⭐ What are streams?

API to process collections in functional style.

4. Difference between map() and flatMap()?

map = transform, flatMap = flatten + transform.

5. ⭐ What is Optional?

Wrapper to avoid NullPointerException.

6. What is functional interface?

Interface with one abstract method (@FunctionalInterface).

7. ⭐ Difference between intermediate and terminal stream ops?

Intermediate = return stream, Terminal = return result.

## 10. Spring Basics

1. ⭐ What is Spring Framework?

Java framework for building enterprise apps.

2. What is Dependency Injection?

Externalizing object creation and dependencies.

3. ⭐ Types of dependency injection?

Constructor, Setter.

4. What is IoC Container?

Manages lifecycle of beans.

5. ⭐ Difference between BeanFactory and ApplicationContext?

ApplicationContext = superset with more features.

6. What are Spring annotations?

@Component, @Autowired, @Configuration, etc.

7. ⭐ What is Spring Boot?

Simplifies Spring with auto-configuration.

## 11. SQL & Java

1. ⭐ Difference between SQL and MySQL?

SQL = language, MySQL = database software.

2. What is JDBC?

API to connect Java with databases.

3. ⭐ Steps to connect DB in JDBC?

Load driver, establish connection, create statement, execute query, close.

4. What is PreparedStatement?

Precompiled SQL query for efficiency & security.

5. ⭐ Difference between Statement and PreparedStatement?

Prepared prevents SQL injection, faster.

6. What is ResultSet?

Object holding query results.

7. ⭐ What is transaction?

Group of operations executed as single unit.

8. What are ACID properties?

Atomicity, Consistency, Isolation, Durability.

## 🔑 Core Java – Interview Q&A

### OOPs Concepts

### 1. ⭐ What are the four pillars of OOP in Java? Explain with examples.

- **Encapsulation**: Wrapping data + methods inside a class (e.g., private fields with getters/setters).
- **Inheritance**: Reusing parent class features in child class (extends).
- **Polymorphism**: Many forms → Compile-time (method overloading), Runtime (method overriding).
- **Abstraction**: Hiding implementation using abstract classes & interfaces.

---

### 2. ⭐ Difference between Encapsulation and Abstraction?

- **Encapsulation**: Hides **data** using access modifiers (focus on "how data is accessed").
- **Abstraction**: Hides **implementation details** using abstract classes/interfaces (focus on "what to do").

---

### 3. How is inheritance implemented in Java?

Using the extends keyword for classes and implements for interfaces. Example:

class Animal {}

class Dog extends Animal {}

---

### 4. ⭐ What is method overloading vs method overriding (compile-time vs runtime polymorphism)?

- **Overloading (Compile-time polymorphism):** Same method name, different parameters.
- **Overriding (Runtime polymorphism):** Subclass provides its own implementation of parent class method.

---

### 5. ⭐ Why is multiple inheritance not supported in Java? How is it achieved?

- To avoid **diamond problem** (ambiguity).

- Achieved via **interfaces** (implements multiple interfaces).

### 6. What are real-world examples of OOP concepts in Java?

- **Encapsulation**: ATM machine hiding PIN logic.
- **Inheritance**: Car extends Vehicle.
- **Polymorphism**: draw() method works differently for Circle, Square.
- **Abstraction**: Abstract class Shape defines draw(), implementation hidden.

### 7. What is the difference between is-a and has-a relationship?

- **is-a**: Inheritance (Dog is-a Animal).
- **has-a**: Composition (Car has-a Engine).

### 8. ⭐ Can we override a private / static / final method? Why or why not?

- **Private**: No (not visible to subclass).
- **Static**: No (they belong to class, not object → method hiding happens).
- **Final**: No (final means cannot be changed).

## Class, Object, Constructor

### 9. ⭐ What is the difference between class and object in Java?

- **Class**: Blueprint/template (e.g., Car class).
- **Object**: Instance of class (e.g., Car myCar = new Car()).

### 10. What are the types of constructors in Java?

1. **Default constructor** – No args, created automatically if not defined.
2. **Parameterized constructor** – Accepts parameters.
3. **Copy constructor** – Copies another object (not built-in, manually defined).

### 11. ⭐ What is a copy constructor in Java? Is it built-in?

- Java does not have a built-in copy constructor like C++.
- You can create manually:

```
class Student {
 String name;
 Student(Student s) { this.name = s.name; }
}
```

### 12. Can a constructor be final, static, or abstract? Why?

- **Final**: Not allowed (constructors aren’t inherited).
- **Static**: Not allowed (constructors are tied to objects).
- **Abstract**: Not allowed (constructors can’t be incomplete).

### 13. ⭐ Difference between constructor overloading and method overloading?

- **Constructor overloading**: Multiple constructors with different parameter lists.
- **Method overloading**: Multiple methods with same name but different params.

### 14. What is the role of default constructor?

Initializes objects with default values. Created automatically if no constructor is defined.

### 15. ⭐ What happens if you don’t define any constructor in a class?

The compiler automatically provides a **default no-arg constructor**.

## Static & Final

### 16. ⭐ What is the use of the static keyword in Java?

- Belongs to class, not object. Used for:
    - Static variables → shared by all objects.
    - Static methods → called without object.
    - Static blocks → run once when class loads.

**17. What is the difference between static variables, static methods, and static blocks?**

- **Static variable**: Common across all objects.
- **Static method**: Belongs to class, can’t access instance variables directly.
- **Static block**: Executes once when class is loaded.

**18. ⭐ Difference between final, finally, and finalize()?**

- **final**: Keyword → constant, no override, no inheritance.
- **finally**: Block → executes always in exception handling.
- **finalize()**: Method → called before garbage collection.

**19. Can we declare a constructor static or final? Why?**
No (explained in Q12).

**20. What happens if you declare a final variable but don’t initialize it?**
Compiler error → must be initialized at declaration or inside constructor.

## This & Super Keywords

**21. ⭐ Difference between this and super keywords.**

- **this**: Refers to current object.
- **super**: Refers to parent class (methods/constructors).

**22. Can we use this() and super() in the same constructor? Why?**
No → both must be first statement, so only one can exist.

**23. ⭐ What is the role of this keyword in constructor chaining?**
Used to call another constructor within same class.

### 24. How does super() work in constructor calls (default vs parameterized)?

- **Default super()**: Automatically called if not written.
- **Parameterized super()**: Must be explicitly called if parent doesn't have default constructor.

---

## Method Overloading vs Method Overriding

### 25. ⭐ What are the key differences between overloading and overriding?

- Overloading → Compile-time, same name but diff params, same class.
- Overriding → Runtime, same name + params, parent-child classes.

---

### 26. Can we overload main() method in Java?

Yes, but JVM always calls the standard public static void main(String[] args) only.

---

### 27. ⭐ Can we override a method with different return types?

Yes, but return type must be **covariant** (subclass of parent return type).

---

### 28. Difference between overloading + overriding + hiding?

- **Overloading**: Same method name, different params.
- **Overriding**: Subclass changes parent's method.
- **Hiding**: Subclass defines static method with same signature as parent.

---

### 29. ⭐ What is covariant return type in overriding?

Subclass method can return a more specific type than parent. Example:

```
class A { A get() { return this; } }

class B extends A { B get() { return this; } }
```

---

## Access Modifiers

### 30. ⭐ Explain all four access modifiers in Java.

- **public**: Accessible everywhere.

- **private**: Within same class only.
- **protected**: Same package + subclasses.
- **default (no modifier):** Same package only.

---

### 31. ⭐ Difference between default and protected access modifier.

- **Default**: Only within same package.
- **Protected**: Same package + subclasses (even in other packages).

---

### 32. Can we reduce the visibility of an overridden method?

No → but we can **increase visibility** (e.g., protected → public).

---

### 33. ⭐ What is the difference between public class and default class?

- **public class**: Can be accessed from any package.
- **default class**: Accessible only within same package.

## Class, Object, Constructor

### Q11. ⭐ What is the difference between class and object in Java?

- A **class** is a blueprint or template that defines attributes (fields) and behaviors (methods).
- An **object** is an instance of a class created in memory using the new keyword.

👉 Example:

```
class Car {
  String color;
  void drive() { System.out.println("Car is driving"); }
}

public class Main {
  public static void main(String[] args) {
    Car obj = new Car();  // obj is an object of class Car
  }
}
```

### Q12. What are the types of constructors in Java?

1. **Default constructor** – No parameters, provided by compiler if not defined.
2. **Parameterized constructor** – Accepts arguments to initialize objects.
3. **Copy constructor (user-defined)** – Copies values from another object.

### Q13. ⭐ What is a copy constructor in Java? Is it built-in?

- Java does **not provide a built-in copy constructor** (unlike C++).
- You can define one manually:

```
class Student {
  String name;
  Student(Student s) { this.name = s.name; }  // Copy constructor
}
```

### Q14. Can a constructor be final, static, or abstract? Why?

- **Final** ❌ – Not possible, because constructors are not inherited.
- **Static** ❌ – Not possible, because constructors are called when creating objects, not at class level.
- **Abstract** ❌ – Not possible, because constructor must be implemented and cannot be abstract.

### Q15. ⭐ Difference between constructor overloading and method overloading?

- **Constructor overloading** – Multiple constructors with different parameters.
- **Method overloading** – Same method name but different parameter list.

**Q16. What is the role of default constructor?**

- Initializes objects with default values (null, 0, false).
- If no constructor is defined, compiler adds a default one automatically.

---

**Q17. ⭐ What happens if you don't define any constructor in a class?**

- Compiler automatically provides a **default constructor** with no arguments.
- Example:

```
class Test { }  // compiler adds Test() { }
```

**Static & Final**

**Q18. ⭐ What is the use of the static keyword in Java?**

- Belongs to class, not object.
- Used with variables, methods, blocks, nested classes.

**Q19. What is the difference between static variables, static methods, and static blocks?**

- **Static variable** – shared across all objects.
- **Static method** – can be called without object, can't use this.
- **Static block** – runs once when class is loaded.

**Q20. ⭐ Difference between final, finally, and finalize()?**

- **final** – keyword (variable = constant, method = cannot override, class = cannot inherit).
- **finally** – block that always executes in try-catch-finally.
- **finalize()** – method called by GC before destroying object.

**Q21. Can we declare a constructor static or final? Why?**

- ❌ Not possible (constructors are tied to objects, not class).

**Q22. What happens if you declare a final variable but don't initialize it?**

- Compiler error (must be initialized at declaration or in constructor).

---

### This & Super Keywords

**Q23. ⭐ Difference between this and super keywords.**

- **this** – refers to current class instance.
- **super** – refers to immediate parent class instance.

**Q24. Can we use this() and super() in the same constructor? Why?**

- ❌ No, because both must be the first statement in a constructor.

**Q25. ⭐ What is the role of this keyword in constructor chaining?**

- Used to call another constructor in the same class.

**Q26. How does super() work in constructor calls (default vs parameterized)?**

- Default: super() is inserted automatically.
- Parameterized: explicitly call super(args) to parent constructor.

**Q27. ⭐ Key differences (this vs super)?**

- this() → same class constructor.
- super() → parent class constructor.

---

✅ That covers up to **Q27**.
Do you want me to **dump the full Q1–Q90 with answers in a single doc format (like a PDF/Word)** so you get everything at once without cutoff?

**You said:**

continue

**ChatGPT said:**

Perfect 👍 we already finished till **Q27 (This & Super)**.
Now let's move to the next block:

---

### Method Overloading vs Method Overriding

**Q28. ⭐ What are the key differences between overloading and overriding?**

- **Overloading (Compile-time polymorphism):** Same method name, different parameter list, resolved at compile time.

- **Overriding (Runtime polymorphism):** Child class redefines parent class method, resolved at runtime.

**Q29. Can we overload main() method in Java?**

- ✅ Yes, but JVM only calls public static void main(String[] args) as entry point.

**Q30. ⭐ Can we override a method with different return types?**

- ✅ Yes, but only with **covariant return type** (child type of original return).

**Q31. Difference between overloading + overriding + hiding?**

- **Overloading** – same method name, diff parameters.
- **Overriding** – child redefines parent method.
- **Method hiding** – static methods in child hide parent static methods (not true overriding).

**Q32. ⭐ What is covariant return type in overriding?**

- Overriding method can return subclass of the return type declared in the parent class.

---

### Access Modifiers

**Q33. ⭐ Explain all four access modifiers in Java.**

- **public** – accessible everywhere.
- **protected** – accessible within package + subclasses outside package.
- **default** – package-private, accessible only in same package.
- **private** – accessible only within class.

**Q34. ⭐ Difference between default and protected access modifier.**

- **default** – only within package.
- **protected** – package + subclasses (outside package).

**Q35. Can we reduce the visibility of an overridden method?**

- ❌ No. We can only **increase** visibility (private → protected → public).

**Q36. ⭐ What is the difference between public class and default class?**

- **public class** – accessible everywhere, must match filename.

- **default class** – accessible only within package, no access modifier.

---

### Interfaces & Abstract Classes

**Q37. ⭐ Difference between interface and abstract class.**

- **Interface** – 100% abstraction (till Java 7), multiple inheritance, only method signatures.
- **Abstract class** – partial abstraction, can have constructors, fields, methods.

**Q38. ⭐ Can an interface have a constructor? Why not?**

- ❌ No, because interfaces are not instantiable.

**Q39. What are marker interfaces in Java?**

- Interfaces with no methods (e.g., Serializable, Cloneable). Used to provide metadata to JVM/compiler.

**Q40. ⭐ What is functional interface (Java 8)? Give examples.**

- Interface with exactly **one abstract method**.
- Examples: Runnable, Callable, Comparator.

**Q41. Can an interface extend multiple interfaces?**

- ✅ Yes (supports multiple inheritance of type).

**Q42. ⭐ Difference between default methods and static methods in interface.**

- **Default method** – has body, can be overridden by implementing class.
- **Static method** – belongs to interface, cannot be overridden.

---

### Packages

**Q43. ⭐ What is the difference between built-in packages (like java.util) and user-defined packages?**

- **Built-in** – provided by Java (java.util, java.io).
- **User-defined** – created by developer using package keyword.

**Q44. How do we import a package in Java?**

- Using import package_name.*; or import package_name.ClassName;.

### Q45. ⭐ Can we have two classes with the same name in different packages?

- ✅ Yes, but must be accessed with fully qualified name if conflict.

---

## Inner Classes

### Q46. ⭐ What are the types of inner classes in Java?

1. Member inner class
2. Static nested class
3. Local inner class
4. Anonymous inner class

### Q47. What is the difference between static nested class and inner class?

- **Static nested** – doesn't need outer class object.
- **Inner class** – needs outer class object to access members.

### Q48. ⭐ What are anonymous inner classes?

- Inner classes without a name, used for **one-time use** (often with Runnable, listeners).

### Q49. Can an interface be defined inside a class?

- ✅ Yes, interface inside class is implicitly **static**.

## Collections Framework

### Q50. ⭐ What is the difference between Collection and Collections?

- **Collection** – root interface of Java Collections framework (List, Set, Queue).
- **Collections** – utility class with static methods (e.g., sort(), reverse()).

### Q51. ⭐ Difference between List, Set, and Map?

- **List** – ordered, allows duplicates (ArrayList, LinkedList).
- **Set** – unordered, no duplicates (HashSet, TreeSet).
- **Map** – key-value pairs, keys unique (HashMap, TreeMap).

### Q52. Explain differences between ArrayList and LinkedList.

- **ArrayList** – fast random access, slower insertion/deletion.
- **LinkedList** – faster insertion/deletion, slower random access.

**Q53. ⭐ Difference between HashMap and HashTable.**

- **HashMap** – non-synchronized, allows one null key, faster.
- **Hashtable** – synchronized, doesn’t allow null keys/values, legacy.

**Q54. ⭐ Difference between HashSet and TreeSet.**

- **HashSet** – unordered, uses hash table.
- **TreeSet** – sorted, uses Red-Black tree.

**Q55. What is the difference between Iterator and ListIterator?**

- **Iterator** – works for Collection, only forward traversal.
- **ListIterator** – works only on List, supports bidirectional traversal + add/set.

**Q56. ⭐ Fail-fast vs Fail-safe iterators.**

- **Fail-fast** – throws ConcurrentModificationException (e.g., ArrayList iterator).
- **Fail-safe** – works on cloned copy, no exception (e.g., CopyOnWriteArrayList).

---

### Comparable vs Comparator

**Q57. ⭐ Difference between Comparable and Comparator.**

- **Comparable** – natural ordering, compareTo(), only one sorting.
- **Comparator** – custom ordering, compare(), multiple sortings possible.

**Q58. Can a class implement both Comparable and Comparator?**

- ✅ Yes, but usually implements Comparable and has separate Comparator classes.

**Q59. ⭐ Real-world example where Comparator is useful.**

- Sorting employees by **salary, name, age** dynamically.

---

### Generics in Java

**Q60. ⭐ What are generics? Why are they used?**

- Provide **type-safety** and **eliminate type-casting**.

- Example: List<String> list = new ArrayList<>();.

### Q61. ⭐ What is type erasure in generics?

- At compile time, generics are enforced; at runtime, they are erased (type replaced by Object).

### Q62. Difference between bounded and unbounded wildcards?

- **Unbounded** – <?> → any type.
- **Bounded** – <? extends Number> or <? super Integer>.

### Q63. ⭐ Can we use primitive types with generics?

- ❌ No. Generics only support objects. Use wrapper classes (Integer, Double).

---

## Multithreading & Concurrency

### Q64. ⭐ Difference between process and thread?

- **Process** – independent execution, heavy.
- **Thread** – lightweight, shares process memory.

### Q65. ⭐ Difference between Thread class and Runnable interface?

- **Thread class** – extend class, override run().
- **Runnable** – implement interface, pass to Thread object.

### Q66. ⭐ What is synchronization in Java?

- Used to control thread access to shared resources (via synchronized keyword).

### Q67. What is deadlock?

- When two or more threads wait forever on resources locked by each other.

### Q68. ⭐ Difference between sleep() and wait()?

- **sleep()** – pauses thread for given time, doesn't release lock.
- **wait()** – releases lock, waits until notified.

### Q69. ⭐ Difference between volatile and synchronized?

- **volatile** – ensures variable value is read from main memory.
- **synchronized** – provides lock to control thread access.

### Q70. What is Executor framework in Java?

- Provides thread pool management (ExecutorService, Executors).

---

## Exception Handling

### Q71. ⭐ What is the difference between checked and unchecked exceptions?

- **Checked exceptions** – checked at compile time (e.g., IOException, SQLException).
- **Unchecked exceptions** – runtime errors (e.g., NullPointerException, ArithmeticException).

### Q72. ⭐ Difference between throw and throws.

- **throw** – used to explicitly throw an exception (throw new Exception() inside method).
- **throws** – declares exceptions in method signature (void m() throws IOException).

### Q73. ⭐ Difference between final, finally, and finalize().

- **final** – keyword (constant, prevent override/inheritance).
- **finally** – block in exception handling, always executes.
- **finalize()** – method called by GC before destroying object.

### Q74. Can we have try block without catch?

- ✅ Yes, if followed by **finally**.

try { … } finally { … }

### Q75. ⭐ What happens if exception occurs in finally block?

- If exception is not handled, it **overrides** exception from try/catch.

### Q76. What are custom exceptions?

- User-defined exceptions by extending Exception or RuntimeException.

---

## Memory Management

### Q77. ⭐ Difference between stack and heap memory?

- **Stack** – stores local variables, function calls, faster.
- **Heap** – stores objects, shared among threads, garbage collected.

### Q78. ⭐ How does Garbage Collection (GC) work in Java?

- GC removes unreachable objects from heap.
- Uses algorithms like **Mark and Sweep**.

### Q79. What is the difference between finalize() and Garbage Collector?

- finalize() – called before GC destroys object (but not guaranteed).
- GC – automatic memory management, frees unused objects.

### Q80. ⭐ Can we force Garbage Collection in Java?

- ❌ No guarantee. We can **request** with System.gc(), but JVM decides.

### Q81. ⭐ What is memory leak in Java?

- When objects are not used but still referenced, preventing GC from reclaiming them.

## Strings

### Q82. ⭐ Why are Strings immutable in Java?

- For **security, caching, synchronization, and class loading**.
- Immutable means once created, cannot be changed → ensures thread safety & efficient memory via **String pool**.

### Q83. Difference between String, StringBuilder, and StringBuffer.

- **String** – immutable, stored in string pool.
- **StringBuilder** – mutable, not thread-safe, faster.
- **StringBuffer** – mutable, thread-safe (synchronized).

### Q84. ⭐ What is String Pool in Java?

- Special memory area inside **heap** where string literals are stored.
- Reuses objects → saves memory.

### Q85. What is the difference between == and .equals() in Strings?

- == → compares **reference (memory address)**.
- .equals() → compares **content (values)**.

### Q86. ⭐ How to make Strings mutable?

- Use **StringBuilder** or **StringBuffer**.

## Java 8 Features

### Q87. ⭐ What are lambda expressions in Java 8?

- Short way to implement functional interfaces.

```
(x, y) -> x + y
```

### Q88. ⭐ What is Streams API in Java 8?

- Functional-style operations on collections (map, filter, reduce).
- Improves readability and performance.

### Q89. ⭐ What are functional interfaces?

- Interface with **only one abstract method**.
- Examples: Runnable, Callable, Comparator.

### Q90. ⭐ What are default and static methods in interface (Java 8)?

- **default** – method with implementation inside interface.
- **static** – utility methods inside interface.

### Q91. ⭐ What is Optional in Java 8?

- Wrapper class to avoid **NullPointerException**.
- Example:

```
Optional<String> name = Optional.ofNullable(null);
```

## Java 11+ Features

### Q92. What is the var keyword in Java 10/11?

- Local variable type inference.

```
var list = new ArrayList<String>();
```

### Q93. ⭐ What are new String methods in Java 11?

- isBlank(), lines(), repeat(int n), strip().

### Q94. What is the difference between Java 8 and Java 11?

- Java 8 → Lambdas, Streams, Optional.

- Java 11 → var, new String methods, HTTP Client API, removed JavaFX.