Interview Questions

JAVA-112: Polymorphism

Answering interview questions is crucial in your journey of applied learning. Review them to ensure your understanding of important topics covered in the previous session and to prepare yourself for upcoming challenges. Remember that it's important to answer these questions on your own before viewing the solution. The solutions are hyperlinked to community posts on our platform.

**Note: The questions below have been sourced from previous interviews**

Questions

1. [Explain Method Overloading and Method Overriding.](https://crio.do/learn/crio-community/topic/explain-method-overloading-and-method-overriding/169124)
2. [What is another name for method overloading?](https://crio.do/learn/crio-community/topic/what-is-another-name-for-method-overloading/169100)
3. [What will happen if we change the access modifier of a method that is overriding a method from the parent class? Is this allowed?](https://crio.do/learn/crio-community/topic/what-will-happen-if-we-change-the-access-modifier-of-a-method-that-is-overriding-a-method-from-the-parent-class-is-this-allowed/169112)
4. [What is polymorphism and its types?](https://crio.do/learn/crio-community/topic/what-is-polymorphism-and-its-types/169116)
5. [How to make a class immutable?](https://crio.do/learn/crio-community/topic/how-to-make-a-class-immutable/169122)
6. [How does Java implement polymorphism?](https://crio.do/learn/crio-community/topic/how-does-java-implement-polymorphism/252726)
7. [What is the difference between method overloading and method overriding?](https://crio.do/learn/crio-community/topic/what-is-the-difference-between-method-overloading-and-method-overriding/252831)

**1. Explain Method Overloading and Method Overriding.**

**Method Overloading:**

* It happens when two or more methods in the same class have the **same name but different parameter lists** (different type, number, or order of parameters).
* It is an example of **compile-time (static) polymorphism**.

class Calculator {

int add(int a, int b) { return a + b; }

double add(double a, double b) { return a + b; }

}

**Method Overriding:**

* It happens when a **subclass provides a new implementation** of a method that is already defined in the parent class.
* It is an example of **runtime (dynamic) polymorphism**.

class Animal {

void sound() { System.out.println("Animal makes a sound"); }

}

class Dog extends Animal {

@Override

void sound() { System.out.println("Dog barks"); }

}

**2. What is another name for method overloading?**

👉 Method overloading is also known as **Compile-time Polymorphism** or **Static Polymorphism**.

**3. What will happen if we change the access modifier of a method that is overriding a method from the parent class? Is this allowed?**

* In **overriding**, the child class **cannot reduce the visibility** of the parent method.
* Allowed: You can increase visibility (e.g., from protected → public).
* Not allowed: You cannot decrease visibility (e.g., public → protected or private).  
  👉 If you try, the code will not compile.

Example:

class Parent {

protected void show() {}

}

class Child extends Parent {

@Override

public void show() {} // ✅ Allowed (visibility increased)

}

**4. What is polymorphism and its types?**

**Polymorphism:**  
It means "many forms." In Java, it allows an object to take multiple forms, enabling the same method or operator to behave differently based on context.

**Types of Polymorphism in Java:**

1. **Compile-time polymorphism** → Achieved by method overloading.
2. **Runtime polymorphism** → Achieved by method overriding and dynamic method dispatch.

**5. How to make a class immutable?**

To make a class immutable in Java:

1. Declare the class as final (so it cannot be subclassed).
2. Make all fields private and final.
3. Don’t provide setters.
4. Initialize fields through a constructor.
5. Return deep copies of mutable objects instead of exposing them directly.

Example:

final class Student {

private final String name;

private final int age;

public Student(String name, int age) {

this.name = name;

this.age = age;

}

public String getName() { return name; }

public int getAge() { return age; }

}

**6. How does Java implement polymorphism?**

* **Compile-time (static):** Method overloading, operator overloading (e.g., + for strings).
* **Runtime (dynamic):** Method overriding, achieved using **dynamic method dispatch**, where the JVM determines at runtime which method to invoke based on the actual object type.

**7. What is the difference between method overloading and method overriding?**

| **Feature** | **Method Overloading** | **Method Overriding** |
| --- | --- | --- |
| Definition | Same method name, different parameter list | Subclass provides new implementation of parent method |
| Polymorphism Type | Compile-time polymorphism | Runtime polymorphism |
| Return Type | Can be same or different (must change params) | Must be same (or covariant) |
| Access Modifiers | No restrictions | Cannot reduce parent’s visibility |
| Binding | Static binding (at compile time) | Dynamic binding (at runtime) |
| Inheritance | Not required | Requires inheritance |