Method Overloading in Java (In-Depth)

1. What is Method Overloading?

Method Overloading in Java means **defining multiple methods with the same name in the same class**, but with **different parameter lists**.

The parameter list can differ by:

1. Number of parameters
2. Type of parameters
3. Order of parameters

The return type alone is **not considered** for overloading.

---

2. Why Method Overloading is called Compile-Time Polymorphism

In method overloading, the **compiler decides which method to execute** by looking at the method call and its arguments **before the program runs**.

- Method binding happens at **compile time**
- JVM only executes the already decided bytecode

Hence, method overloading is known as:

- Compile-Time Polymorphism
- Static Polymorphism

---

3. Why Method Overloading is also called False Polymorphism

In method overloading, we often assume that:

One method is doing multiple tasks

But in reality:

- There are **multiple different methods**
- They only share the **same method name**
- The compiler chooses **one exact method**

There is **no dynamic behavior at runtime**.

Because of this assumption-versus-reality mismatch, method overloading is also called **False Polymorphism**.

---

4. Compiler Method Selection Rules (Very Important)

When a method call is made, the compiler follows these rules:

1. Exact match
2. Widening (type promotion: int → double)
3. Var-args
4. If more than one method matches equally → **compile-time error**

---

5. Type Promotion Example (No Exact Match)

```
class Test {

    void add(int a, int b) {
        System.out.println("int, int");
    }

    void add(int a, int b, double c) {
        System.out.println("int, int, double");
    }

    public static void main(String[] args) {
        Test t = new Test();
        t.add(4, 4, 4);
    }
}
```

Explanation:

- Method `(int, int, int)` does not exist
- `int` can be promoted to `double`
- Compiler selects `(int, int, double)`

6. Best Match Rule Example

```
class Test {

    void add(int a, int b, double c) {
        System.out.println("int int double");
    }

    void add(double a, double b, double c) {
        System.out.println("double double double");
    }

    public static void main(String[] args) {
        Test t = new Test();
        t.add(44, 44, 44);
    }
}
```

Explanation:

- `(int, int, double)` needs 1 promotion
- `(double, double, double)` needs 3 promotions
- Compiler selects the method with **minimum conversions**

---

7. Ambiguity in Method Overloading

```
class Test {

    void show(int a, double b) {
        System.out.println("int double");
    }

    void show(double a, int b) {
        System.out.println("double int");
    }

    public static void main(String[] args) {
        Test t = new Test();
        t.show(10, 20);
    }
}
```

Explanation:

- Both methods require one type promotion
- Compiler cannot decide the best match
- Results in **compile-time error: ambiguous method call**

8. Overloading the main() Method

Yes, the `main()` method **can be overloaded**.

```
class MainOverload {

    public static void main(String[] args) {
        System.out.println("Original main");
        main(10);
        main("Java");
    }

    public static void main(int a) {
        System.out.println("int main");
    }

    public static void main(String s) {
        System.out.println("String main");
    }
}
```

Important Points:

- JVM always starts execution from `public static void main(String[] args)`
- Overloaded main methods are **not invoked by JVM automatically**
- They must be called explicitly

---

9. Method Overloading vs Method Overriding (Quick Contrast)

Feature	Overloading	Overriding
Binding	Compile-time	Runtime
Polymorphism	Static / False	Dynamic / True
Inheritance	Not required	Required
Method signature	Must differ	Must be same

---

10. Important Interview Traps

- Return type alone cannot overload a method
- JVM does not resolve overloaded methods
- Ambiguity leads to compile-time error
- Overloading does not depend on object type
- Var-args has the **lowest priority** in method selection

11. One-Line Interview Definition

Method overloading is called compile-time or false polymorphism because the compiler resolves the method call based on parameter matching before execution, and no runtime decision is involved.

---

12. Key Memory Tip

- Overloading → Compiler → Early binding → Compile time
- Overriding → JVM → Late binding → Runtime