

Method Overloading in Java – Corrected Notes and Interview Preparation

1. Definition of Method Overloading

Method Overloading in Java is defined as:

Having **multiple methods with the same name** in the **same class**, but with a **different parameter list**.

A method is considered overloaded when the parameter list differs by:

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

Method overloading is resolved by the **Java compiler at compile time**, therefore it is known as:

- **Compile-Time Polymorphism**
 - **Static Polymorphism**
-

2. Example of Method Overloading

```
class Calculator {  
  
    void add(int a, int b) {  
        System.out.println(a + b);  
    }  
  
    void add(double a, double b) {  
        System.out.println(a + b);  
    }  
  
    void add(int a, int b, int c) {  
        System.out.println(a + b + c);  
    }  
}
```

Explanation:

- All methods have the same name: **add**
 - Overloading is achieved using **different parameter types and counts**
 - The compiler decides which method to call based on the arguments passed
-

3. Important Rules of Method Overloading

Method overloading **cannot be achieved** by changing only:

- Return type
- Access modifier

The **parameter list must change**.

Invalid Example

```
int add(int a, int b) { }  
double add(int a, int b) { }    // Compile-time error
```

Reason:

- Method signature is the same (add(int, int))
 - Return type is **not part of the method signature**
-

4. Method Overloading Across Languages

C Language

- Does **not support** method/function overloading
- C is **procedure-oriented**
- Functions are identified only by their names

C++ Language

- Supports method overloading
- C++ is an **object-oriented programming language**

Java Language

- Supports method overloading
- Provides **compile-time polymorphism**
- Encourages readable and reusable code

Java is sometimes informally described as “C++ **minus complexity**” because:

Java adds:

- Garbage Collection
- Platform independence
- JVM-based execution

Java removes or restricts:

- Pointers (direct memory access)
 - Multiple inheritance using classes
 - Operator overloading
-

5. Naming Conventions in Java

Following naming conventions improves:

- Code readability
 - Maintainability
 - Industry-level consistency
-

5.1 Class Naming Convention

- First letter should be uppercase
- Each word should start with an uppercase letter
- Uses **PascalCase**

Correct:

```
Student
StudentInfo
BankAccountDetails
```

Incorrect:

```
studentinfo
student_Info
studentInfo
```

5.2 Variable Naming Convention

- First letter should be lowercase
- Follow **camelCase**
- Use meaningful names

Correct:

```
studentName
totalMarks
accountBalance
```

Incorrect:

```
StudentName
student_name
```

Note:

- Snake case is common in C, Python, and databases
 - It is **not recommended** in Java
-

5.3 Method Naming Convention

- First letter should be lowercase
- Follow **camelCase**
- Method names should usually be verbs

Correct:

```
calculateSum()  
getStudentName()  
printDetails()
```

Incorrect:

```
CalculateSum()  
get_student_name()  
PrintDetails()
```

6. Additional Important Points (Exams and Interviews)

- Method overloading improves code readability and flexibility
- It supports polymorphism at compile time
- JVM does **not** resolve method overloading
- The **compiler** resolves method overloading

The `main()` method can also be overloaded:

```
public static void main(int a) { }  
public static void main(String[] args) { }
```

Only the standard `main(String[] args)` method is used as the JVM entry point.

7. One-Line Interview Answer

Method overloading allows multiple methods with the same name but different parameters, and it is resolved at compile time to achieve static polymorphism.