**1. Method Overloading**

**Definition:**  
Same method name in the **same class** (or in parent + child) but with **different parameter lists**.

**Why used:**

* To perform similar operations with different types or numbers of inputs.
* Improves **code readability** (no need to create different method names like addInt, addDouble).

**Key points:**

* Happens **at compile-time** → **Compile-time polymorphism**.
* Must have **different**:
  + Number of parameters **OR**
  + Types of parameters
* Return type **alone** cannot be different.
* Can overload **static**, **final**, and **private** methods.

**Example:**

class MathOps {

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

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

}

**2. Method Overriding**

**Definition:**  
Same method name, **same parameters**, but in **parent & child classes**.  
Child class **redefines** parent's method.

**Why used:**

* To provide a **specific implementation** for a method in the child class.
* Supports **runtime polymorphism**.

**Key points:**

* Happens **at runtime** → **Runtime polymorphism**.
* Must have **same**:
  + Method name
  + Parameter list
  + Return type (or covariant type)
* Access level **cannot be more restrictive**.
* Only **inherited methods** can be overridden.
* Use @Override annotation for clarity.

**Example:**

class Parent {

void greet() { System.out.println("Hello from Parent"); }

}

class Child extends Parent {

@Override

void greet() { System.out.println("Hello from Child"); }

}

**Quick Comparison Table**

| **Feature** | **Overloading** | **Overriding** |
| --- | --- | --- |
| **Polymorphism Type** | Compile-time | Runtime |
| **Parameter List** | Must be different | Must be same |
| **Return Type** | Can be different | Must be same/covariant |
| **Access Modifier** | Can be anything | Cannot be more restrictive |
| **Inheritance Needed** | No | Yes |