Polymorphism in Java is one of the core concepts of Object-Oriented Programming (OOP). It means **"many forms"** — the ability of a single function, method, or operator to behave differently based on the context.

There are **two types of polymorphism in Java**:

1. **Compile-time Polymorphism (Static Binding / Method Overloading)**
2. **Run-time Polymorphism (Dynamic Binding / Method Overriding)**

1. Compile-Time Polymorphism (Method Overloading)

Definition:
Method overloading occurs when multiple methods in the same class have the **same name** but **different parameters** (number, type, or sequence).

- **Example:**

```
class Calculator {
    // Overloaded methods
    int add(int a, int b) {
        return a + b;
    }

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

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

public class Main {
    public static void main(String[] args) {
        Calculator calc = new Calculator();
        System.out.println("Add 2 int: " + calc.add(10, 20));
```

```
        System.out.println("Add 2 double: " + calc.add(5.5, 4.5));
        System.out.println("Add 3 int: " + calc.add(1, 2, 3));
    }
}
```

🔹 **Output:**

Add 2 int: 30

Add 2 double: 10.0

Add 3 int: 6

2. Run-Time Polymorphism (Method Overriding)

Definition:
Method overriding occurs when a subclass provides a **specific implementation** of a method that is already defined in its superclass.

It is determined at **runtime**, depending on the **type of object** that is referred to by the reference variable.

🔸 **Example:**

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

class Dog extends Animal {
    @Override
    void sound() {
        System.out.println("Dog barks");
    }
}

class Cat extends Animal {
```

```
    @Override
    void sound() {
        System.out.println("Cat meows");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal a;

        a = new Dog();
        a.sound();  // Output: Dog barks

        a = new Cat();
        a.sound();  // Output: Cat meows
    }
}
```

- **Output:**

Dog barks

Cat meows

Here, although the reference is of type Animal, the method executed depends on the actual object (Dog or Cat) — demonstrating **run-time polymorphism**.

🔍 **Summary Table:**

Feature	Compile-Time Polymorphism	Run-Time Polymorphism
Other Name	Static Binding	Dynamic Binding
Achieved By	Method Overloading	Method Overriding
Resolution	At Compile Time	At Runtime
Flexibility	Less flexible	More flexible due to dynamic behavior
Inheritance	Not required	Required