Java Polymorphism

Table of contents

What is polymorphism? 1
Compile-time polymorphism (method overloading) 1
Runtime polymorphism (dynamic method dispatch) 2
Subtype polymorphism 2

Introduced in 2026-03-12-object-oriented-programming-ii[1] (Lecture, Week 3).

What is polymorphism?

From the Greek “poly” (many) and “morphe” (form). In programming, it allows the same method call to produce different behaviour depending on the object that executes it.

Compile-time polymorphism (method overloading)

Multiple methods share a name but differ in parameter list (type, number, or both). Which one runs is decided at **compile time**, based on the method signature (see java-encapsulation[2]#method-signatures):

```java
public class CompPolymorphism {
    static String print(int value) { return "number"; }
    static String print(String value) { return "text"; }

    public static void main(String[] args) {
        System.out.println(print(21));   // "number"
        System.out.println(print("21")); // "text"
    }
}
```

[1] courses/csse2002/2026-03-12-object-oriented-programming-ii.pdf
[2] courses/csse2002/java-encapsulation.pdf

Runtime polymorphism (dynamic method dispatch)

Also called dynamic method dispatch. The method that gets executed is determined at **runtime**, based on the object's actual type, not the variable's declared type — implemented through method overriding (see java-inheritance[3]):

> The subclass's overriding method executes, regardless of the compile-time type of the subclass instance. — *Effective Java*

```
public class Animal {
    public void makeSound() { System.out.println("Animal makes a sound"); }
}
public class Cat extends Animal {
    @Override public void makeSound() { System.out.println("Cat meows"); }
}
public class Dog extends Animal {
    @Override public void makeSound() { System.out.println("Dog barks"); }
}

Animal myAnimal = new Dog();
myAnimal.makeSound(); // "Dog barks" - even though the variable's type is Animal
```

Subtype polymorphism

Objects of different subclasses can be treated as objects of a common superclass or interface, since subclasses are subtypes of their superclass. Achieved through inheritance (`extends`) or interface implementation (`implements`) — code works with the abstraction (supertype) rather than a specific implementation:

```
public static void performAnimalSound(Animal animal) {
    animal.makeSound();
}

performAnimalSound(new Dog()); // "Dog barks"
performAnimalSound(new Cat()); // "Cat meows"
```

The same works via an interface instead of a shared superclass:

[3] courses/csse2002/java-inheritance.pdf

```java
public interface Animal { void makeSound(); }
public class Cat implements Animal { ... }
public class Dog implements Animal { ... }

// Can pass any class that implements the interface
performAnimalSound(new Dog());
performAnimalSound(new Cat());
```