Polymorphism

Josh Wilcox (jw14g24@soton.ac.uk)

February 12, 2025

Contents

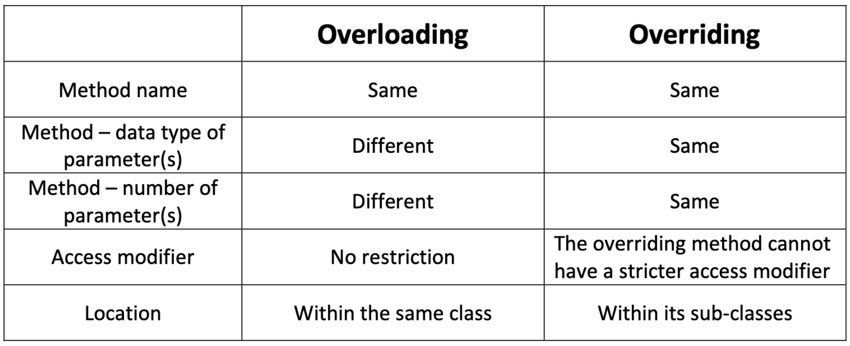
1 Overriding Methods

- If a subclass defines its own versions of methods that are defined in superclasses, we say it **overrides** those methods
- As `Dog extends Animal`, the `roam()` and `eat()` methods defined in `Dog` override the same methods defined in `Animal`
- When overriding, the signatures of the methods should be the same
- An overriding method's access modifier in a subclass can be more permissive (e.g., protected to the public) than the overridden method in the superclass. However, reducing the access level (e.g., making a protected method private) is not allowed and will result in a compile-time error.

```java
// A Simple Java program to demonstrate
// Overriding and Access-Modifiers
class Parent {
    // private methods are not overridden
    private void m1()
    {
        System.out.println("From parent m1()");
    }

    protected void m2()
    {
        System.out.println("From parent m2()");
    }
}

class Child extends Parent {
    // new m1() method
    // unique to Child class
    private void m1()
    {
        System.out.println("From child m1()");
    }

    // overriding method
    // with more accessibility
    @Override public void m2()
    {
        System.out.println("From child m2()");
    }
}

class Geeks {
    public static void main(String[] args)
    {
        // parent class object
        Parent P = new Parent();
        P.m2();
        // child class object
        Parent C = new Child();
        C.m2();
    }
}
```

Output:s

```
From parent m2()
From child m2()
```



	Overloading	Overriding
Method name	Same	Same
Method – data type of parameter(s)	Different	Same
Method – number of parameter(s)	Different	Same
Access modifier	No restriction	The overriding method cannot have a stricter access modifier
Location	Within the same class	Within its sub-classes

2 Dynamic Binding

- In Java, dynamic bin ding (or late binding) determines at runtime which overridden method to call.
- It is a key feature of polymorphism, ensuring that the method corresponding to the object's actual type is executed.
- This mechanism allows a superclass reference to invoke subclass methods that override the superclass methods.
- Dynamic binding promotes flexibility and extensibility, enabling behavior to be modified by subclassing.

2.1 Upcasting

Upcasting involves treating a subclass object as an instance of its superclass. When a method is invoked on an upcasted object, Java uses dynamic binding to determine which implementation to execute at runtime. The process is as follows:

- At compile time, Java verifies that the method exists in the declared type (the superclass). If the method is absent, a compile-time error occurs.
- At runtime, Java identifies the actual object's class and invokes the overridden method from that class.

2.1.1 Larger Hierarchy of Classes

- In a larger hierarchy, the principles of method overriding and dynamic binding still apply.
- Each subclass can override methods from its superclass, and dynamic binding ensures the correct method is called at runtime.

```java
// A Java program to demonstrate a larger hierarchy
class Animal {
    void sound() {
        System.out.println("Animal makes a 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");
    }
}

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

public class Main {
    public static void main(String[] args) {
        Animal myAnimal = new Animal();
        Animal myDog = new Dog();
        Animal myCat = new Cat();
        Dog myBulldog = new Bulldog();

        myAnimal.sound();  // Animal makes a sound
        myDog.sound();     // Dog barks
        myCat.sound();     // Cat meows
        myBulldog.sound(); // Bulldog growls
    }
}
```

Output:

```
Animal makes a sound
Dog barks
Cat meows
Bulldog growls
```

3 Substitution

- Substitution asserts that objects of a superclass should be replaceable with objects of a subclass without affecting the correctness of the program.
- Also known as the Liskov Substitution Principle (LSP), it establishes that a subclass must extend the behavior of its superclass without introducing unexpected behavior.
- This ensures that any function expecting an object of the superclass should work correctly with an object of the subclass.
- For example, if you have a method that processes an object of type Animal, it should work seamlessly even if the object is of type Dog or Cat, as long as they extend Animal appropriately.
- By following this principle, code becomes more modular, reusable, and easier to maintain.

4 Polymorphism

- Substitution, Overriding, and Dynamic Binding gives us **Polymorphism**
- Polymorphism means we can create methods that deal with **superclasses** and then:
  - We can pass them as instances of sub-classes (substitution)
  - When methods on those sub-classes are called, there are more specific methods defined (via overriding)
  - The function call gets diverted at run-time to the most specific method (dynamic binding)