Lesson 7: Inheritance

Marcin Kurzyna

Lecture Goals

By the end of this lesson, students should be able to:

- Explain what **inheritance** is in object-oriented programming.
- Create classes that inherit from other classes.
- Use the keyword `super`.
- Understand and apply **method overriding**.
- Understand class hierarchies.

1 What Is Inheritance?

Inheritance allows you to create new classes based on existing ones.

A new class:

- is called a **subclass** (child class),
- extends an existing **superclass** (parent class).

Inheritance enables:

- reusing code,
- organizing code more efficiently,
- building class hierarchies.

2 Basic Inheritance Example

```java
class Animal {
    public void eat() {
        System.out.println("Animal is eating");
    }
}

class Dog extends Animal {
    public void bark() {
        System.out.println("Dog is barking");
    }
}
```

Usage:

```
Dog d = new Dog();
d.eat();  // inherited method
d.bark(); // method from Dog class
```

3 The extends Keyword

To make one class inherit from another, you use the `extends` keyword.

```
class ChildClass extends ParentClass {
    // ...
}
```

A subclass:

- inherits all **public** and `protected` members,
- does not inherit constructors.

4 Constructors and the super Keyword

Although constructors are not inherited, a subclass can call its superclass constructor using `super()`.

```
class Person {
    protected String name;

    Person(String name) {
        this.name = name;
    }
}

class Student extends Person {
    private int grade;

    Student(String name, int grade) {
        super(name);      // calls superclass constructor
        this.grade = grade;
    }
}
```

5 Method Overriding

A subclass can **override** a method from its superclass to change its behavior.

```
class Animal {
    public void makeSound() {
        System.out.println("Some generic sound");
    }
}
```

```java
class Cat extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Meow");
    }
}
```

Rules of overriding:

- same method name,
- same parameter list,
- same or wider access level (e.g., `public` instead of `protected`).

6 Class Hierarchies

You can build entire inheritance trees:

```java
class Vehicle { ... }
class Car extends Vehicle { ... }
class ElectricCar extends Car { ... }
```

The class `ElectricCar` inherits everything from both `Car` and `Vehicle`.

7 The `final` Keyword

`final` can be used to restrict inheritance:

- `final class` – cannot be extended,
- `final method` – cannot be overridden.

```java
final class Robot { }
// class Android extends Robot {} // Error!
```

Summary

In this lecture you learned:

- Inheritance lets you create subclasses that extend existing classes.
- The `extends` keyword defines inheritance.
- `super()` is used to call a superclass constructor.
- Method overriding allows subclasses to change method behavior.
- Class hierarchies help structure code.

8 Exercises

1. Create a class `Animal` and subclasses `Dog` and `Cat`. Add appropriate methods and test inheritance.
2. Implement the following classes:
   - `Vehicle` (field: `speed`),
   - `Car` extending `Vehicle`,
   - `Truck` extending `Vehicle`.
3. Create classes `Person` and `Employee`. Use `super()` to call the superclass constructor.
4. Override the method `makeSound()` in several animal subclasses.
5. (Challenge) Design an RPG character class hierarchy:
   - `Character`,
   - `Warrior`, `Mage`, `Archer`,
   - override the `attack()` method in each subclass.