Lesson 5: Encapsulation and Access Modifiers

Marcin Kurzyna

Lecture Goals

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

- Understand and explain the concept of **encapsulation**.
- Use **access modifiers** to control visibility of fields and methods.
- Create **getter and setter** methods.
- Understand the use of the keyword `this`.

1 What is Encapsulation?

Encapsulation is one of the core principles of Object-Oriented Programming.
It means:

- Keeping the internal data of an object **hidden** (private).
- Providing controlled access through **public** methods.

2 Access Modifiers

Java provides several access levels:

- **public**: accessible from anywhere.
- **private**: accessible only inside the same class.
- **protected**: accessible in the same package and subclasses.
- (default): accessible in the same package.

3 Example Without Encapsulation (Not Recommended)

```
class BankAccount {
    String owner;
    double balance;
}
```

Anyone can change `balance` directly, even to negative values, which is unsafe.

4 Encapsulated Version (Recommended)

```
class BankAccount {
    private String owner;
    private double balance;

    BankAccount(String owner, double initialBalance) {
        this.owner = owner;
        this.balance = initialBalance;
    }

    public void deposit(double amount) {
        balance += amount;
    }

    public void withdraw(double amount) {
        if(balance >= amount) {
            balance -= amount;
        } else {
            System.out.println("Not enough money!");
        }
    }

    public double getBalance() {
        return balance;
    }
}
```

5 The `this` Keyword

The keyword `this` refers to the **current object**.

```
class Person {
    private String name;

    Person(String name) {
        this.name = name; // 'this.name' is the field, 'name' is
           the constructor parameter
    }
}
```

6 Getters and Setters

Getters return the value of fields. **Setters** allow controlled modification.

```
class Person {
    private String name;
    private int age;

    Person(String name, int age) {
```

```
        this.name = name;
        setAge(age);
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        if(age >= 0) {
            this.age = age;
        }
    }
}
```

7 Why Use Encapsulation?

- Protects data from invalid changes.
- Makes code easier to maintain.
- Allows changes inside the class without affecting other code.

Summary

In this lecture you learned:

- Encapsulation hides internal object data and controls access.
- Use **private** fields to protect data.
- Use **public getters and setters** to access data safely.
- The keyword `this` refers to the current object.

8 Exercises

1. Modify your `Car` class from the previous lecture to make all fields private.
2. Add getters and setters to your `Car` class.
3. Create a class `Temperature` that stores a temperature in Celsius. Add methods to convert to Kelvin and Fahrenheit.
4. Write a class `Student` with a private field `name` and private field `averageGrade`. Add validation so the grade must be between 1 and 6.

5. Challenge: Create a class `Password` that stores a private password string and has methods `setPassword(String p)` and `check(String p)`.