Java Programming 1 - Week 10 Notes

Hia Al Saleh

November 7th, 2024

Contents

1 Object-Oriented Programming (OOP)

Object-Oriented Programming is a programming paradigm based on the concept of "objects", which can contain data in the form of fields, and code, in the form of procedures or methods. OOP focuses on using objects and classes to structure programs, enabling modular and reusable code.

1.1 Understanding Classes and Objects

To truly grasp OOP, it's essential to understand two key concepts:

- **Class:** A blueprint or template for creating objects.
- **Object:** An instance of a class, representing a specific entity with unique properties.

While classes define characteristics and behaviors, objects are the real instances of these classes. Every object generated from a class may share its structure but can possess unique values for its properties.

1.2 Example: Car as a Class

Consider a `Car` class:

- Properties like `wheels`, `color`, and `engineType` define the general attributes of any car.
- Methods like `accelerate()`, `brake()`, and `turn()` define actions or behaviors the car can perform.

When creating a `Car` object, we instantiate it with specific values, such as four wheels, red color, and an electric engine, giving the car its unique identity.

2 Components of a Class

A class in Java consists of various components, such as:

- **Properties (Attributes):** Characteristics of the class.
- **Methods:** Functions or behaviors associated with the class.
- **Constructor:** A special method for initializing new objects.
- **Access Modifiers:** Keywords that define the visibility and accessibility of classes, methods, and variables.

2.1 Class Example: Movie Class

- Properties: `title`, `director`, `releaseDate`, `actors`, etc.
- Methods: Actions like `play()`, `pause()`, `stop()`.

Using the `Movie` class template, each object (movie) will have its unique values for title, director, and so on.

3 Constructors

A constructor is a unique method that is automatically called when an object is created. It typically has the same name as the class and no return type. Constructors are used to initialize objects with specific values.

3.1 Creating a Constructor

```
public class Movie {
    private String title;
    private String releaseDate;

    public Movie(String title, String releaseDate) {
        this.title = title;
        this.releaseDate = releaseDate;
    }
}
```

In this example, `Movie` objects are initialized with a title and release date upon creation.

3.2 Default and Parameterized Constructors

- **Default Constructor:** A constructor with no parameters, providing default values to properties.
- **Parameterized Constructor:** Accepts parameters, allowing specific initialization of properties.

A `Movie` class can have both types to offer flexible object creation.

4 Encapsulation and Data Hiding

Encapsulation is the practice of restricting access to certain details and exposing only the necessary parts of an object. This is done by setting properties to private and using public methods (getters and setters) to access or modify them.

4.1 Private, Public, and Protected Access Modifiers

- **Private:** Only accessible within the same class.
- **Public:** Accessible from any class.
- **Protected:** Accessible within the same package or subclasses.

Using these modifiers allows control over the accessibility of class components, enhancing data security and modularity.

4.2 Example: Getter and Setter Methods

Getters and setters provide controlled access to private fields.

```
public class Movie {
    private String title;

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }
}
```

5 Inheritance in OOP

Inheritance allows a class to inherit properties and methods from another class. This facilitates code reuse and establishes a relationship between classes, such as parent (superclass) and child (subclass) classes.

5.1 Creating a Subclass

```
public class ActionMovie extends Movie {
    private int explosions;

    public ActionMovie(String title, String releaseDate,
        int explosions) {
        super(title, releaseDate);
        this.explosions = explosions;
    }
}
```

Here, `ActionMovie` inherits from `Movie` and adds its unique property, `explosions`.

6 Polymorphism

Polymorphism enables objects to be treated as instances of their parent class. This allows methods to have different implementations based on the object's actual class.

6.1 Method Overloading and Overriding

- **Overloading:** Defining multiple methods in the same class with the same name but different parameters.
- **Overriding:** Providing a specific implementation in a subclass for a method declared in its superclass.

Polymorphism is essential for designing flexible and reusable code.

7 Homework

For further understanding of the concepts covered, read Chapter 9 of the textbook, focusing on:

- Class creation and constructors.
- Access modifiers and encapsulation.
- Inheritance and polymorphism.

8 Topics for Next Week

In the upcoming week, we will cover:

- Superclasses, Subclasses, and Extending Classes.
- Advanced Inheritance concepts.
- Method Overloading and Overriding in more depth.