Encapsulation Exercises

1.**Basic Understanding**

1.1 Class: A blueprint or definition for creating objects.
Object: An actual instance of a class with real data and behavior

1.2 Key Differences of instance and class variables are:

Characteristic	**Instance Variable**	**Class Variable (Static Variable)**
Keyword	No special keyword	Declared with the `static` keyword
Memory	Each object has its own copy	Only one copy shared by all objects
Access	Accessed via object reference	Accessed via class name or object
Lifecycle	Exists as long as the object exists	Exists as long as the class is loaded
Scope	Specific to a single object	Common across all instances
Use Case	Store data unique to each object	Store data shared by all objects

1.3 Java keywords are integral to the structure of the language.
Each keyword has a specific function and usage in the program.
They cannot be used as variable names, class names, or method names.

2. Constructors and Initialization:

2.1 A **constructor** in Java is a special type of method that is used to initialize objects.

2.2 1. Default Constructor:

- A **default constructor** is either provided automatically by the Java compiler or explicitly defined by the programmer.
- It has **no parameters** and is used to initialize an object with default or predefined values.
- If no constructor is explicitly defined in a class, Java will automatically generate a default constructor.

2.2.2. Parameterized Constructor:

- A **parameterized constructor** is a constructor that accepts one or more parameters. These parameters allow the programmer to provide specific values when creating an object.
- The **parameter list** inside the constructor is used by Java to distinguish it from the default constructor.

2.3 Constructor chaining in Java refers to the process of calling one constructor from another within the same class or from a superclass. It allows multiple constructors to reuse code and ensure proper initialization of the object by calling a sequence of constructors, each adding to the initialization process.

The `this()` keyword is used to call one constructor from another within the same class.

3. In-depth Conceptual:

3.1 In Java, encapsulation is achieved primarily through:

1. **Private access modifier**: Making class fields (variables) private to hide them from outside classes.
2. **Public getter and setter methods**: Providing public methods to access and modify the private fields, which allows control over how the fields are accessed or modified.

By using encapsulation, you can:

- **Hide the internal state** of an object.
- **Control access** to the data by providing controlled methods (getters and setters).
- **Improve code maintainability** by keeping data and behavior together.
- **Safeguard data integrity** by preventing unauthorized or unexpected changes to the object's state.

3.2 Yes, a class in Java can have **multiple constructors**, and this is achieved through **constructor overloading**. Constructor overloading allows you to define multiple constructors within the same class, each having a different parameter list. This provides flexibility when creating objects with different initialization values or approaches.

Constructor Overloading in Java

Constructor overloading is similar to method overloading. It is achieved by defining multiple constructors with the same name (the class name) but with different parameter lists (different types, number, or order of parameters). Java differentiates between constructors based on their signatures, which include the number, type, and order of parameters

3.3 Significance of `static` Keyword

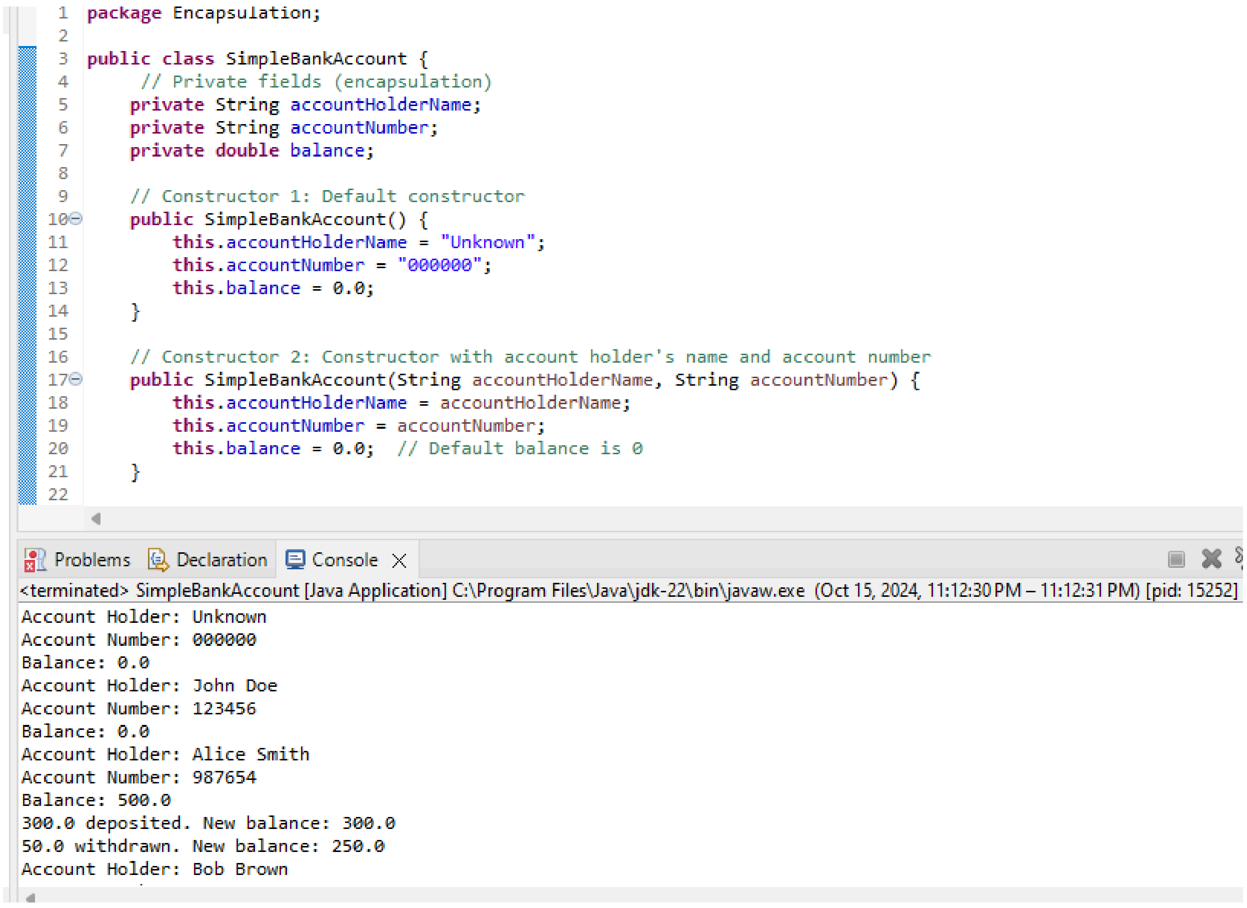
Static Variables (Class Variables):

- A static variable is a **class-level variable** that is shared by all objects of the class.
- Only one copy of the static variable exists, regardless of how many instances (objects) of the class are created.

- It is initialized when the class is loaded into memory, and it can be accessed using the class name or through any object of the class.

4. Real-world Design Questions:

4.1 Simple Bank-Account



```java
package Encapsulation;

public class SimpleBankAccount {
     // Private fields (encapsulation)
    private String accountHolderName;
    private String accountNumber;
    private double balance;

    // Constructor 1: Default constructor
    public SimpleBankAccount() {
        this.accountHolderName = "Unknown";
        this.accountNumber = "000000";
        this.balance = 0.0;
    }

    // Constructor 2: Constructor with account holder's name and account number
    public SimpleBankAccount(String accountHolderName, String accountNumber) {
        this.accountHolderName = accountHolderName;
        this.accountNumber = accountNumber;
        this.balance = 0.0;  // Default balance is 0
    }

```

Problems Declaration Console

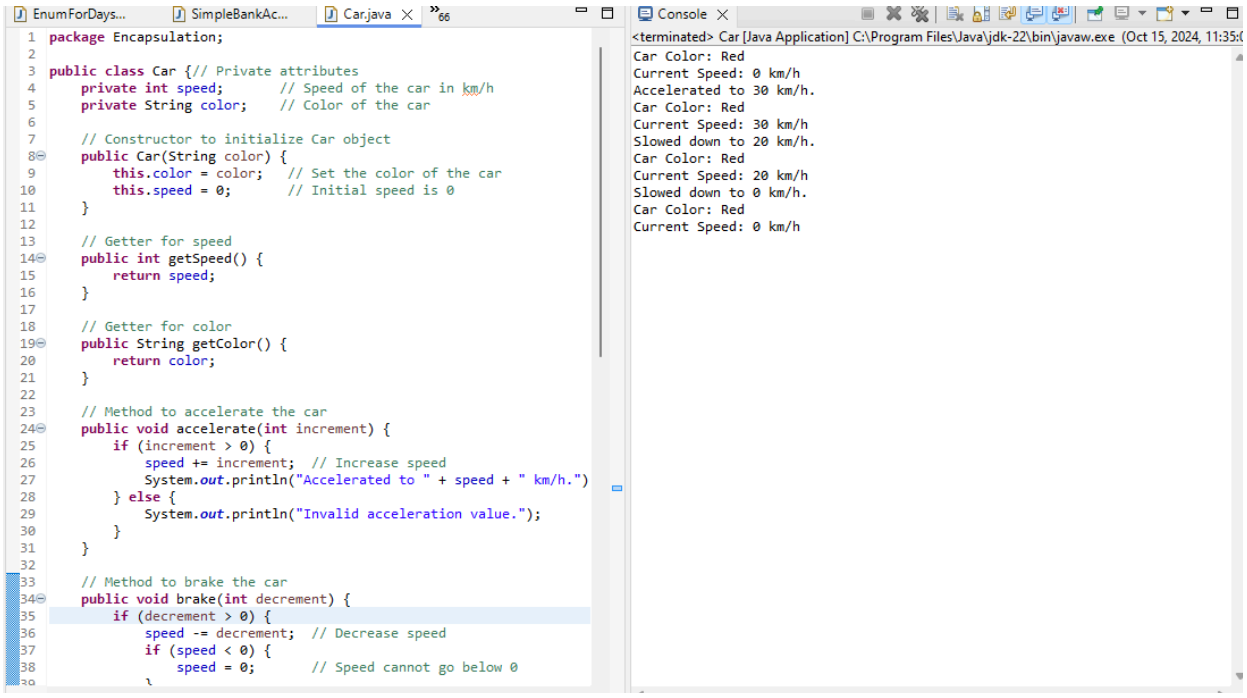
<terminated> SimpleBankAccount [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (Oct 15, 2024, 11:12:30 PM – 11:12:31 PM) [pid: 15252]

```
Account Holder: Unknown
Account Number: 000000
Balance: 0.0
Account Holder: John Doe
Account Number: 123456
Balance: 0.0
Account Holder: Alice Smith
Account Number: 987654
Balance: 500.0
300.0 deposited. New balance: 300.0
50.0 withdrawn. New balance: 250.0
Account Holder: Bob Brown
```

4.2 How would you represent a Point in a 2D space using classes? Extend it to 3D

To represent a **Point** in a 2D space using classes in Java, you can create a `Point2D` class that holds `x` and `y` coordinates. Then, to extend it to 3D, you can create a `Point3D` class that includes a `z` coordinate along with `x` and `y`.

4.3 Model a simple car class



```java
package Encapsulation;

public class Car {// Private attributes
    private int speed;       // Speed of the car in km/h
    private String color;    // Color of the car

    // Constructor to initialize Car object
    public Car(String color) {
        this.color = color;    // Set the color of the car
        this.speed = 0;        // Initial speed is 0
    }

    // Getter for speed
    public int getSpeed() {
        return speed;
    }

    // Getter for color
    public String getColor() {
        return color;
    }

    // Method to accelerate the car
    public void accelerate(int increment) {
        if (increment > 0) {
            speed += increment;  // Increase speed
            System.out.println("Accelerated to " + speed + " km/h.")
        } else {
            System.out.println("Invalid acceleration value.");
        }
    }

    // Method to brake the car
    public void brake(int decrement) {
        if (decrement > 0) {
            speed -= decrement;  // Decrease speed
            if (speed < 0) {
                speed = 0;       // Speed cannot go below 0
```

```
<terminated> Car [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (Oct 15, 2024, 11:35:
Car Color: Red
Current Speed: 0 km/h
Accelerated to 30 km/h.
Car Color: Red
Current Speed: 30 km/h
Slowed down to 20 km/h.
Car Color: Red
Current Speed: 20 km/h
Slowed down to 0 km/h.
Car Color: Red
Current Speed: 0 km/h
```

5 **Advanced Concepts:**

5.1 Anonymous inner classes in Java are a powerful feature that allows you to create and instantiate classes on-the-fly without explicitly defining a separate named class.

5.2 Memory management in Java is primarily handled through automatic garbage collection, which is a process that helps manage memory by automatically reclaiming memory that is no longer in use, thereby preventing memory leaks and optimizing the use of available memory.

5.3 **JavaBeans** are a specific type of Java class that adhere to certain conventions and are designed to be reusable software components. They are often used in graphical user interface (GUI) applications, as well as in enterprise applications.

6 **Scenario-based Questions:**

6.1 To model a relationship where a student can enroll in multiple courses and a course can have multiple students, we need to create a many-to-many relationship between the `Student` class and the `Course` class. In this relationship, each `Student` can be associated with multiple `Course` objects, and each `Course` can have multiple `Student` objects enrolled in it.

6.2 To design classes for a simple game where players can move within rooms, we need to create two main classes: `Player` and `Room`. The `Player` class will represent each player in the game, while the `Room` class will represent the different rooms that players can move between. Here's a structured approach to designing these classes: