Lab sheet 01: Java and Object-Oriented Programming

Learning Objectives

After completing this lab, students will be able to:

- Explain the main OOP concepts: **Class, Object, Encapsulation, Inheritance, Polymorphism, Interface**
- Analyze and design programs using OOP principles
- Apply OOP in writing Java programs
- Handle exceptions safely

Part 1 — Concept Review

Instructions: Answer each question clearly and concisely.

1. What is the difference between a **class** and an **object** in Java? *Answer:* .....
2. Why is **encapsulation** important in programming? *Answer:* .....
3. What Java keyword allows one class to **inherit** another? *Answer:* .....
4. Explain the concept of **polymorphism** with an example. *Answer:* .....
5. When would you use an **interface** instead of inheritance? *Answer:* .....

Part 2 — Code Comprehension

Study the following Java code and answer the questions.

```java
class Animal {
    void speak() {
        System.out.println("Some sound...");
    }
}

class Cat extends Animal {
    void speak() {
        System.out.println("Meow!");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal a = new Cat();
        a.speak();
    }
}
```

Questions:

1. What will this program print? *Answer:* .....
2. Which OOP concept does this demonstrate? *Answer:* .....
3. If you create another class `Dog extends Animal`, what method should it have? *Answer:* .....

Part 3 — Exception Handling

Read the following code:

```java
public class Demo {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3};
        try {
            System.out.println(arr[3]);
        } catch (Exception e) {
            System.out.println("Something went wrong!");
```

```
        } finally {
            System.out.println("Program ended.");
        }
    }
}
```

Questions:

1. What kind of error happens in this program? *Answer:* ……
2. What will be printed on the screen? *Answer:* ……
3. Why is the `finally` block important? *Answer:* ……

Part 4 — Code Writing Practice

Task: Write a Java program that models a **Car system** using OOP concepts.

Requirements:

- Class `Car`
  - Fields: `brand`, `speed`
  - Methods: `accelerate()`, `brake()`
- Use **Encapsulation** for the `speed` field (getter and setter).
- Create a subclass `ElectricCar` that overrides `accelerate()`.

Bonus (5 points): Add an **interface** `Chargeable` with a method `chargeBattery()` and implement it in `ElectricCar`.

Write your code below:

```
// Your code here
```

Mini Project Challenge

Scenario: Design a simple **Student Registration System** using OOP.

Requirements:

- Classes: `Student`, `Course`, `Registration`
- Use **Encapsulation** to protect student data.
- Use **Inheritance** if needed (for example, `OnlineCourse` extends `Course`).
- Throw an **Exception** when a student tries to register for a full course.

Expected Output Example:

```
Course full! Registration failed.
```

Sketch your class relationships or UML diagram below:

```
(Draw or describe relationships here)
```

Outline your main method or pseudocode here:

```
(Write or describe steps here)
```