SWE 4301 - OBJECT ORIENTED CONCEPTS II

OOC-2 LAB 01 HANDOUT AND TASKS

Prepared by:
Md. Jubair Ibna Mostafa
Assistant Professor, IUT CSE

Department of Computer Science and Engineering
Islamic University of Technology

Contents

1 OCP

The Open-Closed Principle (OCP) is one of the five SOLID principles of object-oriented programming, introduced by Bertrand Meyer. The principle states that

> software entities (classes, modules, functions, etc.) should be open for extension but closed for modification.

In other words, once a module is developed and working, it should be able to accommodate new functionality without altering its existing codebase.

The essence of the OCP is to design software systems in a way that allows for easy and seamless extension without causing cascading changes or disruptions in existing components. This is achieved by employing abstraction, inheritance, interfaces, and polymorphism.

Example of OCP violation

```
class AreaCalculator {
    static double calculateCircleArea(Circle circle) {
        return Math.PI * circle.radius * circle.radius;
    }

    static double calculateRectangleArea(Rectangle rectangle) {
        return rectangle.width * rectangle.height;
    }
}
```

Listing 1: Partial Java code

Code 1 presents a code snippet that is not close to modification. For example, if a new type of shape is added. The AreaCalculator class needs to add that code as another method.

Another Example of OCP violation

```
void processOrder(Order order) {
        switch (order.type) {
            case "regular":
                System.out.println("Processing regular order: $" +
   order.amount);
```

```
                break;
            case "premium":
                System.out.println("Processing premium order: $" +
    order.amount);
                break;
            // New order types require modification of existing code
            default:
                System.out.println("Unknown order type: " + order.type)
    ;
        }
    }
```

Listing 2: Partial Java code

Code 2 presents a code snippet that is not close to modification. If a new type of order, for example, seasonal is added another case needs to be added in the switch statement.

Example discussed in the class

```
public enum VehicleType {
    SEDAN, MOTOR_BIKE, SEVEN_SEATER
}

public class Trip {
    private  VehicleType vehicleType;
    private  int distanceKM;
    private  int timeMinutes;
    private  int numberOfPassengers;

    public Trip(VehicleType vehicleType,
                int distanceKM,
                int timeMinutes,
                int numberOfPassengers) {
        this.vehicleType = vehicleType;
        this.distanceKM = distanceKM;
        this.timeMinutes = timeMinutes;
        this.numberOfPassengers = numberOfPassengers;
```

```
19      }
20
21      public int perHeadFare()
22      {
23          int fare = -1;
24          switch (vehicleType) {
25              case SEDAN:
26                  fare = (50 + distanceKM * 30 + timeMinutes * 2) /
    numberOfPassengers;
27                  break;
28              case MOTOR_BIKE:
29                  fare = Math.max(25, distanceKM * 20) /
    numberOfPassengers;
30                  break;
31              default:
32                  if (distanceKM < 10)
33                      fare = 300 / numberOfPassengers;
34                  else
35                      fare = distanceKM * 30 / numberOfPassengers;
36                  break;
37          }
38
39          return fare - (fare % 5);
40      }
41
42      public boolean canTakeTrip()
43      {
44          if (numberOfPassengers < 1)
45              return false;
46
47          switch (vehicleType)
48          {
49              case SEDAN:
50                  return numberOfPassengers <= 4 && distanceKM <= 25;
51              case SEVEN_SEATER:
```

```
52                  return numberOfPassengers <= 7 && distanceKM >= 10;
53              default:
54                  return numberOfPassengers <= 1 && distanceKM <= 10;
55          }
56      }
57  }
```

2 TASKS

Section A & B **Marks 10**

- Write the above code.
- Write 6 test cases to check the correctness of the program.
  - satisfy sedan car for taking a trip
  - satisfy seven-seater car for taking a trip
  - satisfy motorbike for taking a trip
  - satisfy per head fair calculation for seven-seater
  - satisfy per head fair calculation for motorbike
  - satisfy per head fair calculation for sedan
- Refactor the code so that it follows OCP.

Section B

Scenario **Marks 15**

You are asked to build a Shape drawer application. A shape can be a circle, rectangle, or square. A circle can be represented using a point (x,y) and a radius. A rectangle can be represented by a point (x,y) that denotes the top left point of the rectangle, a length, and a width. Similarly, a square can be represented using a top left point, and a side length. Each shape area can be calculated using the known formula. Moreover, each shape is drawable. A canvas can receive a list of drawable shapes, and draw all of those.

1. Write C# or Java code that does not follow OCP.
2. Write 3 test cases to check the correctness of your program.
3. Refactor your code so that it follows OCP.