Assignment 1

CSE306 Software Engineering II

Deadline: 5th April 2020, 23:59:00
Cut-off Date: 5th April 2020, 23:59:00
Assignment Type: Individual
Submission: ICE Assignment Upload

For this assignment, write answers to the questions below and upload to ICE on the assignment upload link. The answer should be prepared and properly formatted in MS Word. This is an **individual** assignment. You are to complete this assignment on your own, although you may discuss the *concepts* with your classmates.

Design Pattern

This assignment requires you to demonstrate understand of design patterns. Note that you are being expected to do research because most of the patterns have not been covered on lectures.

Part A: Identifying Pattern (15 Marks)

For each of the following Java code snippets, identify the design pattern best represented by that code. Briefly explain your reasoning.

1.

```
public class Employee {
    ...
}
public class Manager extends Employee {
    private List<Employee> subordinates;
    public Manager(List<Employee> subordinates) {
        this.subordinates = subordinates;
    }
    public List<Employee> getSubordinates() {
        return this.subordinates;
    }
}
```

2.

```
public interface SomeObject {
    void process();
}

public class SomeObjectImpl implements SomeObject {

    public SomeObjectImpl() {
        ...
    }

    @Override
    public void process() {
        ...
    }
}

public class AnotherObject implements SomeObject {
    private SomeObject object;

    @Override
    public void process() {
        if (object == null) {
            object = new SomeObjectImpl();
        }
        object.process();
    }
}
```

3.

```
public interface Relay {
  public void send(String m, User u);
}

...

public abstract User{
  private Relay relay;
  public User(Relay r) {
    relay = r;
  }

  public void send(String message) {
    relay.send(message, this);
  }
  ...
  public abstract void receive(String message);
}

...

public class ApplicationRelay implements Relay {
  private ArrayList<User> users;
  public ApplicationRelay() {
    users = new ArrayList<User>();
  }

  ...

  public void send(String m, User originator) {
    for(User user: users) {
      if(user != originator) {
        user.receive(m);
      }
    }
  }
}
```

Part B: Correcting Problems (15 marks)

For the following code snippets, identify which design pattern *should have been used* to improve the quality of the code. Rewrite the code using design pattern that you've identified. Your answer should include i) the statement to describe the design pattern, ii) the rewritten Java code, and iii) the test result from the rewritten Java code.

```
public class FitnessCustomer {
    private static enum Level {
        BRONZE, SILVER, GOLD
    }
    private Level level;
     public void setLevel(Level level) {
         this.level = level;
     }

    public double getFees() {
        switch (level) {
            case BRONZE: return CustomerConstants.BRONZE_FEES;
            case SILVER: return CustomerConstants.SILVER_FEES;
            case GOLD: return CustomerConstants.GOLD_FEES;
        }
        throw new IllegalStateException("How did I get here?");
    }

    public boolean canAccessPool() {
        return (level == Level.SILVER || level == Level.GOLD);
    }

    public boolean hasOwnLocker() {
        return (level == Level.GOLD);
    }

    public double getEquipmentDiscount() {
        switch (level) {
            case BRONZE: return CustomerConstants.BRONZE_DISCOUNT;
            case SILVER: return CustomerConstants.SILVER_DISCOUNT;
            case GOLD: return CustomerConstants.GOLD_DISCOUNT;
        }
        throw new IllegalStateException("How did I get here?");
    }
    ...
```

Part C: Combining Design Patters (15 marks)

Variations on the standard design patterns which can be made by overlaying two different patterns (think overlaying their UML diagrams at a high level). This sometimes leads to useful new applications. For the following pairs of design patterns there is an elegant overlay possible, draw a UML diagram of that overlay and briefly describe it. What you don't want to do is to use the two patterns "next to" each other, you want to use them "on top of" each other, combining features of both.

1. **Deposite: Decorator and Composite;**
2. **Prodapter: Proxy and Adapter;**
3. **Stategy: State and Strategy.**