Practical Object oriented design (Strategy,Observer,Decorator&Iterator)

Solve the following problems.

Problem1: SalesTax Computation

Consider the development of a software package that must account for the different procedures for handling sales tax computation in each of the 27 states in India.

a) One possibility would be to use switch statements to handle these procedures. For example, when determining a state's sales tax rate, a routine could look like this:

```
float getSalesTaxRate(state s, merchandiseType m) {
        switch (s) {
                case AP: {
                        if (m == PrescriptionDrugs) return 0.0;
                        else return 0.04;
                }
                case MP: {
                        return 0.0;
                }
                case Telangana: {
                        if ( (m == Food) || (m == PrescriptionDrugs) ) return 0.0;
                        else return 0.056;
                }
                // And so forth...
        }
}
```

Find the issues with switch-case based approach.

b) A second possibility would be to exploit inheritance, with each state's way of handling commercial transactions encapsulated within the subclass associated with that state. For instance, the subclasses could look like:

```
        class APCommerce extends StateCommerce {
                float getSalesTaxRate(merchandiseType m);
        }
        class MPCommerce extends StateCommerce {
                float getSalesTaxRate(merchandiseType m);
        };
        // And so forth...
```

Again, find what's wrong with this inheritance-based approach.

c) Use the Strategy pattern, which favors aggregation over inheritance, to address the problems that you observed in the two previous questions. Include a UML class diagram that illustrates how the Strategy pattern would be applied here.

Problem2: Computing Average Score

We want to implement a class to represent the student scores from a class they are enrolled in. The class must provide at least the following methods:

```
void addAssignmentScore (double as); //0 or more assignments
void addExamScore (double es); //0 or more exams
double getAverage(); //the final class average
```

The algorithm to compute the average can be selected at runtime. It also must be possible to add new algorithms to compute the average to the program without modifying the above class. Your task is to design such class that satisfies the above requirements.

Use the following two algorithms for computing the average in your implementation:
A. The Assignment average contributes 40%, and the Exam average contributes 60% to the final class average.
B. Use the same percentages as the first algorithm, but first drop the lowest Assignment score.

Problem3: Facebook

In principle a social network service focuses on building online communities of people who share interests and/or activities, or who are interested in exploring the interests and activities of others. Facebook support groups that people can join. Each group has a title, administrative members, a group type (open/ closed), and a list of related groups. If somebody writes on the wall page of the group, the information is broadcasted to all the members and it is visualized in the news feeds of the members. Users should be able to join a group as well as leave a group if they get bored. Once a user has joined a group it will automatically receive any updates that are published on the wall. Which design pattern is the most appropriate to handle this basic functionality of such a Facebook group? Provide the Design as well. The following test code provides details about the group operation:

```
public static void main(String[] args) {

        System.out.println("Testing the Facebook Application");

        //Create a group
        FacebookGroup dp = new FacebookGroup();

        //Create users
        FacebookUser user1 = new FacebookUser(dp, "XYZ", 23);
        FacebookUser user2 = new FacebookUser(dp, "ABC", 20);
        FacebookUser user3 = new FacebookUser(dp, "AXY", 25);

        //Add users to the newly created group
        dp.addUser(user1);
        dp.addUser(user2);
        dp.addUser(user3);

        //write something on the wall
        dp.setState("Hello World");

        //users can also write on the wall
        user1.writeOnTheWall("Hi");
}
```

Problem4: Groupon

Groupon webpage shows deeply discounted offers from different business located in the city where you are living. As an example, it might show a beauty salon offer that has a 50% discount on women's haircut. You do nothing if you are not interested. If you like the Groupon deal, you select "buy" before the expiration date and your credit card is charged. If enough people select the deal, you will be sent a link for your rebate coupon at the end of the offer. Once you receive your coupon you can claim it at the specified store, usually within a period of time before it expires as noted on the website. If enough people do not join for that deal, the Groupon is cancelled and no one gets it and in this case you will get a refund for your money. In case that you have changed your mind you can also cancel an already made order and you will get a full refund. Provide a design for the specified functionality.

Problem5: HTML Tags

We wish to create HTML expression composed of some plain text and any combination of following html tags:,<a>,,<u></u>,. Both a and span tags require extra information namely a value for href and style parameters

respectively. Provide OO design to create HTML expressions easily from the client code using the above tags. Here is sample output:

<u>Hello Design</u>

Problem6: Printing Cars by brand, mileage, price, year and whole list

Design a CarCollection object that allows us to add cars and print car objects by whole/brand/mileage/year/price. The client will specify the feature for iteration and subdetails of that feature while creating iterator object. Here are the details of required printing:

Feature	Subdetail
whole	empty argument
brand	brandname
mileage	upper_limit_mileage
year	year_value
price	upper_limit_price