Assignment 5

Deadline: Friday, 27 November 2020 at 6pm

Before you start: create a new folder, called "*Assignment_5*" for this assignment in your GitHub repository.

Exercise 1 (5 points)

Part 1

Using the **Composite** and **Decorator** design patterns, implement a system to model a chain of bakeries with the following entities and requirements. Assume that in the future the company will acquire new restaurants of different specialities. Therefore, it is important to use design patterns to ensure the system can be easily extended.

Bakeries: There are three different types of bakeries:

1. **Normal bakeries**, which sell both bread and sweets
2. **Bakeries specialized in bread**, which sell only bread
3. **Bakeries specialized in sweets**, which sell only sweets

Furthermore, each bakery has a unique name (e.g., "Bakery Crunchy Bread"), an address (including the street name and number, postal code, and city).
Each bakery has a method "printName" to print the name of the bakery.

City office: All the bakeries in the same city are coordinated by a city office. The city office has a list of all the bakeries in the city. A city office has a unique name specified as follows: "*city office of [city name]*".
A city office has a method "printAllBakeriesNames" that prints the name of all the bakeries associated with the city office.

Central office: The headquarter of our chain of bakeries. There is only one central office and it has a list of all city offices that belong to the company.
The central office has a method "printAllOffices" that prints the name of each city office and the names of all bakeries associated to the specific city office in the following way:

[name of city office 1], [name of bakery 1 belonging to city office 1], [name of bakery 2 belonging to city office 1], …, [name of city office 2], [name of bakery 1 belonging to city office 2], …

Cake: Only bakeries selling sweets can sell cakes (i.e., normal bakeries and bakeries specialized in sweets). Each cake costs 3 CHF.
Furthermore, a cake can come with the following additional toppings for an additional price: (1) Strawberries (1 CHF each), (2) cream (1.5 CHF per portion), and (3) chocolate (2 CHF per portion).

There is no limit to the number of toppings that it is possible to add. Once a client has selected all the toppings they want, it must be possible to print the correct price (the price of the cake with all the chosen extras).

Sandwich: Only bakeries selling bread can sell sandwiches (i.e., normal bakeries and bakeries specialized in bread).
Furthermore, a sandwich can come with the following additional fillings for an additional price: (1) Ham (2 CHF each slice), (2) tomatoes (1.5 CHF per portion), (3) cheese (1 CHF per slice), and (4) tuna (3 CHF per portion).
There is no limit to the number of toppings that it is possible to add. Once a client has selected all the toppings they want, it must be possible to print the correct price (the price of the cake with all the chosen extras).

Part 2

Considering the code written in Exercise 1, complete the following points:

- Write a natural language description of **how and why** the patterns are implemented in your code. Address these points in a **ANSWERS.md** file and add it to the assignment folder.
- Document all methods and classes using Javadoc. The Javadocs can be auto-generated, but you need to complete it with all method information and its responsibilities.

Exercise 2 (5 points)

Part 1

Using the **Strategy** design pattern, implement the system of an airport shuttle service with the following entities and requirements. Assume that in the future the shuttle service will acquire new vehicles. Therefore, it is important to use design patterns to ensure the system can be easily extended.

Vehicles: There are four types of vehicles:

1. **Micro car**: The cheapest private car (12 CHF/h). It only fits a small luggage (1 bag). It has a normal speed and it is an electric car.
2. **Family car**: Middle price (15 CHF/h) private car that fits two small and two large bags. It has a normal speed and consumes gasoline.
3. **Supercar**: The most expensive private (30 CHF/h) car and fits a small and a large bag. It has a fast speed and consumes gasoline.
4. **Bus**: The cheapest means of transportation (5 CHF) where customers share the ride. They can bring as much luggage as they want. It has a slow speed and consumes diesel.

Customer: A customer can choose one of the vehicles and ride to the airport on a specific date. A customer entity has a method "ride" that, depending on the type of car to drive, prints the following "[name of the car] + [bags allowed] + [speed] + [price]".
For instance, "Micro car; 1 bag; normal speed; 12 CHF/h" or "Supercar; 1 small and 1 large bags; fast speed; 30 CHF/h".

Part 2

Considering the code written in Exercise 2, complete the following points:

- Write a natural language description of **how and why** the pattern is implemented in your code. Address these points in a **ANSWERS.md** file and add it to the assignment folder.
- Document all methods and classes using Javadoc. The Javadocs can be auto-generated, but you need to complete it with all method information and its responsibilities.

Part 3

Write unit tests for all the exceptions that might arise in the methods of the entities in the system. Refer to the system implemented in Part 1 of Exercise 2.
Remember to consider the corner cases of your implementation. For example, check for null pointer exceptions.

Important notes

Write all your answers in the **ANSWERS.md** file. Do not use a file with a different name. Also, please ensure that it is clear which answers belong to each exercise of the assignment.

Please note that no commits can be added to the assignment folder on GitHub after the deadline. Any commit added after the deadline **will not** be counted in the evaluation of the assignment.