Software Engineering 2022/2023

Assignment 4 – Design Patterns

Teacher: Tom Pepels – tom.pepels@maastrichtuniversity.nl
Lab date: **25 April 2022** - Due date: **30 April 2022**

This assignment awards **25** points toward the total 100 assignment points.

Important: *For this course, you work on all assignments in* ***teams*** *of* ***two****. Before submitting the assignment, you must join one of the* ***prepared groups*** *in Canvas (click on* ***People*** *in the course menu).*

Do not make your own group in Canvas, nor submit the assignment without joining a group first!

Design Patterns Exercise

In lecture we talked about a number of design patterns. In this exercise, you apply them to some small Java examples.

Part 1: Questions: Decide which Pattern

Before we implement different design patterns, let's first ensure we know when to use each pattern.

Which design pattern best meets each of these situations below, and why?

1. A model that stores a collection of blocks. Blocks can be metal or wood; can be painted, sanded, or chrome plated; and can sometimes be radioactive or magnetized. What design pattern would allow the system to easily add new types of blocks without changing existing code and how?

2. A model for a game that represents a robot. The robot navigates a maze that has obstacles. While playing the game, the robot can be upgraded with new parts that change its abilities like speed, weapons, and shields. Which design principle allows the robot object to change its behavior at runtime in flexible ways and how?

3. A model stores a phone number, and the UI (a keypad) which allows the user to enter digits one at a time. A different part of the UI wants to respond to each key being entered; however
   a) the two parts of the UI should be decoupled, and
   b) the model should be decoupled from the UI (model knows nothing about the UI).
   Which pattern would allow this and how?

4. A library supports recursively searching a directory for files. It allows the client code to provide it an object to filter the results. For each file which the library finds, it will ask the filter object if that file should be accepted or rejected. (See Java's FileFilter). Which design pattern would solve this problem, why and how?

Part 2: Implementing Patterns

Follow the directions for each section. **You may refactor the provided code** any way you like as long as you implement the expected patterns. The provided functionality may in fact be made such that you will have to restructure it to solve the problem.

Start with the code attached with the assignment on Canvas.

a) Telephone

The **telephone** package in the starter project contains partial code for making some parts of a (pretend!) telephone.

Change the code so that it uses the Observer Design Pattern as follows:

- In the model, define an interface for the observers.
- Have the model notify the observers whenever a new digit is entered for the phone number.
- Have the UI (the Screen class) create two observers:
    1. The first observer prints the newest digit out to the screen
    2. The second observer prints "Now dialing 12345678901..." out to the screen (where the number is the number the model has).
    You'll have to implement a new method in the PhoneModel to achieve this and call it from the main. This also means that you have two types of observers. Think about how you can distinguish them (hint: check out how the java enum type works)

Sample output from `telephone` package's Main:

```
Pressing: 2
2
Pressing: 5
5
Pressing: 5
5
Pressing: 4
4
Pressing: 8
8
Pressing: 1
1
Pressing: 7
7
Pressing: 0
0
Pressing: 9
9
Pressing: 2
2
Now dialing 2554817092...
```

Constraints

- Only the UI can print to the screen
- The model must be decoupled from the UI (model must not know about the UI).

b) Web Search

The websearch package in the provided source code contains partial code for a pretend web search engine. It will read a data file and "pretend" that each line is a query someone sent to a search engine. Our goal is to extract "interesting" queries meeting certain conditions.

Note that the code already uses the Observer Pattern to notify the Snooper of each query.
The Snooper then prints everything out.

Change the code so that it uses the Strategy Design Pattern as follows:

- In the model, create a new interface which describes the interface for a policy object that will define a query filter.
- A query filter policy object will have one method which will be passed a string (the query) and return true if the model should notify the observer about this query; returns false if the observer is not interested in this string (the query). For inspiration, see the Java FileFilter class.
- Change the model so when an observer is registered, the registration method to also accept a query filter policy object alongside it.
  Find a solution to store the Policy together with the Observer so that each observer can have its own attached filter.
- Change the model to, for each query (string from the file), check if an observer is interested in the query before notifying it.
- Change the client (Snooper.java) to create two query observers:
  1. One prints out "Oh yes! <query>" whenever the query contains the word 'friend' (case insensitive).
  2. One prints out "So long....! <query>" whenever the query is more than 60 characters long.

Sample output from `websearch` package's Main.java:

```
Oh Yes!     Friends to this ground.
So long....     Enter KING CLAUDIUS, QUEEN GERTRUDE, HAMLET, POLONIUS, LAERTES,
VOLTIMAND, CORNELIUS, Lords, and Attendants
Oh Yes!     And let thine eye look like a friend on Denmark.
Oh Yes!     Sir, my good friend; I'll change that name with you:
Oh Yes!     Those friends thou hast, and their adoption tried,
Oh Yes!     For loan oft loses both itself and friend,
Oh Yes!     O'ermaster 't as you may. And now, good friends,
Oh Yes!     As you are friends, scholars and soldiers,
Oh Yes!     A worthy pioner! Once more remove, good friends.
So long....     Or 'If we list to speak,' or 'There be, an if they might,'
Oh Yes!     May do, to express his love and friending to you,
Oh Yes!     As thus, 'I know his father and his friends,
So long....     That's not my meaning: but breathe his faults so quaintly
So long....     As 'twere a thing a little soil'd i' the working, Mark you,
Oh Yes!     'Good sir,' or so, or 'friend,' or 'gentleman,'
Oh Yes!     At 'closes in the consequence,' at 'friend or so,'
So long....     Enter KING CLAUDIUS, QUEEN GERTRUDE, ROSENCRANTZ, GUILDENSTERN, and
Attendants
Oh Yes!     Welcome, my good friends!
Oh Yes!     Friend, look to 't.
Oh Yes!     My excellent good friends! How dost thou,
So long....     Guildenstern? Ah, Rosencrantz! Good lads, how do ye both?
Oh Yes!     my good friends, deserved at the hands of fortune,
So long....     substance of the ambitious is merely the shadow of a dream.
```

```
Oh Yes!     beaten way of friendship, what make you at Elsinore?
Oh Yes!     thank you: and sure, dear friends, my thanks are
So long....     Why did you laugh then, when I said 'man delights not me'?
Oh Yes!     to see thee well. Welcome, good friends. O, my old
Oh Yes!     friend! thy face is valenced since I saw thee last:
Oh Yes!     Follow him, friends: we'll hear a play to-morrow.
Oh Yes!     Dost thou hear me, old friend; can you play the
Oh Yes!     My good friends, I'll leave you till night: you are
So long....     Enter KING CLAUDIUS, QUEEN GERTRUDE, POLONIUS, OPHELIA,
ROSENCRANTZ, and GUILDENSTERN
So long....     The courtier's, soldier's, scholar's, eye, tongue, sword;
So long....     How now, my lord! I will the king hear this piece of work?
So long....     To feed and clothe thee? Why should the poor be flatter'd?
So long....     Danish march. A flourish. Enter KING CLAUDIUS, QUEEN GERTRUDE,
POLONIUS, OPHELIA, ROSENCRANTZ, GUILDENSTERN, and others
So long....     Enter a King and a Queen very lovingly; the Queen embracing him,
and he her. She kneels, and makes show of protestation unto him. He takes her up,
and declines his head upon her neck: lays him down upon a bank of flowers: she,
seeing him asleep, leaves him. Anon comes in a fellow, takes off his crown, kisses
it, and pours poison in the King's ears, and exit. The Queen returns; finds the
King dead, and makes passionate action. The Poisoner, with some two or three Mutes,
comes in again, seeming to lament with her. The dead body is carried away. The
Poisoner wooes the Queen with gifts: she seems loath and unwilling awhile, but in
the end accepts his love
So long....     ashamed to show, he'll not shame to tell you what it means.
Oh Yes!     The poor advanced makes friends of enemies.
Oh Yes!     For who not needs shall never lack a friend,
Oh Yes!     And who in want a hollow friend doth try,
Oh Yes!     you deny your griefs to your friend.
So long....     Do you see yonder cloud that's almost in shape of a camel?
Oh Yes!     Leave me, friends.
So long....     Enter KING CLAUDIUS, QUEEN GERTRUDE, ROSENCRANTZ, and GUILDENSTERN
Oh Yes!     Friends both, go join you with some further aid:
Oh Yes!     Come, Gertrude, we'll call up our wisest friends;
So long....     and wife is one flesh; and so, my mother. Come, for England!
So long....     There's tricks i' the world; and hems, and beats her heart;
```

etc etc..

Constraints

The model should not know anything about the implementation of the query filter policy objects, other than they implement the required interface.

c) Bakery

The bakery package in the starter project contains partial code for a bakery. The bakery makes two types of cakes: vanilla and chocolate. They now want to make more complex cakes such as a "Multi-layered Vanilla cake with sprinkles that says 'Hello World!'"

Change the code so that an order can contain such complex cakes using the Decorator Pattern:

- Create the necessary decorator classes:
  - For multi-layered cakes, add $5 and print "Multi-layered" out at the front of the name.
  - For sprinkles, add $2 and print "with sprinkles" at the end of the name.
  - For a cake with the saying *X*, add nothing to the cost, and print "with saying '*X*'" at the end of the name.
- Add the new type of cake: strawberry cake, which costs twice as much as a Cake.
- Change the client to add the following to an Order, and print the Order:
  - Chocolate cake
  - Vanilla cake with saying "PLAIN!"
  - Vanilla cake with sprinkles with saying "FANCY"
  - Multi-layered Strawberry cake with double sprinkles and two sayings "One of" and "EVERYTHING"
    Suggested output is: Multi-layered Strawberry cake with sprinkles with sprinkles with saying "One of" with saying "EVERYTHING"

Sample output from `bakery` package's Main.java:

```
10  Chocolate cake
10  Vanilla cake with saying "PLAIN!"
12  Vanilla cake with sprinkles with saying "FANCY!"
29  Multi-layered Strawberry cake with sprinkles with sprinkles with saying
"One of" with saying "EVERYTHING"
```

Constraints

- Adding a new type of cake does not change any existing code (except to instantiate it)
- Adding a new way to decorate or style the cake (such as multi-layer, or with sprinkles) does not change any existing code (except to instantiate it)

Submission Guidelines

Submit deliverables as a zip wile with filename: SE_ASS4_***GroupN_IDStudent1_IDStudent2***.zip substituting ***GroupN*** with your group number, and ***IDStudent1***, and ***IDStudent2*** with your student id's.

Make sure that you also write your:

- group name,
- student names,
- student Id's,

in **all submitted** files (write a comment at the top of all source code files)! Failure to do this may result in ***NG*** (no grade) for the assignment.

Grading

- Exercise marked on coming up with a reasonable solution using the indicated patterns.
- Output formatting not important; only the design is important.
- For part 1 each exercise is graded with a maximum score of 2 points (max 8 points):
    - 2: Correct pattern and explanation
    - 1: Incorrect pattern but reasonable explanation
    - 0: No pattern or no explanation or no submission
- For part 2, each exercise is graded with a maximum score of 6 points (max 18 points):
    - 6: Great! It is the pattern and works.
    - 5: Reasonable in most places, but missed a part of the pattern, mostly works.
    - 4: A start on the pattern, but did not get more than half way, mostly works.
    - 3: Some changes, but not using the pattern, partially works.
    - 2: Poor changes, pattern incorrectly implemented.
    - 1: Not the correct pattern, code or solution does not work.
    - 0: Not submitted
- Your final grade for the assignment is ***min***(***total score***, ***25***)