Object Oriented Software Engineering

Practical 08

Paolo Ballarini

Learning outcomes:

- Observer Pattern
- Visitor Pattern
- MVC Pattern

Exercise 1. Code Design - Observer Pattern

You are requested to realize a virtual bulletin board system (BBS) through which registered users can post/read message on a board. The BBS system should be designed so to fulfil the following requirements:

> **R1** : the board may contain (at most) a single message
>
> **R2** : Each time the message on the board is changed the system should automatically inform all registered users that the message on the board has changed (hence passing on the new message to each registered reader).

In order to practice with the Observer design pattern you are requested to realize the following two versions of the program

Q1) First version of the program:

- you are not allowed to use the `Observer/Observable` interface/class provided by JDK.
- Develop a UML class diagram representing the classes architecture for your solution.
- Hence you have to implement the `Observer` design pattern, for this particular case, from scratch. Your solution should include the definition of a `MessageBoard` class and of a `BoardReader` class.
- write a simple JUnit Test where you create a `MessageBoard` object and a couple of `BoardReaders` and where you tests the functionalities of your classes. Let the program post a couple of messages on the message board and control if the readers get notified correctly (for that you should include in the notification the display of the updated message).
- Extend the above design by adding the possibility for each `BoardReader` to post a message on the board he/she is registered at.
- Modify the Test program by letting different readers posting different messages at different times. Control that the messages posted by each user are correctly received by any other users.

Q2) Second version of the program:

- you are now asked to reproduce your first solution by using the `Observer/Observable` classes available in JDK.
- Develop a UML class diagram representing the classes architecture for this second solution.

- Write a JUnit test using the same approach you used for the first version of the program.

Exercise 2. Tax calculator visitor

You are required to build a tax calculator system for an airport. This system is used to calculate tax/import duty to the products that are bought (imported) at the airport by passengers. System will calculate the import duty on the basis of some rules. For the sake of simplicity let us consider only three kinds of products that passengers of an airline may buy: `Book`, `Cosmetic`, and `Wine`. At present, the Airlines application only provides a single class of passengers, i.e passengers can only travel on `Normal` class ticket, but in future it will be expanded to `Corporate` and `Executive` passengers. The tax rate to be applied to `Normal` passengers are given in Table 1.

Type of passenger	Book	Cosmetic	Wine
Normal	10%	30%	32%

Table 1: Tax rate for different kind of products to be applied to `Normal` passengers

Goal: based on the Visitor pattern you are required to design and implement a program that allows for computing the total amount of money that a `Normal` passenger of an airline has to pay for the items (books, car, wine) he/she has purchased at the airport. The program should display both the total price for of all bought items including their tax costs as well as the total amount of taxes payed.

Q1) Draw a UML class diagram based on the Visitor pattern for your solution

Q2) Write the code for the classes you have conceived in the UML class diagram.

Q3) Write a simple client application where you simulate the computation of the total cost and total taxes for a given number of items bought by a `Normal` passenger.

Q4) Extend the above design by adding the tax computation functionalities for the `Corporate` and `Executive` kind of passengers. The tax rate for `Corporate` and `Executive` passengers are given in Table 2.

Type of passenger	Book	Cosmetic	Wine
Corporate	7%	20%	20%
Executive	5%	10%	10%

Table 2: Tax rate for different kind of products to be applied to `Corporate` and `Executive` passengers

Exercise 3. A cursor-move game by application of MVC pattern

Description. By application of the MVC pattern develop an interactive game through which a *cursor* is moved around a table of a given size which is displayed on the console. One cell of the table contains a cursor (displayed by a specific character) and the user can move around the cursor within the table by inputing simple commands like `left` (to move the cursor to the left cell w.r.t. its current position), `right` (to move the cursor to the right cell w.r.t. its current position) and so on. The MVC design should support different *views* of the table representing the game (see Figure 1) allowing the player to switch from one view to another.

Figure 1 and Figure 2 show an overview of the game execution (initial state and after a `left`).

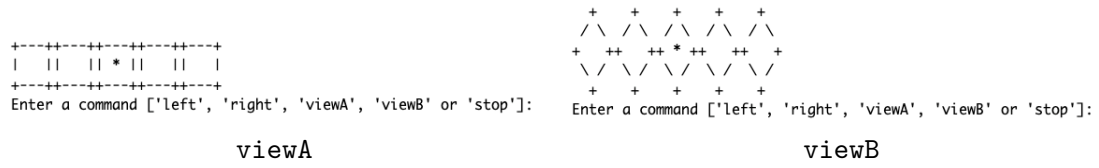

Figure 1: Initial state of the console of the table-cursor game

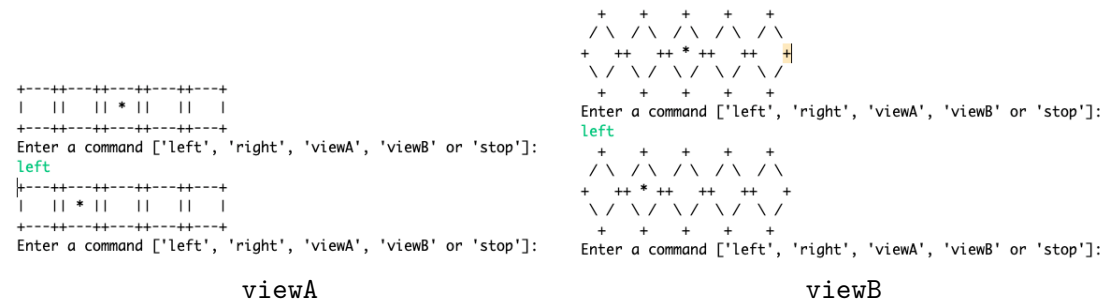

Figure 2: Console of the table-cursor game after a `left` move

Specifications:

- **S1**: the game should provide the user with the following commands
  - `left`: through which the cursor is moved one cell to the left w.r.t. to its current position

- `right`: through which the cursor is moved one cell to the right w.r.t. to its current position
- `viewA`: through which the table is display with the style depicted in Figure 1
- `viewB`: through which the table is display with the style depicted in Figure 3
- `stop`: through which the game is stopped

`right`, `left`

Q1) Develop a UML class diagram for the move-cursor game based on application of MVC pattern. Identify the Model, the View and the Controller for this game and name them properly (e.g. Player, Table, View, for example). The design should allow for easy configuration of the cursor-game: the size of the table should be chosen by configuration as well as the character that represent the cursor. Think carefully which class should be responsible for setting the size of the table and which one for setting the character with which the cursor is displayed on the console.

Q2) Work out the code for each class of the MVC that you identified for the cursor game.

Q3) Modify the view of the cursor game so that the table is depicted as in Figure fid:tablegame3: what part of the code you have to modify ?

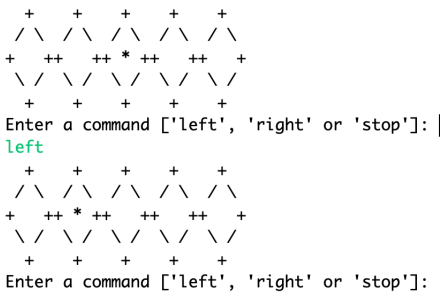

Figure 3: Using a different view for the table-cursor game