Seminar 4

Object-Oriented Design, IV1350

Adalet Adiljan, adalat@kth.se

22 May 2023

Contents

1. Introduction

Welcome to the last seminar report of the course. The seminar 4 assignments were divided into 4 different tasks, which is explained as following. The first task was about throwing exception(s) to handle alternative flow in 3-4a in Process Sale that was presented in seminar 1. An example of a thrown exception is for when an item identifier doesn't exist in the inventory catalog. The partial task b) was to use exception(s) to indicate that a database is not running. To achieve this, we need to be creating an imitated database to simulate the scenario to indicate a database failure for each time that a search is made for a hardcoded item identifier. It is required to log the error by writing and printing it to a file.

The first task of task 2 was to implement the Observer pattern to add another functionality of the program that calculates the sum of the costs of all sales during the runtime since the program started which required us to create two new classes named TotalRevenueView and TotalRevenueFileOutput. The second half of the task consisted about implementing the singleton pattern for the Register class. The pattern makes sure that the class has only one instance and provides a global point of access to that instance. It was also being recommended to implement the strategy pattern for the Discount and Sale class, since they enhance the flexibility and maintainability of the codebase.

The tasks were successfully completed with my fellow teammates *Haron Osman* & *Ali Kazimov.*

2. Method

This chapter of the report covers the following evaluations and discussions of how we as a group reasoned when writing the handling exceptions presented in the following tasks. The solutions for the partial tasks a) and b) of task 1 was created by looking at the different ways of handling curtain exceptions depending on a given code or a user problem that was presented in chapter 7-8 and the lecture recordings.

The thought process behind implementing the exception for the alternative flow was made by closely considering the specific alternative flow scenario. There are different appropriate exception types to choose by depending on the alternative flow, such as IllegalArgumentException, NullPointerException, IOException etc. In this case, we created a class called NoSuchItemException as an extension for the Exception class, meaning that it is a custom exception class, which in this case is customed to handle the exceptional cases where the item with no specified identifier is found. To clarify the exception case, the class has a constructor accepts a message of a String datatype that provides more information about the issue. In this case, it indicates that the expected item doesn't exist in the inventory catalog. In that way, an exception is thrown as the item identifier is invalid and catch it in the appropriate place to handle the alternative flow.

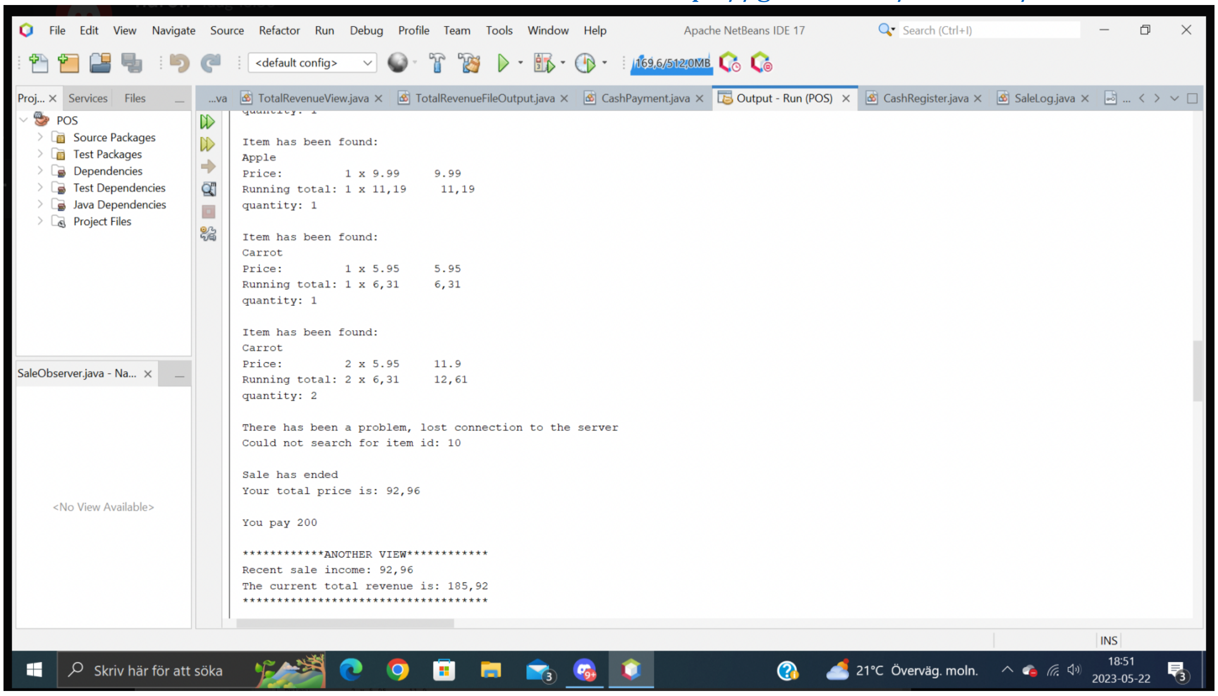
In a similar way to our other partial task b), we utilized the exceptions to indicate that the database cannot be accessed, which can occur if the database is not running. Although there is no real database to depict the exact scenario, we were given the task to create an invented database. The task also provided us an example of consistently throwing a database failure expectation whenever a search is performed for a specific, hardcoded item identifier. Our way of implementing this example was to create two custom exception public classes called ServerConnectionError in our integration layer respective ServerErrorException in our controller layer, which both extends the Exception parent class. Then, a code for an implementation of the ErrorLogger class which implements the Logger interface, was created. This class handles functionality of error logging.

The purpose of task 2a) was to implement the observer design pattern by creating two classes called TotalRevenueView and TotalRevenueFileOutput in the view layer. During the process of implementing the observer pattern, we mainly followed the examples that was covered in the recorded lectures and tried to understand the concrete examples given in the chapters 7-8 to gain an idea behind the implementation details. The group decided to meet up to solve the tasks togheter as usual, except it became a huge improvement of the time management compared to the previous seminars, which is a positive progress.

3.Result

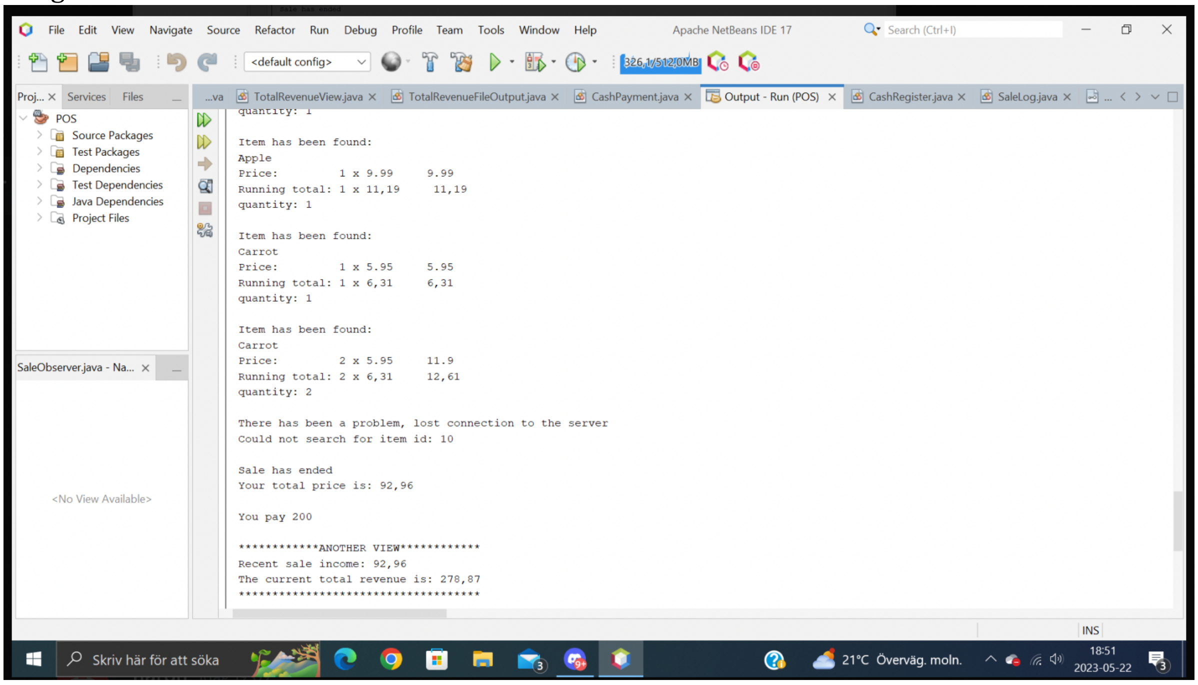
This chapter of the report covers the evaluation of the results by the given assignments for task 1 and 2.

Images 1-3 shows the results of the print-outs by running the program once, where we see the noticeable increase of the current total revenue. The same sale information that is being calculated by "income amount" through TotalRevenueView, the same information is sent to TotalRevenueOutput to a textfile. Along with these following images demonstrates that the serve error is properly caught when, for example, item ID 10 is entered.
For a closer code review, be free to visit our Github: https://github.com/Haaron1/Seminarium4



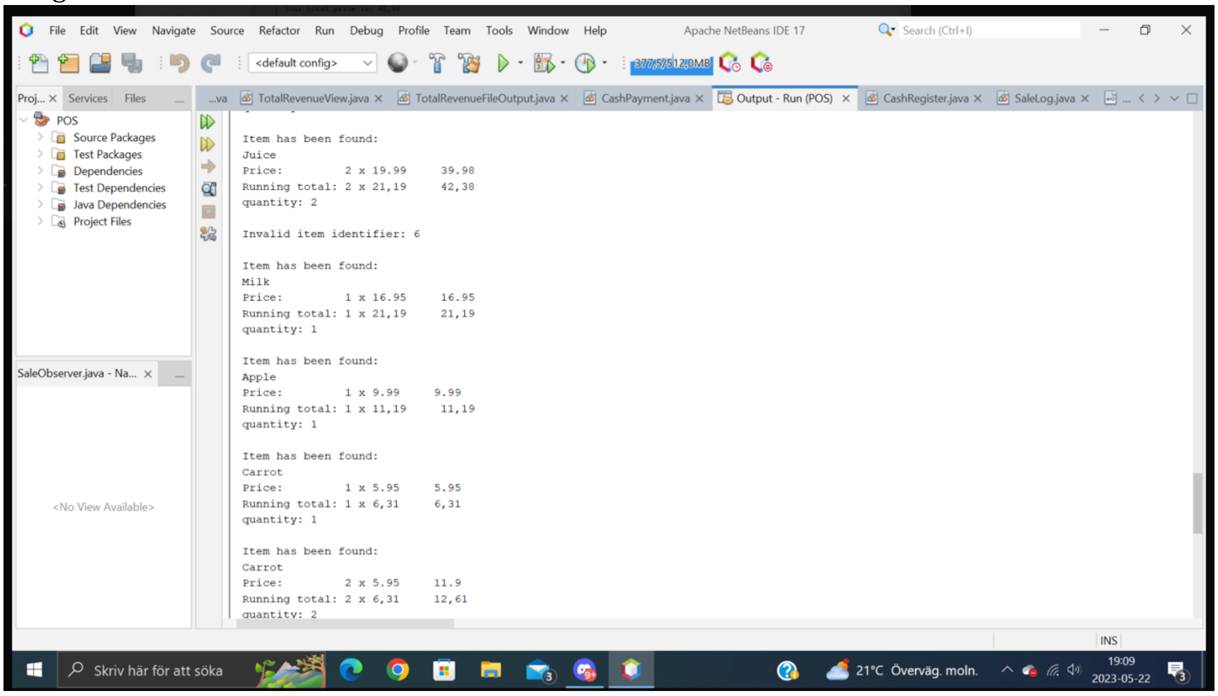
```
Item has been found:
Apple
Price:         1 x 9.99     9.99
Running total: 1 x 11,19     11,19
quantity: 1

Item has been found:
Carrot
Price:         1 x 5.95     5.95
Running total: 1 x 6,31     6,31
quantity: 1

Item has been found:
Carrot
Price:         2 x 5.95     11.9
Running total: 2 x 6,31     12,61
quantity: 2

There has been a problem, lost connection to the server
Could not search for item id: 10

Sale has ended
Your total price is: 92,96

You pay 200

************ANOTHER VIEW************
Recent sale income: 92,96
The current total revenue is: 185,92
************************************
```

Image2:



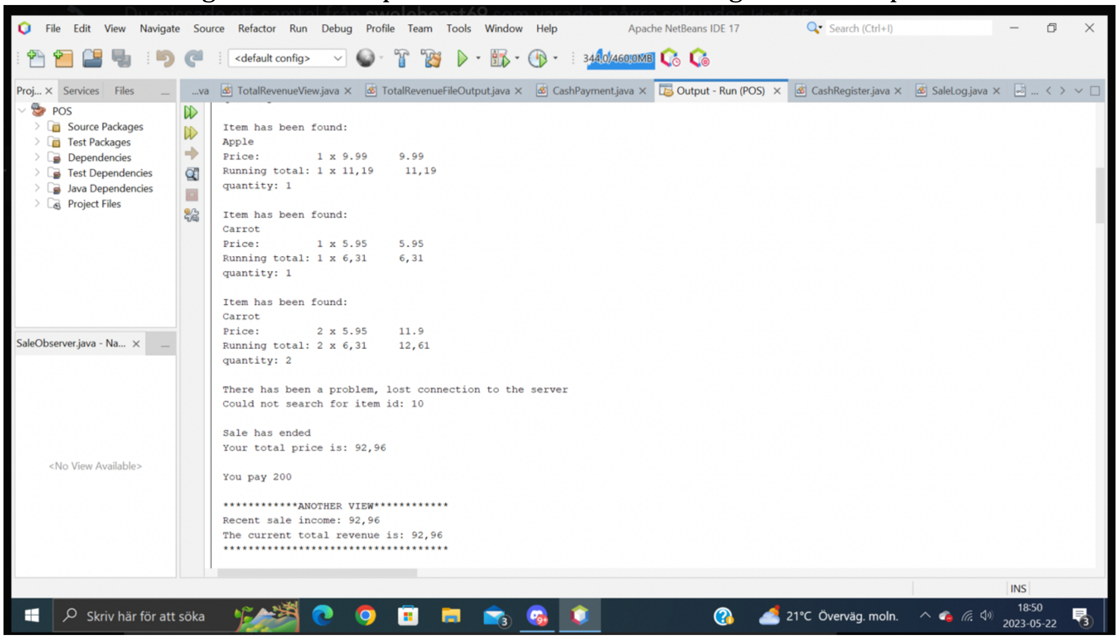
```
quantity: 1

Item has been found:
Apple
Price:          1 x 9.99     9.99
Running total: 1 x 11,19     11,19
quantity: 1

Item has been found:
Carrot
Price:          1 x 5.95     5.95
Running total: 1 x 6,31      6,31
quantity: 1

Item has been found:
Carrot
Price:          2 x 5.95     11.9
Running total: 2 x 6,31      12,61
quantity: 2

There has been a problem, lost connection to the server
Could not search for item id: 10

Sale has ended
Your total price is: 92,96

You pay 200

************ANOTHER VIEW************
Recent sale income: 92,96
The current total revenue is: 278,87
************************************
```

Image 3:



```
Item has been found:
Juice
Price:          2 x 19.99     39.98
Running total: 2 x 21,19      42,38
quantity: 2

Invalid item identifier: 6

Item has been found:
Milk
Price:          1 x 16.95     16.95
Running total: 1 x 21,19      21,19
quantity: 1

Item has been found:
Apple
Price:          1 x 9.99     9.99
Running total: 1 x 11,19      11,19
quantity: 1

Item has been found:
Carrot
Price:          1 x 5.95     5.95
Running total: 1 x 6,31      6,31
quantity: 1

Item has been found:
Carrot
Price:          2 x 5.95     11.9
Running total: 2 x 6,31      12,61
quantity: 2
```

Image 4: Last image shown demonstrates that if an incorrect item ID is entered, the item is not found, indicating that the exception for NoSuchItem is caught as it is expected to.



```
Item has been found:
Apple
Price:         1 x 9.99     9.99
Running total: 1 x 11,19     11,19
quantity: 1

Item has been found:
Carrot
Price:         1 x 5.95     5.95
Running total: 1 x 6,31     6,31
quantity: 1

Item has been found:
Carrot
Price:         2 x 5.95     11.9
Running total: 2 x 6,31     12,61
quantity: 2

There has been a problem, lost connection to the server
Could not search for item id: 10

Sale has ended
Your total price is: 92,96

You pay 200

************ANOTHER VIEW************
Recent sale income: 92,96
The current total revenue is: 92,96
************************************
```

4. Discussion

This chapter of the report covers the evaluation of the solutions according to the assessment criteria that is found in the seminar tasks description in Canvas.
Regarding the first question if the exception handling follows the nine bullets that are presented, I think the answer to that question is yes, majority of them. By taking a close consideration to the presented bullets that was required to be implemented for creating the exception handling from chapter 8. An example of an Exception code that fulfills all the expected criterias are following:

```
“public class NoSuchItemException extends Exception {
                public NoSuchItemException(String msg) {
                   super(msg);
                }
  } “.
```

According to the first bullet, we used an unchecked exception (extends Exception) to allow flexibility in handling exception. Regarding the 2nd bullet: I agree with that the name “ NoSuchItemException” is given the appropriate name, since it indicates the specific error condition related to that the typed item doesn’t exist. The additional necessary information is included by having a constructor that accepts an error message. Another example from the code is when the object doesn’t change state if the exception is thrown, which is true since it is only used for handling and reporting the error condition and doesn’t change the state of any object.
The fourth question deals with the agreeing with the choices of the observer and observed object. Depending on the specific requirement of the program, the choices of the right observer and the observed object varies. There were couple of things that we consider, which sort of guided us to create the right choices. One of them to consider is the observer’s interest, where the interest in the observer needs to be in being notified and the observer’s responsibilies or functionalities. The TotalRevenueView class fulfills this criteria, where the interest lays in the total revenue and recent incomes, which is being indicated in the following code found in the view package: public void totalRevenue(double recentTotalPrice, double totalRevenue) {..}, where observer’s interest receives the values of the total revenue and recent income of sales from the parameters in the totalRevenue method. Another example is the loose coupling, where the class depends on the SaleObserver interface, providing a level of abstraction between the observed object and the observer according to the following code: “public class TotalRevenueView implements SaleObserver {..} “. A third example of the design being implemented as expected is when the single responsibility is achieved, where the class is only focused on its task for displaying the total revenue and recent income and no unrelated responsibilities.

The sixth question follows what data is passed from the observed object to the observer. When implementing the observer pattern, it is good to know what the observer pattern serves for purpose. Typically, the design provides a way for observers to register themselves and receive updates in cases when events occur that changes the state of the observed object. Along with this, the methods are invoked that are also defined in the observer interface.

The importance of not breaking the encapsulation is also essential when implementing the pattern, since in that scenario, could be exposing its internal state directly to the observer. A way of keeping the encapsulation is by only providing the expected and relevant data to the observer without giving any of its own implemented details. The importance of having a well-designed interface could not be more emphasized here, since the interface decides / sets the necessary methods for the observer that are important enough to be implemented. The same thing goes for limited access of amount of data to only be providing the relevant ones and using appropriate access modifiers to restrict access to state and methods of the observed object. These are some of the requirements to consider when making sure that the observer pattern is maintaining its encapsulation, which I think it does in our case.