Seminar 4

Object-Oriented Design, IV1350

Maximilian Petersson | maxpet@kth.se

08-05-2021

Contents

1 Introduction

The goals for seminar 4 are to evaluate the tests created in seminar 3 and to put in measures that prevent errors from occurring, mainly by throwing exceptions and catching them with try-catch blocks. An observer is added in the project to practise polymorphism.

The author collaborated with Daniel Mebrahtu, Nahom Solomon and Harry Lazaridis when discussing and solving the seminar tasks.

2 Method

The team used git when collaborating on this project.
The tests from seminar 3 showed what exceptions each method could throw in different circumstances. With this knowledge we created our own exceptions, since all exceptions are thrown in vital parts of the program, the team decided to make them checked, so that they cannot be ignored. The introduction of exceptions has made it possible to replace previously used methods to signal errors.
To widen our test classes to also catch exceptions,
Implementation of the observer object was done once the team had a common understanding of its use, see figure 1 in results.

3 Result

GitHub link: https://github.com/PeterssonM/IV1350-KTH.git

This program writes to file, check your program folder for a file named “amountPaid.txt”.

Seminar 4 is the product of all seminars combined. The team has developed a functional and efficient cash register system with appointed chapters in the course literature used as guidelines .

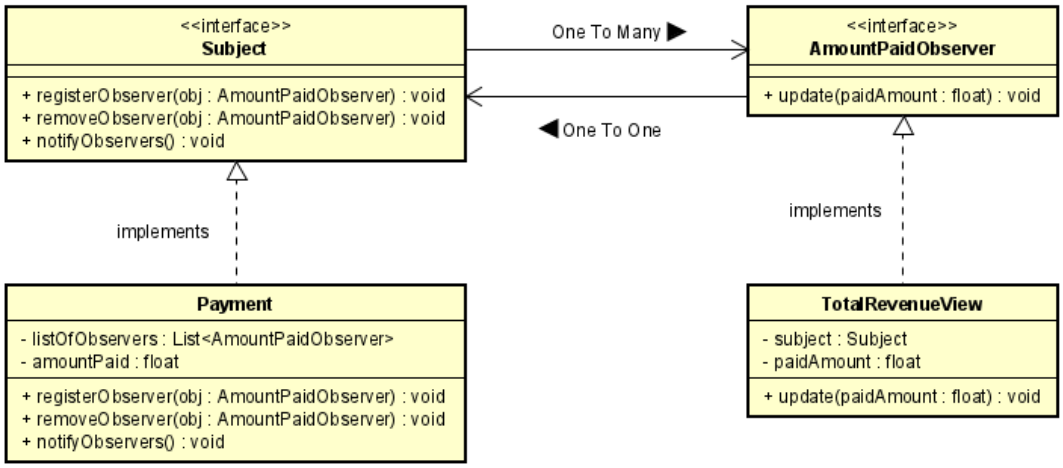

Figure 1 – Observer design pattern used in seminar 4. Designed by the author with inspiration from Dinesh Varyani on youtube. The figure was drafted in the early stages of an observer implementation, the class TotalRevenueFileOutput was later added next to TotalRevenueView in the mind of the author, although this was not needed to be modelled since everyone understood.

```
[A new sale has been started]

[Scanning items]
Item(s):        Price:  Tax[%]: Quantity:
red milk        12.5    3.0     5

[Is checking for discount]
Eligable discount: 25.0%

[Ending sale and showing a conclusion]
red milk        12.5SEK         VAT: 3.0%
Items cost:     62.5SEK  Total VAT: 15.0%

[Printing Receipt]

[Notifying that a new customer has paid]
The most recent customer paid: 500.0 SEK
Total revenue: 500.0 SEK
Writing to file 'amountPaid.txt'

Thanks for your purchase!
red milk        12.5SEK         VAT: 3.0%
Items cost:     62.5SEK  Total VAT: 15.0%
2021-06-09
See you soon!
Best regards Amigo
Isafjordsgatan 22, 16440, Stockholm
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.002

Results:

Tests run: 16, Failures: 0, Errors: 0, Skipped: 0


--- maven-jar-plugin:2.4:jar (default-jar) @ se.kth.IV1350.seminar4 -
Building jar: D:\Program\OneDrive\KTH!\Period 4\IV1350 Objektorienter

--- maven-install-plugin:2.4:install (default-install) @ se.kth.IV135
Installing D:\Program\OneDrive\KTH!\Period 4\IV1350 Objektorienterad
Installing D:\Program\OneDrive\KTH!\Period 4\IV1350 Objektorienterad
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  3.436 s
Finished at: 2021-06-09T21:20:17+02:00
------------------------------------------------------------------------
```

Figure 2 – Except for a simulation of a sale, this extract shows a successful observer implementation.

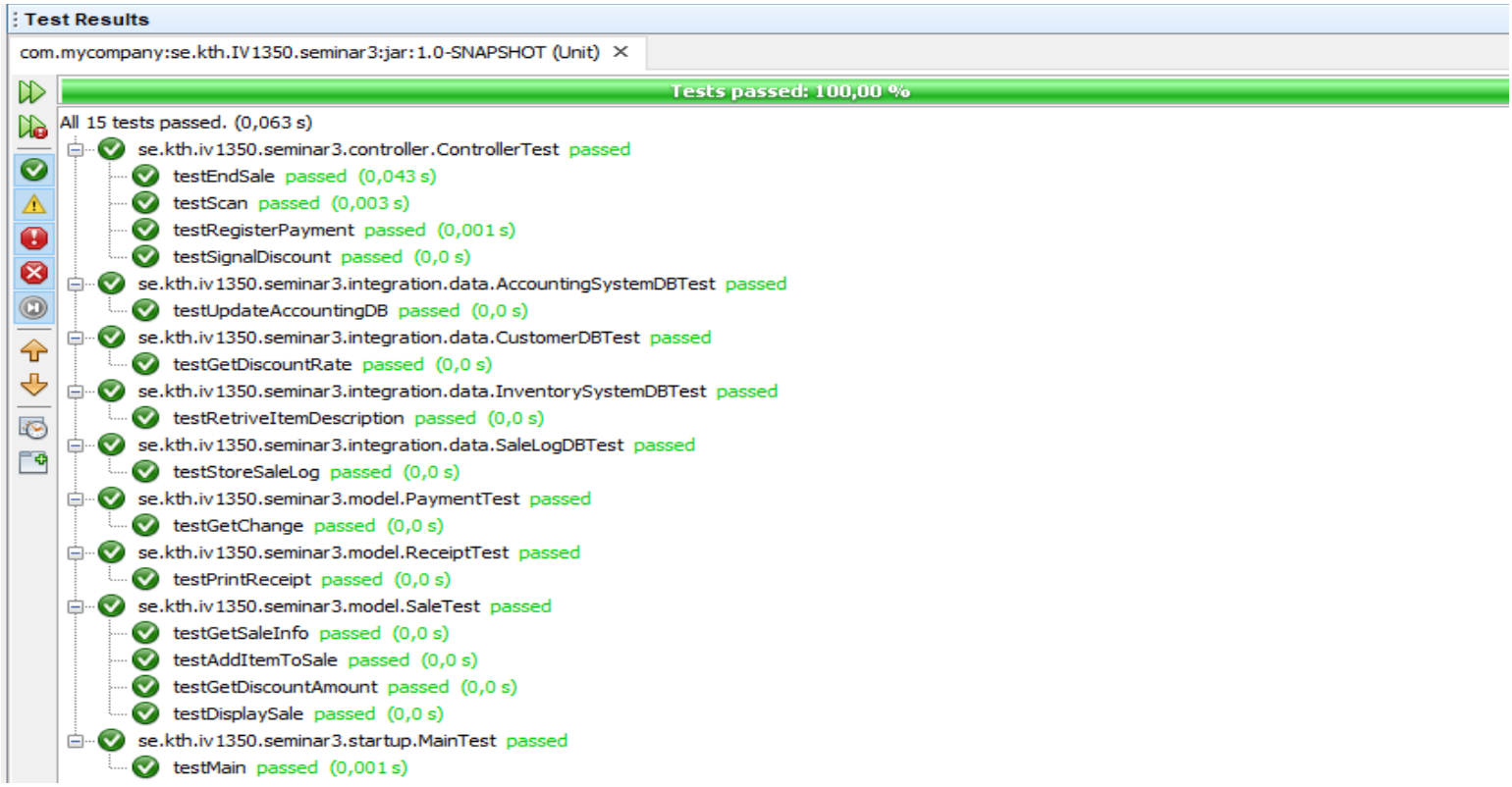

Figure 3 – In seminar 4, tests were widened to catch exceptions. This way, the team could test and evaluate the program's exception handling.

4 Discussion

The team settled for 1 point on this seminar, thus task 2, part b is absent.

Flawed exception handling is a security risk. Exceptions should be informative enough for a programmer to understand- and correct-them. It is therefore dangerous to let exceptions thrown low in the layers, arise to the top and meet the user, since this reveals the internals of the program. Low level exceptions should be logged and translated into something generic before reaching the user, this way, the user gets informed of an error without risking revealing system internals.
In our case, the databases were simulated with little to no realism, the exception they threw was "ConnectionTimedOut", which already is a generic and is therefore allowed to be thrown all the way to the user.

There are two obvious benefits with observers, resource efficiency and scalability. Implementation of "task 2, part a" would without an observer consist of method(s) constantly checking to see if a new payment has been done, this is very resource heavy and requires many calculations. What the observer allowed us to do, was to send a notification to a list of subscribed classes which in turn triggered all subscribed classes to perform some action (in our case printing to screen and writing to file).