Seminar 4

Object-Oriented Design, IV 1350

Lucas Larsson
lulars@kth.se

09-06-2021

Contents

1 Introduction 3

2 Method 4

3 Result 5

4 Discussion 7

1 Introduction

This seminar is the fourth seminar in a series of seminars that aims to give a practical experience of well-established theories within object-oriented programming with the focus in this seminar on Exceptions and error handling.
The task at hand in this seminar is to write Exceptions for alternative flows of the process sale specified in the documents from seminar 1.

Collaborations:

Dennis Hadzialic

Jabez Kung Otieno

Other students during "*Handlednings Pass*".

Other students in Discord Server.

2 Method

The method used to implement Exceptions and Error handling is a direct inspiration of the book instructions chapter 8.

Task 1-a:
Use exceptions to handle alternative flow 3-4 a. This task is solved through creating the class InvalidIdentifierException, this exception is invoked if the item identifier inputted from view doesn't exist in the ExternalInventorySystem.

Task 1-b:
Use exception(s) to indicate that the database cannot be called.
Ass suggested in the task, this exception is triggered using a hard coded solution due to not having an actual data base. The hard coded item identifier that triggers this exception is an empty string.

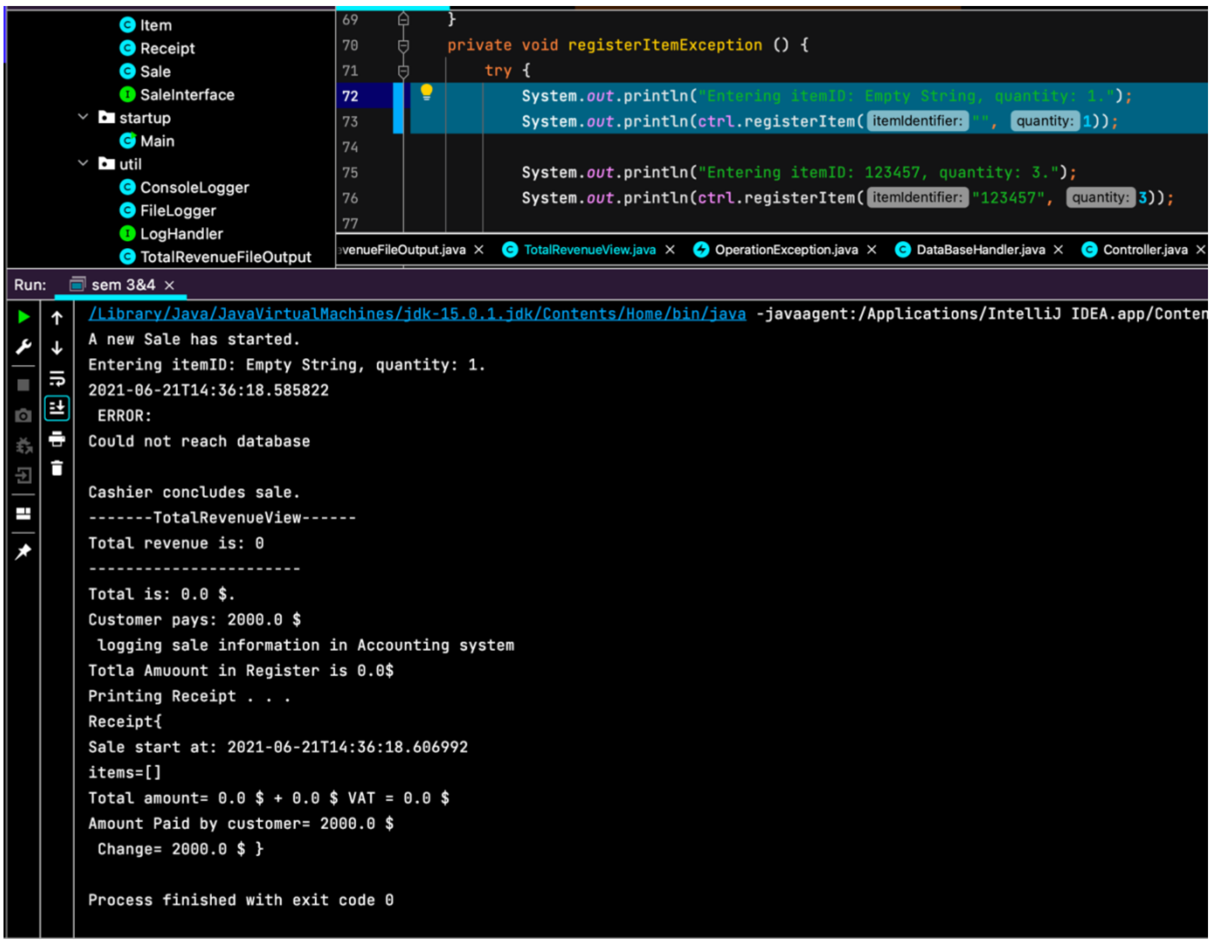
After updating the program according to the solution presented above, when an exception accrues, an informative message is printed to the terminal to inform the user, and a log containing an error report is written into the class FileLogger to inform the developer.

Task 2-a:
use the Observer pattern to implement a new functionality,
namely to show the total amount paid for purchases since the program started.
Considering the Observer pattern needed to be implemented with the classes TotalRevenueView and TotalRevenueFileOutput, the class Sale is chosen as an Observer counterpart the reasoning behind this is, simply to be able to access revenue, Sale is the best fit, another class candidate was Receipt but after discussion with other classmates Sale is determined as the only viable option due to its importance and centric location for many of the operations in the ProcessSale as so the class SaleInterface is created to implement the Observer pattern.

3 Result

As seminar 4 is complementary to seminar 3 the results of it can be viewed in the same file, the program still has the same functionality namely a sale process, with the addition of exception handling. Exceptions added in the program include OperationException, InvalidIdentifierException and ConnectivityException.

ConnectivityException is an exception thrown in case of connection failure to the database, As no database is used in this program an alternative solution is hard codded in the class, it is instead invoked in case an empty string is passed as an item identifier. A sample run is included below figure.



```
    }
    private void registerItemException () {
        try {
            System.out.println("Entering itemID: Empty String, quantity: 1.");
            System.out.println(ctrl.registerItem( itemIdentifier: "", quantity: 1));

            System.out.println("Entering itemID: 123457, quantity: 3.");
            System.out.println(ctrl.registerItem( itemIdentifier: "123457", quantity: 3));
```

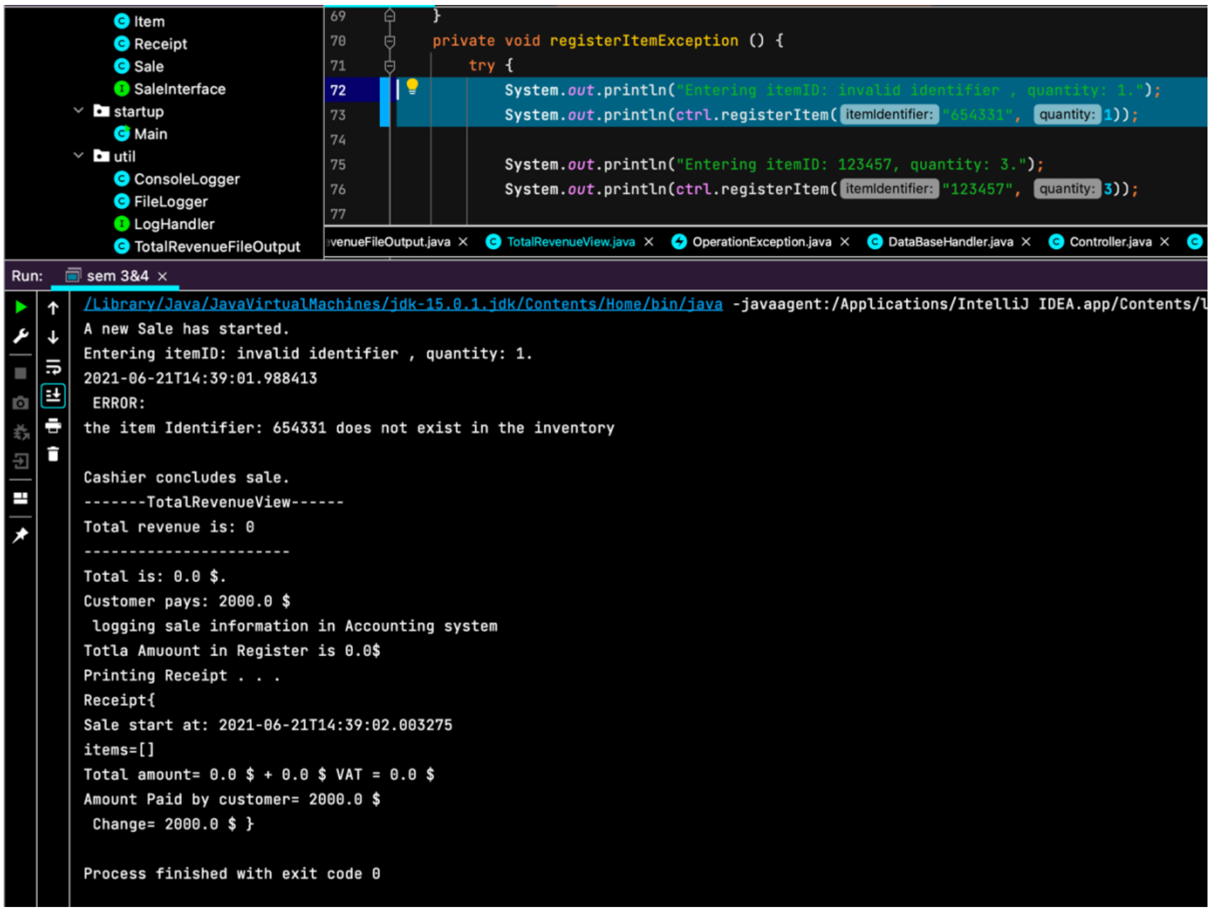
```
/Library/Java/JavaVirtualMachines/jdk-15.0.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Conten
A new Sale has started.
Entering itemID: Empty String, quantity: 1.
2021-06-21T14:36:18.585822
 ERROR:
Could not reach database

Cashier concludes sale.
-------TotalRevenueView------
Total revenue is: 0
----------------------
Total is: 0.0 $.
Customer pays: 2000.0 $
 logging sale information in Accounting system
Totla Amuount in Register is 0.0$
Printing Receipt . . .
Receipt{
Sale start at: 2021-06-21T14:36:18.606992
items=[]
Total amount= 0.0 $ + 0.0 $ VAT = 0.0 $
Amount Paid by customer= 2000.0 $
 Change= 2000.0 $ }

Process finished with exit code 0
```

Figure 3.1: A sample run showing the thrown exception(ConnectivityException).

InvalidItemIdentifierException is an exception invoked in case of input of an item that is invalied i.e does not exict in the ExternalInvintory system, a sample run is included below figure 3.2.
The sample ruOnvitorySystem.



```
    }
    private void registerItemException () {
        try {
            System.out.println("Entering itemID: invalid identifier , quantity: 1.");
            System.out.println(ctrl.registerItem( itemIdentifier: "654331", quantity: 1));

            System.out.println("Entering itemID: 123457, quantity: 3.");
            System.out.println(ctrl.registerItem( itemIdentifier: "123457", quantity: 3));
```

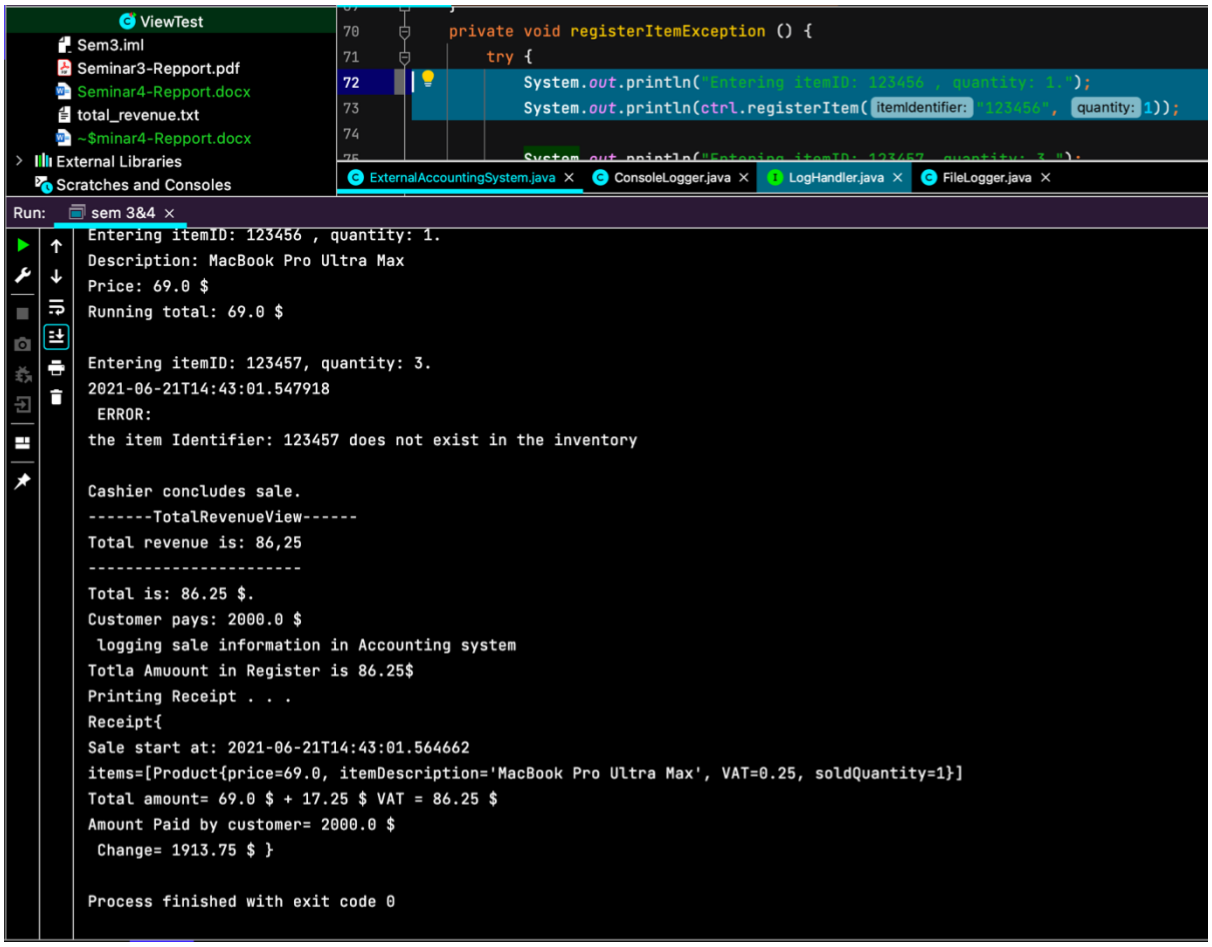
```
/Library/Java/JavaVirtualMachines/jdk-15.0.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/l
A new Sale has started.
Entering itemID: invalid identifier , quantity: 1.
2021-06-21T14:39:01.988413
 ERROR:
the item Identifier: 654331 does not exist in the inventory

Cashier concludes sale.
-------TotalRevenueView------
Total revenue is: 0
-----------------------
Total is: 0.0 $.
Customer pays: 2000.0 $
 logging sale information in Accounting system
Totla Amuount in Register is 0.0$
Printing Receipt . . .
Receipt{
Sale start at: 2021-06-21T14:39:02.003275
items=[]
Total amount= 0.0 $ + 0.0 $ VAT = 0.0 $
Amount Paid by customer= 2000.0 $
 Change= 2000.0 $ }

Process finished with exit code 0
```

Figure 3.2: A sample run showing the thrown exception (InvalidItemIdentifierException).

Lastly the Observer design pattern was implemented to show the revenue as shown in all the figures in the terminal, and even is written to a separate file “total_revenue.txt”.



```
    private void registerItemException () {
        try {
            System.out.println("Entering itemID: 123456 , quantity: 1.");
            System.out.println(ctrl.registerItem( itemIdentifier: "123456", quantity: 1));

            System.out.println("Entering itemID: 123457, quantity: 3.");
```

```
Entering itemID: 123456 , quantity: 1.
Description: MacBook Pro Ultra Max
Price: 69.0 $
Running total: 69.0 $

Entering itemID: 123457, quantity: 3.
2021-06-21T14:43:01.547918
 ERROR:
the item Identifier: 123457 does not exist in the inventory

Cashier concludes sale.
-------TotalRevenueView------
Total revenue is: 86,25
-----------------------
Total is: 86.25 $.
Customer pays: 2000.0 $
 logging sale information in Accounting system
Totla Amouunt in Register is 86.25$
Printing Receipt . . .
Receipt{
Sale start at: 2021-06-21T14:43:01.564662
items=[Product{price=69.0, itemDescription='MacBook Pro Ultra Max', VAT=0.25, soldQuantity=1}]
Total amount= 69.0 $ + 17.25 $ VAT = 86.25 $
Amount Paid by customer= 2000.0 $
 Change= 1913.75 $ }

Process finished with exit code 0
```

4 Discussion

OperationException is a checked exception, so no sample runs show this exception since the input is hardcoded and no business rules were broken. An example could be that if running total was greater than paid amount then an exception would be thrown, this is not necessary since the payment is in cash and handled by the cashier, but it was just an idea, it could be more appealing if it was self-service-POS as in you scan your items and the pay directly to a machine.