Seminar 2

Object-Oriented Design, IV1350

Casper Kristiansson, Casperkr@kth.se

2021-04-22

Contents

1 Introduction

The second seminar included designing a class diagram and communication diagram representing a basic and alternative flow for a process of a sale. The requirements on this assignment was that it should to have high cohesion, low coupling, and good encapsulation. This assignment was completed by the author, but the exercises was discussed with Michell Dib, Fredrik Lundström and Anders Söderlund.

2 Method

The design method followed the steps outlined in lecture 7:

1. Create basic packages
2. Transfer of classes, objects and methods from DM and SSD from seminar 1
3. Create communication diagrams

All along the process the ambition should was to strive for encapsulation with a small, well defined public interface, high cohesion and low coupling.

The first step was to design the class diagram. The author created the diagram by following the class template, see figure 2.1. After building the template he followed the Domain Module and System Sequence Diagram that was developed in seminar 1. It made it very easy to develop the diagram by following what we had done earlier. The different packages consisted of view, controller, model, startup, dbhandler and the author made some changes so it would fit the current objective such as changing the data layer to integration.

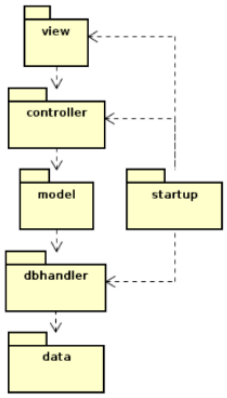

Figure 2.1: Packages

In the next step the author designed the controller and all the different methods that would be called inside it. This includes both basic and alternative flow. After designing the control the next step was to design the dbhandler package where all the data, such as ExternalInventorySystem, ExternalAccountSystem, DiscountSystem, was supposed to be processed. The author also designed the different methods for these classes such as searching and updating the system.

When designing the class diagram, the ambition was to create all the packages and a layout to simply visualize an overview of the program so it could easily be understood. The startup package has association with the view, controller, and integration packages, which in their end have association with other parts of the program.

The design of the package model, which was supposed to process all the information such as the saleInformation and ItemInformation, was completed in the next step. Due to the fact that these two types include many objects, the author created an ItemDTO and SaleDTO to make it easier to process the information. The next part of the seminar included creating the communication diagram. The author created one communication diagram for each part of the control, which consisted of discount, endSale, enterItem, pay, startSal. Another diagram was designed for the Startup.

3 Result

Seminar 5, Improve Your Score

There was a total of 6 mistakes in seminar 2.

- The first mistake was that the controller was missing the parameters that should have been included. This was fixed by adding the parameters printer, eas, eis and dc.
- The second mistake was that all commands in the controller were voids, which means that the view will not receive any data about the sale. This was fixed by reconstructing the methods so they will return the right appropriate value. For example, we return the saleDTO when we enter an item and we return a double value when we pay.
- The third mistake was that in the old figure 3.3 we created the SaleDTO but then never used it. This could be fixed by simply removing the saleinformation. We also added a return value to the operation sale() which then later is used in the receipt.
- The fourth mistake was that the author missed to send in quantity when adding an item. This was simply fixed by adding it as a parameter when calling the method. The second part was that the author didn't describe how the alternative flow 3-4a, 3-4b was handled. This was fixed by adding notes when the program checks if an item has already been entered or if an item doesn't exist.
- The fifth mistake was that the wrong parameter was sent in when the receipt created. This was fixed by creating the receipt in the correct way by sending in the sale object in startSale. The next step is to get the receipt and print it. In the payment package the getReceipt returns a receipt which in turns is sent to the print method.
- The last mistake was that the discount handler was not modeled the correct way. The correct way was to first get the sale information of the sale. Then look up if a discount exists for the customer. Finally, an applyDiscount method is called which updates the sale information.

All figures below have been updated accordingly.

The result of the design exercise is outlined in the class and communication diagram below.

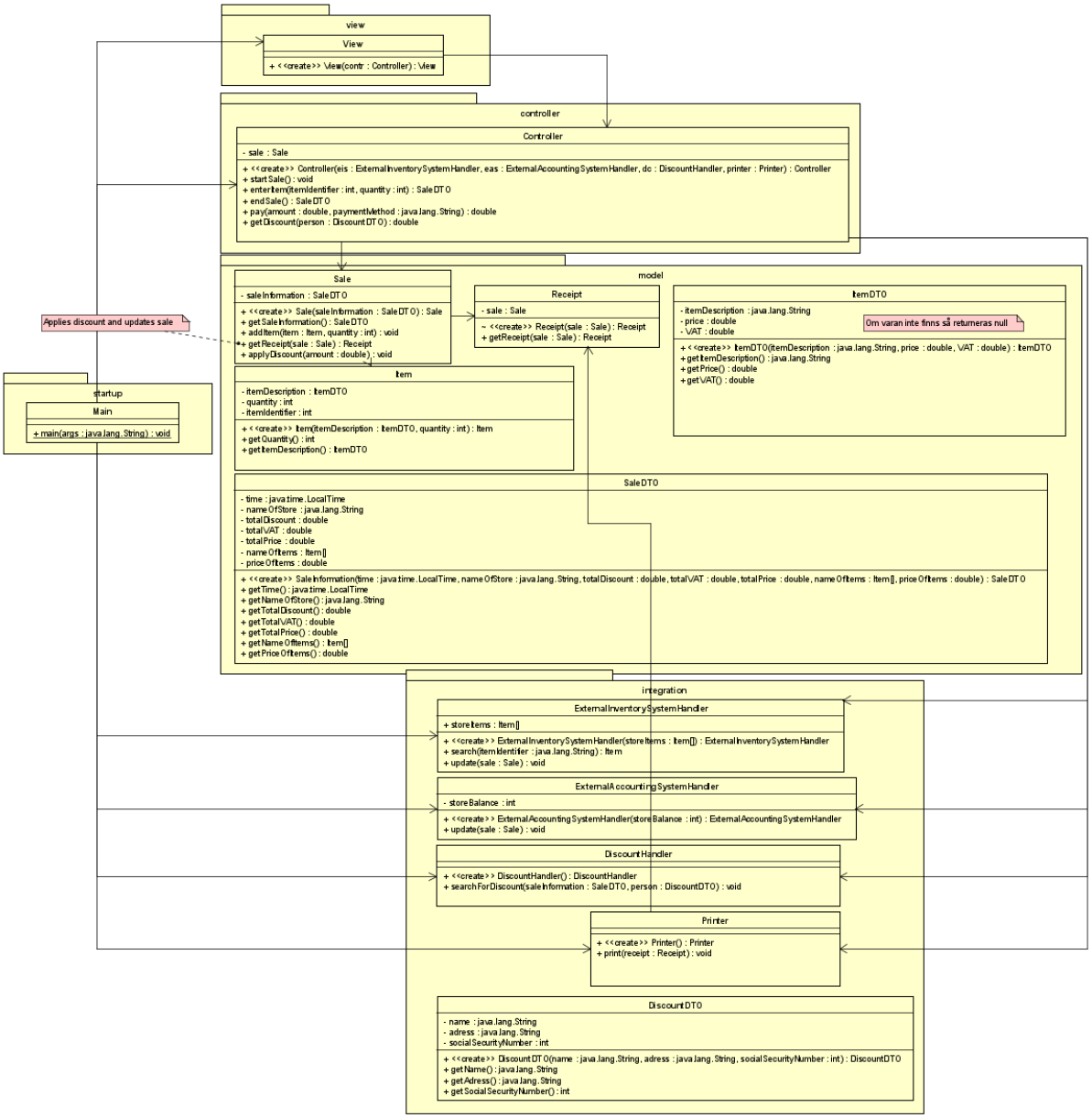

Figure 3.1: The figure represents the class diagram and all the different classes and methods.

The next step was to design the communication diagrams. The diagrams that was needed to be created was the different methods that was inside the Controller package. This includes

discount, endSale, enterItem, pay, startSale and Startup. The author also made sure that all the different classes were created when they were supposed to.

Start up process

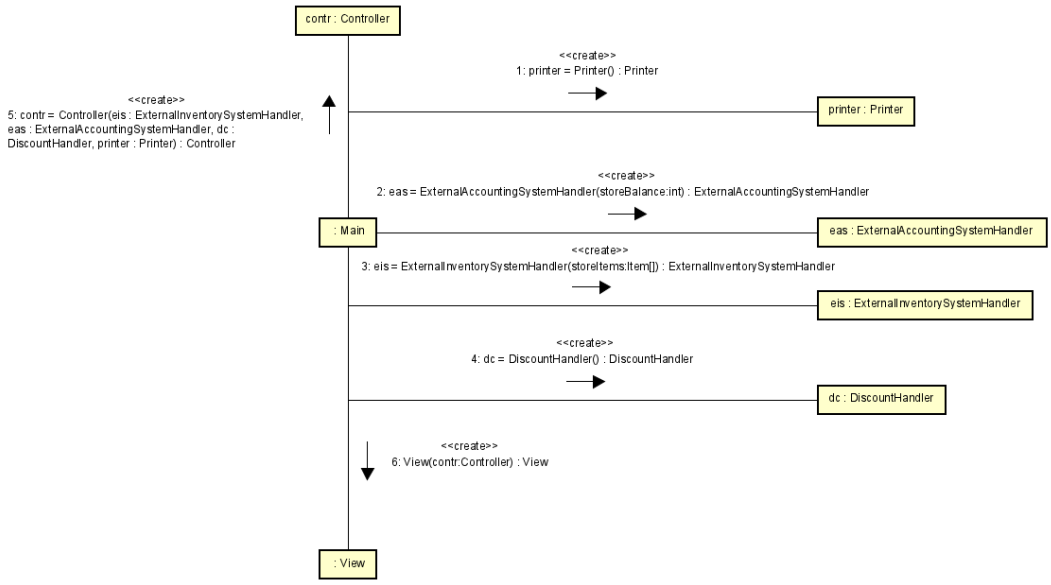

Figure 3.2: Communication Diagram of startUp

The Startup process communication diagram contains the main function and how it connects to all the other different classes. It shows that the different instances for the classes are created.

Start Sale process

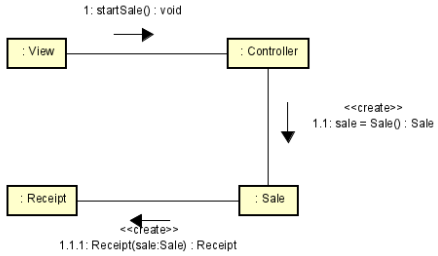

Figure 3.3: Communication Diagram of startSale

The Start sale process communication diagram shows how the startSale() method works and how it invokes getSaleInformation and Receipt.

1. Initiation of startSale()
2. startSale() calls for the constructor Sale()
3. Sale calls for getSaleInformation()
4. Sales calls for Receipt()

Enter item process

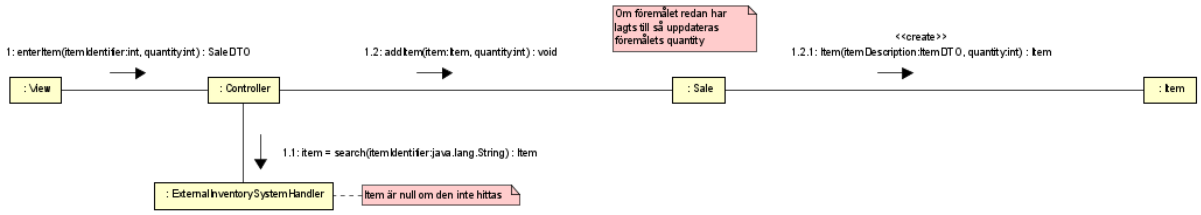

Figure 3.4: Communication Diagram of enterItem

The Enter item process communication diagram shows the process of entering an item and how the item is being identified by searching the ExternalInventorySystemHandler. The method addItem used to update the sale information.

1. Initiation of enterItem()
2. enterItem() calls for Search()
3. enterItem() calls for addItem()
4. The constructor Sale() calls for the item() information

End sale process

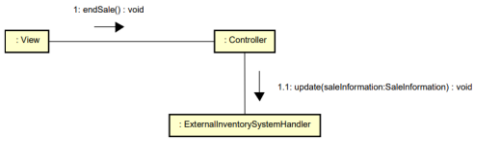

Figure 3.5 Communication Diagram of endSale

The End sale process communication diagram shows the process of endSale which updates the externalInventorySystem.

1. Initiation of endSale()
2. endSale() calls for update()

Pay process

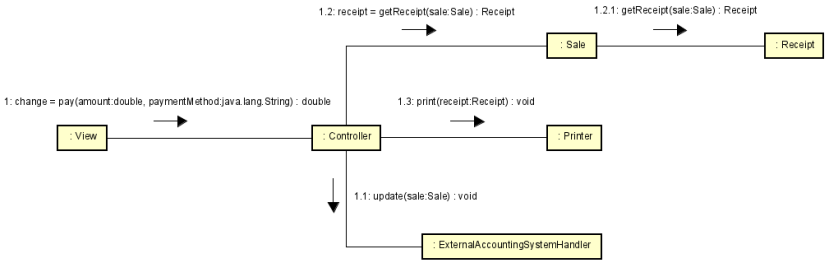

Figure 3.6: Communication Diagram of pay

The Pay process communication diagram outlines how the action pay is processed and how it updates the ExternalAccountingSystemHandler and finally prints out a receipt.

1. Initiation of pay()
2. pay() calls for update()
3. pay() calls for print()

Get discount process

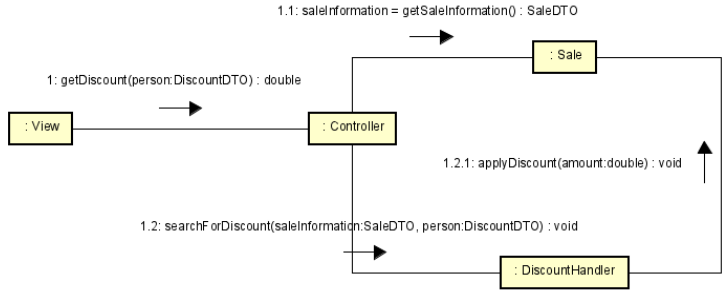

Figure 3.7: Communication Diagram of getDiscount

The get discount process describes the process of getDiscount. It first step is to search the DiscountHandler for the discount and if a discount is found it updates the system.

1. Initiation of getDiscount()
2. getDiscount() calls for searchForDiscount()
3. DiscountHandler calls for the update()

4 Discussion

When designing the class diagram the author designed it to be easily understood. This included the best layout for the different packages. By using the template, it is easy to follow the startUp packages to the other packages. By following the template, we follow the MVC (Model-View-Controller), so the system is divided into the different subsystems/packages, such as the Model, View and Controller. When the class diagram was designed the author had in mind how the process of a sale would look like in code. This way the diagram got the perfect number of layers and classes and the controller doesn't handle any logic it shouldn't.

The class diagram has high cohesion where the classes fit together very well. Example of this is the classes Sale, Receipt, and Item and how they have association to each other. This way it is easy to understand the cohesion between the different classes and to understand the purpose programs. The diagram was as well designed to have low coupling where the program doesn't have any unnecessary association between the different classes. For example, between the classes controller and the different integration parts such as ExternalInventorySystemHandler, ExternalAccountSystemHandler, DiscountHandler and Printer. In this way it is easy to understand how these classes are used in the program. The program also has good encapsulation where the methods that are not needed for other parts of the program are private. IBy doing so the data can be separated and protected.

The author designed the diagram so that all the different classes have the right parameters, return values and types. Example "pay (amount : double, paymentMethod : java.lang.String) : void" where it is specified what the different parameters and their types are and if the method should return any value.

A "problem" with Java is how it manages different objects and that the list of objects can be very long. A solution to this is to create data transfer objects (DTO) to return parameter lists. DTO:s are also really great for keeping a high encapsulation. One example is how the author created the DTO:s ItemDTO, SaleDTO and DiscountDTO.