Seminar 2

Object-Oriented Design, IV1350

Sebastian Rone, sebgro@kth.se

2022-04-12

Contents

1 Introduction

The task for this seminar was to design a program that could handle all the parts from seminar 1, the different flows of the “Process of sale” scenario.

2 Method

When designing the program I worked with classmate Sushil KC, we had many discussions about what was needed and what seemed to be unnecessary. We followed the methods that was given as guidelines in the book. Which meant to do MVC and Layer patterns, which basically means to create one package for each layer. Then we made sure to only do one system operation at a time while doing this we always tried our best to strive for the 3 things that have been said repeatedly under the lectures "high cohesion, low coupling, and a high degree of encapsulation with a small, well-defined public interface"

3 Result

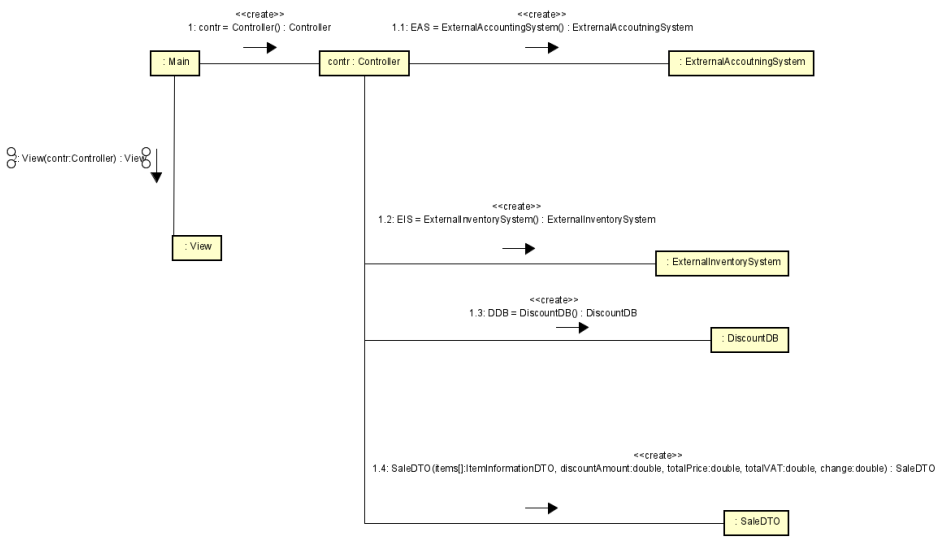

Figure 3.1: Communication Diagram.

This diagram above illustrates the startup of the process of a sale. Where the main method first calls on Controller which then calls on the other four objects and later is passed to View.

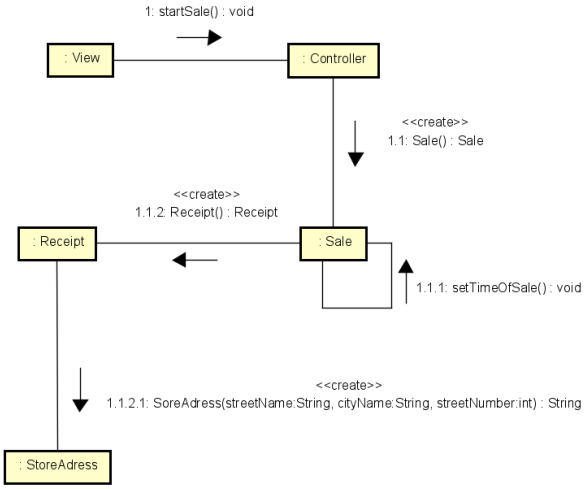

Figure 3.2. Communication Diagram.

The diagram above shows the process of starting a sale. Where the View is calling on the controller to “startSale” which in return makes the controller create an object of the type sale, and sale saves the date and time which lastly calls on receipt whom stores the store address there.

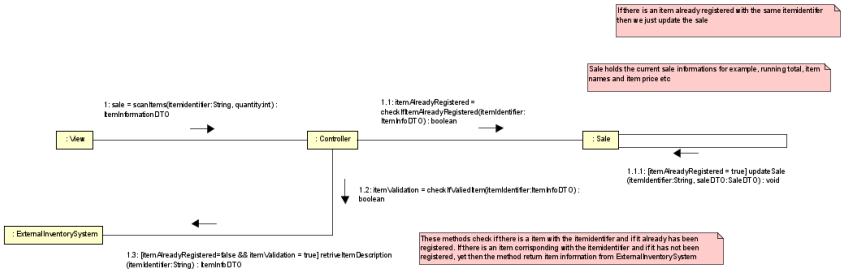

Figure 3.3 Communication Diagram.

The diagram above shows the process of scanning items. Where we have Viewer “cashier” who passes the item Identifier to Controller which checks if the item has been scanned before or not if it has it send it to sale which adds the item to the list of items. If it is not checked it’s sent to the external inventory to see if it’s a valid item. If it’s valid but has not already been registered its then given the information about the item, prices etc.

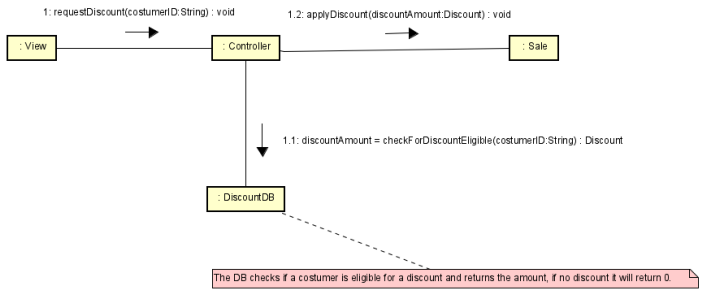

Figure 3.4 Communication Diagram.

The diagram above shows the process of requesting a discount. The method requestDiscount is sending costumerID in its parameters to controller, controller then passes it to the DiscountDB to see if costumer is eligible and how much. It's then saved in discountAmount and the method applyDiscount is called and added to the amount

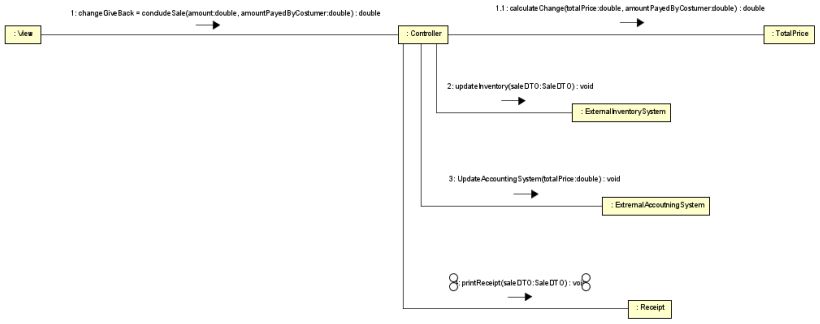

Figure 3.5 Communication Diagram

The diagram above shows the log completed sale. View sends controller how much amount has been paid and gets back how much to costumer back. Controller then updates all inventory system and accounting system by sending the parameter saleDTO which has all the information about the sale and lastly, we print out the receipt. Here I could have made another class called printer but decided not to.

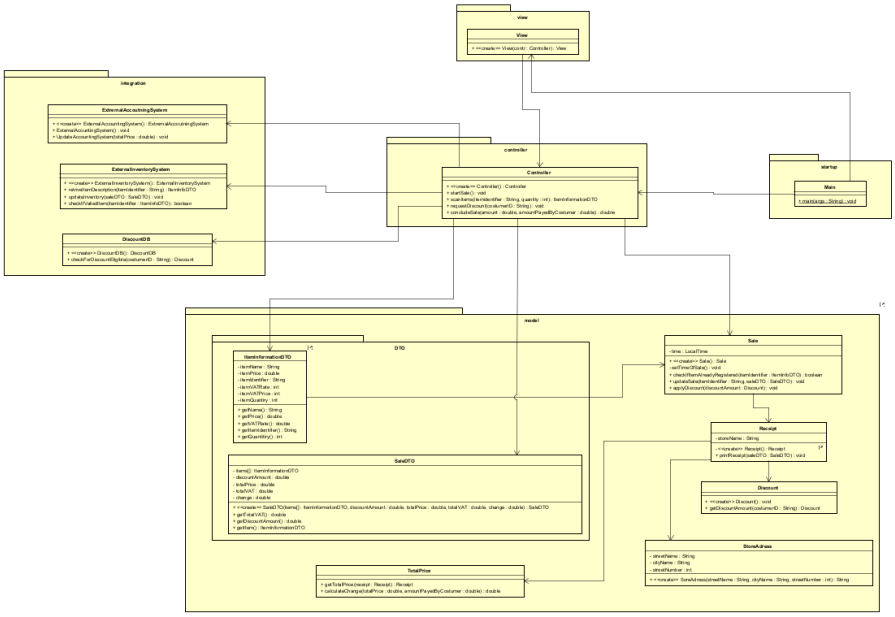

Figure 3.6 Class diagram

The diagram above is illustrating the whole design of the process of sale where all the classes are divided into different packages “layers” and have associations between them.

4 Discussion

The design is easy to understand for me and hopefully for others as well, I tried my best to make it as readable as possible, but it is many classes that might have needed some more explanation to them.

I followed the guidelines for MVC and layer patterns which resulted in 6 packages and a reasonable number of classes. The controller doesn't handle any logical operations, most of the logical operations are in model. There is no view related code in the model and all functions, operations and classes are in the appropriate packages.

I strived for low coupling, high cohesion, and good encapsulation, but it was not so easy to get low coupling because it's so many different associations which leads to a different scenario and more references. I would say that it's something in between of high and low coupling, but I also felt that all of them were quite necessary. The encapsulation is followed to the best of abilities but almost all methods needed to be public for the different packages to be able to access them.

I've mostly used objects instead of primitive data. The design follows all the java conventions to make sure it's easy to follow if you understand java programming. There are no static methods/attributes used in the design.