Seminar 2

Object-Oriented Design, IV 1350

Lucas Larsson
lulars@kth.se

15-05-2021

Contents

1 Introduction

This seminar is the second 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 design and architecture of a program.

The task at hand in this seminar is to design a program that perform all the system operation of the process sale from the previous seminar i.e., seminar 1, in this exercise it is expected to design a class diagram that represents the whole program and multiple SSD/communication diagrams that represent each single operation, including the alternative flow showed in SSD diagram from the seminar 1 analysis.

Collaborations:

Dennis Hadzialic

Daniel Chouster

Other students during "*Handlednings Pass*".

Other students in Discord Server.

2 Method

The method used throughout this report is a direct practical implementation of the theories from the course literature and the lectures giving by Leif Lindbäck.

As instructed in the Course literature "A First Course in Object Oriented Development, Leif Lindbäck" the first step was starting with the MVC Layers method, that is cerate layers to separate the different type of code into the appropriate layer, as in view have the interface, controller handles the logic operations and model form a model of the application and includes program-specific classes, with program-specific its meant classes as Sale, Receipt and SoldItem that are specific for this program .

Step two was the design of the system operations from the SSD diagram designed in seminar 1, for each operation an interaction diagram was created. One system operation at a time.

Interaction diagrams serve as a ground for how the programing of the actual application is going to look like, it shows in detail how different classes interact and communicate internally, which methods are executed in which order, as well as the parameter passed in the method and the return variables.
With the above statement in consideration, the operation is designed in the order of execution i.e., in this case the first method to be designed is "startSale" from the bottom up because if "View" has to call some method in Controller it has to have reference to it.

All the interaction diagrams describing the system operations were designed with the three fundamental design principles, encapsulation, high cohesion and low coupling, examples with explanation found in the result section.

After finishing one system operation the main Class diagram representing the whole application is updated to keep track of the progress and the remaining tasks.

Following the update of the class diagram the system operation is implemented in code. This part was not mandatory nor recommended for seminar 2 but since I missed the deadline for the seminar, I had the time and wanted to try a different approach, as it is the way Object Oriented programing is inducted in work environment.

3 Results

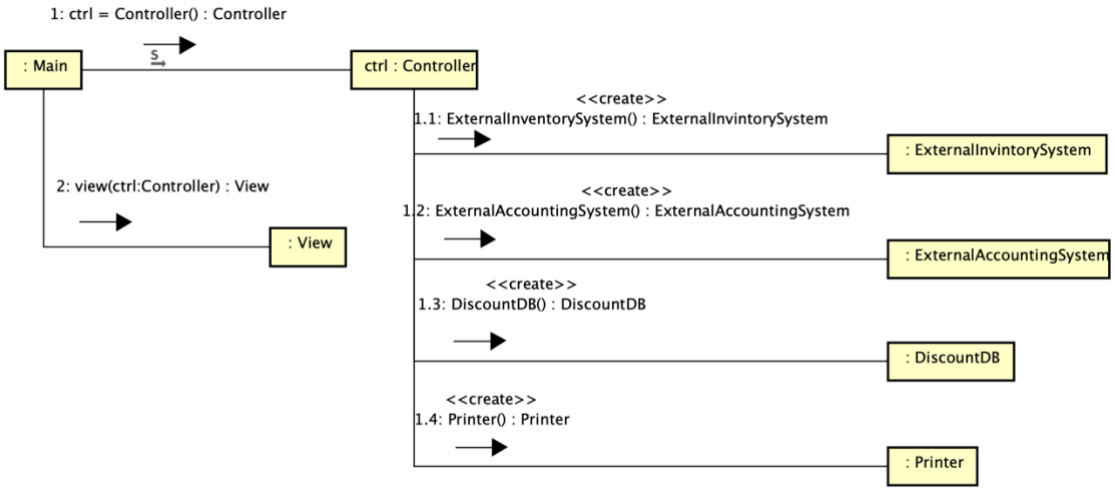

Figure 3.1: communication Diagram describing the operation startUp

The first diagram describes the operation start up as stated above, it shows in what order the calls are executed for an example an instance of the class controller “ctrl” is created in the first call and then passed to the View class, the controller in its turn create instances of the above-mentioned classes as shown in the figure 3:1.

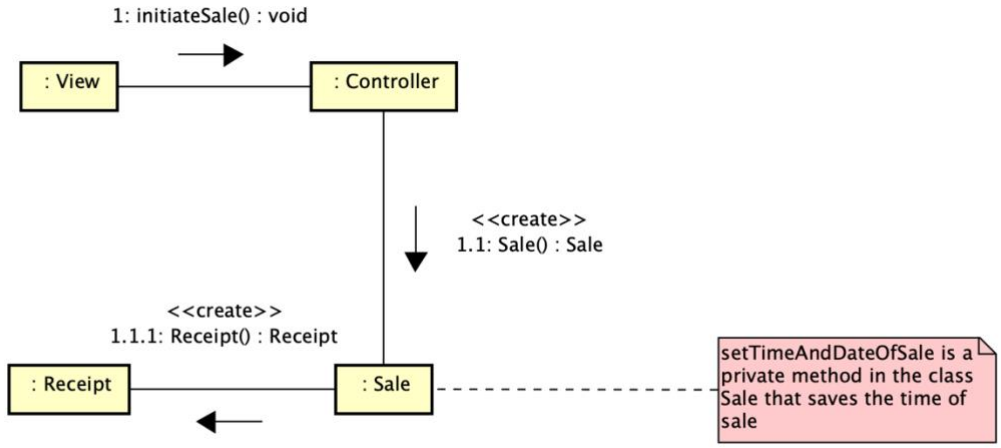

Figure 3.2: communication Diagram describing the operation initiateSale

Figure 3:2 shows the operation initiateSale where the first call is from the view i.e, the cashier to the controller to start a new sale, the controller creates an instance of the class Sale, Sale saves the date and time of the sale and then creates an instance of the class Receipt.

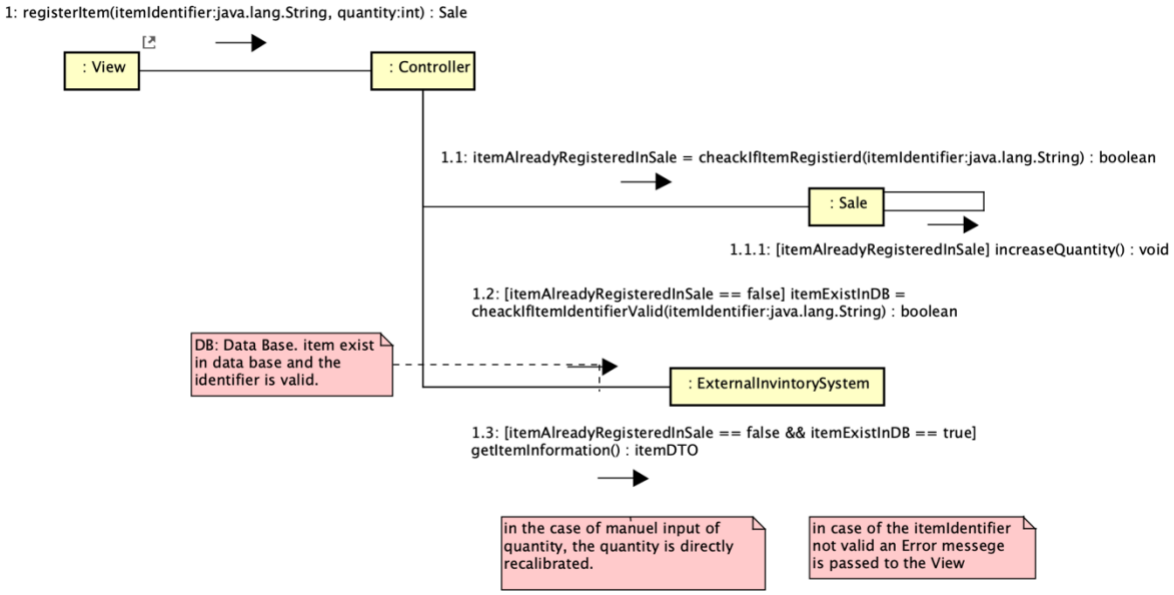

Figure 3.3: communication Diagram describing the operation registerItem

Register item is the operation where the cashier enters the item identifier to register an item in the sale list.
the item identifier is passed to the class controller, controller checks first if the item is registered in the sale list. if true the quantity is increased in the list of items.

If false, the item identifier is passed to External inventory system to check if the item identifier is valid. This method in its turn has 2 cases:
Case 1.the item is not valid, in that case an Error message is returned to the View.
Case 2 the item identifier is valid, and the items information is fetched and registered in list of items.

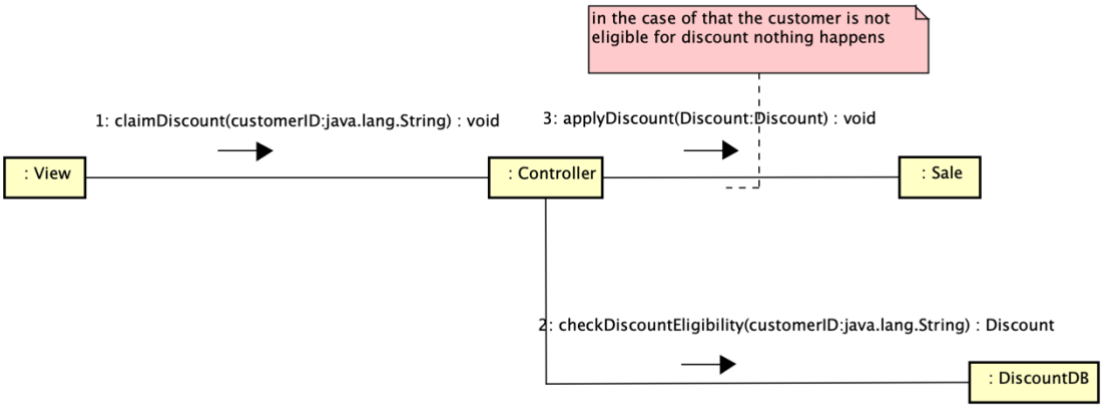

Figure 3.4: communication Diagram describing the operation discount

Claim Discount parameter is the customers unique store ID, it is passed to the controller, and the controller passes it to the Discount Date Base to check what kind of discount is the customer eligible to, if they are at all eligible.

If they are eligible for discount, the method apply discount is called, if not nothing happens.

The method that checks customer eligibility to discount is not specified more since we don't have Data base in This exercise.

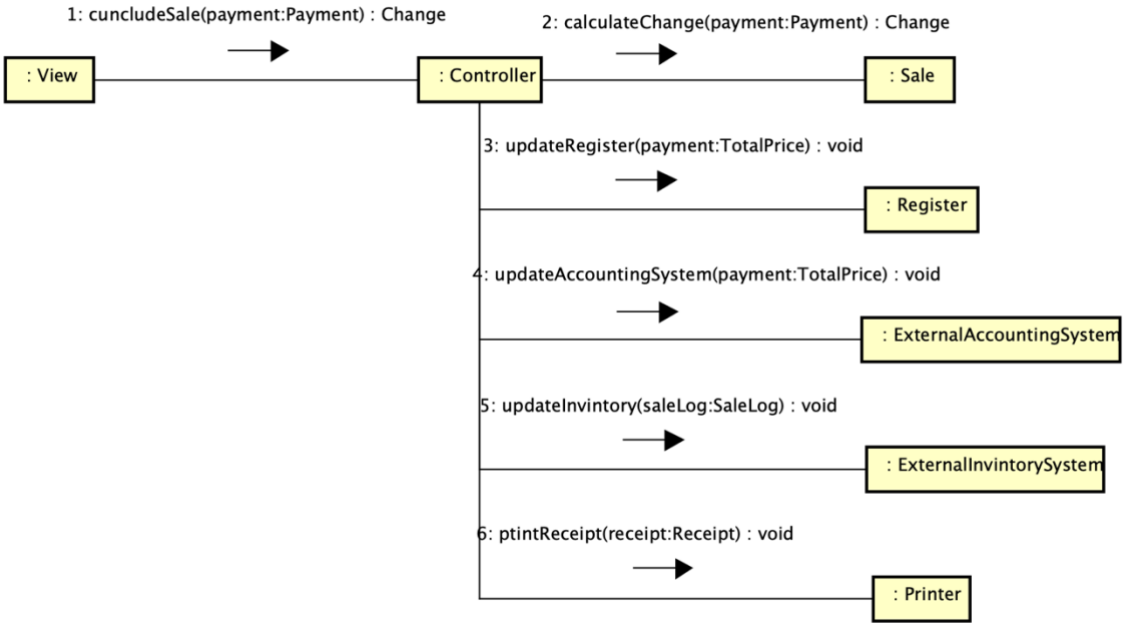

Figure 3.5: communication Diagram describing the operation cuncludeSale

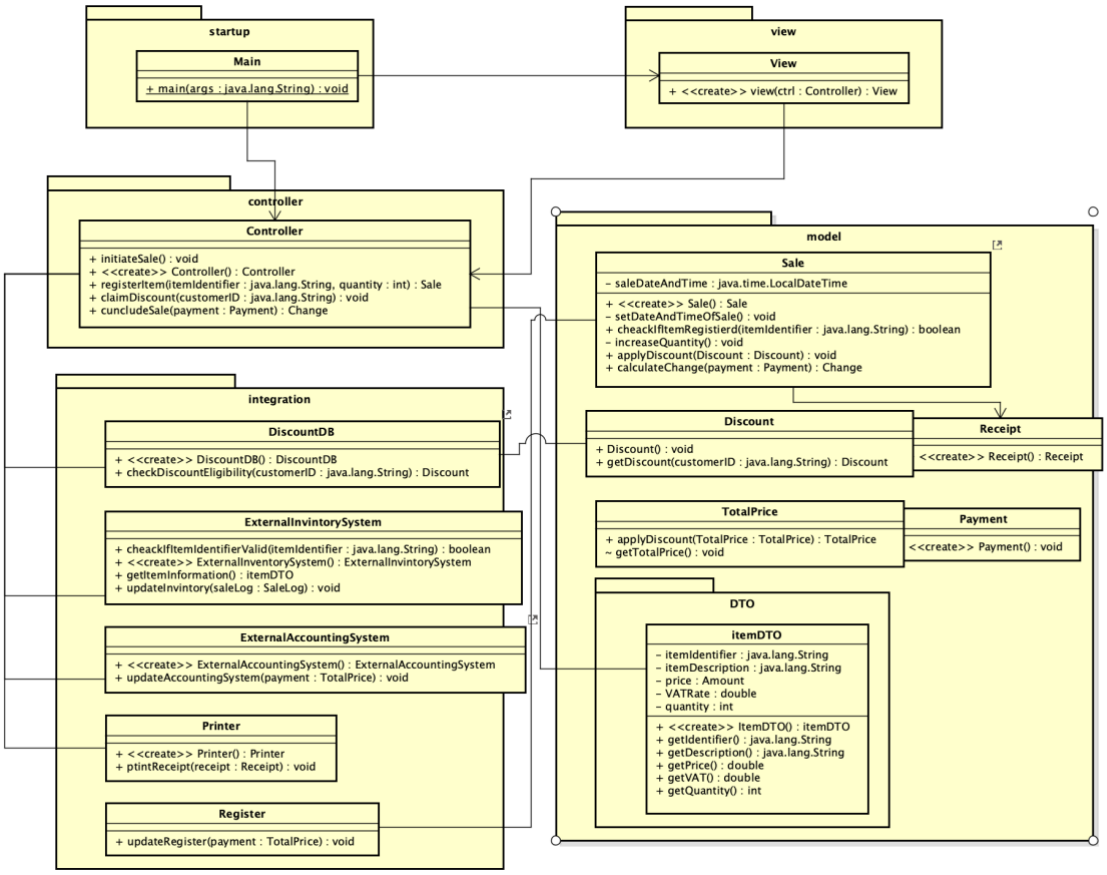

Figure 3.6: A class diagram describing the end result of the whole program

The end result is as stated above, the class diagram shows how the different classes are divided into layers and the associations between them.

4 Discussion

- *Is it easy to understand the design?*
  - The design is easy to understand according to me, but off course that could be because I've designed the whole model, so in effort to test if it's easy to understand, I showed the design to some classmates that agreed that it's easy to understand.

- *Are the MVC and Layer patterns used correctly? Are there enough layers? Enough packages? Enough classes? Does the controller handle any logic (it should not)? Is there view-related code in the model (there should not be any)?*
  - The MVC and Layer patterns are used correctly, the design uses 5 main packages and an adequate number of classes. The controller doesn't handle any logic, all logic operations are handled in the appropriate classes. There's no view-related code in the model, all view-related classes and code is designed to be in the view Layer.

- *Is there low coupling, high cohesion and good encapsulation?*
  - Yes, an example is figure 3:2, where a class creates an instance of a class and the new created class creates an instance of another class.
- *Is the use of low coupling, high cohesion and encapsulation explained in the report?*
  - Yes, it is, a example is the answer of the previous question and even the in the result section where an explanation of the diagram is provided.

- *Are parameters, return values and types correctly specified? Is it clear from where values come?*
  - Yes, this can be shown in the figures 3:1-3:6, in the mentioned communication diagrams it is shown in detail, all method calls, parameters and return types/variables and Void.

- *Are objects used instead of primitive data as much as possible?*
  - Yes, objects of classes are used throughout the program instead of primitive data as much as needed.

- *Are there static methods or attributes? If so, is that a mistake?*
  - There are no static methods/attributes in this version of the design.
- *Are Java naming conventions followed?*
  - Yes, all naming conventions are correctly followed, i.e. CamelCase.
- *Is the method and result explained in the report? Is there a discussion? Is the discussion relevant?*
  - The method and the result are explained in the report.
  - Yes there's a discussion, it is mainly composed of answers to the assessment criteria provided by the teacher, but yeah it is relevant.