Seminar 1

Object-Oriented Design, IV1350

Ayda Ghalkhanbaz
aydag@kth.se

March 27, 2024

Contents

1 Introduction

The first task of the project is to design a domain model and a system sequence diagram for a retail store.

The model is based on actions during the sale process. It starts by the customer's arrival with the bag of items that the customer intends to purchase. Then the cashier starts the sale process by scanning the items. At the end of the sale, the customer may request a discount and then pay for the items in cash. The cashier then gives the receipt to the customer. A more detailed description is provided through the task.

This report mainly covers how the domain model and the system sequence diagram were designed, and the results will be shown and discussed in the later sections. I cooperated with Naveed Rahman and Nora Say to accomplish this task.

2 Method

The methods used for both the domain model and SSD (System Sequence Diagram) were explained thoroughly in the book '*Object Oriented Development'*, but a short explanation of the used methods are as described below.

2.1 Domain Model

Designing a domain model involves several steps in order to find class candidates and their associations. The steps are as follows:

1. Noun identification
2. Category list
3. Remove unnecessary entities
4. Find attributes
5. Find associations

The methods used for both the domain model and SSD (System Sequence Diagram) were explained thoroughly in the book "Object Oriented Development", but a short explanation of the used methods are as described below.

Domain model: Designing a domain model involves several steps in order to find class candidates and their associations. The steps are as follows:

The first step of the design is to find appropriate class candidates and in order to do that, all nouns in the project description were identified. However, it is possible to miss a candidate during the noun identification, therefore a category list (provided in chapter 4 of the book) is used to find further possible class candidates.

In the third and fourth steps, all unnecessary candidates are removed or assigned as an attribute to the remaining classes. After assigning attributes to all classes, associations were created between classes.

2.2 SSD

The only tricky part to complete this task was to identify the *actors* that have interactions with the system. There were no special methods to do that unlike the previous task.

The first step in this task is to identify all the *actors* that have a interaction with the system directly. Then using the project specification, all of the interactions between actors and the system were determined.

3 Result

The designed domain model for the first task is illustrated in figure 3.1 while the system sequence diagram is shown in figure 3.2. These models are designed with the consideration of both basic flow and alternative flow. In the two upcoming sections, the model for each task is briefly explained.

3.1 Domain Model

The main process is illustrated through the associations between the `Cashier`, `Customer`, `Item`, `Sale`, `DiscountCatalog` and `Payment` classes. The customer arrives with items to purchase, the cashier starts a sale and scans the items. If the customer asks for a discount, the cashier sends a request to the discount catalog (database). The discount catalog may affect the total price of the sale. The sale then ends with the customer's payment.

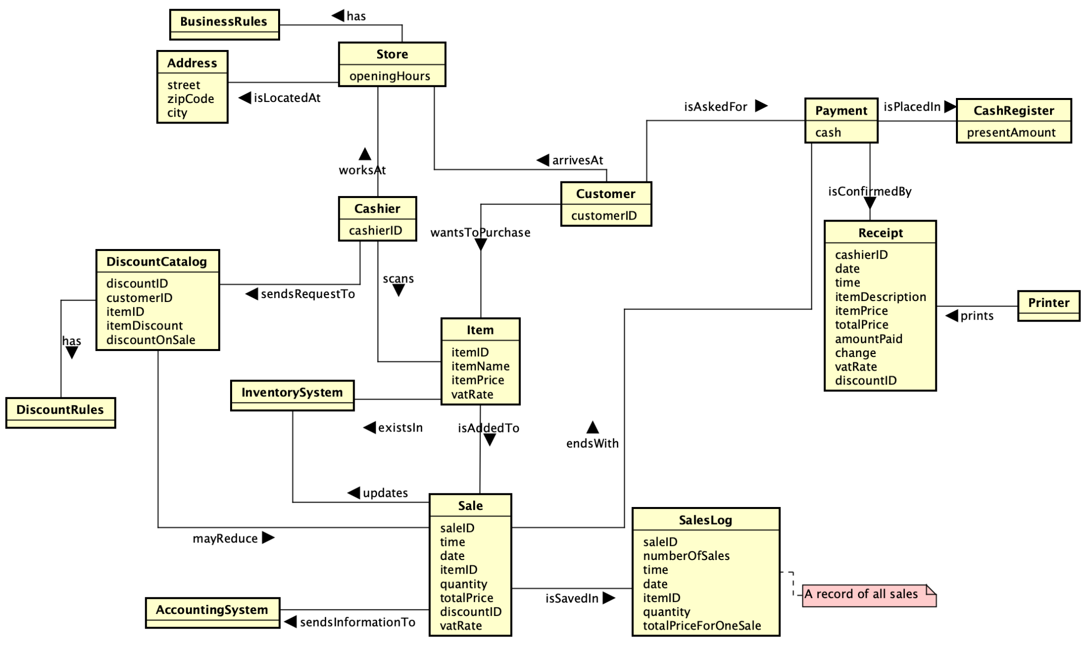

Figure 3.1: A sample diagram to illustrate caption (this text), numbering and reference in text.

3.2 SSD

The SSD model mainly illustrates the flows and interactions between actors (which are objects of classes) and the system. Probably the most confusing part of this model is if-statements and the loop.

Starting with the loop (since there might be more than one item for the cashier to scan), the cashier starts scanning an item. The first if-statement covers the possibility of entering an invalid item ID, which results in a "invalid ID" message from the system.

In case of a valid item ID, there is another scenario, that is, when the cashier enters an item ID that is already scanned (by mistake or intentionally), this will cause an update in the inventory system.

Another case scenario is when the cashier wants to scan multiple items with the same ID. In this case, instead of scanning each item, the cashier enters the item ID and the quantity of the item.

Otherwise, if none of these scenarios are actual, the system simply fetches the information about the item from the inventory system.

Applying a discount on the sale is also done with a simple if-statement, where the system checks whether there are any discounts applicable for the customer, if so, then the total amount of discount is returned through the system to the cashier.

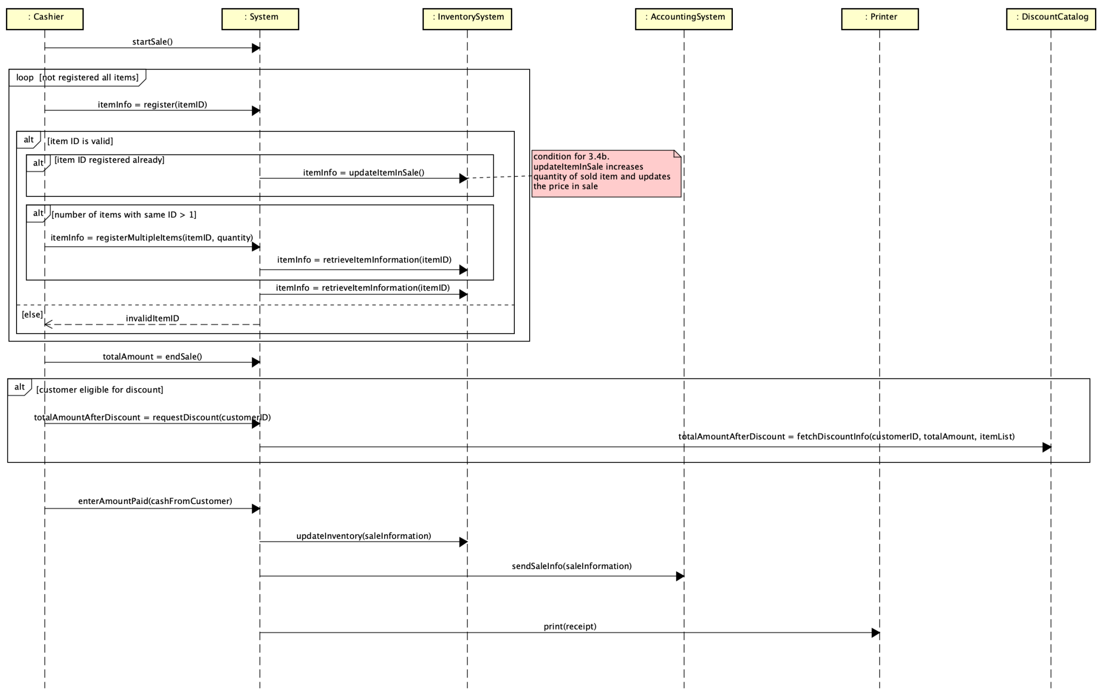
: Cashier
: System
: InventorySystem
: AccountingSystem
: Printer
: DiscountCatalog
startSale()
loop [not registered all items]
itemInfo = register(itemID)
alt [item ID is valid]
alt [item ID registered already]
itemInfo = updateItemInSale()
condition for 3.4b. updateItemInSale increases quantity of sold item and updates the price in sale
alt [number of items with same ID > 1]
itemInfo = registerMultipleItems(itemID, quantity)
itemInfo = retrieveItemInformation(itemID)
itemInfo = retrieveItemInformation(itemID)
[else]
invalidItemID
totalAmount = endSale()
alt [customer eligible for discount]
totalAmountAfterDiscount = requestDiscount(customerID)
totalAmountAfterDiscount = fetchDiscountInfo(customerID, totalAmount, itemList)
enterAmountPaid(cashFromCustomer)
updateInventory(saleInformation)
sendSaleInfo(saleInformation)
print(receipt)

Figure 3.2: A sample diagram to illustrate caption (this text), numbering and reference in text.

4 Discussion

The designed domain model is neither programmatic or naïve. The reason that the DM is not programmatic is that the model demonstrates real-world objects. We constantly tried to keep us within the frame of reality, i.e. what the customer "*actually*" sees with the arrival at the point-of-sale.

For instance, there are no details on how the inventory system or the accounting system works on the back scene, since the customer can not see these details, instead, the customer sees the payment process or how the cashier scans the items. No internal software problems or classes are included in the model.

I believe the model is not naïve either because we tried to not include any straightforward, obvious and not sophisticated associations.

It is not a 'spider-in-the-web' either since there is no class with many associations while other classes have zero or few associations. All of the main classes have at least 3 associations that are reasonable.

In my opinion, the model has a reasonable number of classes and includes all necessary classes and attributes. To make sure that all important classes are included, we went through the project specification and the category list several times to find all possible candidates for classes and attributes. For instance, in a real scenario, a receipt contains information such as date and time, the cashier's ID/name, the item description, total price, etc. Which in this case, the `Receipt` class contains all of these information and attributes.

Although designing this model took several days and a full concentration, one might have a different opinion on the design which is totally respectful and I am open for improvements.

When designing the domain model and SSD the naming conventions were followed carefully. For instance, all classes are in the upper camel case (`SalesLog`) and all attributes and messages in the SSD are in lower camel case (`itemPrice`, `startSale()`).

The objects used in the SSD are those that have a direct interaction with the `system`. For example, the `cashier` works with the `system` without any intermediator. An example of an incorrect object would be the `customer` who doesn't interact with the system directly.

Additionally, the external systems such as `inventory system` and `accounting system` are also included in the SSD, since they are also actors when it comes to the interaction with the `system`.

As an example, everytime that the `cashier` scans a new item, the `system` fetches the corresponding information from the external `inventory system` and returns the information on the screen to the `cashier`.