Seminar 1

Object-Oriented Design, IV1350

Zak Ora, Zako@kth.se

2025-03-24

Project Members:
[Zak Ora, zako@kth.se]

Declaration:

1 Introduction

The goal of this seminar is to get acquainted with a UML modeling tool, practice creating a domain model, and practice creating a system sequence diagram (SSD). The requirements include creating a domain model for a retail store based on the Process Sale requirements specification, and drawing an SSD illustrating the basic and alternative flows of Process Sale.

2 Method

2.1 Literature Study

Before starting the tasks a literature study was made to ensure that enough knowledge on the subject had been collected to complete the assignment correctly. The study mostly consisted of watching the lecture videos created by Leif Lindbäck, which taught the process of creating a Domain Model Diagram and System Sequence Diagram.

2.2 Domain Model Diagram

The Domain Model (DM) was designed using the Astah Professional program, using the built in Unified Modeling Language and Class Diagram editor to create the finished model. Information regarding the process of creating a domain model was collected in the literature study and used as a step-by-step guide. First, the nouns in the project description was used to create a list of key words, which was then converted into classes in the Astah application. By doing this, the processes was eased and ensured that the entire system was in consideration from the start of the designing process. Secondly, a category list was used as a reference and inspiration point, to see if any more classes should be made to create the diagram. Thirdly, unnecessary entities and ambiguous class names were removed from the diagram. This helped in making the diagram more concise and centred around the most important topics of the task. Next, attributes were created for each class, in addition to removing classes that was better fit as an attribute for some other class. Lastly, relations between the classes were added to elucidate the diagram and make it easier to understand the importance of each class and how they connect together.

2.3 System Sequence Diagram

After the literature study and the creation of the domain model, the system sequence diagram (SSD) was ready to be made. By following the video lecture by Leif Lindbäck and learning how to use the program, the basic outline of the SSD could be done. More over, by using the seminar 1 task description of the basic flow and alternative flows, messages between each class was made to ensure that it follows the logic and requirements of the assignment. Lastly, notes were also added to ensure that there was no confusing messages, loops or if-statements.

3 Result

The seminar aimed to develop a Domain Model Diagram (DM) and a System Sequence Diagram (SSD) to accurately represent the “Process Sale” use case. The results reflect a structured approach in modeling the problem domain, ensuring correctness, clarity, and adherence to UML standards.

3.1 Domain Model Diagram

The final Domain Model Diagram consists of essential classes that accurately represent the problem domain without introducing unnecessary complexity. The model includes entities such as *Sale*, *Item*, *Customer*, *Payment*, and *Store*. These classes are connected by meaningful associations, ensuring that each class plays a distinct role in the system.

A systematic approach was employed to extract key entities from the problem description. The first step involved identifying significant nouns, which were later transformed into classes. Further refinement was conducted to eliminate redundancies and ensure appropriate abstraction levels. Additionally, attributes were carefully assigned to maintain a balance between class complexity and relevance.

Appropriate relations between classes were defined, ensuring accurate representation of constraints within the system. For example, assuring that the cashier has relations that accurately depicts the real world, such as the Register and the Store. The relations were also named and connected (with the arrows) in a way that makes it easier for the reader to understand the flow of the diagram.

The domain model can be seen under figure 1, or the GitHub repository linked at the end of the result chapter.

3.2 System Sequence Diagram

The SSD illustrates the interactions between external actors and the system during the execution of the “Process Sale” use case. The sequence of messages and their return values ensures logical coherence, enabling a smooth flow of the transaction process.

The diagram includes the key objects: *Cashier*, *System*, and *Register*, each playing a defined role in the transaction process. The main operations such as *enterItem*, *endSale*, *makePayment*, and *issueReceipt* are appropriately included, with return values ensuring clarity in system responses.

Notably, the SSD adheres to UML notation guidelines, following proper syntax and visual clarity. The placement of messages and activation bars aligns with best practices, ensuring that the sequence of interactions is accurately represented. The diagram can be seen in figure 2 in this report, or under the GitHub repository.

GitHub Repository

4 Discussion

The developed models were evaluated against the assessment criteria to ensure correctness, completeness, and adherence to object-oriented principles. This section discusses key aspects of the solution, potential improvements, and how it aligns with best practices in object-oriented design.

4.1 Evaluation of the Domain Model Diagram

One of the critical evaluation points in the DM is ensuring that it is neither a programmatic DM nor a naive DM. The developed DM avoids these pitfalls by maintaining a conceptual level rather than a technical implementation focus. The diagram effectively captures the essential entities of the problem domain, ensuring clarity and ease of understanding. A common issue in domain modeling is the presence of "spider-in-the-web" classes—overly central classes with too many dependencies. The designed DM mitigates this issue by distributing responsibilities appropriately. For example, *Sale* interacts with *ExternalInventorySystem*, *Receipt*, and *Discount*, but does not dominate the model excessively. Another critical aspect of evaluation is ensuring a reasonable number of classes and associations, with around 1.2 associations per class. The DM includes only necessary classes, avoiding redundancy while ensuring completeness. Furthermore, attributes were assigned carefully to distinguish between classes and data elements. Lastly, UML standards were rigorously followed, ensuring the correct use of notation, naming conventions, and relationship multiplicities. This adherence enhances clarity and allows for seamless communication of the model to stakeholders.

4.2 Evaluation of the System Sequence Diagram

The SSD was evaluated on its ability to accurately capture interactions during the "Process Sale" use case. The inclusion of correct objects, operations, and return values ensures that the model represents system behavior effectively. One key strength of the SSD is its adherence to the basic and alternative flows of the "Process Sale" use case. The diagram systematically captures each step, from the initiation of a sale to the processing of payment and receipt issuance. This ensures that all relevant operations are included and logically sequenced. A potential improvement in the SSD is the addition of explicit handling of exceptional cases, such as invalid item scanning or insufficient payment. While the core functionality is well-represented, incorporating error-handling scenarios would enhance the robustness of the model. Naming conventions and UML compliance were strictly adhered to in the SSD. Each message is clearly labeled, ensuring readability and adherence to best practices in UML modeling.

Figures

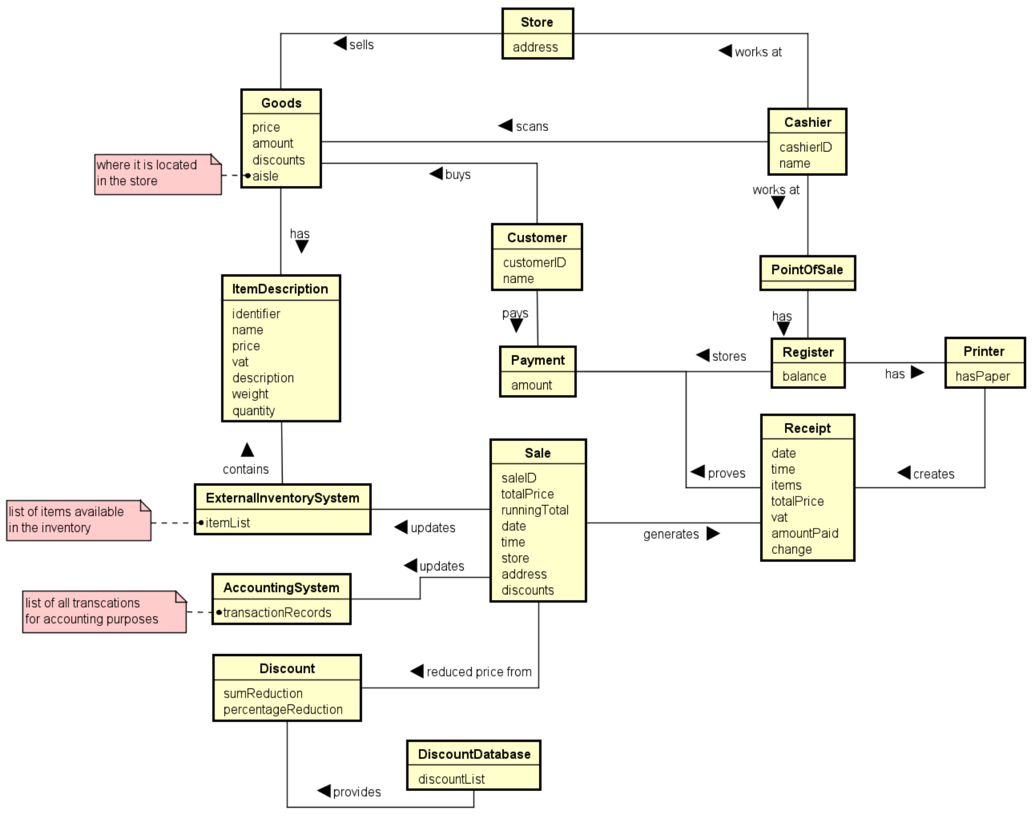
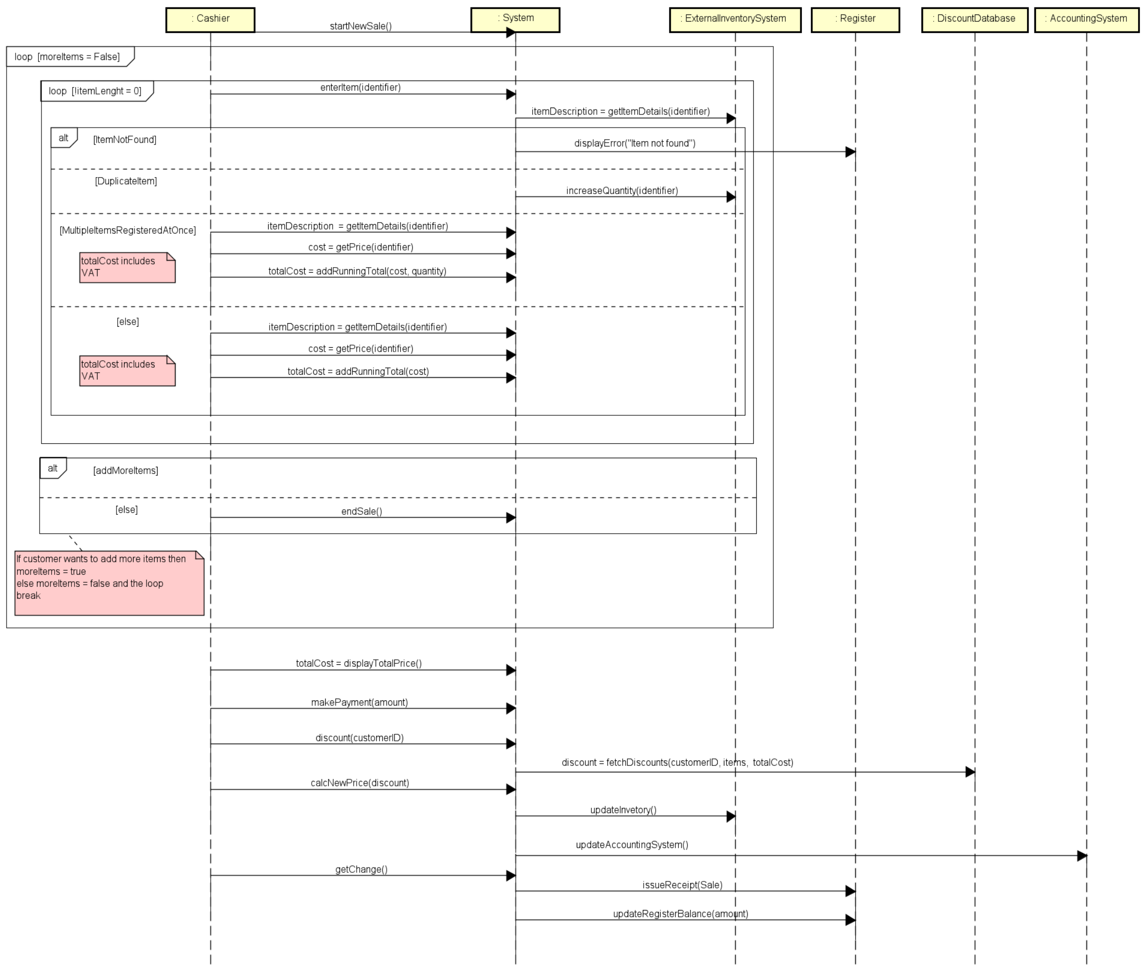


Figure 1: Domain Model Diagram created for Seminar 1.



: Cashier
: System
: ExternalInventorySystem
: Register
: DiscountDatabase
: AccountingSystem
startNewSale()
loop [moreItems = False]
loop [itemLenght = 0]
enterItem(identifier)
itemDescription = getItemDetails(identifier)
alt
[ItemNotFound]
displayError("Item not found")
[DuplicateItem]
increaseQuantity(identifier)
[MultipleItemsRegisteredAtOnce]
itemDescription = getItemDetails(identifier)
cost = getPrice(identifier)
totalCost includes VAT
totalCost = addRunningTotal(cost, quantity)
[else]
itemDescription = getItemDetails(identifier)
cost = getPrice(identifier)
totalCost includes VAT
totalCost = addRunningTotal(cost)
alt
[addMoreItems]
[else]
endSale()
If customer wants to add more items then moreItems = true else moreItems = false and the loop break
totalCost = displayTotalPrice()
makePayment(amount)
discount(customerID)
discount = fetchDiscounts(customerID, items, totalCost)
calcNewPrice(discount)
updateInvetory()
updateAccountingSystem()
getChange()
issueReceipt(Sale)
updateRegisterBalance(amount)

Figure 2: System Sequence Diagram created for Seminar 1.

References

1. `https://www.youtube.com/watch?v=5HQEdhA3vMQ&feature=youtu.be`
2. `https://www.youtube.com/watch?v=S5mvmkCGsb8&feature=youtu.be`
3. `https://www.youtube.com/watch?v=AjKiV2QsQ9Y&feature=youtu.be`
4. `https://www.youtube.com/watch?v=EdXjr4ddqqE&feature=youtu.be`