1st SEMESTER PROJECT

Vestbjerg Byggecenter Management System

UNIVERSITY COLLEGE
OF NORTHERN DENMARK

Computer Science AP Degree

Dmai0919 - Group 3

2019/2020 autumn semester

25.11.2019 - 13.12.2019

UNIVERSITY COLLEGE OF NORTHERN DENMARK

COMPUTER SCIENCE AP DEGREE
CLASS: DMAI0919 - GROUP 3

1ST SEMESTER PROJECT REPORT

VESTBJERG BYGGECENTER MANAGEMENT SYSTEM

2019/2020 - AUTUMN SEMESTER
25.11.2019 - 13.12.2019

Group members:

- Dino Maras
- Erik Petra
- Krisztián Henrik Papp
- Marek Strúčka
- Pavel Mokraček

Supervisors:

- Dimitrios Kondylis
- Gianna Belle
- Mogens Holm Iversen

Repository path: https://kraka.ucn.dk/svn/dmai0919_1Sem_project_3
Revision number: r118

Normal pages / characters: 14.1[1] pages / 33 863 characters[2]

[1] Characters / 2400 = X.X normal pages
[2] A normal page is defined as 2.400 characters incl. whitespaces and footnotes. Frontpage, title page, table of contents, literature list and appendices are not included

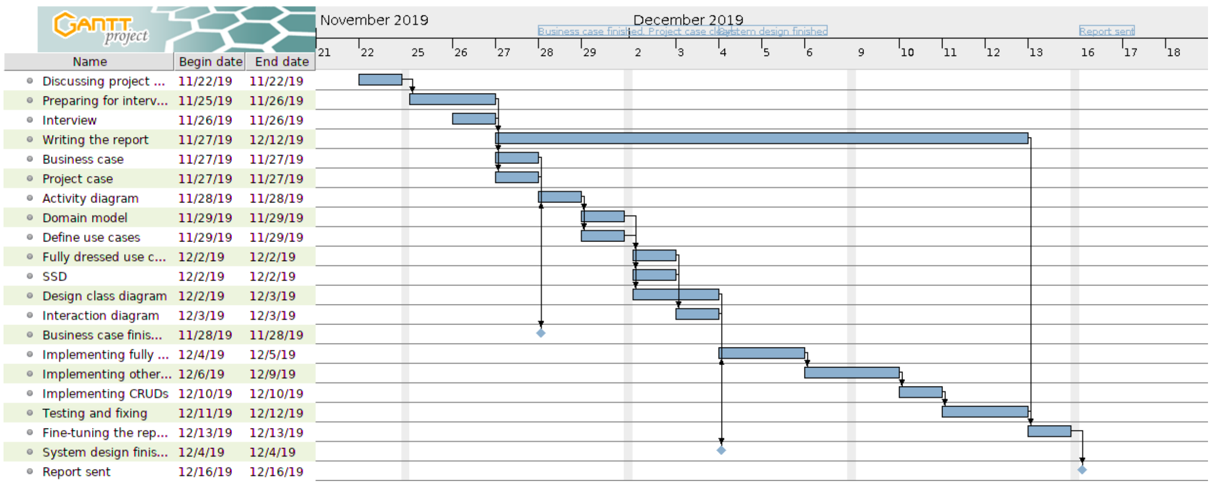
Table of contents

Introduction

This report contains the whole workflow description of developing and implementing a solution for a Vestbjerg Byggecenter company. This company, although well-known in northern Denmark, has not updated all the work environment constantly which resulted in pretty old and insufficient UNIX system they kept using until now. Also, since the company wanted their business part to be improved as well, they asked us to provide a business analysis for the company too. The report covers the team developing a business analysis after an interview with the company's sales representative, developing the diagrams and suggesting improvements for the new system and finally the process of implementing the solution.

Time schedule

We created a time schedule via Gantt chart that we were presented with during business and system development lessons. We used a program called GanttProject. GanttProject is GNU licensed and offers necessary Gantt functionality and multiple export options.



Name	Begin date	End date
Discussing project ...	11/22/19	11/22/19
Preparing for interv...	11/25/19	11/26/19
Interview	11/26/19	11/26/19
Writing the report	11/27/19	12/12/19
Business case	11/27/19	11/27/19
Project case	11/27/19	11/27/19
Activity diagram	11/28/19	11/28/19
Domain model	11/29/19	11/29/19
Define use cases	11/29/19	11/29/19
Fully dressed use c...	12/2/19	12/2/19
SSD	12/2/19	12/2/19
Design class diagram	12/2/19	12/3/19
Interaction diagram	12/3/19	12/3/19
Business case finis...	11/28/19	11/28/19
Implementing fully ...	12/4/19	12/5/19
Implementing other...	12/6/19	12/9/19
Implementing CRUDs	12/10/19	12/10/19
Testing and fixing	12/11/19	12/12/19
Fine-tuning the rep...	12/13/19	12/13/19
System design finis...	12/4/19	12/4/19
Report sent	12/16/19	12/16/19

Being presented with the project case on Friday, all of us went through the case on our own during the weekend and we met on Monday. We discussed our understanding of the project and we prepared first interview questions together with an initial time schedule. On Tuesday we performed an interview with the sales representative and got crucial feedback for our questions and disputes we had before. Everything we did was also written in the report.

We steadily moved on and created most of the business analysis by Friday. This took us a bit longer that we expected. We have also started discussing the use cases and agreed on the most complex ones. By next Tuesday we had the initial fully dressed use cases completed and we also started developing the domain class diagram.

We steadily implemented the most important use cases and fine-tuned the report on the way. We agreed to stop coding Tuesday 10th, leaving 2 days to finalize the report. At the beginning, we were mostly ahead of our schedule and as the time went by and the implementation of most important use case has been done, we were happy that we managed to complete all the tasks by the deadline date at the end.

Project case

Introduction

Vestbjerg Byggecenter is a company that has been in the business for more than 30 years already. In 2005 it changed its structure to a limited liability company with the two company founder's sons joining the management of the company.
The company spans over 2 departments, a timber one and a DIY department. The company has around 30 employees in total, therefore its structure is pretty straightforward. It has one managing director, two department managers and a few other office employees, assistants and warehouse workers. Nevertheless, we would suggest a slightly improved structure for the company here.
The most crucial problem for the company at the moment is its old UNIX system which does not meet the information handling criteria in a company with a turnover of almost 70mil. DKK.
Therefore, we will mostly present our solutions for this issue, designing and implementing a brand new stock handling system for the company, trying our best to provide a complex but user friendly responsive system that will be easy to navigate in.

Interview with the customer representative

During the very first days we got to know the Vestbjerg Byggecenter's case. It was written pretty vaguely, so we had a lot to ask and clarify. We have come up with some initial SWOT analysis, company hierarchy graph, we wrote down the most important features of the system we were going to implement and, most importantly, we prepared a lot of questions we had to resolve before we started working on the project.

Although the customer representative was busy, we interviewed him for 30 minutes which helped us a lot. The most important points were noted down. Those were:

- Concerning the business part, the DIY warehouse is small in size and the company therefore has to store some equipment in the timber warehouse and cannot track it
- They would also like to establish a new fireplace department which was not mentioned in the case
- The company wants to create stronger bonds with the building companies
- Concerning the system implementation part, the most crucial thing that is missing is holding the location for all the items and automated loan of equipment.
- Auto generated emails and notifications come with this issue as well
- The company holds info about more than 1000 registered customers, which are by now divided into 22 groups. The number of groups should be diminished to 4-6.
- There are a lot of discounts, but the system should be able to cap the discount at 20% or value set by management
- The system has to hold a list of the orders, must be able to create a pre-order, must be able to find an order by a customer

All the information we found out during the interview were of high importance to us. Having finished the interview, we were left mostly with our own knowledge inside the group and we had to come up with a proper business case and then with a functioning system. Our solution is documented below.

Stakeholders

Stakeholder	**Contribution**	**Interest**	**Influence / attitude**	**Importance**
Anders Olesen	Finances, Decisions, Knowledge, Feedback	A better, more user friendly system	High / positive	High
Thomas Olesen			High / positive	High
Casper Olesen			High / positive	High
Other managers	Decisions, Knowledge, Feedback		High / positive	High
Office employees	Knowledge, Feedback		Low / positive	Medium
Warehouse employees	Feedback		Low / negative	Medium
Customers			Low / neutral	Low

At first, based on the project case, we assumed that everyone has a positive attitude towards the new system, but after finishing the interview we felt like the warehouse employees fear automation and thus they have a fear of being replaced or simply fired. That is one of the threats, but we will solve this by implementing some business strategies that would increase the sales and thus increase the workload.

Project constraints / risk analysis

Risk	**Probability**	**Impact**	**Priority**	**Solution**
Data transfer issues	3	3	9	Try to understand the system / hire experts on that system.
UX not infeasible	3	2	6	Show mockups to managers and workers, get feedback and update it based on that.
Warehouse workers	3	2	6	Create learning sessions to adjust

worried about new system				workers, implement some business and marketing changes so that there will be more work for them to be done.
Requirement is not precise	2	3	6	Keep the client updated.

After the interview we realised that we can't contact the old system creator, so there is a high risk in transferring the data from the old system. We also have to aim for a user friendly system, since (mostly) the workers are not as familiar with using computers as we expected them to be. Also, time has to be invested at the beginning to teach all the employees how to properly manage the whole system so there are no problems later on concerning the system handling.

Product Features

Having familiarized with the case, one of the tasks we used to better understand and direct our group work was creating a product features table that helped us see the features the system had to possess and their priority. It furthermore helped us in determining the most crucial use cases as we had the priorities sorted out.

Feature	**Description**	**Priority**
K1	The system can CRUD info about an item	High
K2	The system can CRUD info about a loan item	High
K3	There should be a database with sufficient storage for the company	Medium
K4	The system has to be user friendly	High
K5	The system has to be error preventative	Medium
K6	The system has to be fast and responsive	High
K7	The system should have different features based on the user	Medium
K8	The system should handle the discount calculations well	High
K9	The item must have it's specific location, number in stock and prices	High
K10	The system must be able to print sales tags and price labels	High

Costs / Benefits of the project

New Hardware	700 000 DKK
New Software	2 100 000 DKK*
Education of workers	100 000 DKK

* the price of a software was calculated: 700 Kr salary / hour * 6 hours / day * 100 days * 5 people

Costs: The Company's budget of 3M DKK should be met, according to our initial budget we should fit into this budget. The most expensive part will be the software itself, which will take a long time to finish. New hardware is also needed, therefore money should be reserved for that as well. Last but not least, we need to educate the workers and get them familiarized with the system, which should not take so much time and money, since our aim for the system is to be as user friendly as possible and there are not that many workers to be educated.

- New hardware
- New software
- Education of workers

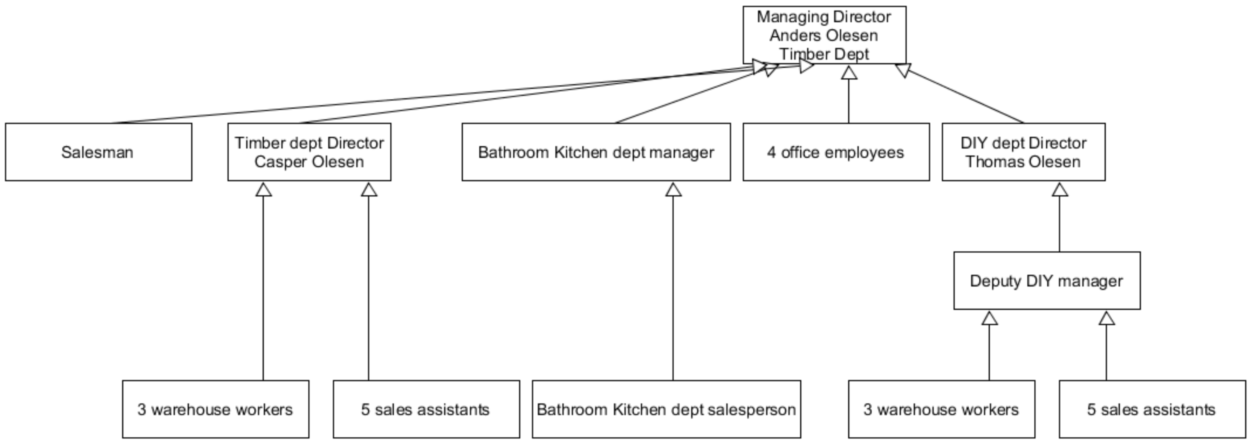
Benefits: The benefit which will impact the company's workflow the most is definitely the new system, which will be both faster and more user friendly. Also the storage of information would be more tabular and the information and statistics the managers could see would be more appropriate. Last but not least, the customer satisfaction should also be there, since the sales would be done quicker than before.

- Faster system
- Better statistics information
- Sales done quicker

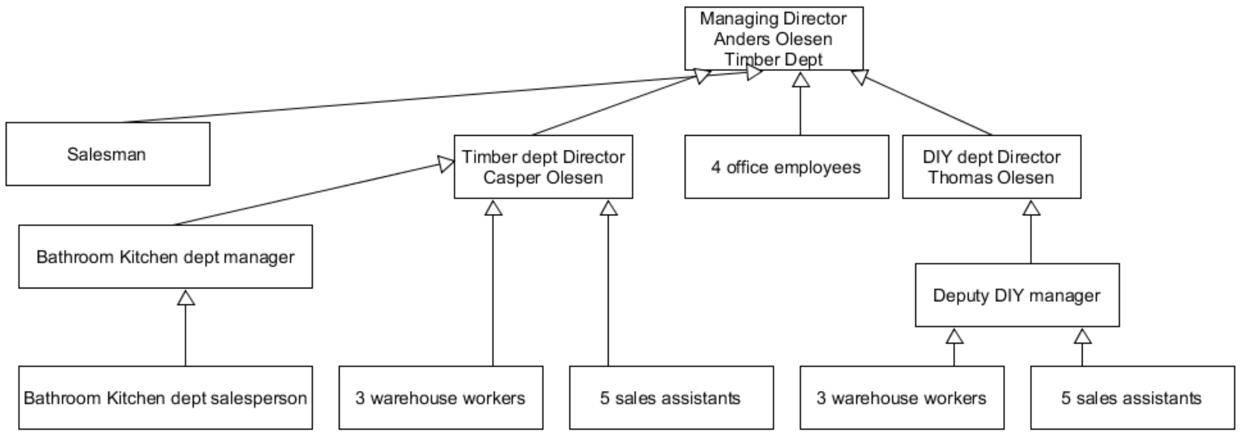
Business case

Company hierarchy

We find the company hierarchy as it is now being pretty well-structured, although we think that there is a space for improvement. As we have found out, the managing director, Anders Olesen, is slowly delegating the responsibility to his sons. This looks partially true, but we still see a crucial point concerning the hierarchy that would unburden Anders even more. That is the manager of the kitchen and bathroom department, who reports directly to the managing director as of now. We find this being both unnecessary and inefficient and we therefore suggest restructuring the hierarchy a bit so that the manager will report to the timber department manager, Casper Olesen.
Although they (Anders and Casper) both have their offices at timber department, Anders would benefit a lot from this improvement and it would not affect Casper too much, since the manager is a responsible person who works in a company for a long time, there are no problems he makes. Current hierarchy looks like this:



The hierarchy we suggest:



Description of the organizational structure

- 1 managing director, his 2 sons are directors of the 2 departments, 1 deputy manager for DIY
- 2 managers for the most important departments - bathroom and kitchen
- 1 salesman, 4 office managers for bookkeeping, wages 8 + 8 employees
- Manager has only 2 department directors reporting to him. There are also 2 managers for the 2 most crucial departments (Timber and DIY department) which they also specialize in. Employees report directly to the managers which means that the hierarchy is well defined within the system.
- There is only one salesman for the whole company and that probably means that the salesperson is overloaded with work,this means that the salesman should get a more efficient program to work in or even better a co-worker to help him out with it.
- The company has 4 office employees responsible for bookkeeping and wages. Since the company is fairly small the level of formalization is also very low because everyone knows each other and the environment is friendly because of that.
- Departmentalization is implemented within the company, that way everyone has their fixed responsibilities and that way efficiency and communication is improved between the departments of the company.
- Casper Olesen has 8 employees at Timber reporting to him, since it is a lot of employees and 2 departments it could be pretty hard on him.
- Narrow and flat organisational structure enables employees to make decisions easier and reduces the company's budget costs for the cost of employees reporting certain information a specific boss, in the case of a confusion which can lead to power struggles among management.
- Managing director Anders delegates responsibility to the department managers, motivates them and solves only the problems that he is familiar with. Others have the freedom to make decisions on their own.
- Thomas Olesen speaks to the employees a lot and respects the feedback that the employees provide him with. He is motivating when it comes to the employees, he cares about them and has a friendly management system. He is also holding yearly interviews with the employees to help him with his management but he is also strict when it comes to the quality of work
- The company has a multifunctional structure divided into 2 departments with different functions. The functions are well divided, everyone is highly skilled in what they are doing, and efficiency is also on a high level.
- The company has a clearly defined hierarchy with only a few levels of depth to it. All of the employees that are employed there have been working for multiple years which means that job security is well established, for that reason employees are loyal and have good communication and cooperation between each other.
- The company has no issues affecting its structure, everyone is working in their own departments, they have fixed working hours and strict but friendly management.

Organizational culture and description of the managers behavior

Organizational culture here reminds of Power culture, as it consist of informal communication and trust, yet managers still hold the power. Since the 3 of the main managers have family ties, this might cause problems when solving disputes and finding the right direction for the company as it might be hard to outvote them (3 votes).

- **Anders Olesen:**
  - Motivating employees
  - Distributes responsibilities
- **Thomas Olesen:**
  - Cares about employees (yearly interviews)
  - Team Building organiser
- **Casper Olesen:**
  - Detailed and controlling but helpful

SWOT analysis

A SWOT analysis was the tool we used to identify the strengths, weaknesses, opportunities and threats to the Vestbjerg Byggecenter company. Most of the SWOTs were pretty clear to us after the initial reading of the case, but a few more were added after the interview with the customer representative.

- **Strengths**
  - High number of items in stock
  - They have been in business for more than 30 years and therefore they have gained a lot of reputation
  - Member of XL - byg chain
  - good work culture and environment
- **Weaknesses**
  - Current IT system
    - With the current system, they cannot keep track of the stock of items in each warehouse
    - It’s over 20 years old, outdated, a lot of tasks are not included
  - System not integrated in XL - byg chain
  - Leases are managed manually
- **Opportunities**
  - With a new computer system the company could be able to increase their efficiency and therefore increase their revenue
  - Once the XL - byg website is integrated with the company’s own system, the sales through the website will be faster and more efficient
- **Threats**
  - With increasing amount of employees organisational change would be needed
  - Competitors with a more efficient system or lower prices

Strategic goals for the company

The company needs to think of its future. So it's important to have strategic goals of the company in mind.
First time we read the case we wrote about the company, but after having the interview we changed some parts about the strategic goals of the company. These are some of our old pointers.

- Global expansion outside of Denmark for more customer reachability
- Additional warehouses all over Denmark

These are our post interview pointers we gathered.

- Get a better system to control item locations between departments and their availability.
- Expansion of customers and products.
- Completely new system that will help with the productivity and organisation of everything and everyone that's connected to the company in any way.
- Build a website so that customers have a better view of the availability and department location of the specific item they enquire.
- Possibly get an additional warehouse because of the growing number of customers and products.

Strategic Alignment

Plan	Goals / Objectives	Relationship to project
2020 Strategic plan for Vestbjerg Byggecenter A/S system	Real-time updated item information.	This project will allow for real-time information about the products to be available. That information consists of availability of the product and the department where they are available.
2020 Strategic plan for Vestbjerg Byggecenter A/S system	Engage the employees and make their work more efficient.	This project will give the employees access to the system which will make them more productive and as well make their job a lot easier.
2020 Strategic plan for Vestbjerg Byggecenter A/S system	Future proof for company expansion.	If the Vestbjerg Byggecenter A/S company decides to acquire another warehouse and expand their business, the system will be future proof meaning it will be easy to add another warehouse to the system and check the availability of the products inside it.

Business problem & opportunity

- As the current system is very outdated, both the staff and the stakeholders think it's important that a new system is developed for the company.
- One of the major downsides of it is that it's not compatible with the XL-byg website's system and therefore it's really slow to process orders from there. Lots of processes are manual (for example the leases of machines) which also slows down the work. It takes a long time to carry out a few tasks, which could be faster or even automated.
- With a new system, which is up to today's standards the company should be able to boost their revenue as it would be more efficient and faster than the current one.

Suggested marketing strategy

In order for the company to efficiently expand their customer base, they have to apply a good marketing strategy which targets the Northern Denmark region.
As nowadays most people (and therefore most company representatives as well) like to do their "shopping" online, it's important that the company develops a website which is then advertised on different platforms (like Facebook, Google Ads, Twitter, etc.). In order to target not just the younger generations, "offline" ads are just as important as the previously mentioned online ads. These offline ads could be for example advertisements in local newspapers or billboards in this area. These offline ads are more important for local expansion.
Also, the company should aim more for providing some kind of a daily or weekly discounts for different parts of the shop- since there it would be easy to navigate in the system, the prices should be easy to alter even this often.
The company should also try to attract more people to get registered, providing them some kind of special discounts for registered users and so, so that their database of registered customers would expand drastically.

System Design

System requirements

Before we started designing the actual system, first we had to take a look at the requirements set up by the project case and the customer representative.
The following requirements were set by the company:

- Customer management
  - Storing, editing and using customer data for both sales and leases.
  - Grouping customers into 4-5 groups (groups can be set-up by managers)
  - Setting discount schemes for groups
- User management
  - Creating and editing users (employees) for the system
  - Multiple user types, each with different permissions
  - Keep track of sales made by the salesperson to calculate their commission
- Product management
  - Keeping track of stock at each warehouse
  - For each item store information about its location, cost price, recommended retail price
  - For each product store information about its amount in stock, stock at each location, barcode number
  - Create product bundles from more different products and sell them as an individual product
- Sales
  - Create and view all sales made by each employee
  - Add multiple items to order
  - Create quote which is due for payment by the customer in 14-30 days (set for each customer individually by managers, default 14 days)
  - Automatically send emails to customer for every step of the order (payment received, packing up order, sent out to delivery, delivered, etc)
- Lease management
  - Keep track of ~30 machines, their status, daily lease price, and value
  - Customer base for leases and sales is the same
- In store features
  - Possibility to print out sales tags and price labels for products

Use cases

When we started working on system design, the first thing that we decided to work on, were use cases.

Use cases:

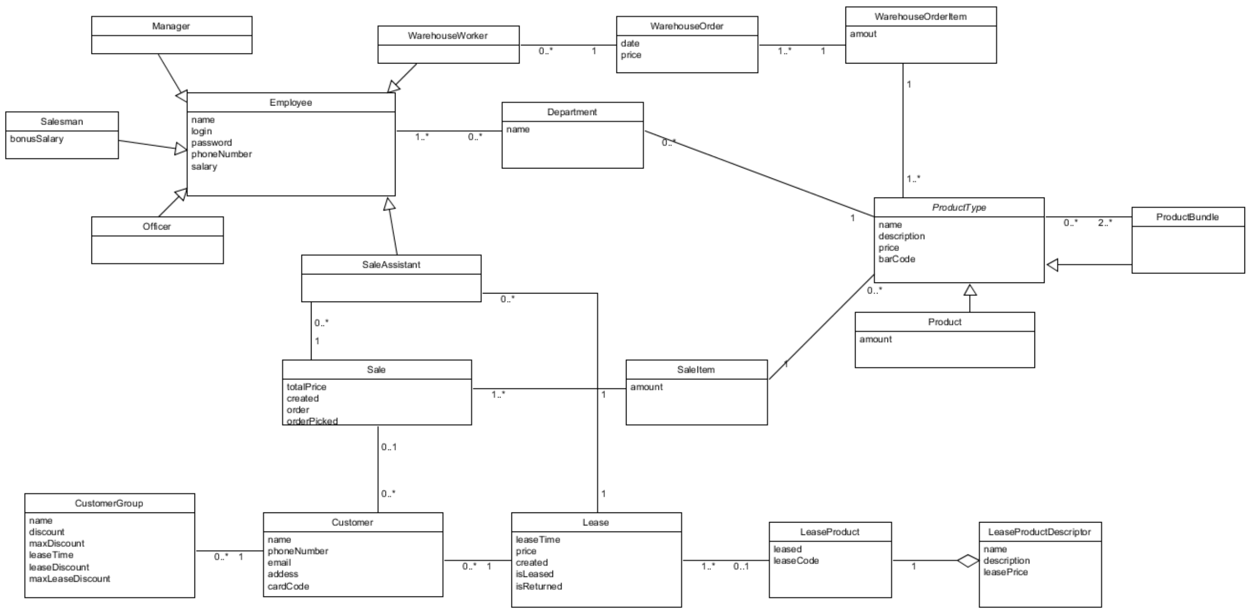
- CRUD Product
- CRUD Product bundle
- CRUD Customer
- CRUD Customer groups
- CRUD Employee
- CRUD Department
- Sell products
- Order products
- Accept products
- Lease products
- Return lease

When we agreed on use cases, we went on to determine the use case priority. We omitted CRUDs and marked most important ones with yellow.

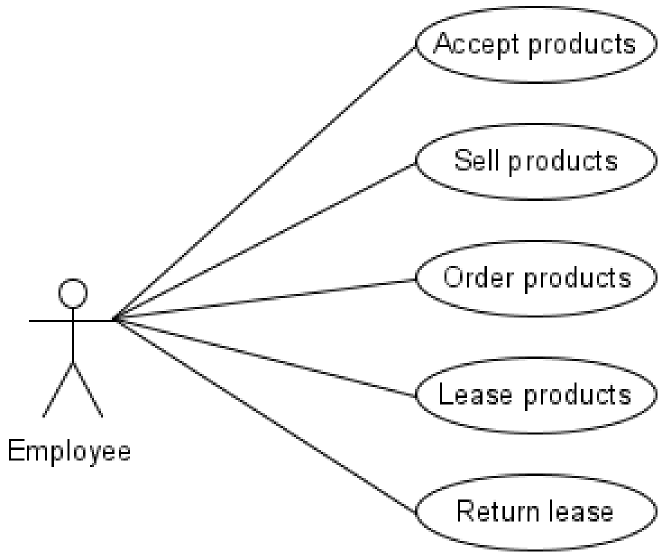
Use case priority:

Use case	**Architectural importance**	**Business value**	Priority
Accept products	10	5	50
Sell products	10	10	100
Order products	7	10	70
Lease products	6	10	60
Return lease	5	6	30

Domain model

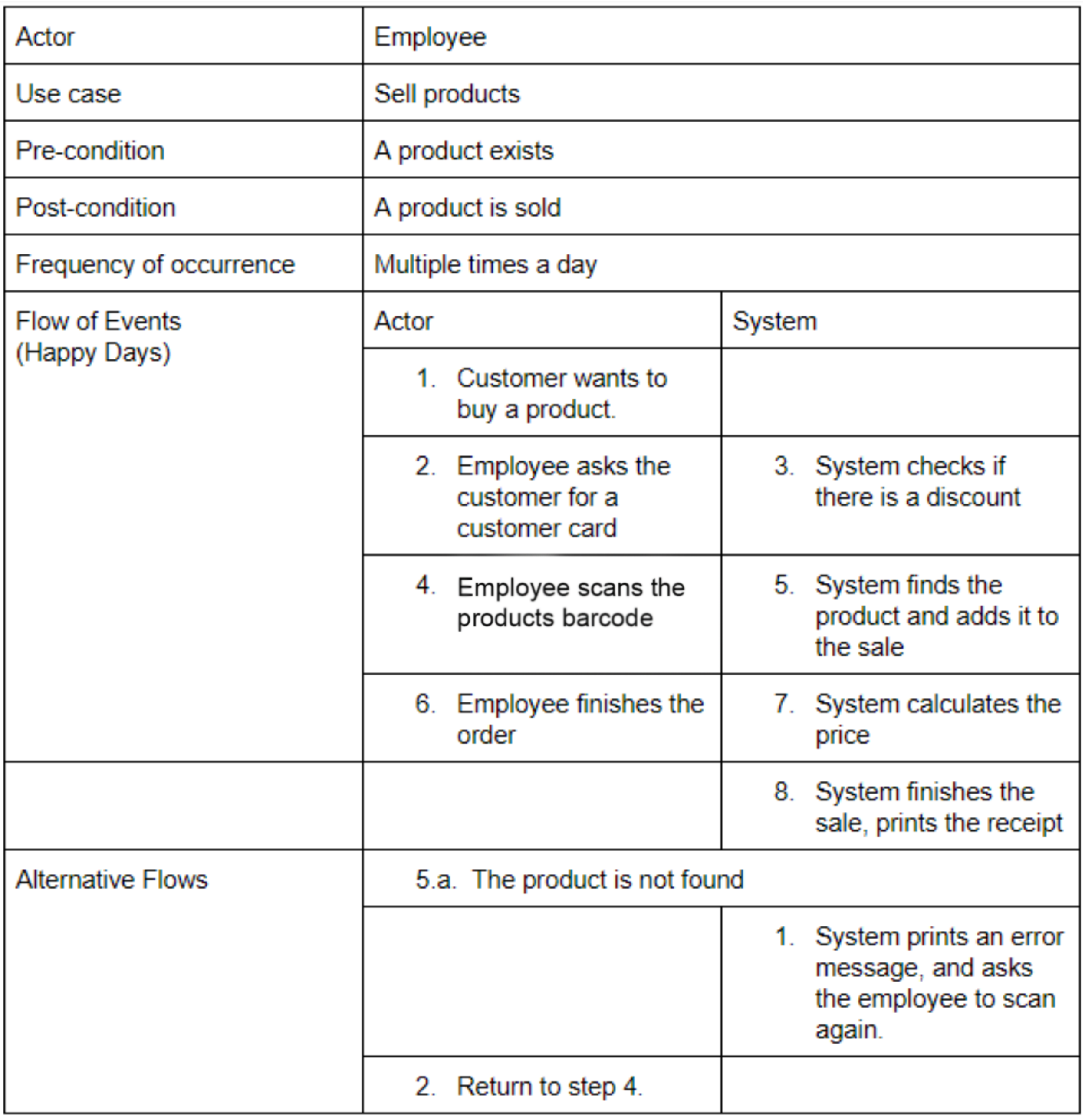


Use case diagram for the highest priority use cases



Fully dressed description for the most complex use case - Sell products

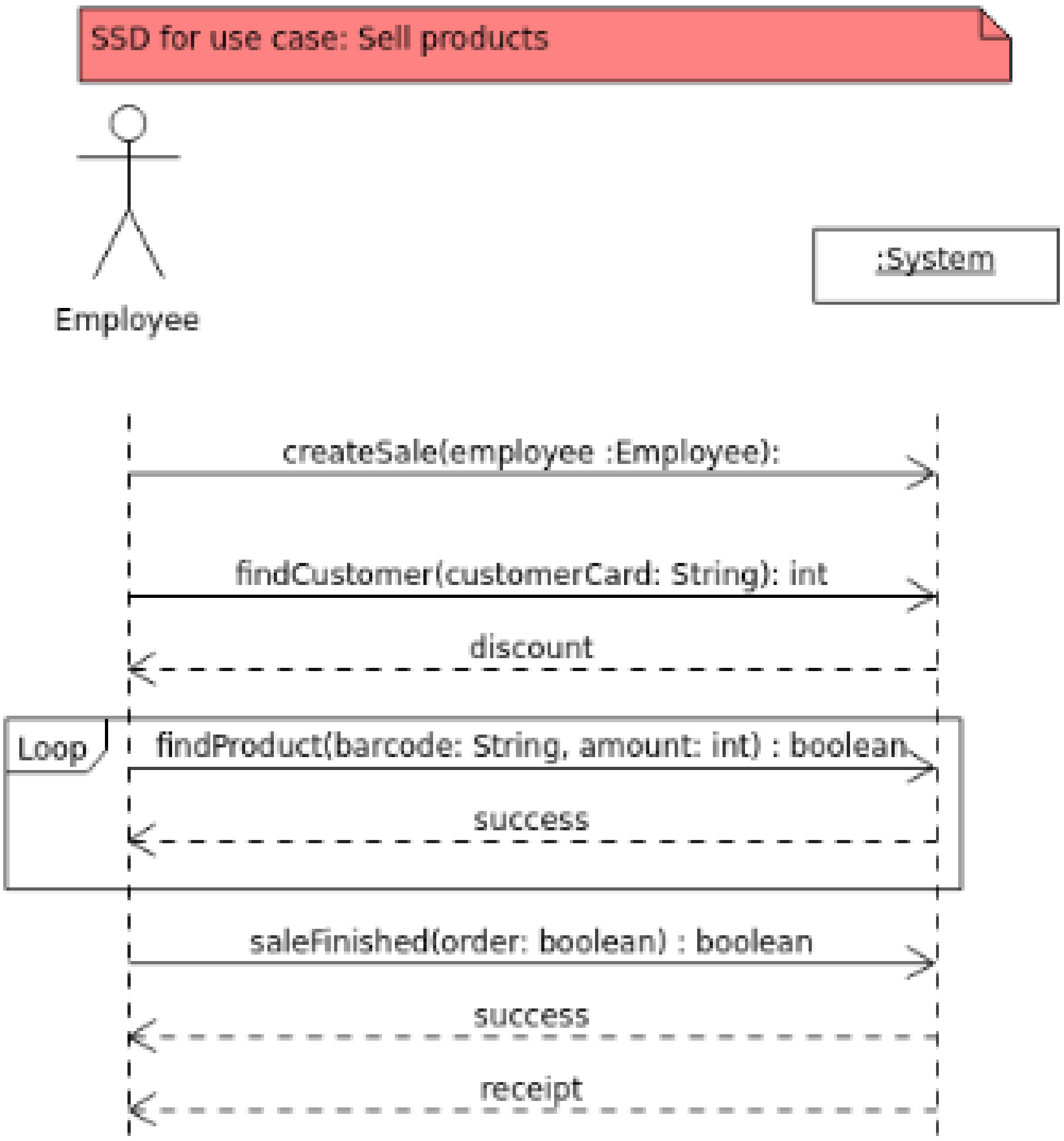
After we agreed on the table we decided to move on to make the fully dressed use case for “sell products” use case- the one with the highest priority score for us.

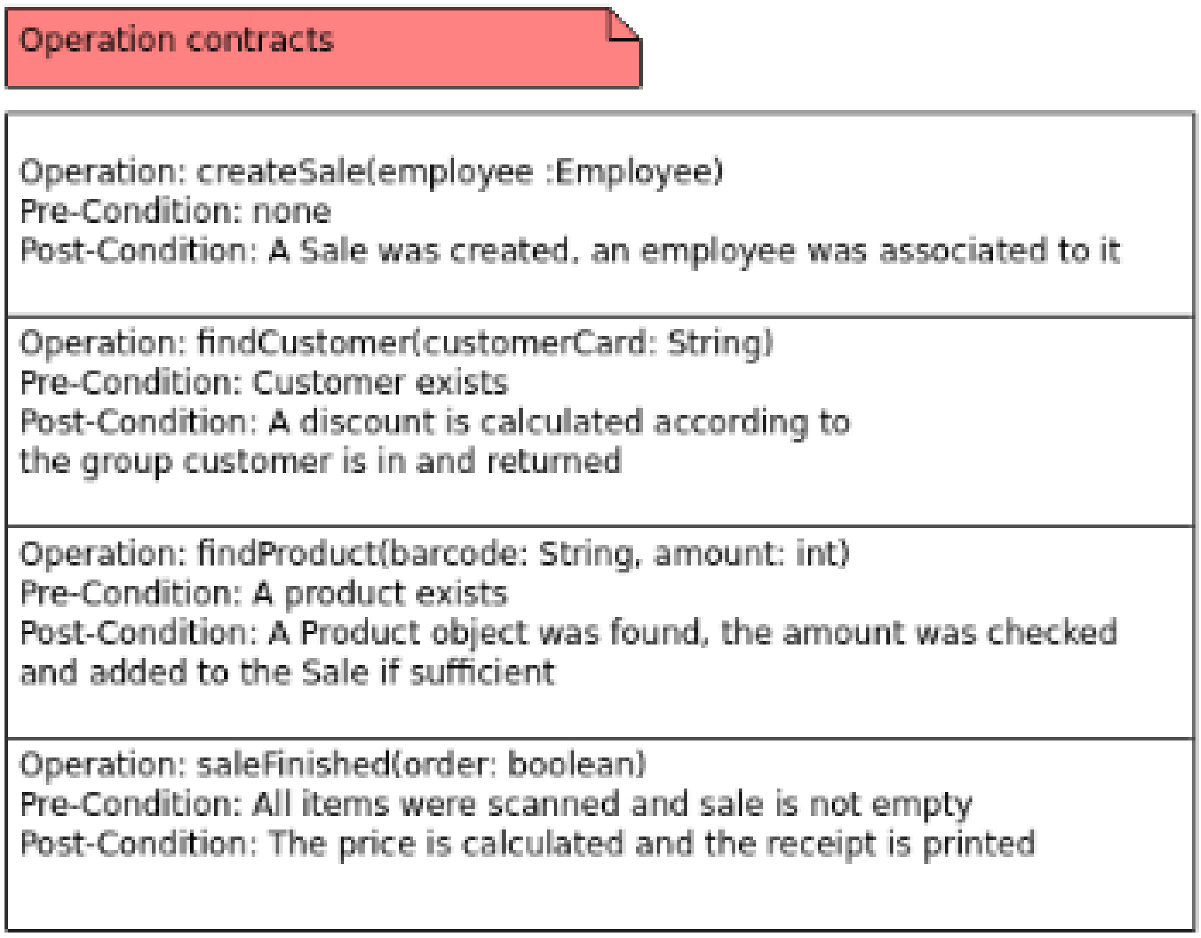


Actor	Employee	
Use case	Sell products	
Pre-condition	A product exists	
Post-condition	A product is sold	
Frequency of occurrence	Multiple times a day	
Flow of Events (Happy Days)	Actor	System
	1. Customer wants to buy a product.	
	2. Employee asks the customer for a customer card	3. System checks if there is a discount
	4. Employee scans the products barcode	5. System finds the product and adds it to the sale
	6. Employee finishes the order	7. System calculates the price
		8. System finishes the sale, prints the receipt
Alternative Flows	5.a. The product is not found	
		1. System prints an error message, and asks the employee to scan again.
	2. Return to step 4.	

System sequence diagram & operation contracts

After we were done with the fully dressed use case we decided to proceed with making an SSD as well as the operation diagram and the interaction diagram for the "sell products" use case. The SSD with the loop was nicely visible and conditions for the operation contract were too. The communication diagram was also pretty straightforward to us, since we were experienced with these from the past.


SSD for use case: Sell products
Employee
:System
createSale(employee :Employee):
findCustomer(customerCard: String): int
discount
Loop
findProduct(barcode: String, amount: int) : boolean
success
saleFinished(order: boolean) : boolean
success
receipt



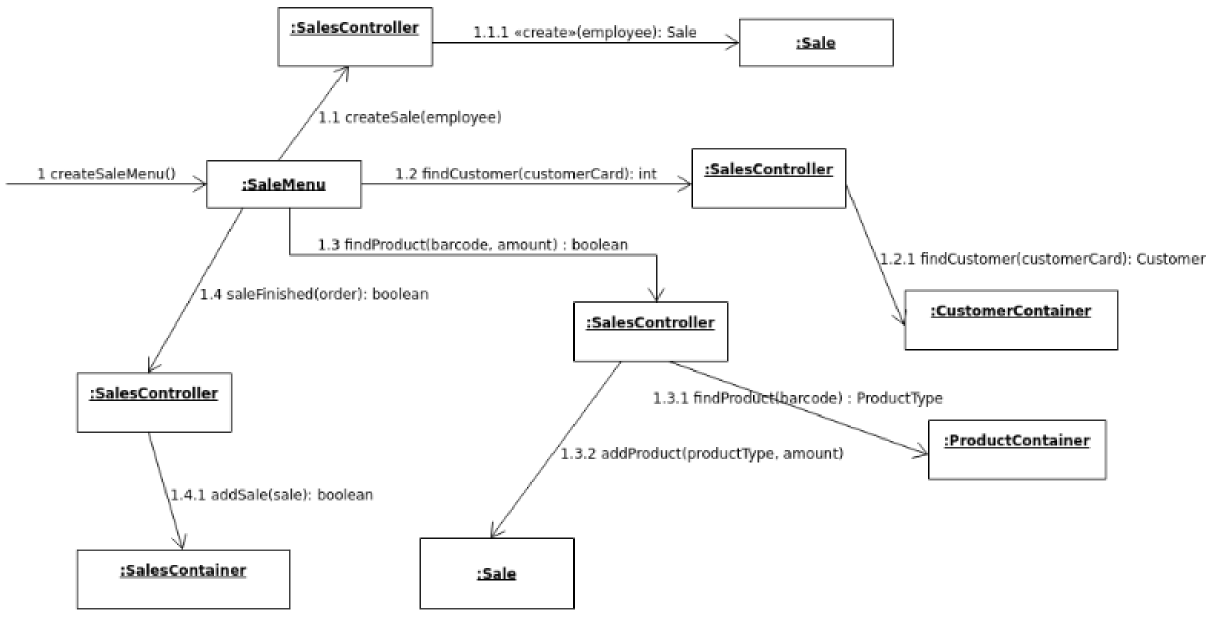
Operation contracts

Operation: createSale(employee :Employee)
Pre-Condition: none
Post-Condition: A Sale was created, an employee was associated to it

Operation: findCustomer(customerCard: String)
Pre-Condition: Customer exists
Post-Condition: A discount is calculated according to
the group customer is in and returned

Operation: findProduct(barcode: String, amount: int)
Pre-Condition: A product exists
Post-Condition: A Product object was found, the amount was checked
and added to the Sale if sufficient

Operation: saleFinished(order: boolean)
Pre-Condition: All items were scanned and sale is not empty
Post-Condition: The price is calculated and the receipt is printed


:SalesController
1.1.1 «create»(employee): Sale
:Sale
1.1 createSale(employee)
1 createSaleMenu()
:SaleMenu
1.2 findCustomer(customerCard): int
:SalesController
1.3 findProduct(barcode, amount) : boolean
1.2.1 findCustomer(customerCard): Customer
1.4 saleFinished(order): boolean
:SalesController
:CustomerContainer
:SalesController
1.3.1 findProduct(barcode) : ProductType
:ProductContainer
1.3.2 addProduct(productType, amount)
1.4.1 addSale(sale): boolean
:SalesContainer
:Sale

Fully dressed use case: Order products

Having finished the first fully dressed use case with all the diagrams, we moved on to the next important one, “Order products”. After creating the initial fully dressed use case, we came up with a maybe a bit lazy, but definitely handy and rational solution- the “Order products” would be exactly the same as the “Sell products”, with a minor difference: At the end, the Sale’s Boolean values for “ordered” and “picked” attributes would be opposite to what they were in the “Sell products” use case to differentiate between them. All the other things would be the same.

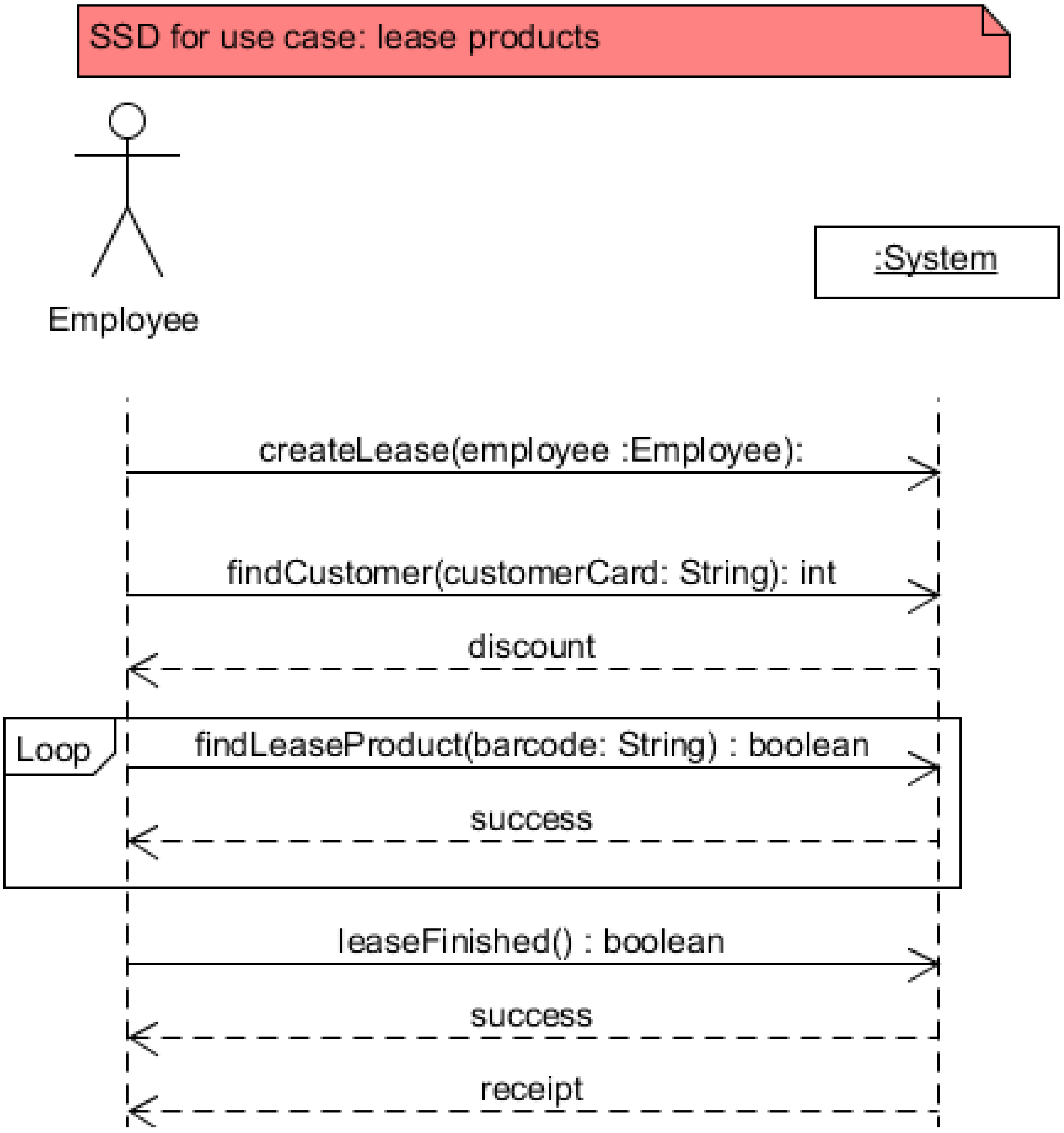
Fully dressed use case: Lease products

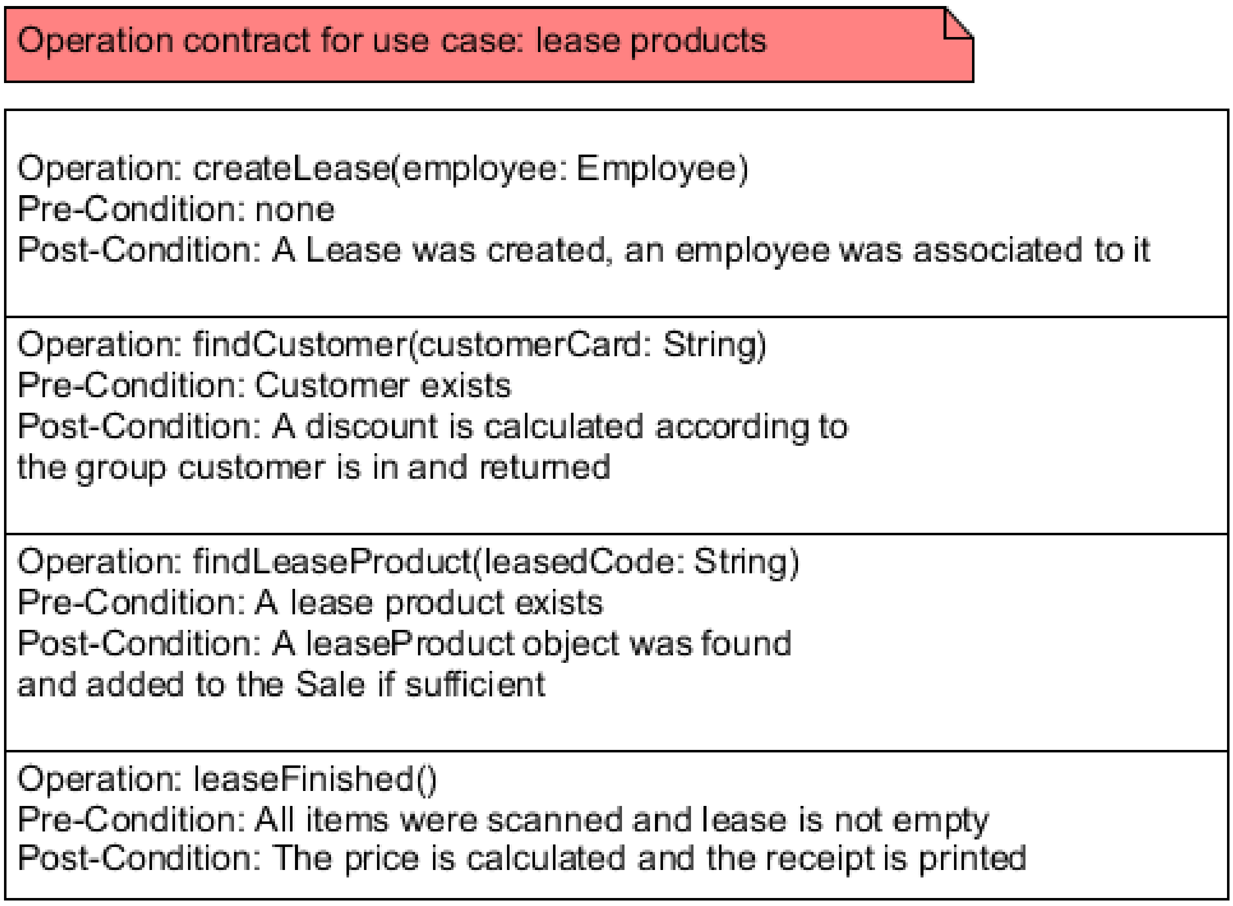
Lease products fully dressed use case, just as important as the first two, in this use case we also kept the things pretty much the same with a few minor changes. The customers still approach the employee to lease a product, employee asks for the customer card, adds discount if possible, scans the products that are being leased and finishes the order.

Actor	**Employee**	
Use case	Lease products	
Pre-condition	A product is available for a lease	
Post-condition	A product is leased	
Frequency of occurrence	Multiple times a year	
Flow of Events (Happy Days)	**Actor**	**System**
	1. Employee gets asked by the customer to lease a product	
	2. Employee asks the customer for a customer card	3. System checks if there is a discount
	4. Employee scans the products lease code	5. System finds the product and adds it to the sale
	6. Employee scans the product that is being Leased	7. System returns success
	8. Employee finishes the order	9. System calculates the price
		10. System finishes the sale, prints the receipt

Alternative Flows	5. a. The product is not found	
		1. System prints an error message, and asks the employee to scan again.
	2. Return to step 4.	

Lease products SSD


SSD for use case: lease products
Employee
:System
createLease(employee :Employee):
findCustomer(customerCard: String): int
discount
Loop
findLeaseProduct(barcode: String) : boolean
success
leaseFinished() : boolean
success
receipt



Operation contract for use case: lease products

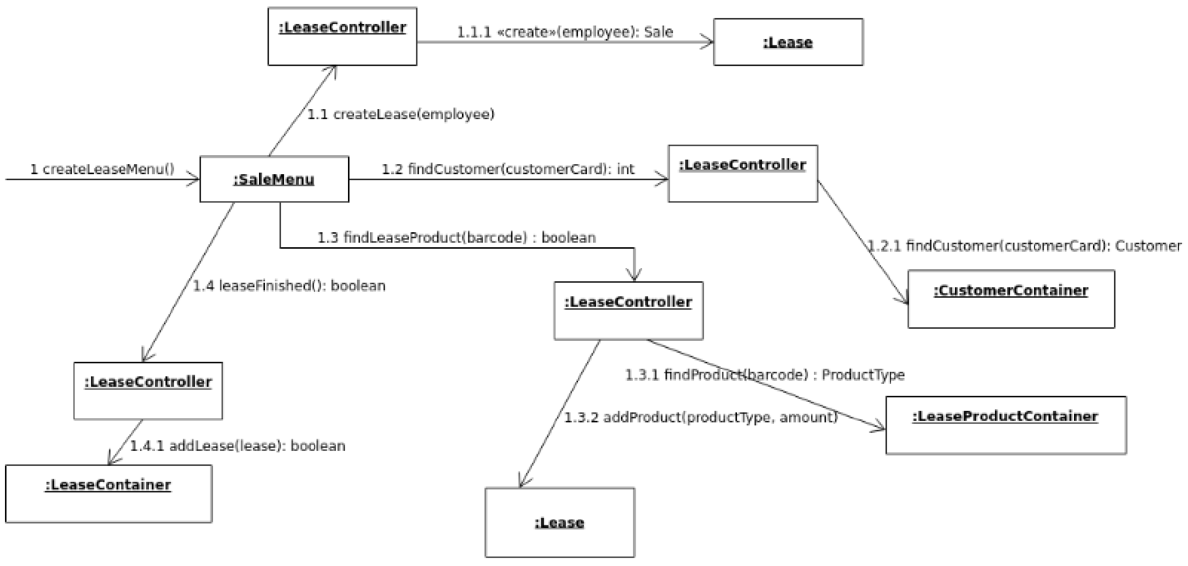
Operation: createLease(employee: Employee)
Pre-Condition: none
Post-Condition: A Lease was created, an employee was associated to it

Operation: findCustomer(customerCard: String)
Pre-Condition: Customer exists
Post-Condition: A discount is calculated according to
the group customer is in and returned

Operation: findLeaseProduct(leasedCode: String)
Pre-Condition: A lease product exists
Post-Condition: A leaseProduct object was found
and added to the Sale if sufficient

Operation: leaseFinished()
Pre-Condition: All items were scanned and lease is not empty
Post-Condition: The price is calculated and the receipt is printed

Lease operation diagram


:LeaseController
1.1.1 «create»(employee): Sale
:Lease
1.1 createLease(employee)
1 createLeaseMenu()
:SaleMenu
1.2 findCustomer(customerCard): int
:LeaseController
1.2.1 findCustomer(customerCard): Customer
1.3 findLeaseProduct(barcode) : boolean
:CustomerContainer
1.4 leaseFinished(): boolean
:LeaseController
:LeaseController
1.3.1 findProduct(barcode) : ProductType
1.3.2 addProduct(productType, amount)
:LeaseProductContainer
1.4.1 addLease(lease): boolean
:LeaseContainer
:Lease

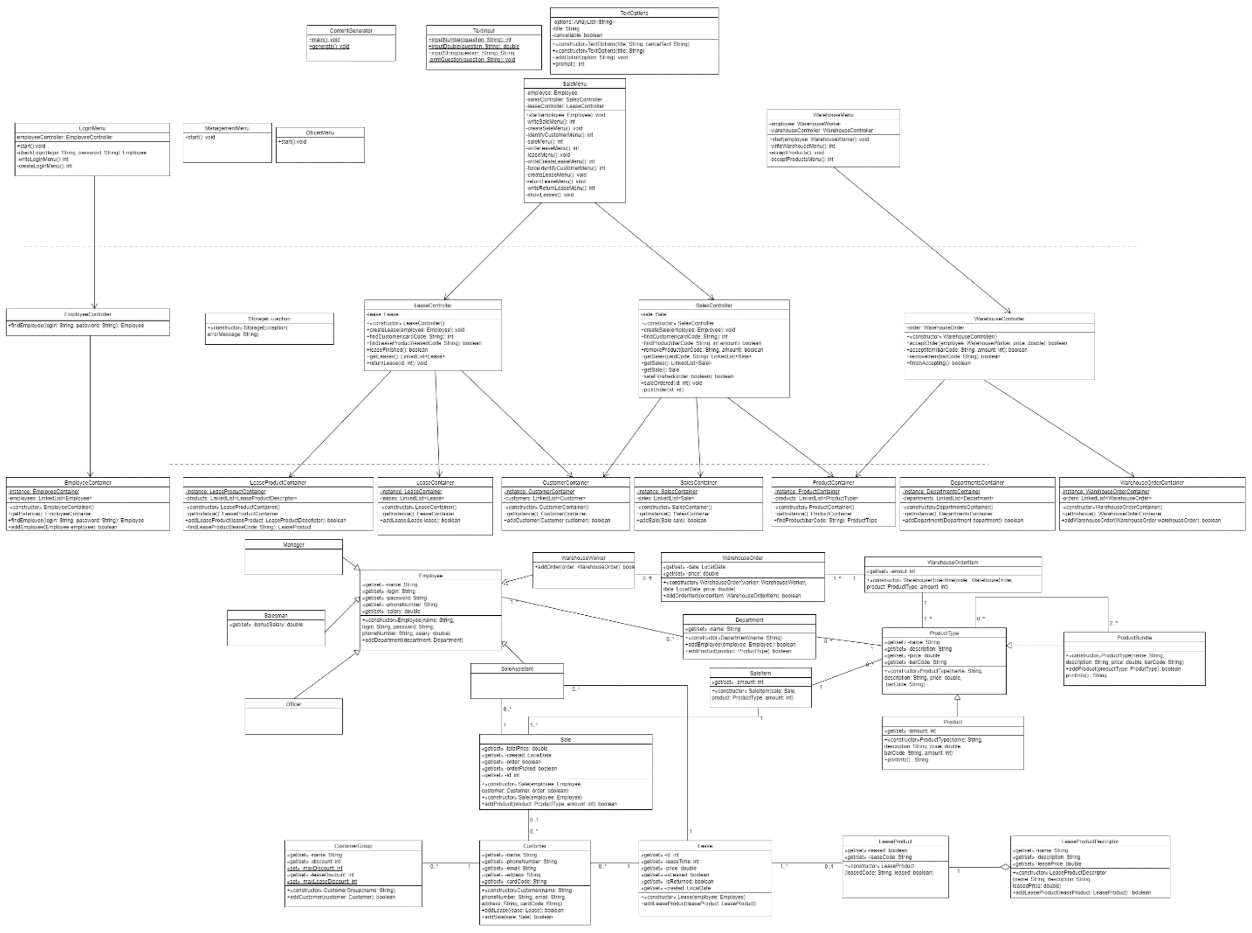
Brief use cases

Since we decided to create a fully dressed use case only for the two most complex ones, we agreed to create brief use cases for all the other use cases (except for the CRUDs) in order to get an idea of what they are about and what functionality they provide.

- **Accept products**: When new products arrive in the warehouse, the warehouse workers have to register them into the system. They scan the barcode and write all the necessary information concerning the product together with its location in the warehouse.
- **Return products**: The customer has to return the equipment he leased until the lease period expires. When everything is OK, the lease is finished and the equipment is returned to the stock.
- **Sell product bundle**: When the customer sees a pre-build kitchen he wants to buy as a whole, the system must be able to store the information about all the products that the given kitchen contains, therefore a product bundle where all the items are stored is created.

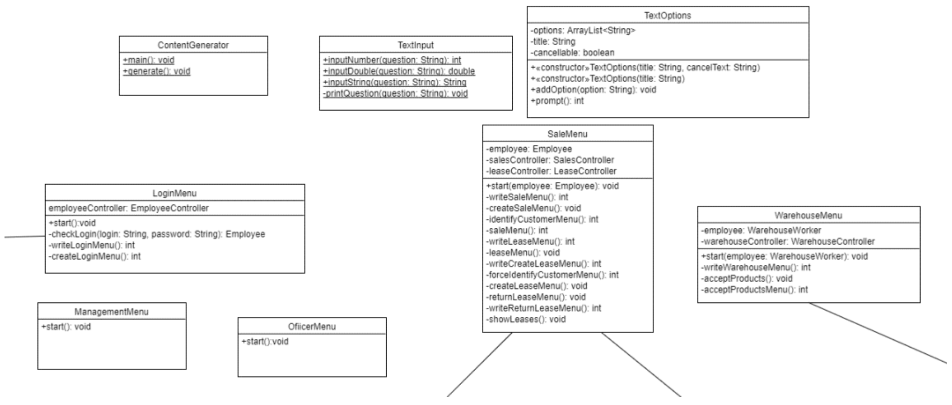
Design class diagram

(Below is the full picture of the design class diagram that we've deconstructed it into smaller parts so that we were able to explain them. In the appendix you can find a more readable version of the full picture)

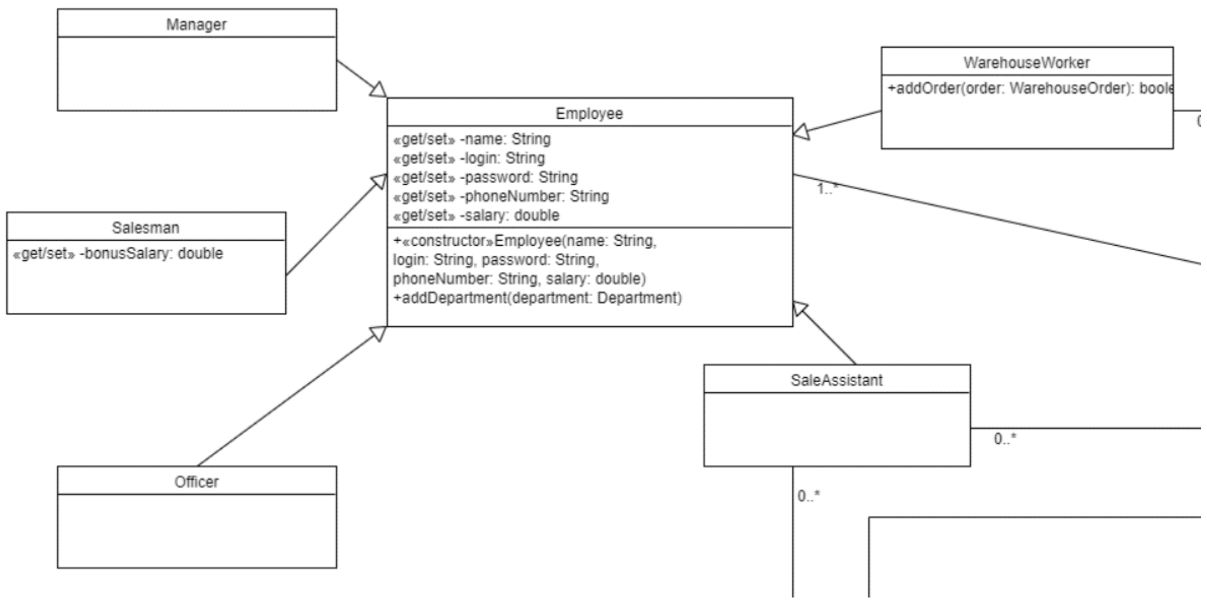


SalesController handles two use cases that are highest priority, but they are really similar, so they use the same methods and workflow. These use cases are Sell products and Order products.
The third most important use case Lease products is handled by **LeaseController**. Controllers for other use cases will be created or implemented when we start developing other use cases.

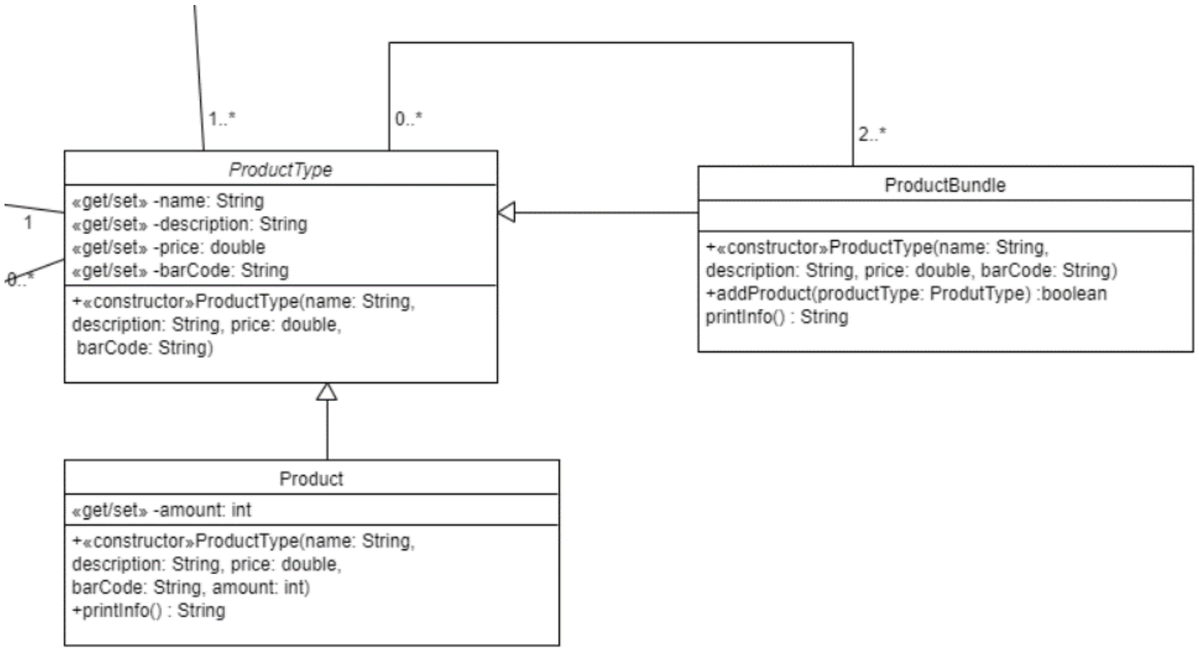
User interface


ContentGenerator
+main(): void
+generate(): void
TextInput
+inputNumber(question: String): int
+inputDouble(question: String): double
+inputString(question: String): String
-printQuestion(question: String): void
TextOptions
-options: ArrayList<String>
-title: String
-cancellable: boolean
+«constructor»TextOptions(title: String, cancelText: String)
+«constructor»TextOptions(title: String)
+addOption(option: String): void
+prompt(): int
SaleMenu
-employee: Employee
-salesController: SalesController
-leaseController: LeaseController
+start(employee: Employee): void
-writeSaleMenu(): int
-createSaleMenu(): void
-identifyCustomerMenu(): int
-saleMenu(): int
-writeLeaseMenu(): int
-leaseMenu(): void
-writeCreateLeaseMenu(): int
-forceIdentifyCustomerMenu(): int
-createLeaseMenu(): void
-returnLeaseMenu(): void
-writeReturnLeaseMenu(): int
-showLeases(): void
LoginMenu
employeeController: EmployeeController
+start():void
-checkLogin(login: String, password: String): Employee
-writeLoginMenu(): int
-createLoginMenu(): int
WarehouseMenu
-employee: WarehouseWorker
-warehouseController: WarehouseController
+start(employee: WarehouseWorker): void
-writeWarehouseMenu(): int
-acceptProducts(): void
-acceptProductsMenu(): int
ManagementMenu
+start(): void
OfficerMenu
+start():void

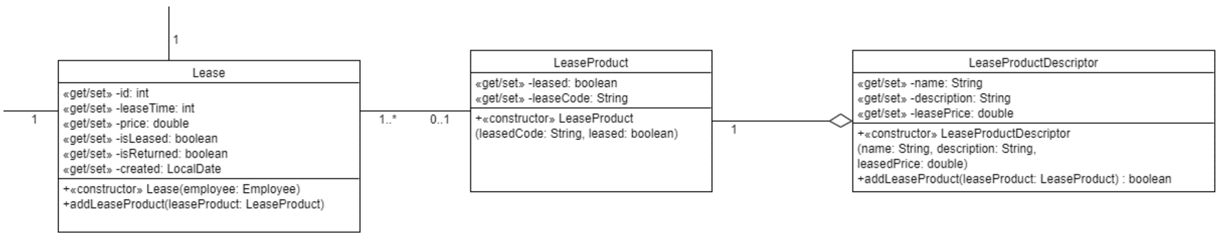
In the user interface, we are using LoginMenu as a way for our employees to access the system. This menu will then start other menus based on type of Employee logged in.
Within the SalesMenu we can create sales, create leases and we're also available to see sales and leases. We can also finish and pay orders.
In the WarehouseMenu we are able to accept products into warehouse.


Manager
WarehouseWorker
+addOrder(order: WarehouseOrder): boole
Employee
«get/set» -name: String
«get/set» -login: String
«get/set» -password: String
«get/set» -phoneNumber: String
«get/set» -salary: double
+«constructor»Employee(name: String, login: String, password: String, phoneNumber: String, salary: double)
+addDepartment(department: Department)
1..*
Salesman
«get/set» -bonusSalary: double
SaleAssistant
0..*
0..*
Officer

All the employees are based off of the employee model. This enables us to save them easily and together inside EmployeeContainer.

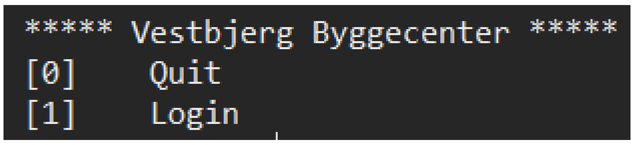

1..*
0..*
2..*
ProductType
«get/set» -name: String
«get/set» -description: String
«get/set» -price: double
«get/set» -barCode: String
+«constructor»ProductType(name: String, description: String, price: double, barCode: String)
1
0..*
ProductBundle
+«constructor»ProductType(name: String, description: String, price: double, barCode: String)
+addProduct(productType: ProdutType) :boolean
printInfo() : String
Product
«get/set» -amount: int
+«constructor»ProductType(name: String, description: String, price: double, barCode: String, amount: int)
+printInfo() : String

All of the product models are based off of the base product type model, ProductBundle contains ProductType and that gives us an opportunity to create ProductBundles containing bundles. We encountered some problems while solving this, but we managed to create getAmount and printInfo methods, that made our implementation easier.

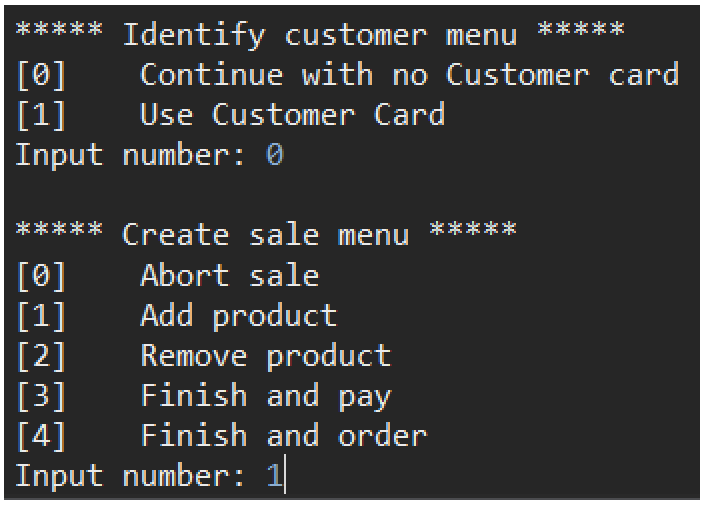

1
1
Lease
«get/set» -id: int
«get/set» -leaseTime: int
«get/set» -price: double
«get/set» -isLeased: boolean
«get/set» -isReturned: boolean
«get/set» -created: LocalDate
+«constructor» Lease(employee: Employee)
+addLeaseProduct(leaseProduct: LeaseProduct)
1..*
0..1
LeaseProduct
«get/set» -leased: boolean
«get/set» -leaseCode: String
+«constructor» LeaseProduct (leasedCode: String, leased: boolean)
1
LeaseProductDescriptor
«get/set» -name: String
«get/set» -description: String
«get/set» -leasePrice: double
+«constructor» LeaseProductDescriptor (name: String, description: String, leasedPrice: double)
+addLeaseProduct(leaseProduct: LeaseProduct) : boolean

Leasing is also a feature that the company has. We decided to solve this separately from regular products and used Item descriptor pattern.

A short user manual

1. Sell items to the customer/ create an order for the customer
   a. Press 1, Login with your name and password
   b. 

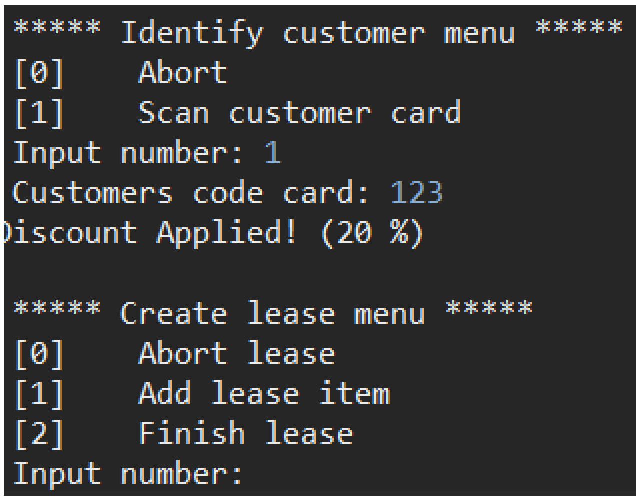
```
***** Vestbjerg Byggecenter *****
[0]    Quit
[1]    Login
```

   c. Press 1 to “create sale”, then identify the customer or continue without card, the discount is automatically applied.
   d. Press 1 to “add products” to the sale, scan the barcode and enter the amount. If you press enter, the amount is automatically 1.
   e. 

```
***** Identify customer menu *****
[0]    Continue with no Customer card
[1]    Use Customer Card
Input number: 0

***** Create sale menu *****
[0]    Abort sale
[1]    Add product
[2]    Remove product
[3]    Finish and pay
[4]    Finish and order
Input number: 1
```

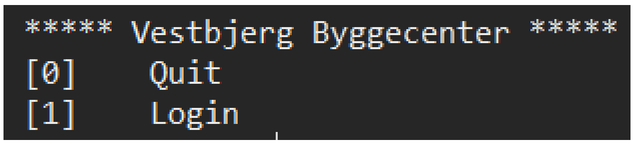
   f. Repeat the process until the order is complete.
   g. Press 3 or 4 depending on whether the customer wants to pay and pick the products or only order them.
   h. Receipt is printed automatically.
2. Create a lease
   a. do steps a,b above to login
   b. press 2 to “Create Lease”
   c. Since you can lease only to registered customer, scan customer card next (press 1)

d. 

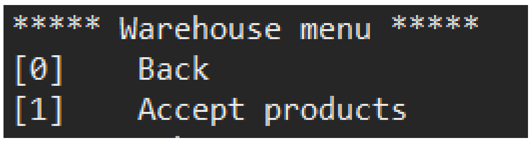
```
***** Identify customer menu *****
[0]    Abort
[1]    Scan customer card
Input number: 1
Customers code card: 123
)iscount Applied! (20 %)

***** Create lease menu *****
[0]    Abort lease
[1]    Add lease item
[2]    Finish lease
Input number:
```

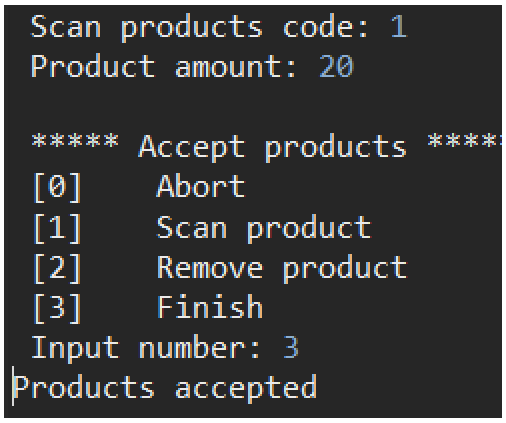
e. Add lease items according to their barcode until all items are added.
f. when all items to lease are added, press "finish Lease" (number 2)

3. Accept new products that arrived to warehouse:
   a. Press 1, Login with your name and password
   b. 

```
 ***** Vestbjerg Byggecenter *****
 [0]    Quit
 [1]    Login
```

   c. Press 1 for accept products, fill in the information about the product
   d. 

```
***** Warehouse menu *****
[0]    Back
[1]    Accept products
```

   e. press 1 to "scan products" if you want to scan more products, press "finish" (3) if the scanning is finished
   f. 

```
 Scan products code: 1
 Product amount: 20

 ***** Accept products ****
 [0]    Abort
 [1]    Scan product
 [2]    Remove product
 [3]    Finish
 Input number: 3
Products accepted
```

   g. Finish, wait for "products accepted" message, then press 0 to log out
4. Show sales/ leases
   a. Login to do system as shown above
   b. press 3 or 4 to see the sales / leases

Code Standards

The code conventions are crucial when a team is developing the same project. In order to maintain some programming practices we checked the original Java code conventions[2] document and based on it we wrote these basic rules:

1. **Variables, methods**
   a. Class variables should be private; usage of getters and setters is important in order to access and modify them easily.
   b. Variables and method names should be written in camelCase with a short but meaningful name.
   c. Each method should be responsible for one task only.
2. **Classes**
   a. Class name should start with a capital letter, the name should be short but meaningful. Class is to have one purpose and should fulfill that one purpose only. Otherwise the class should be split into more classes.
3. **Semantics**
   a. Names of the classes should start with a Capital letter
   b. Variable and method names should follow the camelCase format as written above
4. **Version control**
   a. Everything should be regularly uploaded to a SVN server UCN provides. All the participants should commit their changes using a descriptive commit message and keep their version of the project updated. The project itself, its documentation as well as the diagrams created could be present on the server.
5. **Documentation**
   a. The code should be documented inside the .java code files according to the Javadoc conventions learned during the classes.
   b. The documentation should be a written overview of what the given class/method does and how to work with that.
   c. We used a documentation tool called Doxygen to generate an HTML based documentation for the project. This was really helpful during the implementation. This documentation can be seen at the following URL: https://henkyfen.github.io/dmai0919-group3-1semproject/

References

Literature

[1] C.Larman, "Applying UML and Patterns - An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition)" p. 148, 2004.
[2] Java Code Conventions - December 8, 2019
http://www.oracle.com/technetwork/java/codeconventions-150003.pdf
[3] Doxygen Manual - December 12, 2019
http://www.doxygen.nl/manual/index.html

Technology Used

- Bluej IDE (https://www.bluej.org/)
- Eclipse IDE (https://eclipse.org)
- JetBrains IntelliJ IDE (https://jetbrains.com/)
- SmartSVN Subversion Client (https://www.smartsvn.com/)
- Subversion CLI Client (https://subversion.apache.org/)
- Doxygen documentation generator (http://doxygen.nl/)
- GanttProject project manager (http://ganttproject.biz/)
- KanbanFlow kanban board (http://kanbanflow.com/)

Groupwork

Group Contract

Everyone signed below agrees to the terms that were discussed and agreed upon on the very first day of the project:

1. Everyone will participate in finishing the project
2. Everyone will be present in the group meetings when there are any
3. When someone cannot make it to the meeting, he will inform the group in advance and finish his tasks remotely
4. When someone cannot finish his part on time, he will immediately inform the group and try to resolve the problem
5. When someone disagrees about something, he will start a discussion about the problem and try to suggest a possible solutions
6. When there is a dispute, a vote is held- since there are 5 people in the group, the option with 3 or more votes wins
7. When a member is not participating, he may be sacked from the group

Aalborg, 25.11.2019

....	....	....	....	....
Dino	Erik	Krisztián	Marek	Pavel

Evaluation of group work

At first, we found out about our insights. We discovered that most of us were located in the green part of the circle so the initial estimate was that there would not be a real leader of the group, but the workflow would be steady without any major disputes and arguments, since all of us seem to be easy-going. This turned out to be partially true as the group's workflow was calm and steady, most of the meetings were attended by everyone and we also tried our best to finish the project properly and on time. Almost everyone fell sick during the time we were working on the project so we had to overcome these troubles too, so people had to work remotely from home. Overall, the group work was enjoyable and everyone has learned a lot from the project among all of the different parts of it, since we worked together and everyone helped with his bit to the solution.

Work planning and management

Before we started working, we created a Gantt chart using a free tool called GanttProject. With this tool, we created a timeline, to which we tried to stick. During the design, analysis and implementation, we made a kanban board with the online tool called KanbanFlow. With the help of these two tools, our group work was very effective as everyone knows what they had to do, how long did the task took to finish and what tasks are still needed to be done.

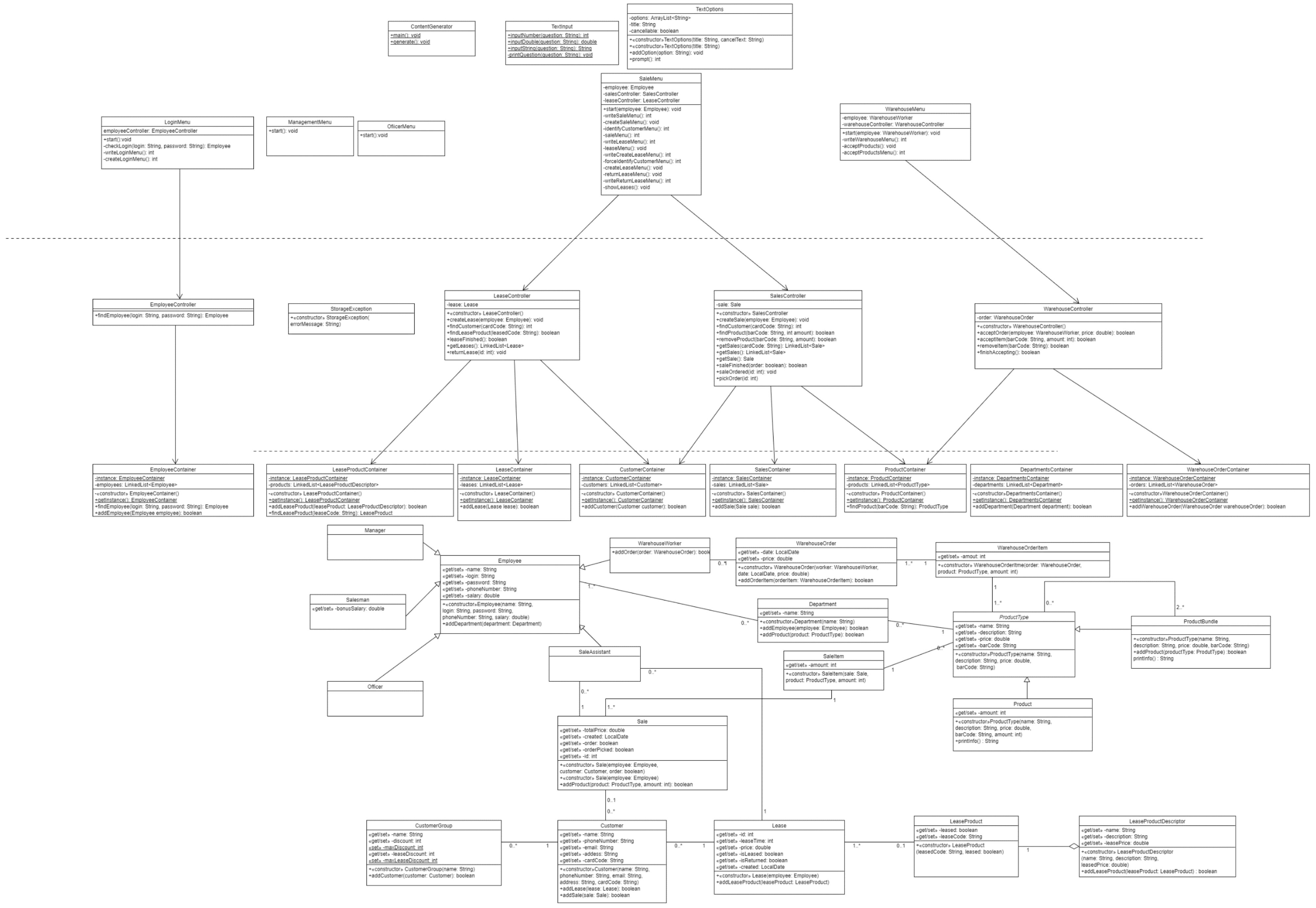
Conclusion

Documented above, there is our suggested solution to the Vestbjerg Byggecenter's case. Divided into business analysis and the implementation of the system itself, the report provides an overview of the current state the company functions in and the improvements we would recommend the company to do in order to simplify the processes inside. Also, we strived for as user friendly system as possible, so that the warehouse workers wouldn't experience any troubles when working with the new system. We also believe that the different locations for the items among the warehouses and departments wouldn't be a problem anymore.

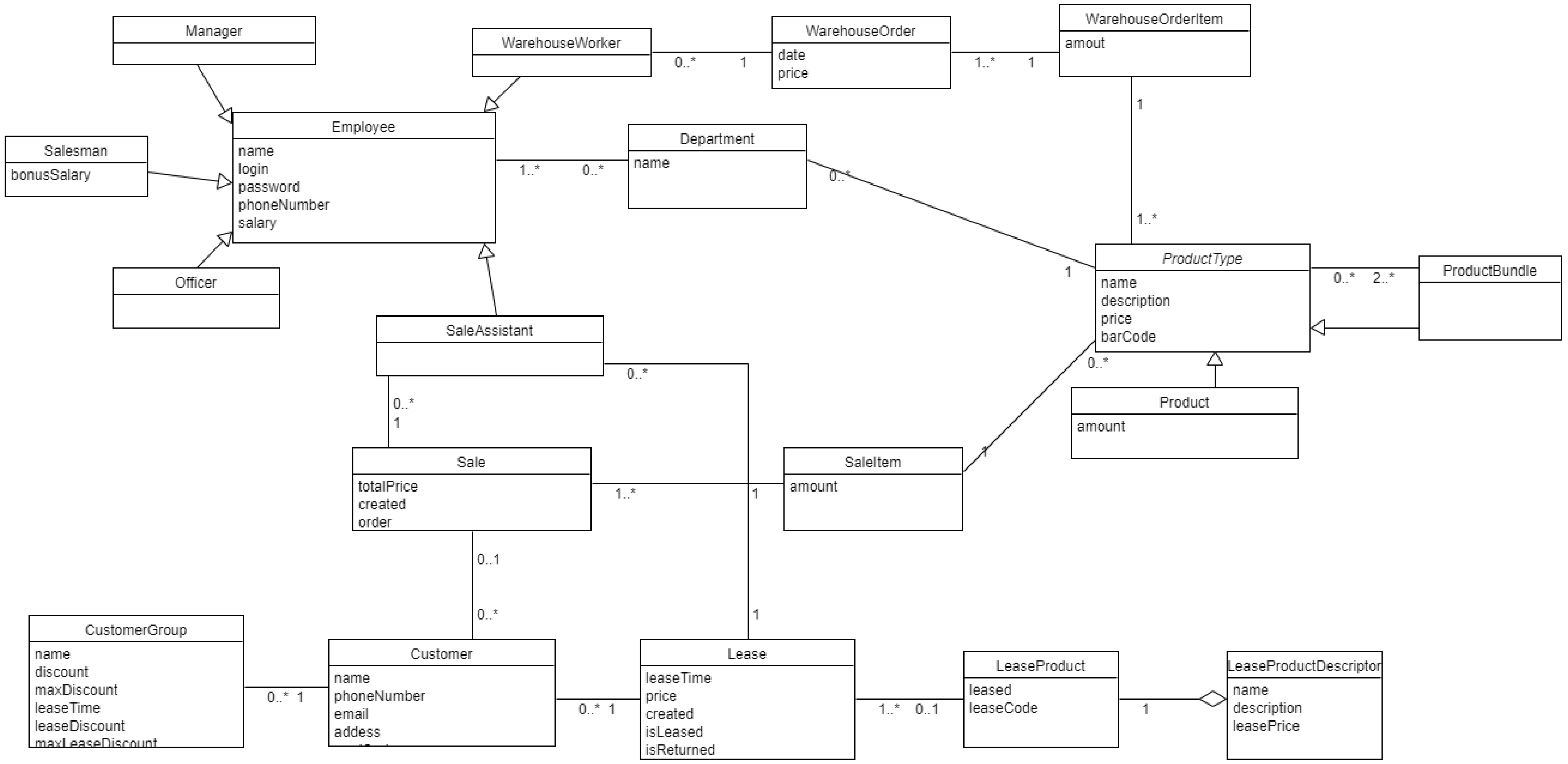
Appendices

[1] Design Class Diagram
[2] Domain model

Appendix 1: Design Class Diagram


ContentGenerator
+main(): void
+generate(): void
TextInput
+inputNumber(question: String): int
+inputDouble(question: String): double
+inputString(question: String): String
-printQuestion(question: String): void
TextOptions
-options: ArrayList<String>
-title: String
-cancellable: boolean
+«constructor»TextOptions(title: String, cancelText: String)
+«constructor»TextOptions(title: String)
+addOption(option: String): void
+prompt(): int
SaleMenu
-employee: Employee
-salesController: SalesController
-leaseController: LeaseController
+start(employee: Employee): void
-writeSaleMenu(): int
-createSaleMenu(): void
-identifyCustomerMenu(): int
-saleMenu(): int
-writeLeaseMenu(): int
-leaseMenu(): void
-writeCreateLeaseMenu(): int
-forceIdentifyCustomerMenu(): int
-createLeaseMenu(): void
-returnLeaseMenu(): void
-writeReturnLeaseMenu(): int
-showLeases(): void
LoginMenu
employeeController: EmployeeController
+start():void
-checkLogin(login: String, password: String): Employee
-writeLoginMenu(): int
-createLoginMenu(): int
ManagementMenu
+start(): void
OfiicerMenu
+start():void
WarehouseMenu
-employee: WarehouseWorker
-warehouseController: WarehouseController
+start(employee: WarehouseWorker): void
-writeWarehouseMenu(): int
-acceptProducts(): void
-acceptProductsMenu(): int
EmployeeController
+findEmployee(login: String, password: String): Employee
StorageException
+«constructor» StorageException(
errorMessage: String)
LeaseController
-lease: Lease
+«constructor» LeaseController()
+createLease(employee: Employee): void
+findCustomer(cardCode: String): int
+findLeaseProduct(leasedCode: String): boolean
+leaseFinished(): boolean
+getLeases(): LinkedList<Lease>
+returnLease(id: int): void
SalesController
-sale: Sale
+«constructor» SalesController
+createSale(employee: Employee): void
+findCustomer(cardCode: String): int
+findProduct(barCode: String, int amount): boolean
+removeProduct(barCode: String, amount): boolean
+getSales(cardCode: String): LinkedList<Sale>
+getSales(): LinkedList<Sale>
+getSale(): Sale
+saleFinished(order: boolean): boolean
+saleOrdered(id: int): void
+pickOrder(id: int)
WarehouseController
-order: WarehouseOrder
+«constructor» WarehouseController()
+acceptOrder(employee: WarehouseWorker, price: double): boolean
+acceptItem(barCode: String, amount: int): boolean
+removeItem(barCode: String): boolean
+finishAccepting(): boolean
EmployeeContainer
-instance: EmployeeContainer
-employees: LinkedList<Employee>
-«constructor» EmployeeContainer()
+getInstance(): EmployeeContainer
+findEmployee(login: String, password: String): Employee
+addEmployee(Employee employee): boolean
LeaseProductContainer
-instance: LeaseProductContainer
-products: LinkedList<LeaseProductDescriptor>
-«constructor» LeaseProductContainer()
+getInstance(): LeaseProductContainer
+addLeaseProduct(leaseProduct: LeaseProductDescriptor): boolean
+findLeaseProduct(leaseCode: String): LeaseProduct
LeaseContainer
-instance: LeaseContainer
-leases: LinkedList<Lease>
-«constructor» LeaseContainer()
+getInstance(): LeaseContainer
+addLease(Lease lease): boolean
CustomerContainer
-instance: CustomerContainer
-customers: LinkedList<Customer>
-«constructor» CustomerContainer()
+getInstance(): CustomerContainer
+addCustomer(Customer customer): boolean
SalesContainer
-instance: SalesContainer
-sales: LinkedList<Sale>
-«constructor» SalesContainer()
+getInstance(): SalesContainer
+addSale(Sale sale): boolean
ProductContainer
-instance: ProductContainer
-products: LinkedList<ProductType>
-«constructor» ProductContainer()
+getInstance(): ProductContainer
+findProduct(barCode: String): ProductType
DepartmentsContainer
-instance: DepartmentsContainer
-departments: LinkedList<Department>
-«constructor»DepartmentsContainer()
+getInstance(): DepartmentsContainer
+addDepartment(Department department): boolean
WarehouseOrderContainer
-instance: WarehouseOrderContainer
-orders: LinkedList<WarehouseOrder>
-«constructor»WarehouseOrderContainer()
+getInstance(): WarehouseOrderContainer
+addWarehouseOrder(WarehouseOrder warehouseOrder): boolean
Manager
Salesman
«get/set» -bonusSalary: double
Officer
Employee
«get/set» -name: String
«get/set» -login: String
«get/set» -password: String
«get/set» -phoneNumber: String
«get/set» -salary: double
+«constructor»Employee(name: String,
login: String, password: String,
phoneNumber: String, salary: double)
+addDepartment(department: Department)
WarehouseWorker
+addOrder(order: WarehouseOrder): boole
0..*1
WarehouseOrder
«get/set» -date: LocalDate
«get/set» -price: double
+«constructor» WarehouseOrder(worker: WarehouseWorker,
date: LocalDate, price: double)
+addOrderItem(orderItem: WarehouseOrderItem): boolean
1..*
1
WarehouseOrderItem
«get/set» -amout: int
+«constructor» WarehouseOrderItme(order: WarehouseOrder,
product: ProductType, amount: int)
1
1..*
0..*
2..*
1..*
0..*
Department
«get/set» -name: String
+«constructor»Department(name: String)
+addEmployee(employee: Employee): boolean
+addProduct(product: ProductType): boolean
0..*
1
0..*
ProductType
«get/set» -name: String
«get/set» -description: String
«get/set» -price: double
«get/set» -barCode: String
+«constructor»ProductType(name: String,
description: String, price: double,
barCode: String)
ProductBundle
+«constructor»ProductType(name: String,
description: String, price: double, barCode: String)
+addProduct(productType: ProdutType) :boolean
printInfo() : String
SaleAssistant
0..*
SaleItem
«get/set» -amount: int
+«constructor» SaleItem(sale: Sale,
product: ProductType, amount: int)
1
1
0..*
1
1..*
Product
«get/set» -amount: int
+«constructor»ProductType(name: String,
description: String, price: double,
barCode: String, amount: int)
+printInfo() : String
Sale
«get/set» -totalPrice: double
«get/set» -created: LocalDate
«get/set» -order: boolean
«get/set» -orderPicked: boolean
«get/set» -id: int
+«constructor» Sale(employee: Employee,
customer: Customer, order: boolean)
+«constructor» Sale(employee: Employee)
+addProduct(product: ProductType, amount: int): boolean
0..1
0..*
1
CustomerGroup
«get/set» -name: String
«get/set» -discount: int
«set» -maxDiscount: int
«get/set» -leaseDiscount: int
«set» -maxLeaseDiscount: int
+«constructor» CustomerGroup(name: String)
+addCustomer(customer: Customer): boolean
0..*
1
Customer
«get/set» -name: String
«get/set» -phoneNumber: String
«get/set» -email: String
«get/set» -addess: String
«get/set» -cardCode: String
+«constructor»Customer(name: String,
phoneNumber: String, email: String,
address: String, cardCode: String)
+addLease(lease: Lease): boolean
+addSale(sale: Sale): boolean
0..*
1
Lease
«get/set» -id: int
«get/set» -leaseTime: int
«get/set» -price: double
«get/set» -isLeased: boolean
«get/set» -isReturned: boolean
«get/set» -created: LocalDate
+«constructor» Lease(employee: Employee)
+addLeaseProduct(leaseProduct: LeaseProduct)
1..*
0..1
LeaseProduct
«get/set» -leased: boolean
«get/set» -leaseCode: String
+«constructor» LeaseProduct
(leasedCode: String, leased: boolean)
1
LeaseProductDescriptor
«get/set» -name: String
«get/set» -description: String
«get/set» -leasePrice: double
+«constructor» LeaseProductDescriptor
(name: String, description: String,
leasedPrice: double)
+addLeaseProduct(leaseProduct: LeaseProduct) : boolean

Appendix 2: Domain Model


Manager
WarehouseWorker
WarehouseOrder
date
price
WarehouseOrderItem
amout
Salesman
bonusSalary
Employee
name
login
password
phoneNumber
salary
Department
name
Officer
SaleAssistant
ProductType
name
description
price
barCode
ProductBundle
Product
amount
Sale
totalPrice
created
order
SaleItem
amount
CustomerGroup
name
discount
maxDiscount
leaseTime
leaseDiscount
Customer
name
phoneNumber
email
addess
Lease
leaseTime
price
created
isLeased
isReturned
LeaseProduct
leased
leaseCode
LeaseProductDescriptor
name
description
leasePrice
0..*
1
1..*
1
1
1..*
1..*
0..*
0..*
1
0..*
2..*
0..*
0..*
0..*
1
1
1..*
1
0..1
0..*
1
0..* 1
0..* 1
1..* 0..1
1