University College Nordjylland
Computer Science
dmaj0915

Workshop Design

15.12.2015

Group 5

Alef Pir, Aurelijus Steigvila,
Joosep Sisas, Marius Untaru,
Nils-Asbjørn Frederiksen

University College Nordjylland

Workshop:
Vestbjerg Byggecenter A/S

Workshop period:
25.11.2015 – 15.12.2015

Workshop group:
Group 5

Members:
Alef Pir

Aurelijus Steigvila

Joosep Sisas

Marius Untaru

Nils-Asbjørn Frederiksen

Teachers:
Kis Boisen Hansen

Mukhtiar Memon

Henrik Munk Hvarregaard

No. of pages: 28

Finished:
15.12.2015

Synopsis:

The main focus of this project is to design and implement an application for the company Vestbjerg Byggecenter A/S.

The entire project includes designing a new application which will allow the user to register products, customers, new employees, sell and lease item, etc. To do so, the project is divided into a few parts which will be done in the span of two weeks. Those are, planning, design, implementation and report writing.

The designing of this project is done using " Umlet" and then implemented in "Bluej". The schedule planning for the project is done using "Microsoft Project 2015".

Table of contents

Contents

Chapter 1. Introduction

The aim of this project is to study, analyze and develop a new IT-system for a company called Vestbjerg Byggecenter A/S. To do so, there are certain phases that have to be looked at including business, design and code implementation. Vestbjerg Byggecenter A/S is a family owned company with many years of experience and expertise. It is owned by Anders Olesen who acts as the CEO and his two sons who are acting managers of the main departments in the company: "Do it yourself" and timbers departments.

Problem statement

Based on the introduction and the assignment given the following problem statements was formulated:

"Analyze the business model and create a new IT-system based on the findings while getting insight into problem based learning and proper team work techniques, especially with focus on work culture."

Chapter 2. Business

2.1 Organizational structure

Vestbjerg Byggecenter A/S seems to follow a functional organizational structure. This type of structure usually involves semi-big organizations which divide the activities into several, different departments, where each one of those departments specializes in particular area. Those can be Human resources, Sales and marketing, finance and administration departments, etc. One of the positives in this type of structure is having a strict chain of command, thus the employees are well informed of what kind of decisions they are allowed to take. The chain of command in Vestbjerg Byggecenter can be seen at (FIG 1) , there is the head of the company Anders Olesen , who carries the role of the managing director, and his two sons who act as chief executives of the two departments, Timber and DIY, respectively. Those three are the leaders of the company and they take all the main and important decisions.

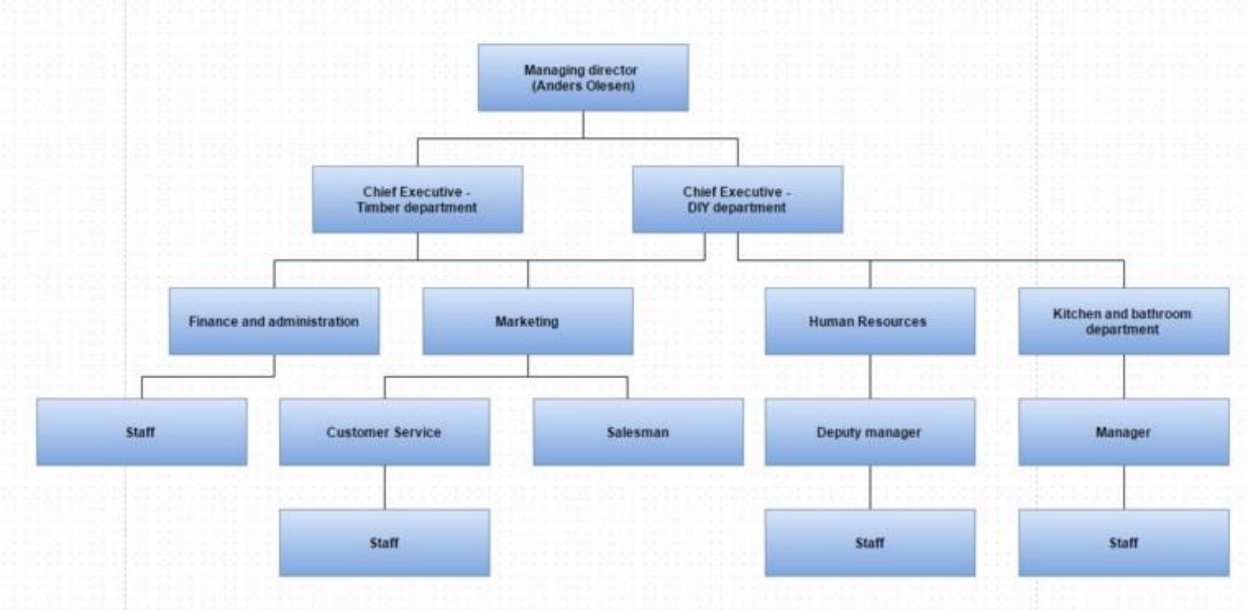


Fig. 1: Organizational structure of Vestbjerg Byggecenter A/S

2.2 SWOT Analysis

Fig. 2: SWOT table of the company

Strengths

Business chain – The company takes part in the XL-byg chain.

Organization structure – It has well established organizational structure where the main decisions are taken by the highest authorities.

Positive working environment - Working environment is really positive there, because hiring process is very thorough and firing is rare thing in this company. This company values its employees.

Weaknesses

Outdated technology – The company uses outdated IT-system , that's one of the main objectives to cover in this project.

Different leadership – There is three different types of leadership due to having three main leaders with different backgrounds. This is considered as a weakness because it can create insecurity and distrust amongst the employees.

Threats

Hard to break out of your comfort zone- New Technology might be hard to be accepted by workers, because it's new and most of the workers have already worked there for a long time and got used to the system that is already there.

Opportunities

To expand their business – Because the company is part of the XL-byg chain there is higher opportunity to expand.

Update technology – New and updated system that has centralized tasks that the staff are using during the day and can fasten work thus decreasing sale time and time used on other tasks.

2.3 Strategic goals

The main goal ahead of the company is to improve the outdated IT system. The initial vision for a new system is that it is user friendly, practical and easily accessible, thus employees can use their personal computers from home. In addition, the new system would look into managing tasks by wider numbers of the personal as well as updating some of the older features of the system such as registering customer, product, etc.

- Staff members gaining access to the new system
- Receiving statistics of the customers, products, etc.
- Improving the registration system
- Adding more functions to the system, such as:

- Make an offer
- Place an order
- Confirm an order
- Register customer
- Register employee
- Register product
- Register leasable product

Chapter 3. Design

3.1 System definition

The following section of the report explains the whole process of designing a system, from getting the original idea, to deciding on the conditions and requirements and finally to planning the implementation of a system that fulfills those requirements.

The purpose of this system that we were aiming for is to make many different functions that could be used while handling the customer, the product, etc. The system has to take a wide variety of functions and it has to be able to hold a lot of information about many entities. It holds information about customer (name, address, phone number, ID, CVR/CPR), product (unit price, weight, size, description, inventory amount) and so on. The user can add, view and remove entities and the user also can find and update information.

The functionality of the system is based on the registration of the customers, products and contractors(company customers) and also sales statistics over the customers, products and co-workers. As an application domain we have support for handling orders and payments.
It is a user-friendly system that is accessible for all co-workers. The technology on which the system will run is four terminals that are used for registration, and five computers are particularly used for word processing. The main objects are customer, salesman, sale, lease, product. System's overall responsibility is represented as an administrative tool for order handling and surveillance of the business process.

3.2 Event tables

Before actually making the SSD's it was important to make an event table and to describe some use cases afterwards. In this table below the steps of the use cases and the main actor were highlighted.

Event	Actor	Use Case	Steps in use case
Client picks up a product at the warehouse	Salesman	Create sale on site	Salesman creates sale Salesman creates saleItem Salesman adds sale items to sale
Order new inventory	InventoryManager	Add new products to the inventory	InventoryManager creates a new product.
Hire new employee	Manager	Create new employee	Manager adds new employee with all the attributes asked for.
Promote employee	Manager	Promote employee to Salesman	Manager picks employee to promote Manager writes name of new title. (Ex. “Salesman”) Employee is promoted to salesman Manager removes old employee object
Create customer	Salesman	Creates customer	Salesman creates a customer. Salesman picks personal or company costumer

Fig. 3: Event table

3.4 Use cases

One of the first steps that were taken in order to get a better idea of how the new system will work was identifying the main use cases and analyzing them in detail. The identified use cases that are shown in the following figures illustrate the fact that the user of the program can complete a large amount of actions.

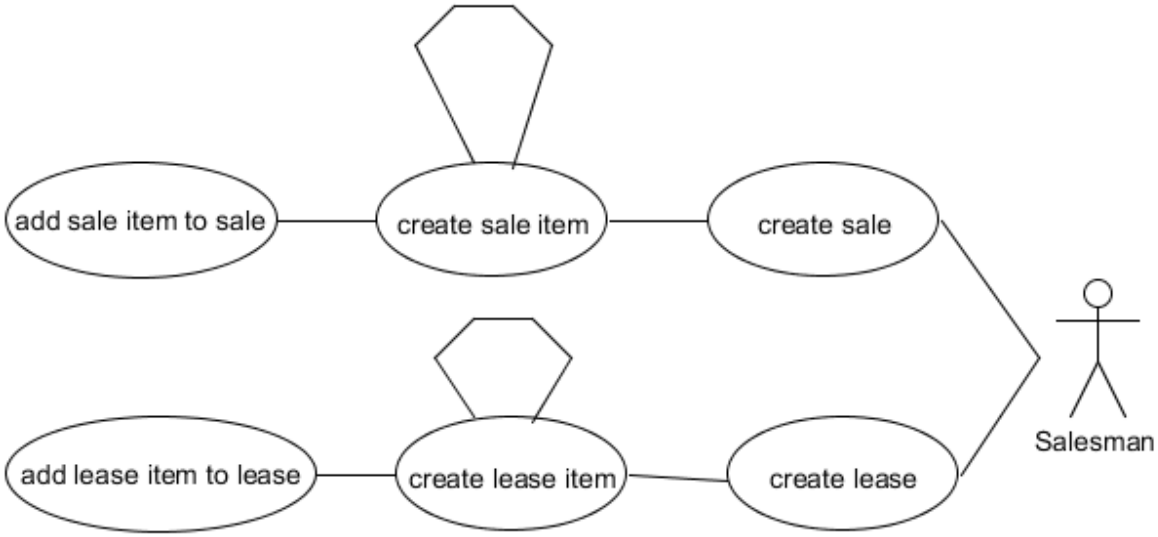


Fig 4: Use case diagram “Sell and lease item”

At Fig.4, it is shown how the create and lease an item. Firstly, the salesman has to create a sale/lease and then create a sale/lease item. Creating sale/lease items is a loop which adds till all the items are created. They can include either a product or leasable item and their total amount. Finally, when the loop is done the newly created item/leasable item is added to the sale/lease.

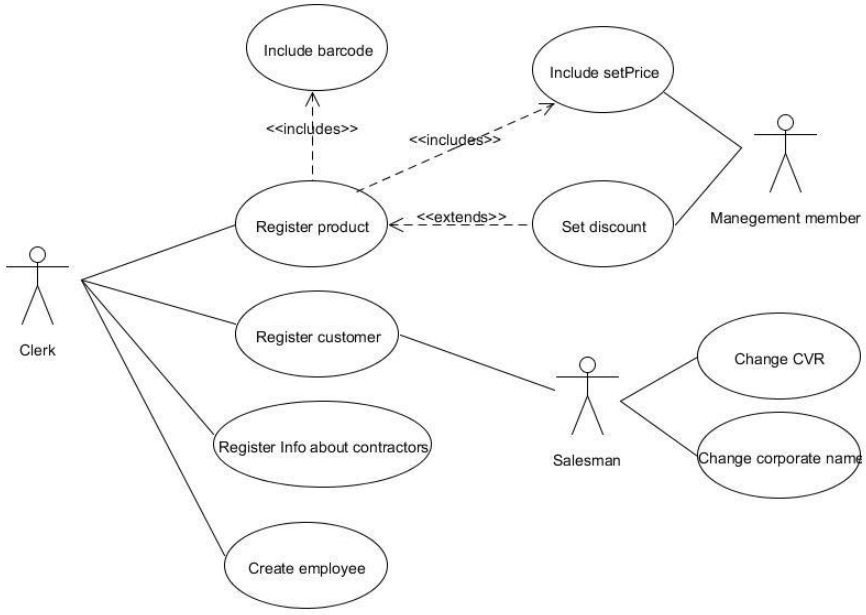


Fig. (5): Use case diagram

3.5 Domain model

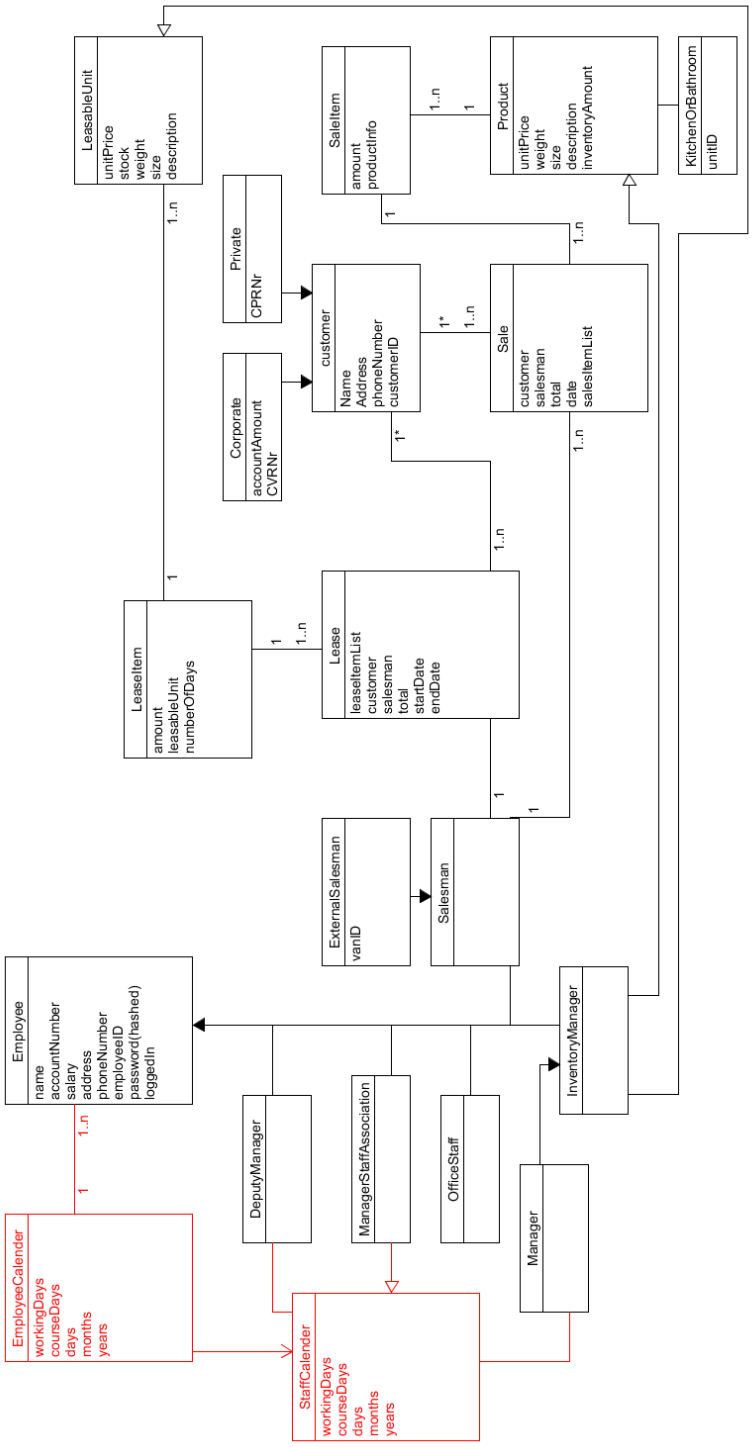


Fig. 6: Domain model

The above domain model got two different classes marked with red because they are not being implemented in the system. We also have a lot of empty classes, because they are only going to hold methods for the employees to inherit and no attributes to speak of. Only eternal salesman adds any attributes to the employee, this is because the eternal salesman gets a car to do home meetings with the customers.

The design holds two very similar parts, this is the sale and lease part. Both of these parts consist of a database (Product, LeasableUnit), a temp holding class (SaleItem, LeaseItem) and in the end a class that holds the entire sale or lease which includes an employee object, a customer object and a list of saleItems or leaseItems. The place these two parts differ is the kitchenOrBathroom class in the sale part, this class is used to show what kitchen or bathroom unit different products are from. For example could a sink belong to a certain full fleshed bathroom, and would then need an indicator to show this, so it is possible to add the entire bathroom to the sale without manually searching from the database.

The last part of the system is the customer database, this got two subclasses (private and corporate) these are used to denominate if a customer is corporate or private, and differ by having a CVR or CPR number, and the corporate customer is able to buy on credit, getting a bill each quarter or each month.

3.6 Fully-dressed use cases

Four use cases were chosen as the most important and were made as fully dressed use cases. The fully dressed use cases describe in details how the whole process of registering a product/customer/leasable unit/sale item works. The primary actors are the salesman and inventory manager. The pre-conditions have to be valid in order to continue to the next step. Post-conditions have to be also valid at the end of the process. The frequency has no impact. Main success scenario is that every single function processes without any errors.

Use case name	**Register customer**	
Actors	Salesman and customer	
Pre-conditions	The salesman must be logged in	
Post conditions	A customer is registered	
Frequency	--	
Main success scenario (Flow of events)	Actor	Local System
	1. Customer wants to buy a product	
	2. The salesman accesses the system	The salesman is logged into the system
	3. The salesman accesses the customer registration	The system provides access to the customer registration details
	4. The salesman adds customer information to the system	The system receives customer information
	5. The salesman registers the customer	The customer is registered into the system
Alternative flows	4a. The salesman has to register the customer either as *Corporate* or as *Personal*	

Fig. 7: Register customer fully-dressed

The primary actor of the register customer use case is the user (salesman). The post-condition is that the information added is valid and the customer is registered successfully.
The salesman right clicks on the class “Corporate” if he wants to create a corporate customer. If the user wants to register a personal customer, he accesses the class “Personal”. The salesman types: the customer’s name, address, phone number, ID and the CVR/CPR. The system adds the customer and all the information about it to the database.
The alternative flow is that the salesman has to register the customer either as the Corporate or as Personal

Use case name	**Register product**	
Actors	Inventory manager	
Pre-conditions	The inventory manager is logged in	
Post-conditions	A product is registered	
Frequency	-	
Main success scenario	Actor	Local System
	1. The inventory manager wants to register the product.	
	2. The inventory manager accesses the system.	The inventory manager is logged in to the system.
	3. The inventory manager accesses the product registration.	The system navigates to the product registration
	4. The inventory manager adds product information.	The system receives product information
	5. The inventory manager registers the product	The product is registered into the system
Alternative flows	-	

Fig. 8: Register product fully-dressed

The primary actor of the register product use case is the user (inventory manager). The post-condition is that the information added is valid and the product is registered.
The user right clicks on the class “Inventory Manager” and chooses the “new product” option and the system asks for input from the user. The inventory manager types: the product’s unit price, weight, size, description and inventory amount. The system adds the product and all the information about it to the database.

Use case name	Create leasable item	
Actors	Salesman and customer	
Pre-conditions	The salesman is logged in	
Post-conditions	A leasable item is created	
Frequency	-	
Main success scenario	Actor	Local System
	1. The customer wants to lease an item	
	2. The salesman accesses the system	The salesman is logged in to the system.
	3. The salesman accesses the leasable item registration	The system navigates to the leasable item registration
	4. The salesman creates a lease.	The system creates a lease
	5. The salesman creates leasable item	The leasable item is added to the system
	6. The salesman adds the leasable item to the lease	The system adds the leasable item to the lease
Alternative flows		

Fig. 9: Create leasable item fully-dressed

Use case name	**Create sale item**	
Actors	Salesman and customer	
Pre-conditions	The salesman is logged in	
Post-conditions	A sale item is created	
Frequency	-	
Main success scenario	Actor	Local System
	1. The customer wants to buy an item	
	2. The salesman accesses the system	The salesman is logged in to the system.
	3. The salesman accesses the sale item registration.	The system navigates through sales page
	4. The salesman creates a sale	The system creates a sale
	5. The salesman creates a sale item	The system creates a sale item
	6. The salesman adds the sale item to the sale	The system stores the sale item in sale
Alternative flows	-	

Table (x): Create sale item fully-dressed

3.7 System sequence diagrams

Using the system sequence diagrams we want to show the interaction and course of events between different actors and the system.

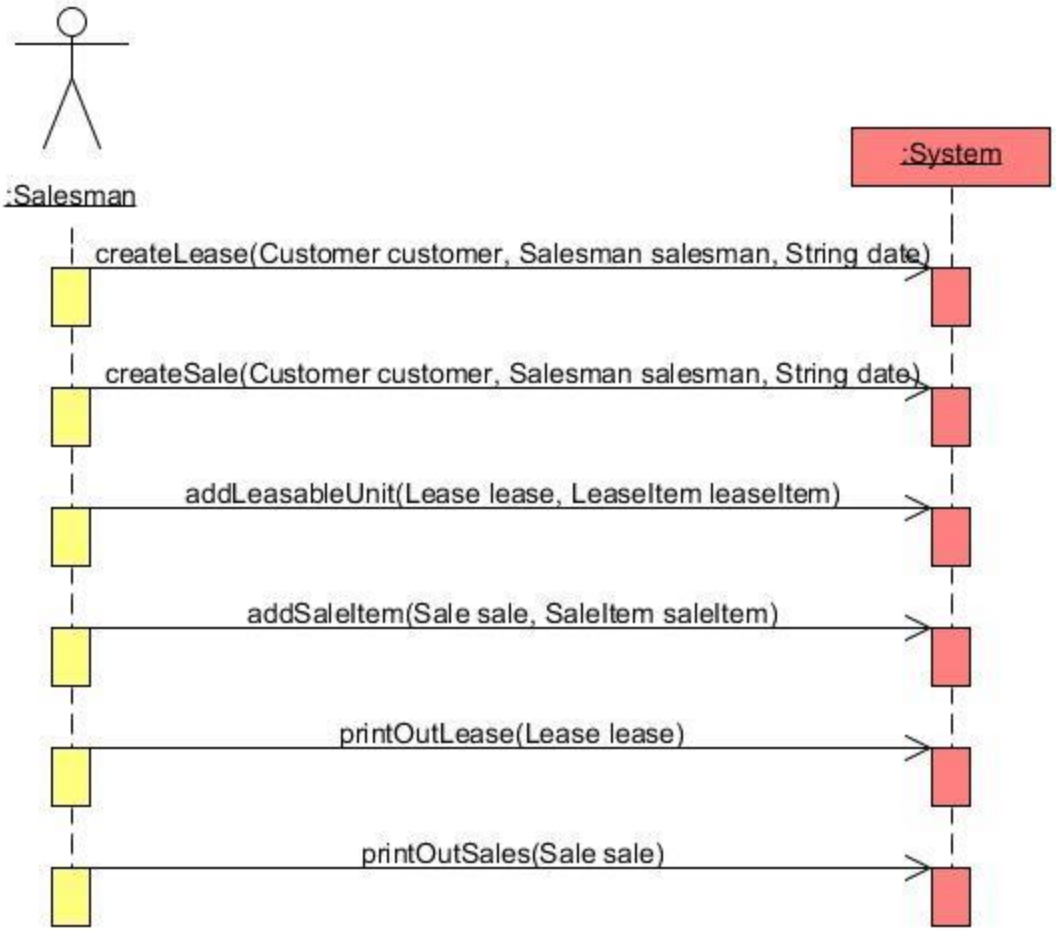
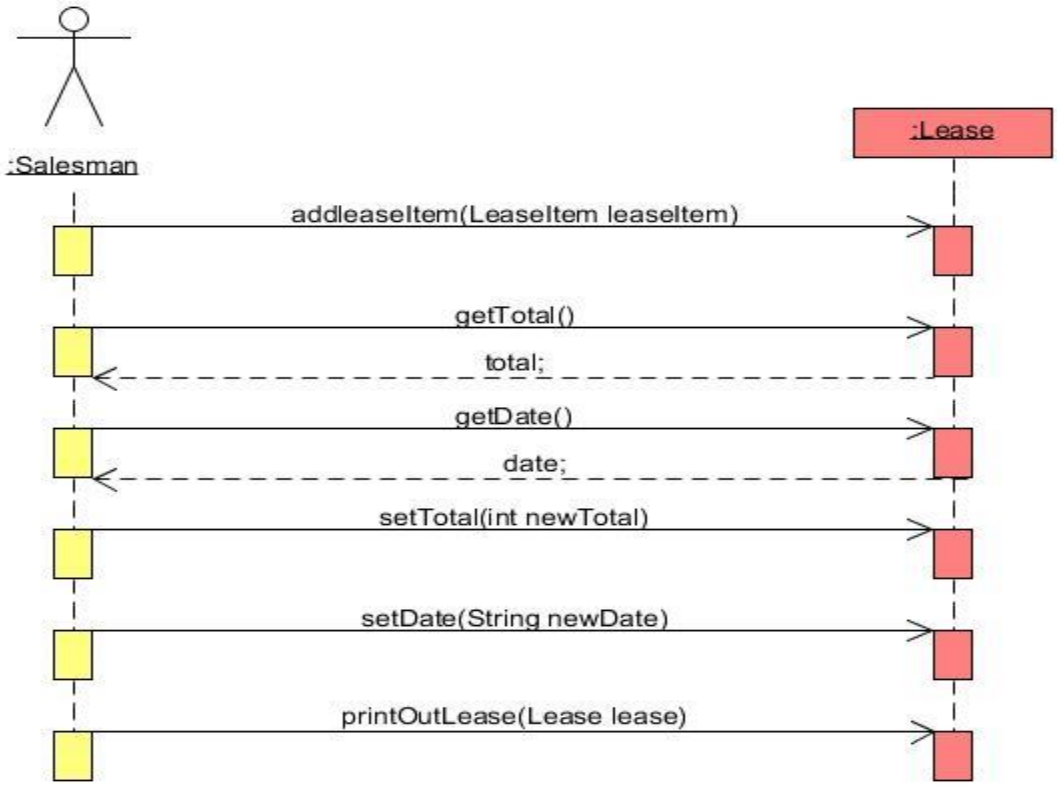


Fig. 10: SSD for all salesman methods

The Salesman can use diverse methods through which he can get usage of the system.

Here we can see that he can create a Lease and Sale object in the presence of a costumer. He can also get information about the Lease and Sales out of the system.

In the second and third image the Salesman can control the Lease and Sale class, setting and getting information.


:Salesman
:Lease
addleaseItem(LeaseItem leaseItem)
getTotal()
total;
getDate()
date;
setTotal(int newTotal)
setDate(String newDate)
printOutLease(Lease lease)


:Salesman
:Sale
addSaleItem(SaleItem saleItem)
getTotal()
total;
getDate()
date;
setTotal(int newTotal)
setDate(String newDate)
printOutSales(Sale sale)

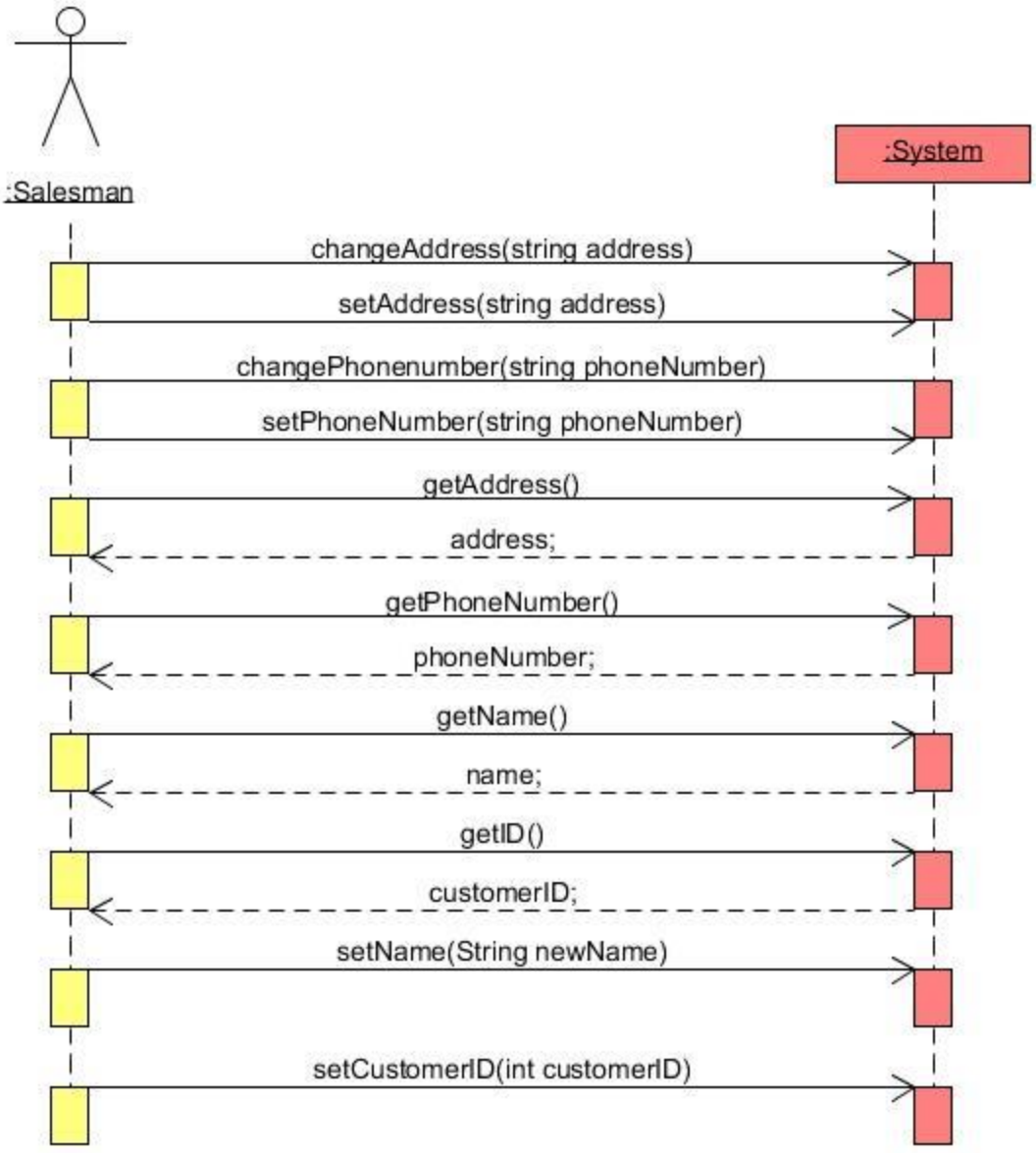


Fig. 11: SSD of employee methods

The methods here are modifying the Costumer class, editing personal information of the costumer like name, address, phone number and costumerID. In case the costumer changes his address or phone number the information can be updated in the system through changeAddress and changePhoneNumber methods.

3.7 Operation contracts

The group made operation contracts as a reminder to pre-conditions and the post-conditions of the desired use cases are. This helps to keep track of each scenario and of each operation and their outputs.

These operation contracts help to identify the system state changes when a certain operation happens. They define what each system operation does. This is always a really good help to the programmer to plan the implementations.
The examples here below show how most of the operation contracts were defined.

Operation: new Customer(name, address, phone, customerID, cvr)
Use case: Customer
Pre-condition: Corporate
Post condition:
If personal:
- name, address, phone, customerID, cpr gets a value from keyboard.
- Object personal is created.

If corporate:
- name, address, phone, customerID, cvr gets a value from keyboard.
- Object corporate is created.

Operation: changeCvrNumber(newCvr)
Use case: Corporate
Pre-condition: A corporate exists.
Post condition: The corporate's cvr number is updated to a new one.

Operation: changeCoorpName(Customer customer, newName)
Use case: Corporate
Pre-condition: A corporate exists.
Post condition: The corporate's name is updated to a new one.

Operation: createLease (itemList, total, date)

Use case: Salesman

Pre-condition: A new lease is being created.

Post condition:

- itemList, total, date gets a value from keyboard
- Object Lease is created

Operation: showLeaseInfo (Lease lease)

Use case: Salesman

Pre-condition: The specific lease is selected and it shows all of its information.

Post condition:

- The objects information gets displayed.

3.8 Class diagram

The class diagram shown at Fig(x) can, like the domain model, be divided into different parts. Employees and the titles, sale, lease and customers. There is also the calendar part, but as mentioned earlier it is not something that is being implemented during this project.

The employee part got all the normal attributes, name, accountnumber … what is obvious to have for the employee object. We also have the password attribute, best case it should be hashed on creation, but because of the time limit for the project it has been chosen to not implement this part of the system, so the password is saved as a normal string. And the loggedIn attribute is a boolean set to true after the correct password is applied. All the other classes in the employee part of the system are titles that an employee can be promoted to.
The promotion method is on the manager object, and should check if the manager is logged in before letting him do anything, actually all methods on the employees are supposed to check the loggedIn boolean.

Sale and lease parts are as mentioned in the domain model text, very similar, all the classes in this part of the system is low on methods, because most of the sale and lease functionality is put in the salesman class instead, this is to avoid checking whether it is a salesman that tries to create a sale or a lease, and instead check if the salesman object is logged in.

The last implemented part is customer, which inherits a corporate or private class based on the customer created. In the design the customer had a few methods to use themselves, but because the employees are the only ones with access to the system all the customer functionality should be put into the salesman employee.


Private
CPRNr
Corporate
accountAmount
CVRNr
changeCVR
changeCustomerName
Customer
Name
Address
phoneNumber
customerID
changePhoneNumber
changeAddress
SaleItem
amount
productInfo
Product
unitPrice
weight
size
description
inventoryAmount
KitchenOrBathroom
unitID
Sale
customer
salesman
total
date
salesItemList
addSaleItem
LeaseItem
amount
leasableUnit
numberOfDays
getLeaseableUnit
Lease
leaseItemList
customer
salesman
total
startDate
endDate
getLeaseableItem
LeasableUnit
unitPrice
stock
weight
size
description
ExternalSalesman
vanID
setVanID
Salesman
createLease
addLeaseItem
createSale
addSaleItem
printOutSales
printOutLease
Employee
name
accountNumber
salary
address
phoneNumber
employeeID
password(hashed)
loggedIn
logIn
logOut
changePassword
getSalary
setAccountNumber
setPhonenumber
setAddress
InventoryManager
addLeasableUnit
changeLeasableUnit (Multiple)
updateStockpile
addProduct
changeProduct (Multiple)
orderInventory
EmployeeCalender
workingDays
courseDays
days
months
years
getCalenderInfo
DeputyManager
ManagerStaffAssociation
setCalenderInfo
OfficeStaff
Manager
setEmployeeSalary
createEmployee
promoteEmployee
setCalenderInfo
StaffCalender
workingDays
courseDays
days
months
years
getCalenderInfo
checkForDoubles

Chapter 4. Implementation

In this chapter some of the implementation in BlueJ will be described.

Calculate endDate for leases

```
else if(month == 4 || month == 6 ||month == 9 ||month == 11)
   {
         if(day > 30)
         {
          month = month + 1;
          day = day - 30;
          endDate = Integer.toString(day)+":"+Integer.toString(month)+":"+Integer.toString(year);
         }
         else
         {
         endDate = Integer.toString(day)+":"+Integer.toString(month)+":"+Integer.toString(year);
         }
   }
```

Snipet 1: Calculate endDate for leases

The above code is a small part of the code that calculates the return date for leases, the entire code part consist of several of these checks.
First we check which month it is, this part of it checks for the 4 months a year with 30 days in (April, Juni, September, November).
After this check it checks whether the variable “day” is above 30, the variable “day” holds the addition of the current date and number of days the lease is intended. If the “day” variable exceeds the number of days in the current month, 1 will be added to the “month” variable indicating that the month has passed. And 30 will be subtracted from “day” variable to find the new day of the month. The new variables for year, month and day is added to a string called endDate. The way this is created adds a condition for leases to not be over one month.

Promote employee

```
  public Employee promoteEmployee(Employee emp, String employeeTitle)
  {
     if(loggedIn)
     {
        if(employeeTitle == "Manager")
        {
           InventoryManager newIM = new InventoryManager(emp);

           Manager newManager = new Manager(newIM);
```

```
            return newManager;

        }
        else if(employeeTitle == "deputyManager")
        {
            DeputyManager newDM = new DeputyManager(emp);
        }
        else if(employeeTitle == "Staffassociation")
        {
            ManagerStaffAssociation newMSA = new ManagerStaffAssociation(emp);

            return newMSA;
        }
         else if(employeeTitle == "OfficeStaff")
        {
            OfficeStaff newOS = new OfficeStaff(emp);

            return newOS;
        }
         else if(employeeTitle == "InventoryManager")
        {
            InventoryManager newIM = new InventoryManager(emp);

            return newIM;
        }
         else if(employeeTitle == "Salesman")
        {
            Salesman newSM = new Salesman(emp);

            return newSM;
        }
         else if(employeeTitle == "ExternalSalesman")
        {
            Salesman newSM = new Salesman(emp);

            ExternalSalesman newESM = new ExternalSalesman(newSM);

            return newESM;
        }
    }
    else
    {
        System.out.print("You need to log in to use this method");
        return emp;
    }
    return null;
}
```

Snipet 2: Promote employees

Another interesting part of the program is the promote employee method in the manager class, the method receives two parameters, the employee object that the manager wants to promote, and a string containing the name of the new employee title.

The method then checks the string and finds the appropriate if sentence and promotes the employee by instantiating a new object of the chosen subclass, given it all the information from the old employee.

There is a bug with this part of the code, the promoted employee object should be deleted after promotion, but this part has not been implemented yet. And it is not possible to promote a salesman to inventory manager or manager. To avoid the lack of promotion possibilities a method to return an employee to the first employee state should be implemented.

Complete sale (two methods)

```
public Sale createSale(Customer customer, Salesman salesman, String date)
{
  if (loggedIn)
  {
    Sale newSale = new Sale(customer, salesman, date);

    return newSale;
  }
  else
  {
    System.out.print("You need to log in to use this method");
    return null;
  }
}

public void addSaleItem(Sale sale, SaleItem saleItem)
{
   if (loggedIn)
   {
     sale.addSaleItem(saleItem);
   }
   else
   {
     System.out.print("You need to log in to use this method");
   }
}
```

Snipet 3: Complete sale Complete sale

The sale consists of two methods, one method to create a sale, with the salesman and customer objects add to the sale object. Afterwards saleItems are created, these objects holds a chosen product and the amount of that product. These sales Items are then added to the sale by the method “addSaleItem”.

There are a few functions missing in these methods, for example is the total price of the sale never calculated.

The program in its entirety can be seen in appendix.

Chapter 5. Conclusion

During the project the group has learned a lot about the different work cultures in the group, we have had a Dane, Bulgarian, Estonian, Lithuanian and Romanian. The differences in these cultures have been interesting and we have learned to express ourselves more clearly towards each other, and not take stuff from our own work culture for granted in others.

The business part of the project has been thoroughly examined, the assignment mentioned different leadership styles from the different bosses and we all agreed that this could lead to insecurities among the employees, mostly because something that could be considered good by one leader might be considered bad by another, leading to the employees changing their methods whenever one manager has taken over by another, during sickness or the like.

The system has been partly implemented, the calendar part has been skipped by choice, but there is still some small stuff that needs to be added. These are so numerous that not all of them will be mentioned here, but some of them are calculating saletotal, leasetotal and make it possible to change an employee's function, from for example salesman to manager.

Appendix

Entire program
Customer

```
import java.util.ArrayList;

public class Customer
{
   public String name;
   public String address;
   public String phoneNumber;
   public int customerID;

   public Customer(String name, String address, String phoneNumber, int customerID)
   {
      this.name = name;
      this.address = address;
      this.phoneNumber = phoneNumber;
      this.customerID = customerID;
   }

   public void changePhoneNumber(String phoneNumber)
   {
      this.phoneNumber = phoneNumber;
   }

   public void changeAddress(String address)
   {
      this.address = address;
   }


 }
```

Cooperate

```
public class Coorporate extends Customer
{
   private static Customer customer;
   private String cvr;

   public Coorporate(Customer customer, String CVRCPR)
   {
      super(customer.name, customer.address, customer.phoneNumber, customer.customerID);
      this.cvr = CVRCPR;
   }

   public Coorporate(String name, String address, String phoneNumber, int customerID, String
CVRCPR)
   {
      super(name, address, phoneNumber, customerID);
      this.cvr = CVRCPR;
   }
```

```
  public void changeCvrNumber(String newCvr)
  {
    cvr = newCvr;
  }

  public void changeCoorpName(Customer customer, String newName)
  {
    customer.name =newName;
  }
}
```

Personal

```
public class Personal extends Customer
{
 private Customer customer;
 private String cpr;

  public Personal(Customer customer, String CVRCPR)
  {
     super(customer.name, customer.address, customer.phoneNumber, customer.customerID);
     this.cpr = CVRCPR;
  }

  public Personal(String name, String address, String phoneNumber, int customerID, String
CVRCPR)
  {
     super(name, address, phoneNumber, customerID);
     this.cpr = CVRCPR;
  }

}
```

Employee

```
import java.lang.Boolean;
import java.lang.System;

public class Employee
{
  public String name;
  public String accountNumber;
  public int salary;
  public String address;
  public String phoneNumber;
  public String employeeID;
  public String password;
  public boolean loggedIn;
```

```
  public Employee(String name, String accountNumber, int salary, String address, String
phoneNumber, String employeeID, String password)
  {

     this.name = name;
     this.accountNumber = accountNumber;
     this.salary = salary;
     this.address = address;
     this.phoneNumber = phoneNumber;
     this.employeeID = employeeID;
     this.password = password;

  }

  public void logIn(String password)
  {
     if (this.password == password)
     {
        this.loggedIn = true;
        System.out.print("You are now logged in");
     }
     else
     {
        System.out.print("You can't log in with the information given");
     }
  }

  public void logOut()
  {
     this.loggedIn = false;
  }

  public void changePassword(String newpass)
  {
     if (loggedIn)
     {
        this.password = newpass;
     }
     else
     {
        System.out.print("You need to log in to use this method");
     }
  }

  public void setPhoneNumber (String phoneNumber)
  {
     if (loggedIn)
     {
     this.phoneNumber = phoneNumber;
      }
     else
```

```
    {
        System.out.print("You need to log in to use this method");
    }
}

public void setAddress (String address)
{
    if (loggedIn)
    {
    this.address = address;
    }
    else
    {
        System.out.print("You need to log in to use this method");
    }
}

public void setAccountNumber(String accountNumber)
{
    if (loggedIn)
    {
    this.accountNumber = accountNumber;
    }
    else
    {
        System.out.print("You need to log in to use this method");
    }
}

public int getSalary()
{
    if(loggedIn)
    {
    return this.salary;
    }
    else
    {
        System.out.print("You need to log in to use this method");
        return 0;
    }
}


/*
public void setSalary(int salary, int safety)
{
    if (safety == 12)
    {
    this.salary = salary;
    }
    else
```

```
      {
      System.out.print("You do not have the authority to do this");
      }
   }
   */

}
```

OfficeStaff

```
public class OfficeStaff extends Employee
{

   public OfficeStaff(Employee employee)
   {
         super(employee.name, employee.accountNumber, employee.salary, employee.address,
employee.phoneNumber, employee.employeeID, employee.password);
   }

}
```

DeputyManager

```
public class DeputyManager extends Employee
{

   public DeputyManager(Employee employee)
   {
         super(employee.name, employee.accountNumber, employee.salary, employee.address,
employee.phoneNumber, employee.employeeID, employee.password);
   }


}
```

InventoryManager

```
/**
* Write a description of class inventoryManager here.
*
* @author (your name)
* @version (a version number or a date)
*/
public class InventoryManager extends Employee
{
```

```
  public InventoryManager(Employee employee)
  {
     super(employee.name, employee.accountNumber, employee.salary, employee.address,
employee.phoneNumber, employee.employeeID, employee.password);
  }

  public LeasableUnit addLeasableUnit(int unitPrice, int weight, String size, String description,
int stock)
  {
     if (loggedIn)
     {
        LeasableUnit newLeasableUnit = new LeasableUnit(unitPrice, weight, size, description,
stock);

        return newLeasableUnit;
     }
     else
     {
        System.out.print("You need to log in to use this method");
        return null;
     }
  }

  public Product addProduct(int unitPrice, int weight, String size, String description, int
inventoryAmount)
  {
     if (loggedIn)
     {
        Product newProduct = new Product(unitPrice, weight, size, description,
inventoryAmount);

        return newProduct;
     }
     else
     {
        System.out.print("You need to log in to use this method");
        return null;
     }
  }

  public void changeProductPrice(Product product,int newPrice)
  {
     if (loggedIn)
     {
        product.unitPrice = newPrice;
     }
     else
     {
        System.out.print("You need to log in to use this method");
     }
```

```
}

public void changeLeasableUnitPrice(LeasableUnit unit,int newPrice)
{
    if (loggedIn)
    {
        unit.unitPrice = newPrice;
    }
    else
    {
        System.out.print("You need to log in to use this method");
    }
}

    public void changeProductWeight(Product product,int newWeight)
{
    if (loggedIn)
    {
        product.weight = newWeight;
    }
    else
    {
        System.out.print("You need to log in to use this method");
    }
}

public void changeLeasableUnitWeight(LeasableUnit unit, int newWeight)
{
    if (loggedIn)
    {
        unit.weight = newWeight;
    }
    else
    {
        System.out.print("You need to log in to use this method");
    }
}

    public void changeProduct(Product product, String newSize)
{
    if (loggedIn)
    {
        product.size = newSize;
    }
    else
    {
        System.out.print("You need to log in to use this method");
    }
}

public void changeLeasableUnitSize(LeasableUnit unit, String newSize)
```

```
{
    if (loggedIn)
    {
        unit.size = newSize;
    }
    else
    {
        System.out.print("You need to log in to use this method");
    }
}

    public void changeProductSize(Product product, String newDescription)
{
    if (loggedIn)
    {
        product.description = newDescription;
    }
    else
    {
        System.out.print("You need to log in to use this method");
    }
}

public void changeLeasableUnitDescription(LeasableUnit unit, String newDescription)
{
    if (loggedIn)
    {
        unit.description = newDescription;
    }
    else
    {
        System.out.print("You need to log in to use this method");
    }
}

public void orderInventory(Product product, int orderInventoryAmount)
{
    if (loggedIn)
    {
        int tp = product.inventoryAmount;
        int ip = tp + orderInventoryAmount;
        product.inventoryAmount = ip;
    }
    else
    {
        System.out.print("You need to log in to use this method");
    }
}

public void updateStockpile(LeasableUnit leasableunit, int orderStock)
{
```

```
      if (loggedIn)
      {
         leasableunit.stock = leasableunit.stock + orderStock;
      }
      else
      {
         System.out.print("You need to log in to use this method");
      }
   }
}
```

Manager

```
public class Manager extends InventoryManager
{

   int safety = 12;

   public Manager(InventoryManager IM)
   {
      super(IM);

   }

   public void setEmployeeSalary(int salary, Employee employee)
   {
      if (loggedIn)
      {
         employee.salary = salary;
      }
      else
      {
         System.out.print("You need to log in to use this method");
      }
   }

   public Employee createAnEmployee(String name, String accountNumber, int salary, String
address, String phoneNumber, String employeeID, String password)
   {
      if (loggedIn)
      {
         Employee newEmployee = new Employee(name, accountNumber, salary, address,
phoneNumber, employeeID, password);

         return newEmployee;
      }
      else
      {
         System.out.print("You need to log in to use this method");
         return null;
```

```
  }
}

public Employee promoteEmployee(Employee emp, String employeeTitle)
{
   if(loggedIn)
   {
      if(employeeTitle == "Manager")
      {
         InventoryManager newIM = new InventoryManager(emp);

         Manager newManager = new Manager(newIM);

         return newManager;

      }
      else if(employeeTitle == "deputyManager")
      {
         DeputyManager newDM = new DeputyManager(emp);
      }
      else if(employeeTitle == "Staffassociation")
      {
         ManagerStaffAssociation newMSA = new ManagerStaffAssociation(emp);

         return newMSA;
      }
        else if(employeeTitle == "OfficeStaff")
      {
         OfficeStaff newOS = new OfficeStaff(emp);

         return newOS;
      }
        else if(employeeTitle == "InventoryManager")
      {
         InventoryManager newIM = new InventoryManager(emp);

         return newIM;
      }
        else if(employeeTitle == "Salesman")
      {
         Salesman newSM = new Salesman(emp);

         return newSM;
      }
        else if(employeeTitle == "ExternalSalesman")
      {
         Salesman newSM = new Salesman(emp);

         ExternalSalesman newESM = new ExternalSalesman(newSM);

         return newESM;
```

```
            }
        }
        else
        {
            System.out.print("You need to log in to use this method");
            return emp;
        }
        return null;
    }
    /*
     * public void setCalenderInfo(int days, int months, int years, Employee employee, String info)
     * {
     *     employee.employeeCalender.setCalenderInfo(days, months, years, info, safety);
     * }
     */
}
```

ManagerStaffAssociation

```
public class ManagerStaffAssociation extends Employee
{
    // instance variables
    /**
     * Constructor for objects of class managerStaffAssociation
     */
    public ManagerStaffAssociation(Employee employee)
    {
      super(employee.name, employee.accountNumber, employee.salary, employee.address,
employee.phoneNumber, employee.employeeID, employee.password);

    }
}
```

Salesman

```
/**
* Write a description of class Salesman here.
*
* @author (your name)
* @version (a version number or a date)
*/
public class Salesman extends Employee
{

    public Salesman(Employee employee)
    {
```

```
    super(employee.name, employee.accountNumber, employee.salary, employee.address,
employee.phoneNumber, employee.employeeID, employee.password);
  }

  public Lease createLease(Customer customer, Salesman salesman, String date)
  {
     if (loggedIn)
     {
        Lease newLease = new Lease(customer, salesman, date);

        return newLease;
     }
     else
     {
         System.out.print("You need to log in to use this method");
         return null;
     }
  }

  public void addLeasableUnit(Lease lease, LeaseItem leaseItem)
  {
     if (loggedIn)
     {
         lease.addleaseItem(leaseItem);
     }
     else
     {
         System.out.print("You need to log in to use this method");
     }
  }

  public Sale createSale(Customer customer, Salesman salesman, String date)
  {
    if (loggedIn)
    {
        Sale newSale = new Sale(customer, salesman, date);

        return newSale;
    }
    else
    {
        System.out.print("You need to log in to use this method");
        return null;
    }
  }

  public void addSaleItem(Sale sale, SaleItem saleItem)
  {
     if (loggedIn)
     {
         sale.addSaleItem(saleItem);
```

```
        }
        else
        {
            System.out.print("You need to log in to use this method");
        }
    }

    public void printOutSale(Sale sale)
    {
        if (loggedIn)
        {
            System.out.print("Salesman: "+sale.salesman.employeeID+"    "+"Customer:
"+sale.customer.customerID+ "  " + "Startdate: " + sale.date +"\n");
            System.out.print("Products: \n");
            for(SaleItem saleItem : sale.salesItemList)
            {
                System.out.print(saleItem.amount + "Product: " + saleItem.productInfo + "\n");
            }

        }
        else
        {
            System.out.print("You need to log in to use this method");
        }
    }

    public void printOutLease(Lease lease)
    {
        if (loggedIn)
        {
            System.out.print("Salesman: "+lease.salesman.employeeID+"    "+"Customer:
"+lease.customer.customerID+ "  " + "Startdate: " + lease.startDate + "Returndate: " +
lease.endDate + "\n");
            System.out.print("Products: \n");

              for(LeaseItem leaseItem : lease.leaseItemList)
            {
                System.out.print(leaseItem.amount + "Product: " + leaseItem.leasableUnit + "\n");
            }
        }
        else
        {
            System.out.print("You need to log in to use this method");
        }
    }


}
```

ExternalSalesman

```
public class ExternalSalesman extends Salesman
{
   // instance variables - replace the example below with your own
   private int vanID;

   /**
    * Constructor for objects of class ExternalSalesman
    */
   public ExternalSalesman(Salesman SM)
   {
      super(SM);
   }

   public void setVanID(int vanID)
   {
      if (loggedIn)
      {
         this.vanID = vanID;
      }
      else
      {
         System.out.print("You need to log in to use this method");
      }
   }

}
```

Product

```
public class Product
{
   public int unitPrice;
   public int weight;
   public String size;
   public String description;
   public int inventoryAmount;

public Product(int unitPrice, int weight, String size, String description, int inventoryAmount)
{
   this.unitPrice = unitPrice;
   this.weight = weight;
   this.size = size;
   this.description = description;
   this.inventoryAmount = inventoryAmount;
}

}
```

KitchenOrBathroom

```
public class KitchenOrBathroom extends Product
{
   // used to differ between instances of bathrooms or kitchens (Example: b1, b2, b3 (bathroom1, bathroom2 ... ) k1, k2, k3 (kitchen1, kitchen3...)
   private String bathroomOrKitchenID;

   /**
    * Constructor for objects of class KitchenOrBathroom
    */
   public KitchenOrBathroom(Product product)
   {
       // initialise instance variables
       super(product.unitPrice, product.weight, product.size, product.description, product.inventoryAmount);
   }

   /**
    * An example of a method - replace this comment with your own
    *
    * @param  y   a sample parameter for a method
    * @return     the sum of x and y
    */
   public void setID(String bORkID)
   {
       // put your code here
       this.bathroomOrKitchenID = bORkID;
   }
}
```

LeasableUnit

```
public class LeasableUnit
{
   public int unitPrice;
   public int weight;
   public String size;
   public String description;
   public int stock;


  public LeasableUnit(int unitPrice, int weight, String size, String description, int stock)
   {

      this.unitPrice = unitPrice;
      this.weight = weight;
      this.size = size;
      this.description = description;
```

```
      this.stock = stock;


   }
}
```

LeaseItem

```
/**
* Write a description of class SaleItem here.
*
* @author (your name)
* @version (a version number or a date)
*/
public class LeaseItem
{
  public int amount;
  public LeasableUnit leasableUnit;
  public int numberOfDays;

  public LeaseItem(int amount, LeasableUnit leasableUnit, int numberOfDays)
  {
      if (leasableUnit.stock > amount)
      {
      this.amount = amount;
      this.leasableUnit = leasableUnit;
      this.numberOfDays = numberOfDays;
      leasableUnit.stock = leasableUnit.stock - amount;
      }
      else
      {
      System.out.print("The requested amount is not in stock");
      }
  }

}
```

Lease

```
import java.util.ArrayList;


public class Lease
{
   public Customer customer;
   public Salesman salesman;
   public int total;
   public String startDate;
   public String endDate;
```

```
  public ArrayList<LeaseItem> leaseItemList = new ArrayList<LeaseItem>();


  public Lease(Customer customer, Salesman salesman, String startDate /*31-12-00*/)
  {
    this.customer = customer;
    this.salesman = salesman;
    this.startDate = startDate;
    System.out.print(startDate);
  }

     public void addleaseItem(LeaseItem leaseItem)
  {
    int year = 0, month = 0, day = 0;
    String[] temp;

    temp = this.startDate.split(":");

    System.out.print(year + " " + month + " " + day + "\n");
    System.out.print("Startdate: " + startDate + "\n");


    year = Integer.parseInt(temp[3]);
    month = Integer.parseInt(temp[2]);
    day = Integer.parseInt(temp[1]);
    day = day + leaseItem.numberOfDays;

    System.out.print(year + " " + month + " " + day + "\n");

    if(month == 2)
    {
       if(year % 4 == 0)
       {
          if(day > 29)
          {
           month = month + 1;
           day = day - 29;
           endDate = Integer.toString(day)+":"+Integer.toString(month)+":"+Integer.toString(year);
          }
          else
          {
          endDate =
Integer.toString(day)+":"+Integer.toString(month)+":"+Integer.toString(year);
          }
        }
       else
       {
          if(day > 28)
          {
           month = month + 1;
           day = day - 28;
```

```
            endDate = Integer.toString(day)+":"+Integer.toString(month)+":"+Integer.toString(year);
           }
           else
           {
           endDate =
Integer.toString(day)+":"+Integer.toString(month)+":"+Integer.toString(year);
           }
        }
    }
    else if(month == 4 || month == 6 ||month == 9 ||month == 11)
    {
           if(day > 30)
           {
            month = month + 1;
            day = day - 30;
            endDate = Integer.toString(day)+":"+Integer.toString(month)+":"+Integer.toString(year);
           }
           else
           {
           endDate =
Integer.toString(day)+":"+Integer.toString(month)+":"+Integer.toString(year);
           }
    }
    else if(month == 1 || month == 3 || month == 5 || month == 7 || month == 8 || month == 10)
    {
           if(day > 31)
           {
            month = month + 1;
            day = day - 31;
            endDate = Integer.toString(day)+":"+Integer.toString(month)+":"+Integer.toString(year);
           }
           else
           {
           endDate =
Integer.toString(day)+":"+Integer.toString(month)+":"+Integer.toString(year);
           }
    }
    else if(month == 12)
    {
           if(day > 31)
           {
            month = 1;
            year = year + 1;
            day = day - 31;
            endDate = Integer.toString(day)+":"+Integer.toString(month)+":"+Integer.toString(year);
           }
           else
           {
            endDate =
Integer.toString(day)+":"+Integer.toString(month)+":"+Integer.toString(year);
           }
```

```
    }
    else
    {
        System.out.print("There is something wrong with the string");
    }

    leaseItemList.add(leaseItem);

  }

  public int getTotal()
  {
     return total;
  }

  public String getDate()
  {
     return startDate;
  }

  public void setTotal(int newTotal)
  {
     this.total = newTotal;
  }

  public void setDate(String newDate)
  {
     this.startDate = newDate;
  }

 // public void getCustomerInfo(Customer customer)
 // {
 //     int theID;
 //     String theName;
  //    String theAddress;
 //     String thePhoneNumber;
 //     theID = salesman.getSalesmanID();
 //     theName = customer.getSalesmanName();
 //     theAddress = customer.getAddress();
 //     thePhoneNumber = customer.getPhoneNumber;
 //     System.out.println("Customer's name:" + theName);
 //     System.out.println("Customer's address:" + theName);
 //     System.out.println("Customer's phone number:" + theName);
  //    System.out.println("Customer's ID:" + theID);
  //}
  /*
  public void getSalesmanInfo(Salesman salesman)
  {
     int theID;
     String theName;
     theID = salesman.getSalesmanID();
```

```
      theName = salesman.getSalesmanName();
      System.out.println("Salesman's name:" + theName);
      System.out.println("Salesman's ID:" + theID);
   }
   */
}
```

Sale

```
import java.util.ArrayList;
/**
* Write a description of class Sale here.
*
* @author (your name)
* @version (a version number or a date)
*/
  public class Sale
{
   public Customer customer;
   public Salesman salesman;
   public ArrayList<SaleItem> salesItemList = new ArrayList<SaleItem>();
   private int total;
   public String date;


   public Sale(Customer customer, Salesman salesman, String date)
   {
      this.date = date;
      this.customer = customer;
      this.salesman = salesman;
   }

   public void addSaleItem(SaleItem saleItem)
   {
       salesItemList.add(saleItem);
   }


}
```

SaleItem

```
public class SaleItem
{
  public int amount;
  public Product productInfo;
```

```
  public SaleItem(int amount, Product product)
  {
      if (product.inventoryAmount > amount)
      {
      this.amount = amount;
      this.productInfo = product;
      product.inventoryAmount = product.inventoryAmount - amount;
       }
       else
       {
           System.out.print("The requested amount is not in inventory");
       }
  }

}
```