

Danish Crown Project

Group “Samsung Headphones”

Aurelian-Cristian Leahu

Marek Lovčí

Peter Majchrák

Umarbek Bakaev

From **Wednesday, 16 March 2016 (8:30 a.m.)**

To **Thursday, 21 April 2016 (12:00 p.m.)**

Class DMU-15I

Business Academy Aarhus - University of Applied Sciences

The date of submission:

Number of pages: 68

Table of contents

BIT

- [Project establishment](#)
- [Team member description](#)
- [Teamwork expectations](#)
- [Company description](#)
- [Business case](#)
 - [Business case table](#)
 - [Truck loading system](#)
 - [Our second idea](#)
 - [Branch in the South Europe](#)
- [Production management](#)
- [Implementation of the solution](#)
- [Teamwork conclusion](#)

SD

- [Requirements](#)
- [Use case diagram](#)
- [Use case description](#)
 - [Use case 1: Sign in](#)
 - [Use case 2: Sign out](#)
- [State machine diagram](#)
- [Class diagram](#)
- [GUI description](#)
- [Class description](#)
 - [ProductType](#)
 - [Order](#)
 - [SubOrder](#)
 - [Dock](#)
 - [Schedule](#)
 - [Trailer](#)
 - [Truck](#)
 - [Driver](#)
 - [Card](#)
- [Test](#)
 - [Service.outOfMargin\(\)](#)
 - [Service.verifyUser\(\)](#)
 - [Service.organizeSchedule\(\)](#)

Programming

- [Decision report](#)
- [GUI](#)
- [Algorithm Description](#)
- [Conclusion](#)

DAOS

- [Description](#)
 - [MSSQL description](#)
 - [Exercise 1](#)
 - [Exercise 2](#)
 - [Exercise 3](#)
 - [Exercise 4](#)

[JAVA description](#)
[Conclusion](#)
[Sources](#)
[Appendix 1](#)
[Group contract](#)
[Appendix 2](#)
[Databases](#)
[MSSQL code](#)
[Java code](#)
[Appendix 3](#)
[Java code](#)
[Model](#)
[Service](#)
[Storage](#)
[Test](#)
[GUI](#)

BIT

Project establishment

We started by making a contract that includes the regular schedule for our meetings and other rules. The contract (Appendix 1) was then signed by all the group members.

Team member description

Cristi (ENTP) - Cristi is the main “Debater” in our group. As second in charge of programming Cristi shows how charismatic, energetic and knowledgeable he can be. During our meetings he was the most argumentative person.

Marek (INTJ) - Marek is “The Architect” the initiator of all our perspective ideas. The GUI design and analyses were assigned to him, because of his natural architectural skills. Marek is the perfectionist as he cares about tiny things which others don't. He was responsible for making our project beautiful and for sure he has done it good.

Peter (ISTP) - “The Virtuoso”. Peter is in programming the most experienced one from our group. He supervises the whole programming part and is responsible of merging commits. In the view of personality Peter showed how practical, creative and rational he really is.

Umar (ESTJ) - As “The Executive” Umar is supposed to be natural leader. During our sessions Umar was the one, who poked us ahead. From the beginning Umar was in charge of arranging meetings with teachers and so on. Thanks to his strong will and dedication we always could rely on him.

Teamwork expectations

Most of the time we work in groups of two people. That means, that programming is going to be done as pair programming. We think, that this will be more efficient and connecting Service and GUI will be less problematic. Umar and Marek are expected to take care mainly of GUI and big part of documentation. On the other hand Christi and Peter are supposed to make “the hard” programming part (like Service and Model).

Company description



Danish Crown is the biggest producer of meat products in Europe and the 2nd biggest on the world. The company's first slaughterhouse was founded in 1887 in Horsens.

The company is owned by individual shareholders (farmers). The entire

Danish Crown group has over 23 000 employees. In Horsens they slaughter approximately 20 000 pigs every day.



The main priority of Danish Crown is quality of their products. The quality is assured on every level of the production process, from breeding the pigs until they are loaded into trucks. The company has its own "Code of practice" that provides hygienic requirements and guidelines.

One of the other priorities of the company is to assure that the pigs have proper care during this whole process. There are rules stating for example the maximum time of transportation and keeping the pigs in groups they are familiar with.

Nowadays it is very important to hear what customers want so the company offers flexible orders to satisfy as much customers as possible.

The output can be in three main forms: boxed, palletised or hanging. There are over 550 trucks loaded every week.

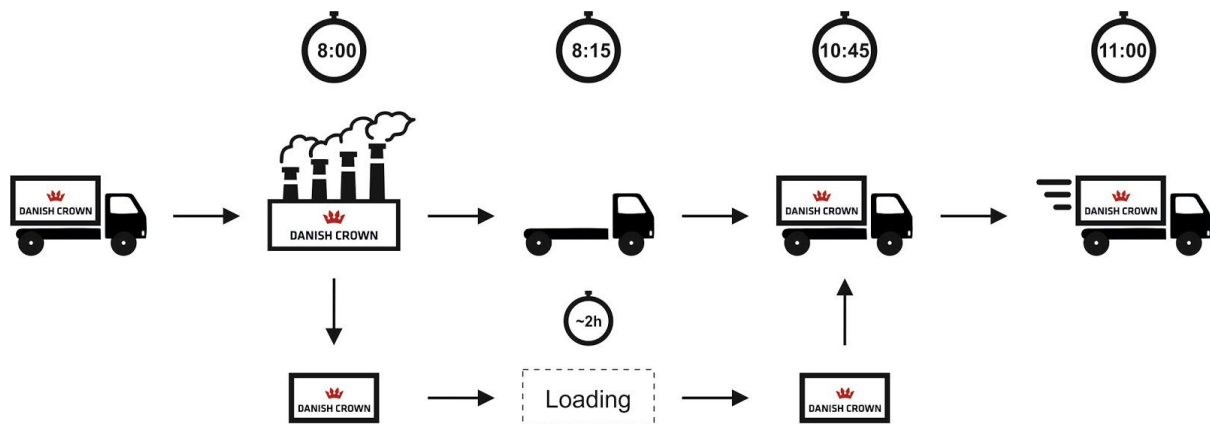
Approximately 90 % of the 20 million pigs per year go to foreign markets and the rest hits the local markets.

At Danish Crown nothing goes to waste. All of the parts of the pig find their uses. If some parts are not suited for human consumption they are sold to environmentally friendly feedstuffs, biodiesel, animal food or even research purposes.

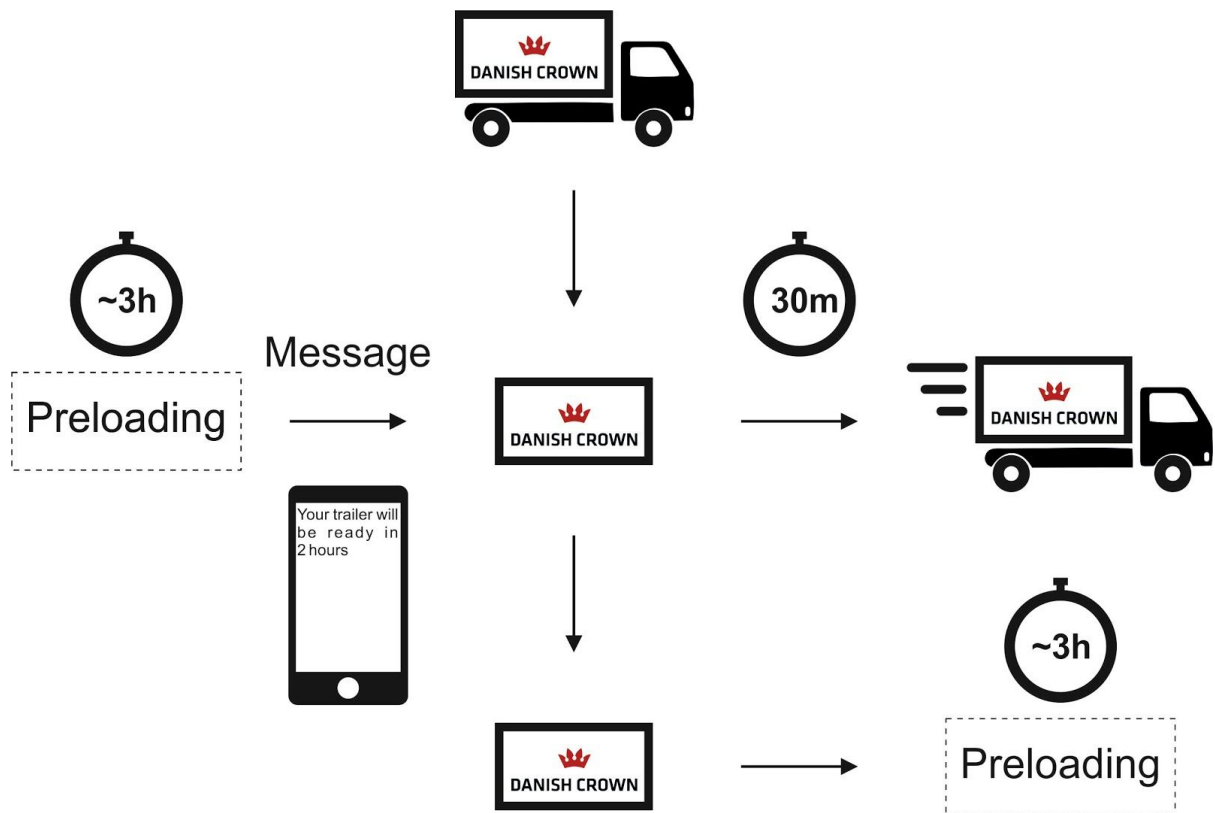


Business case

In the following business case we would like to introduce new Truck loading system which will save resources for DC. Current loading and dispatching system is a challenge for the company.



It can be seen from picture 1 that current system is complicated and can face unexpected issues and expenses. For example it can be truck loading delay, or “traffic” in loading ramps. It is around 120 truck per day passing DC. Which means, that one loading ramp serve five trucks per day. It is around 3 hours per 1 truck without delays. Our main goal is to make structured and organized system which will mitigate possible uncertainties. As you can see from the second picture. It is advised to use preloading system. All trailers will be loaded and ready to dispatch at time when truck driver starts his work day. The principle is that last driver leaves trailer in the ramp in the end of the shift. When first truck arrives to the DC it leaves empty trailer to the ramp and takes loaded trailer and deliver it. All drivers will be informed by the system 2 hours ahead they need to arrive to the DC. By that system DC can avoid unexpected expenses as drivers arrive at exact time and passing out DC in max 30 minutes as it will not be “traffic” in loading ramps.



Business case table

	Truck system loading	Current system
Investment for truck loading system	2 000 000 DKK	-
Payment for app stores	2 000 DKK/year	-
Extra parking/storage place	500 000 DKK	-
Training cost	500 000 DKK	-

Risks		
Delays	-	100.000 DKK/month
System failure/maintenance	100 000 DKK/month	-
Other expenses		

Driver salary per hour	500 DKK	500 DKK
Driving hours per day	2016	2376
Drivers salary per month	22 176 000 DKK	26 136 000 DKK
Expenses per first month	25 278 000 DKK	26 236 000 DKK
Expenses per each other month	22 276 000 DKK	26 236 000 DKK

Profit for first month	958 000 DKK	-
Profit for each other month	3 960 000 DKK	-

Moreover new truck loading system will pay off itself in first month. In advance it could save money for company for new opportunities. Below Business case calculation description.

Investment for truck loading system – developing and implementation system in DC work environment.

Yearly payment for app stores – yearly fee for app store/google store in order to store application.

Extra parking/storage place – reorganisation of current temporary parking place.

Training cost – training fee for train drivers and all needed employees.

Delays – delays risks in average which will cost for DC when goods are not delivered to the customer at time.

System failure/maintenance – system monthly maintenance or unexpected failures to be fixed.

Driver salary per hour – average driver salary per hour in Denmark.

Driving hours per day – total driving hours per day for delivering goods to customers or distribution centers. Around 120 trucks are loaded each day with goods in DC which takes place from 25 loading ramps.

Drivers salary per month – driver salary per hour multiplied by driving hours per day multiplied to 22 working days per month.

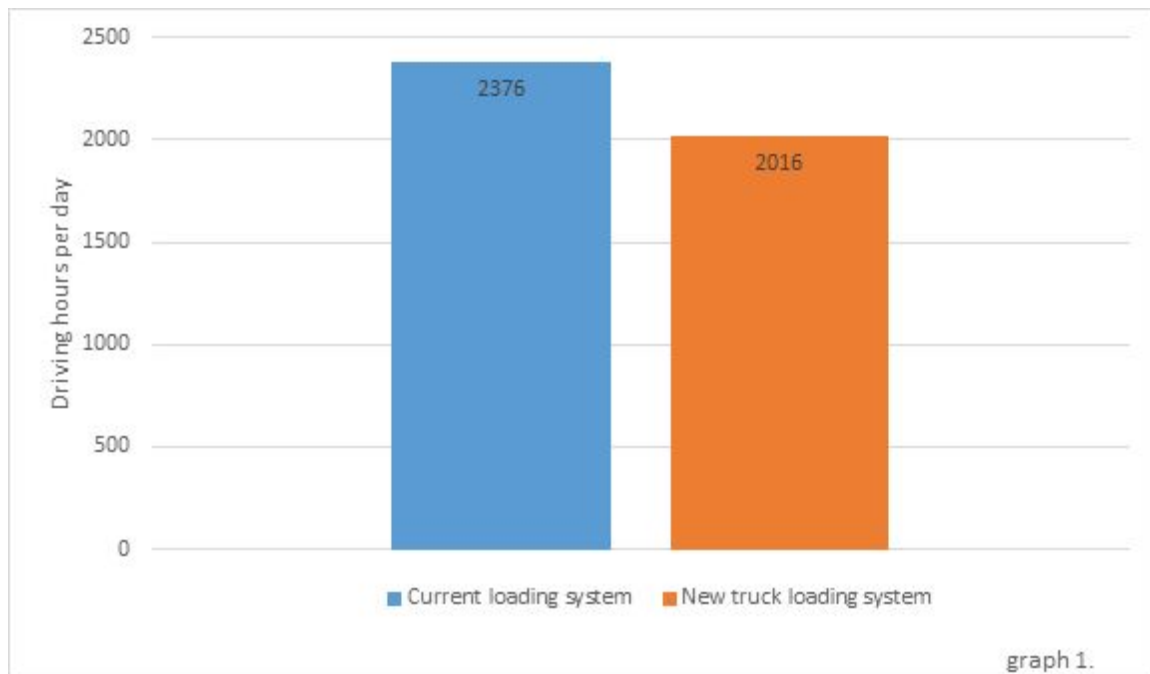
Expenses per first month – sum of all expenses per first month.

Expenses per each other month – sum of all expenses per each other month. Not included first month.

Profit for first month – net profit for first month.

Profit for each other month – net profit for each other month. Not included first month.

In graph 1. Is illustrated how many hours drivers need to drive in order to serve DC per day with current system and new truck loading system.



Truck loading system

Advantages

1. Cost out and save time

The drivers are not wasting time to waiting for loaded trailers. There for the company will avoid unnecessarily payment.

2. Flexibility for company

First in first out. If driver B comes faster than driver A than he can take driver A trailer. They can exchange trailers.

3. Smooth workflow

Production should not wait driver with truck. They will load goods to empty trailers and can move loaded trailer to the temporary parking place. They can start loaded another empty trailer.

Disadvantages

1. Time for training

All drivers and employees from loading department should take “Truck loading system” trainings before system will be implemented.

2. System collapse

It is possibilities that system failures.

3. Inconvenient for drivers

Not fixed schedule.

4. Initial cost for system

System price.

Our second idea

Next good idea we were able to come with was to make a new slaughter house in South Europe. We still think it was very good idea, but when we started to do calculations, we realized, that we would need much more informations to be at least semi-accurate and we would spent too much time with it. But for an idea what we wanted to do is there small part of what we did so far.

Branch in the South Europe

Advantages

1. Opportunity for new market

Opening a new branch will potentially increase the market area. It will allow the company to produce fresh meat for all the southern area. The profits will increase.

2. Customer satisfaction

The customer will have more opportunities for selling the pigs. The branch can buy local pigs and slaughter them there.

3. Save money for transporting

Having a local branch for the souther part will reduce the amount of money spend for transporting the pigs and the goods

4. New job opportunities

The opening of this branch will come with available jobs in different area, which will benefit and satisfy locally people.

Disadvantages

1. Initial costs

Opening a new DC slaughterhouse will require an investment of large amount of money, in order to fit the DC standards.

2. Risk of being unsuccessful

After the investment, if it does not hit the market it will not be great.

Production management

To Danish Crown slaughterhouse in Horsens arrive about 100 trucks with 200 pigs every day. That makes adequate supply of pigs for planning of slaughteries for next day. In general, Danish Crown plans everything on a daily, weekly, monthly and annual basis. This is based on estimations from previous years.

Pigs are usually not staying in the factory over the night. It would make other expenses (food, care, stalls). Usually, orders are planned only 24 hours before implementation to the system. It makes everything as flexible and fluent as possible. All pigs are measured and weighed upon their arrival.

As mentioned earlier, about 550 lorries are sent from DC to their customers every week. After their arrival to DC Horsens they hand over the truck trailer and wait till it is loaded with goods on the one from 25 loading ramps. Drivers have to rest before they leave. It is better for them rest before their way, because they will be able to move the goods into the final destination a little bit faster.

It is very difficult to plan which order should go to the which lorry. Some orders have to split into several units. It makes everything even more problematic. On the other hand, sometimes are trucks loaded with several orders (because of cost of transportation). Those trucks are reloaded at collection point.

Implementation of the solution

Firstly, a developer will come and analyse the loading process, and see the requirements. The developer will describe the challenges to his developer team to come up with the best solution.

The system will consist of the following parts: a mobile log-in application for the drivers to see all the information and a desktop system application for the loaders to see the schedule of every trailer and the belonging ramp.

There will be a person suited to put every registered order of the truck into the system. Now the system will have the list of all the orders that can be seen for both the loaders and the drivers. The driver will get reminder notification 2 hour before the trip. When the driver will arrive in the DC parking area, he can log-in in the mobile application and press the "Arrived" button so the system will know the arrival. After the "Arrived" button will get pressed, the driver will be assigned with the order and get the specific route. The system will have a track on the drivers by the build-in gps mobile app. It will fetch location data and update it to the

system. Later on, when the drivers finish the route, they will press a “Delivered” button which will update the time in the system and prepare the next shipment.

During the development process there will be regular visits and presentations of the system to the end-users in order to get feedback how to improve the system.

The implementation will start with presenting the solution to the employees and provide employees with system user manuals. Furthermore there will hand-on training giving the users chance to put the system in practice so they can learn how to use it and get feedback for possible system improvements.

Teamwork conclusion

Our team is a Problem-solving team as we were working on the project where the problem was to develop software system for organizing loading process in the Danish Crown. All decisions were agreed between at least 3 of 4 team members. Team meetings were hold 3-4 times per week at Academy, beside meetings there were everyday follow up discussions via facebook group which was created for project purpose. Teamwork process was smooth without any long argues or controversies as all ideas were well argued. There were “step back” parts for revision of some decisions and all team members were reacting as professionals. In the our team there were chosen team leader who was responsible for agenda for the meetings and for merging all coding parts. Team leader was chosen according to experience and competences in management. There were arranged meetings for improving team spirit, where team members were talking or discussing subjects not related to the project for example teambuilding activities. All of the meetings were in the friendly atmosphere, it was no pressure or stress to the members who was late with deadlines or was not able to appear in the meeting. There were 90 % of the tasks done within deadlines. In the first team meeting the project contract with meeting dates and start and end time were agreed between team members. Tasks were divided between members according to competences and background. However work was not done individually as project parts were always explained and brain stormed with the whole team. All in all it was great experience for the whole team.

SD

Requirements

ID	Description	Type
1	Get list of schedules	M
2	Send notification SMS	M
3	Display weight out of range error	M
4	Driver sign in	M
5	Driver sign out	M
6	Schedule the loading	M
7	GUI	M
8	Rescheduling of trailers	S
9	Get statistics	C
10	Re-notify the driver when the trailer is loaded and waiting	S

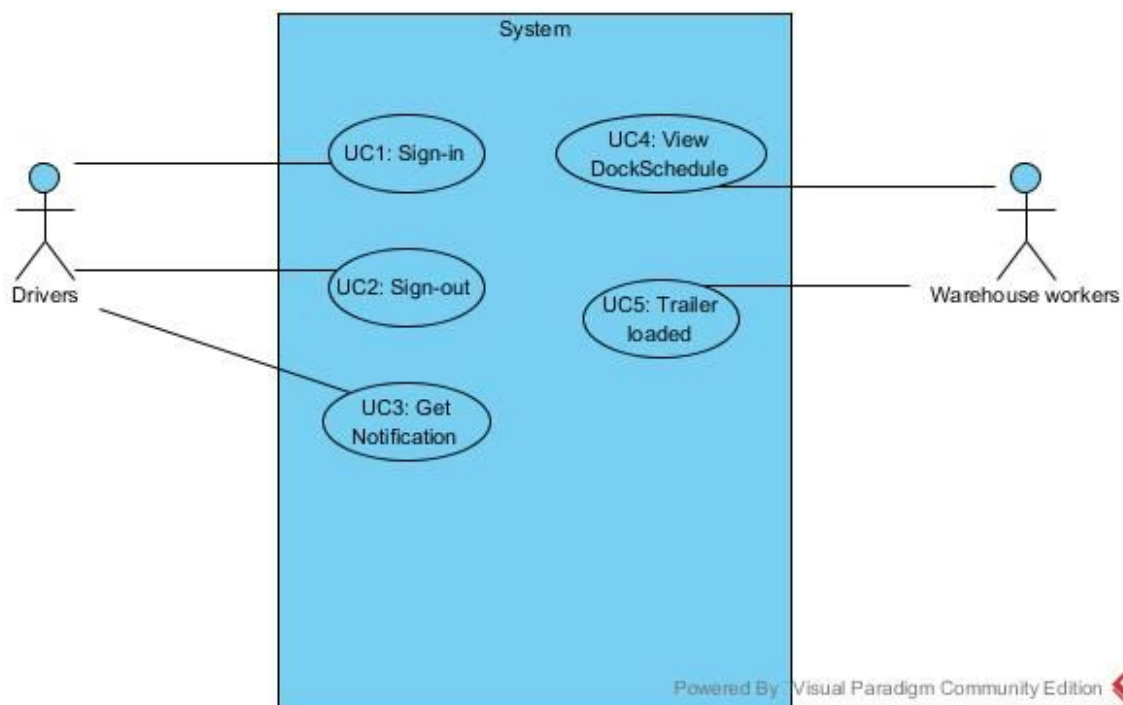
The first step of our development was to create and Requirement list, according to MoSCoW (must, should, could, would). Its purpose was to have a list of requirements, ranked by their “importance” in order to come up with the first working product.

Use case diagram

The second step of our development was to create a use case that will describe all the actors interactions with the system. After the “brainstorming” process we came up with 2 actors: Driver and Warehouse employee.

The actor is able to Sign-in/out (ref. use case descriptions) when he wishes in order to see an eventual schedule plan and to get notifications when the trailer is loaded or mistakes has been made.

A Warehouse employee is able to see the specific dock schedule and to alert the driver that the trailer is loaded, by a simple interaction with a button.



Use case description

Use case 1: Sign in

Use case name	UC01: Sign in	
Triggering event	Driver arrives with his truck into the area of DC	
Brief description	Driver arrives to DC, he signs in and his truck is weighed	
Actors	Driver	
Stakeholders	Warehouse workers	
Postconditions	The driver is signed in	
Main flow	Actor	System
	1. Driver shall choose "Driver" in the user type drop down menu.	1.1. In the main pane "Username" and "Password" fields are enabled.
	2. Driver shall choose his username from "Username" drop down menu.	2.1. System picks chosen driver "Username".
	3. Driver types his password to the password field.	3.1. System verify "Username" and "Password".
		3.2. Driver signed in to the System.
	4. Driver drives to the weighting area.	4.1. System saves trailer weight.
Exceptions	5. The truck is weighed.	5.1. Info about weight is saved to the system.
	1. Input info is incorrect.	1.1. System will pop-up an error notification "Please, type correct username and password".

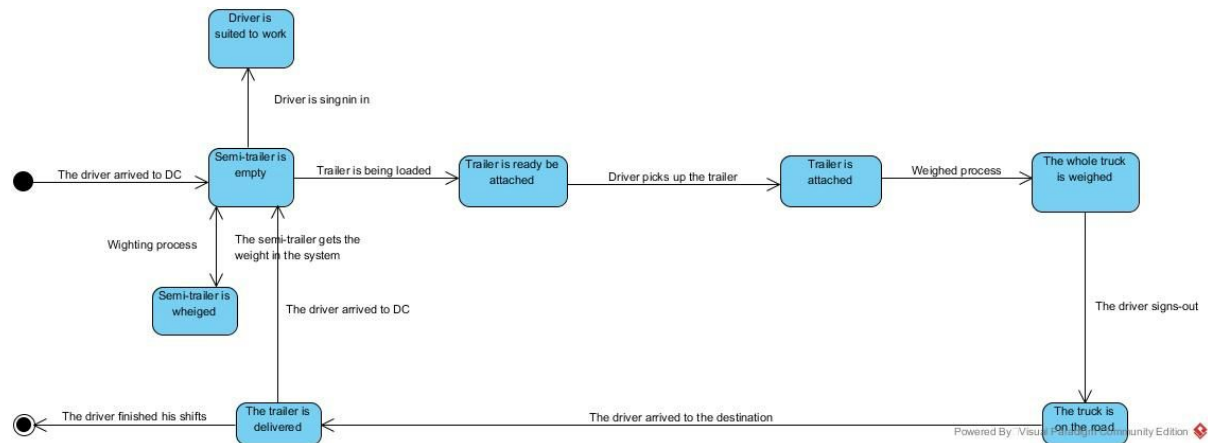
Use case 2: Sign out

Use case name	UC02: Sign out	
Triggering event	Semi-trailer is loaded and ready to deliver	
Brief description	Driver sign out before leaving the DC	
Actors	Driver	
Related use cases	UC01: Sign in; UC05: Semi-trailer loaded; UC03: Get notification	
Preconditions	The driver has signed in to the system The semi-trailer is loaded and ready to deliver	
Postconditions	The semi-trailer weighted and goods will be delivered	
Main flow	Actor	System
	1. The driver drives semi-trailer to the weighting area.	
	2. The semi-trailer is weighted.	2.1. The system assigns the weight to the semi-trailer's ID. 2.2. The weight checked by the system. 2.3. If weight is out of margins system will pop-up the message "Weight of the goods are out of margin. Please, drive back to the loading area".
	3. The driver signs out.	3.1. The system gets information about signing out.
Exceptions	1. The driver take wrong semi-trailer to deliver. 2. Expected weight of semi-trailer are not fits.	2.1. If weight is not according to the rules the system will send warning notification to the driver.

Test case ID	Test source	Initial state	system Input	Expected result	Actual result
U1-1	UC1	Start test from the main menu	Actor chooses "Driver" in the user type drop down menu.	In the main pane "Username" and "Password" fields are enabled	✓
U1-2	UC1	Start test from the main menu	Actor chooses his username from "Username" drop down menu.	System picks chosen driver "Username".	✓
U1-3	UC1	Start test from the main menu	Actor types his password to the password field.	System verify "Username" and "Password". Actor signed in to the System.	✓
U1-4	UC1	Start test from the main menu	Driver drives to the weighting area.	System saves trailer weight.	✓
U1-5	UC1	Start test from the main menu	The truck is weighed	Info about weight is saved to the system.	✓
U2-1	UC2	Start test from the main menu	Actor drives semi-trailer to the weighting area.	-	-
U2-2	UC2	Start test from the main menu	The semi-trailer is weighted	a) The system assigns the weight to the semi-trailer's ID. b) The weight checked by the system. c) If weight is out of margins system will pop-up the message "Weight of the goods are out of margin. Please, drive back to the loading area".	✓
U2-3	UC2	Start test from the main menu	The driver signs out	The system gets information about signing out.	✓

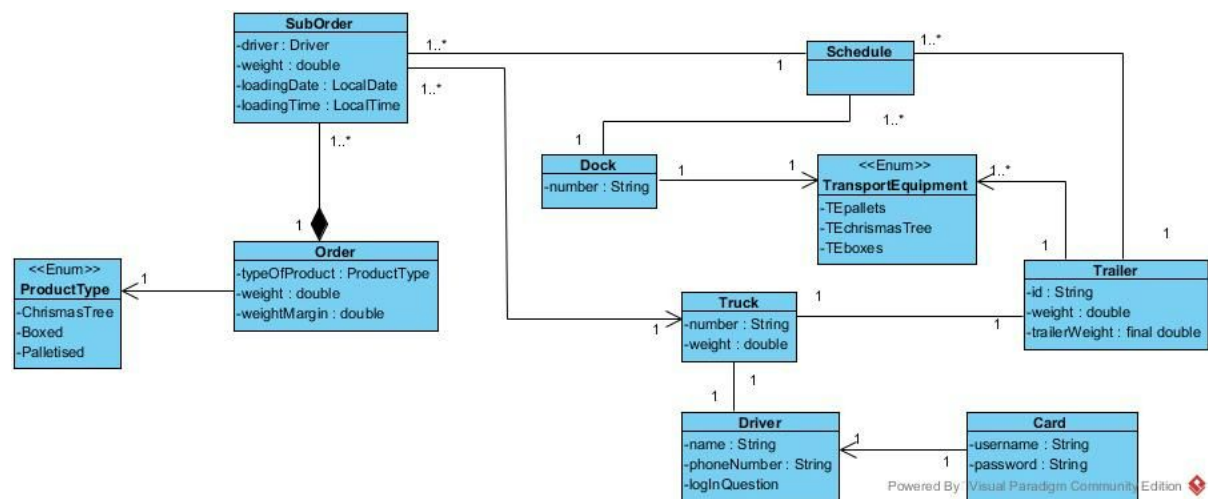
State machine diagram

Also, in our first steps we developed and state machine diagram, so that we can have a clear idea about the whole working process at the DC. It helped us a lot to understand the full functional way of getting the orders done.

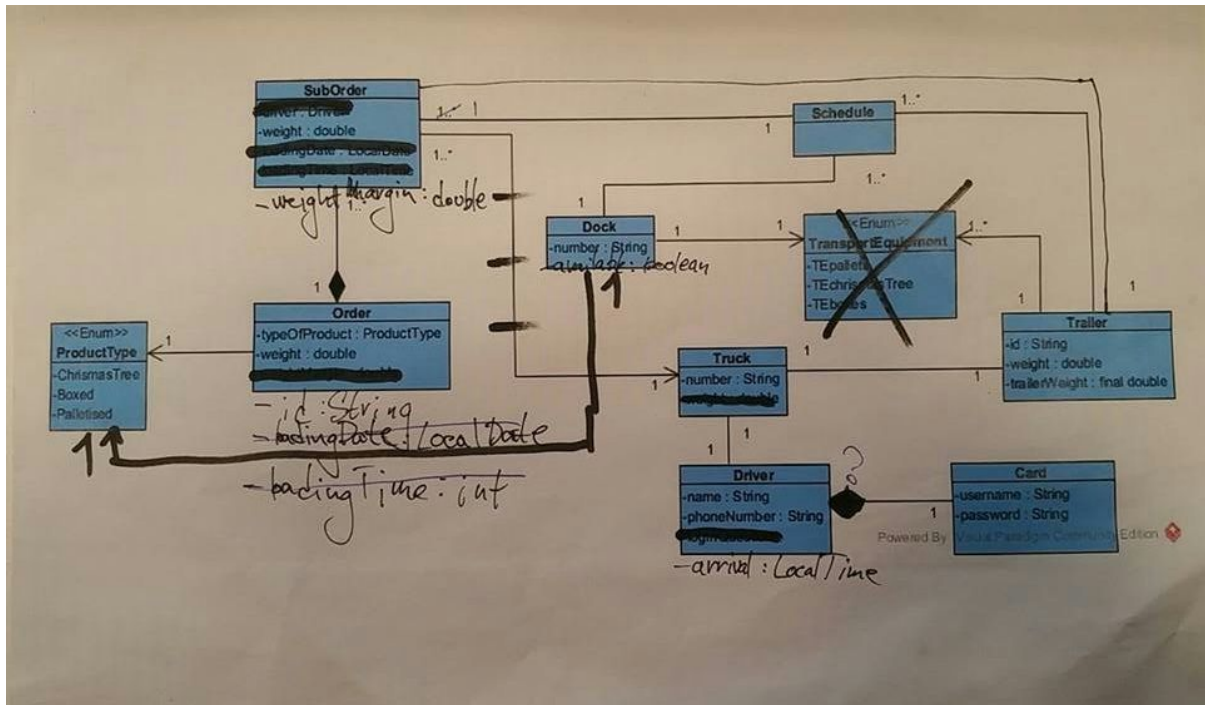


Class diagram

When the system analyses were complete, we start developing the software's architecture. After a long brainstorming process we came up with the first class diagram.



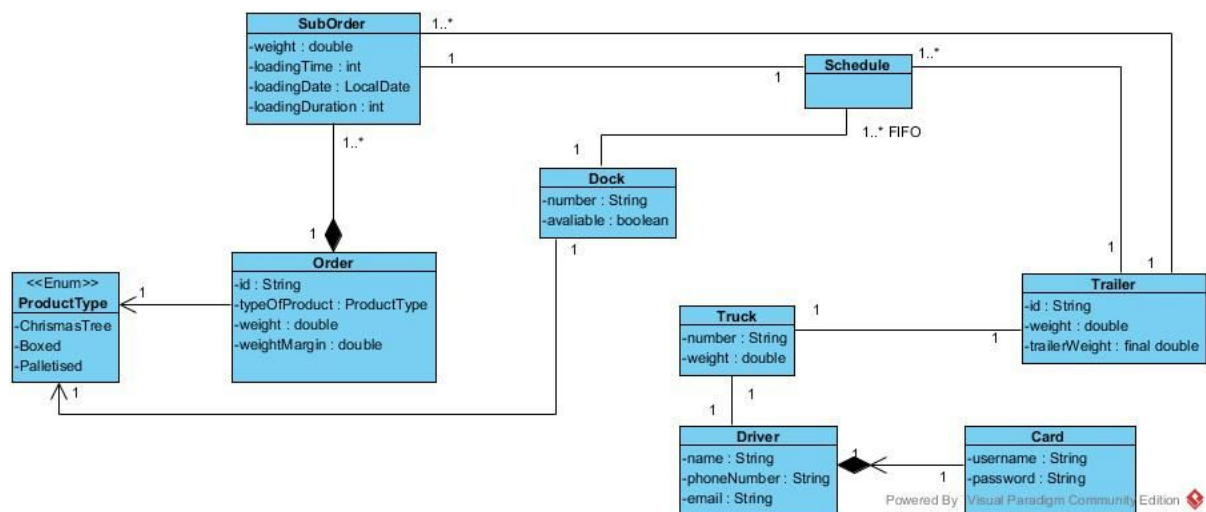
But on the next meeting, we decided to take a better look on it, just to be sure that we can start programming without wasting anymore of our time. And we actually found improvements to this first version of the diagram.



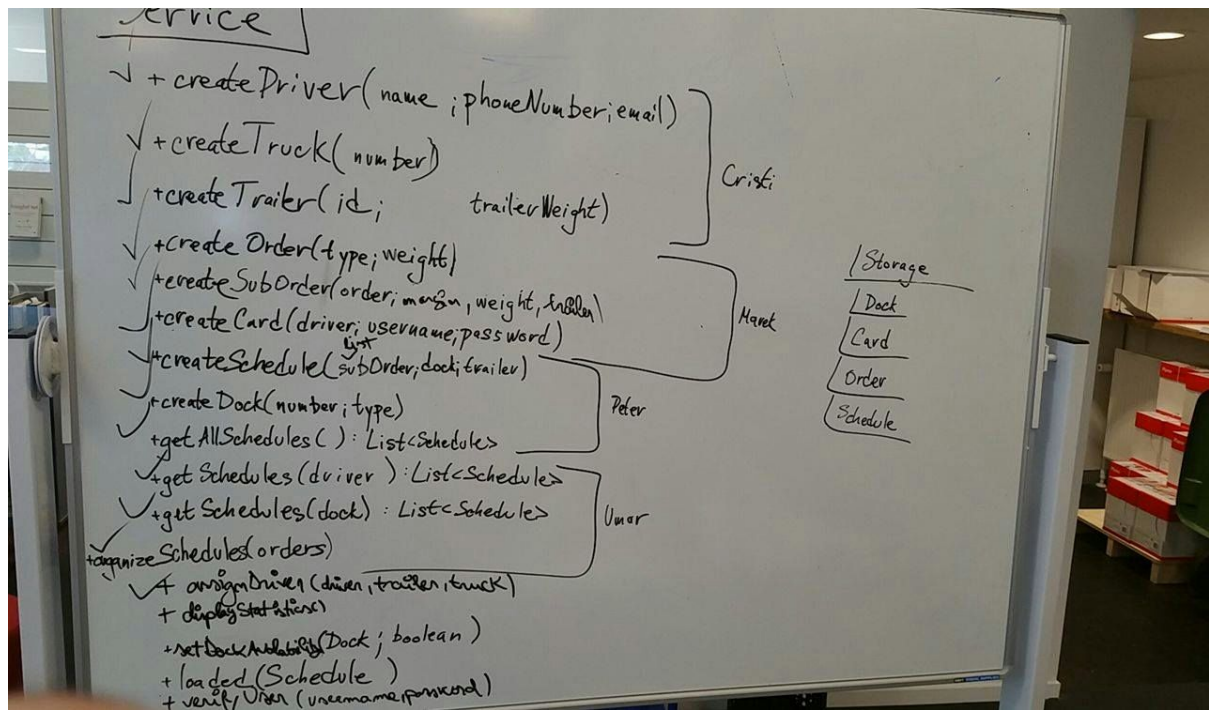
We removed the “TransportEquipment” class from our architecture because we thought it was not necessary in our system, provided we already had an enumeration of product types (In the first development brainstorm we considered that a dock can transport different product types but). We also changed the link from “Truck” to “SubOrder” into “Trailer” and “SubOrder” because it was not influencing the functionality of the system but it helps in further statistics requests.

There are changes in the attributes as well, but they will be explained in the class description. The connection from Driver to Card has been changed to Composition, meaning that the card cannot exist outside the Driver, nor can it be created.

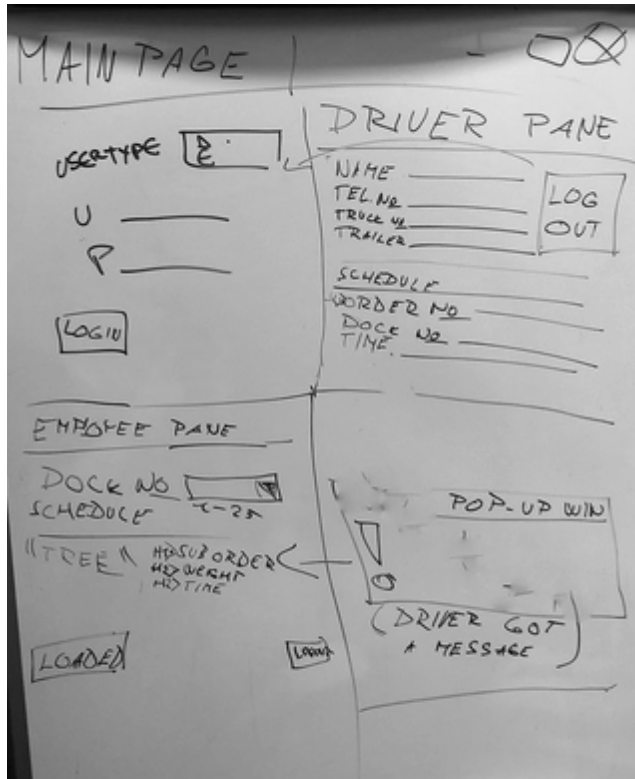
The final architecture looks like this.



The Service class was modeled on a whiteboard, at the library.



GUI description

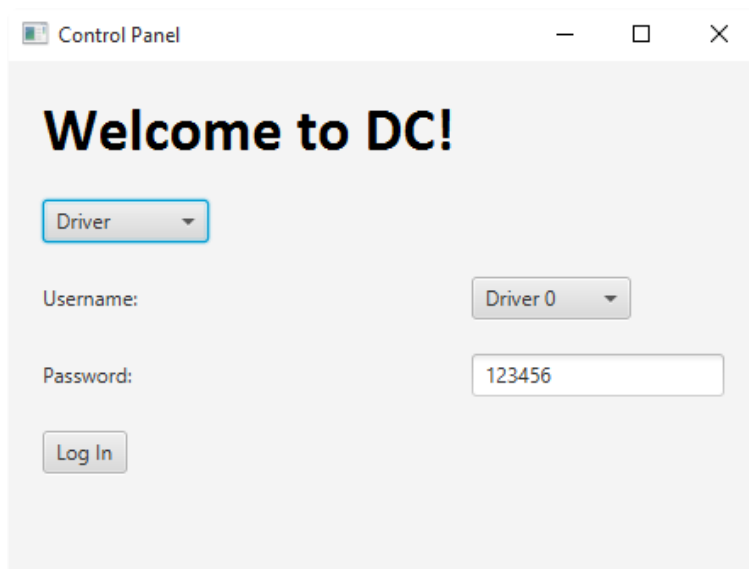


From the beginning we were designing our GUI just like terminal to see all the informations. We are not able to create any objects in our GUI. There is our first design in the picture. It is very simple. In the first window we select user either driver or warehouse worker.

Log in for the driver is simulation of auto filling of username and password done by some card. That means, that in our GUI driver can select his username in dropdown menu, password is filled automatically. When password is correct, new window with informations for driver is opened (and text "You are logged in!" is shown). If password is incorrect, text in GUI says you, that username or password is incorrect by notification: "Invalid password or username!".

If usertype Employee is selected, all fields and controls are deactivated, so you have no other choice than to press login button or select driver as user.

After pressing login button Employee pane is opened. Employee pane shows schedules for docks. There is a Loaded button that will remove the first Schedule from the view (because it is ready to be taken by the driver). Logout button will close window and "Dock broke down" button simulates inactive dock and reschedules schedules to other docks. For clarity, we attach screenshots.



Driver Pane

Name: Driver 0

Phone #: 12323340

Truck #: 101

Trailer #: 0

Weight:

Log Out

▼ Schedule

Dock num: 3

Date: 2016-04-16

Weight: 500.0kg

Loading time start: 08:00

Loading duration: 10min

Loading time end: 08:10

► Schedule

Employee Pane

Dock #: 1

Loaded

Log out

Dock broke down

▼ Schedule 5

Trailer ID: 5

▼ SubOrder No: 5

Weight: 500.0

Loading Time: 22:00

► Schedule 5

► Schedule 13

► Schedule 13

► Schedule 14

► Schedule 14

► Schedule 5

► Schedule 5

► Schedule 13

► Schedule 13

► Schedule 14

► Schedule 14

SMS been sent

Info

Driver will be informed about loading

OK

Weight!

Info

Weight is not a number. Type a proper number

OK

Class description

ProductType

The first class we are describing is “ProductType”, which is an enumeration that consists in 3 attributes: ChristmasTree, Boxed, Palletised (representing the actual DC product packaging types). The class is pretty much “isolated” in the meaning that the class doesn’t have any visibility of other classes that are linked to it.

Order

Order is a class that can contain multiple “SubOrder” and a “productType”. Its attributes are: “id, typeOfProduct, weight, weightMargin” following the example provided in the description. The link to “ProductType” is bidirectional (non nullable) and without visibility from the “ProductType” as described before. The link to “SubOrder” is Composition, meaning that the class cannot exist outside the Order, nor can it be created. If the “Order” dies, so is the “SubOrder”. A `ArrayList<SubOrder>` has been used to store the sub orders by using its interface of `List<SubOrder>`.

SubOrder

SubOrder class takes as attributes “weight, loadingTime, loadingTime”. It is also linked to Order (as described before), Schedule and Trailer. The link to schedule is set to 1, which will conclude that a SubOrder can be only in one Schedule, but the Schedule can take multiple different SubOrders. Same link with same specifications has been set to Trailer as well.

Dock

Dock contains: “number, available” as well as one ProductType to which it is linked, and a `Deque<Schedule>` to store the allocated schedules. The usage of Deque was necessary to plan the whole schedule for each of the docks. It is declared as a Deque but initialised as an `LinkedList`.

Schedule

The Schedule class it’s of high importance, for all the scheduling will be done using it. It takes multiple classes as attributes and links everything according to the demands. Each Schedule will have a SubOrder which will eventually be assigned to a dock. It is also linked to the “Trailer”, for it is in the requirements that the external company can assign a truck and a driver, and it can do so through the “Trailer’s” links.

Trailer

Trailer takes “id, weight, trailerWeight” as attributes, as well as an `List<SubOrder>` and a specific Truck that will get assigned on a Schedule. The further connections from Driver and Truck will be done through the service, by using this class.

Truck

Truck has a number and a weight and also a Driver and a Trailer objects. The links are set to 1, bidirectional.

Driver

Driver class contains a name, phoneNumber, email and arrival. It is linked as described above with the Truck and compositionally linked to the Card. A driver can have only one card that cannot exist without the Driver nor can it be created.

Card

The card class is used for sign-in/out in the system. In our system, it will be simulated by typing a username and a password (its actual attributes). In real life the process of typing username and password will be overtaken by the card usage. The link to the Driver is described above and there are no other links to it.

Test

Service.outOfMargin()

Double actualWeight	Double targetWeight	Double margin	Expected Result
100	100	10	false
110	100	10	false
90	100	10	false
50	100	10	true
150	100	10	true
0	100	10	true
100	0	10	true
100	100	0	false
-100	100	10	true
-100	-100	10	true

Service.verifyUser()

Storage	Driver1: {Benn, 908430175, benn@baaa.dk}
	Driver2: {Mikael, 568968114, mik@baaa.dk}
	CardBenn: {benn, 123456}
	CardMikael: {mikael, 123456}

String username	String password	Expected Result
"benn"	"123456"	CardBenn
"mikael"	"123456"	CardMikael
"benn"	"1241"	null
"mikel"	"123456"	null
"Benn"	"123456"	null
null	"123456"	null
"benn"	null	null

Service.organizeSchedule()

Storage	Order1: {ChristmasTree SubOrder1.1: 500 kg, 10 %, today, 10 SubOrder1.2: 500 kg, 10 %, today, 10 }
	Order2: {Boxed SubOrder2.1: 500 kg, 10 %, today, 20 SubOrder2.2: 500 kg, 10 %, today, 20 }
	Order3: {Palletized SubOrder3.1: 500 kg, 10 %, today, 30 SubOrder3.2: 500 kg, 10 %, today, 30 }
	Dock1: {Boxed}
	Dock2: {Palletized}
	Dock3: {ChristmasTree}

Expected Result

Schedule1: {SubOrder1.1, 8:00-8:10, Dock3}

Schedule2: {SubOrder1.2, 10:10-10:20, Dock3}

Schedule3: {SubOrder2.1, 8:00-8:20, Dock1}

Schedule4: {SubOrder2.2, 10:20-10:40, Dock1}

Schedule5: {SubOrder3.1, 8:00-8:30, Dock2}

Schedule6: {SubOrder3.2, 10:30-11:00, Dock2}

Programming

Decision report

We decided that for our program, the external company won't provide a Truck number and a Driver name, but instead a Trailer ID. There are also methods with Serializable that puts the Storage into a file and reads it, but it is not linked into the GUI. There is an Mail class that will send an email to the Administrator (dummy email address) in case a dock breaks down. A dock can be set to unavailable by an employee when a specific button is pressed, leading to a rescheduling its suborders. We considered that each deliver + the rest time of the driver will take 2 hours and therefore each suborder is scheduled to a minimum 2 hours apart. We also decided using cards, that will be simulated by typing a Username and Password. It was decided that each schedule should have only one Suborder. The driver will can sign-out only if he meets the specific requirements (weight margin). As part of our extensions we also choose to have a pane for the Driver, so he can see the schedules he has, at what time and at which dock. All code is in the Appendix 3.

GUI

We chose 3 parts of GUI: login, employee, driver.

On the login screen there is username and password and Employee log in. The users will be redirected into a corresponding pane. The driver pane provides schedule viewer for the driver, as well as an weight input for when signed in.

Algorithm Description

The main body of the calculations are done in the service method `organizeSchedules`. This method takes in a list of Orders as a parameter and schedules all their suborders accordingly.

The first step in accomplishing this is to get all the suborders from the orders because the schedules have links to suborders. I chose a handy, short syntax by using Java streams.

Then we need to filter the suborders that are supposed to be scheduled for today. And of course we need to filter out those that have already been scheduled.

Then I created a Dock comparator for making sure that the docks will be in order where the one that's available soonest is in front.

Then I go through every type of product. For every type we have to get the the list of docks that support the given type.

And now we just take every suborder from today that matches the type and assign it to the first dock in the sorted list (so the dock that's available the soonest).

Conclusion

In conclusion, we have a fully functional software, probably of version 1.0, that can get external suborders, organise and display them in 2 panes (one for driver and one for employee). We respected the requirements of type Must: Get list of schedules, send notification SMS (simulated by a warning), Display weight out of margin, Driver sign in, Driver sign out, Organisation of schedules and a GUI for the system. We also did the Should ones: Rescheduling of trailers and Re-notifying the driver when something is wrong. We did not do some of the others extensions, for it was decided that we should focus on one good, functional system. A weakness of our system can be the time displayed for the scheduling. We decided to use a 2 hours delay for each driver in the scheduling part, and therefore some of them may have a big time gap. For example, we took some tests and a single driver had a schedule at 09:00 and another one at 20:00). The gui present no signs of errors.

The whole team concluded that this project was helpful to us to understand the whole development process, how to divide tasks and how to manage time. We also earned a lot of experience from it and we can know how to exceed some of the difficulties we faced.

DAOS

Description

MSSQL description

Exercise 1

Our domain class diagram from SD was very database friendly so we didn't need any significant modifications to make it into tables. For the sake of keeping things simple we decided to not utilize time functionality.

As for transferring programming constructs into DB we used a constrained varchar as an enumeration of product types. There are other possibilities for doing this. One of them would be to create a table of different product types and reference them using foreign keys. That approach would be necessary in a bigger projects that have to deal with more demanding and dynamic requirements. But in order to satisfy the requirements of this project it wasn't necessary.

In our class diagram there was no inheritance used. One-to-one associations were not condensed into a single table because it doesn't always make sense and the data model allowed us to retrieve the data without using a lot of joins anyway.

We started our DAOS part of the project by creating the tables using simple "CREATE TABLE" statements.

```
create table Truck(  
  
number varchar(255) PRIMARY KEY not null,  
truckWeight decimal not null)
```

Before initialising the database, we altered the tables in order to put foreign keys in, so that the links in the diagram are realized.

```
alter TABLE Schedule ADD foreign key(dock_number)  
  
references Dock(number)
```

Right after that we initialised our database using simple fields.

```
INSERT INTO Truck VALUES ('1',1000)  
INSERT INTO Truck VALUES ('2',1000)  
INSERT INTO Truck VALUES ('3',1000)  
INSERT INTO Truck VALUES ('4',1000)
```

This procedure can be called using "EXEC getTrucks".

Exercise 2

The exercise 2 will be mainly done with the usage of our already existing code for the programming.

For exercise 2.a we can use, for example our method called organizeSchedule, and whenever they are placed in an order you just call the procedure createSchedule that will store the specific data.

In exercise 2.b we used a stored procedure

```
create procedure showDriverLoading @day DATE as

BEGIN
select so.loadingTime,d.name
from SubOrder so join Trailer tra on so.trailer_id=tra.id
  join Truck t on tra.truck_number=t.number
  join Driver d on t.number=d.truck_number where so.loadingDate=@day;
END

exec showDriverLoading '2016/01/01'
```

Exercise 3

After we finished the creation and initialisation of the database we were able to create the first procedure that will return a required list as for exercise 3 in the paper.

3.a

```
select count(*),productType from Dock dock group by dock.productType
```

3.b

```
CREATE PROCEDURE getTrailersAtDate @date date AS
BEGIN
  SELECT t.* from Schedule s join SubOrder so on s.subOrder_id=so.id join Trailer t on
s.trailer_id=t.id where so.loadingDate=@date;
END

EXEC getTrailersAtDate '2016-01-01'
```

We chose to use a procedure so that we can call it each time we want to with ease, by just typing the specific date it's required. We joined the 3 tables in order to get to the number of docks that can load on the given date. The joins respects the links from our diagram.

Exercise 4

We also created a procedure, regarding to exercise 4, that will create records for the tables and we displayed them as soon as created by using a trigger.

```
create procedure createTruck @number varchar(255), @truckWeight decimal as
BEGIN
INSERT INTO Truck VALUES (@number,@truckWeight)
END

create trigger truck_trigger on Truck AFTER INSERT AS
BEGIN
DECLARE @number as varchar(255)
DECLARE @truckWeight as decimal
SELECT @number=number FROM inserted
SELECT @truckWeight=truckWeight FROM inserted
PRINT @number
PRINT @truckWeight
END
```

The next step was to create some procedures that will return the tables.

```
create procedure getTrucks AS  
BEGIN  
SELECT * from Truck;  
END
```

JAVA description

We choose to display things and work only with the console, so there will not be any GUI used in our project regarding the databases.

The connection to the database by a simple connection lines code as the following.

```
String url = "jdbc:sqlserver://localhost\\SQLEXPRESS;databaseName=Project";
Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
con = DriverManager.getConnection(url, "sa", "123456");
```

Printing the tables is done easily by using a statement and a result set.

```
public static void printTrucks() throws SQLException {
    Statement s = con.createStatement();
    s.executeQuery("EXEC getTrucks");
    ResultSet rs = s.getResultSet();

    while (rs.next()) {
        System.out.print(rs.getString("number") + " ");
        System.out.println(rs.getDouble("truckWeight"));
    }
}
```

For adding something to the database table we use a PreparedStatement, for security purposes.

```
public static void createTruck(String number, Double truckWeight) throws SQLException {
    PreparedStatement ps = con.prepareStatement("Exec createTruck ?,?");
    ps.setString(1, number);
    ps.setDouble(2, truckWeight);
    ps.executeUpdate();
}
```

Conclusion

In conclusion, the connection to the database works. All the queries works and they respect the rules told and required.

Sources

1. Danish Crown AmbA, [n.d.]. *Welcome to Danish Crown - Danish Crown*. [online] Available through: <http://www.danishcrown.com> [downloaded 16. 3. 2016]
2. Rosenberg D. & Stephens M., 2007. *Use Case Driven Object Modeling with UML: Theory and Practice*. Berkeley, CA: Apress, 2007. Print.
3. NERIS Analytics Limited, 2011. *Free personality test, type descriptions, relationship and career advice | 16Personalities*. [online] Available through: <https://www.16personalities.com> [downloaded 18. 3. 2016]
4. Robbins, S. P. & Judge T. A., 2015. *Organizational behaviour*. 16th global edition. Harlow: Pearson Education Ltd. Print.

Appendix 1

Group contract

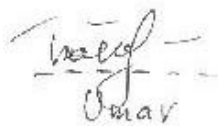
CONTRACT
PROJECT AGREEMENT
GROUP 1

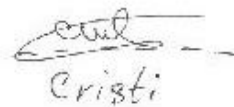
Weekdays : Mon, Wed, Fri

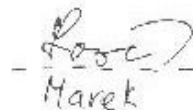
Time : 9:00am

If you break the rules you have to bring something to eat for the others.


Peter M


Omar


Cristi


Marek

Appendix 2

Databases

MSSQL code

```
create table Card(
username varchar(255) PRIMARY KEY not null,
pass varchar(255) not null,
driver_id int not null)

create table Driver(
id int not null IDENTITY(1,1) PRIMARY KEY,
name varchar(255) not null,
phoneNumber varchar(255) not null,
email varchar(255),
truck_number varchar(255) not null
)

create table Truck(
number varchar(255) PRIMARY KEY not null,
truckWeight decimal not null
)

create table Trailer(
id varchar(255) PRIMARY KEY not null,
emptyTrailerWeight decimal not null,
trailerWeight decimal not null,
truck_number varchar(255) not null,
)

create table Schedule(
id varchar(255) PRIMARY KEY not null,
dock_number varchar(255) not null,
trailer_id varchar(255) not null,
subOrder_id varchar(255) not null,)

create table Dock(
number varchar(255) PRIMARY KEY not null,
productType varchar(255) CHECK(productType IN('ChristmasTree','Palletised','Boxed')) not
null,
available bit not null)

create table SubOrder(
id varchar(255) PRIMARY KEY not null,
suborderWeight decimal not null,
loadingTime int not null,
loadingDate date not null,
trailer_id varchar(255) not null,
order_id varchar(255) not null)

create table ExternalOrder(
id varchar(255) PRIMARY KEY not null,
productType varchar(255) CHECK(productType IN('ChristmasTree','Palletised','Boxed')) not
null,
externalOrderWeight decimal not null,
weightMargin decimal not null)

alter table Card ADD foreign key (driver_id)
references Driver(id) on delete cascade

alter table Driver ADD foreign key(truck_number)
references Truck(number)

alter table Trailer ADD foreign key(truck_number)
```

```

references Truck(number)

alter table SubOrder ADD foreign key(order_id)
references ExternalOrder(id)

alter table SubOrder ADD foreign key(trailer_id)
references Trailer(id)

alter table Schedule ADD foreign key(dock_number)
references Dock(number)

alter table Schedule ADD foreign key(subOrder_id)
references SubOrder(id)

alter table Schedule ADD foreign key(trailer_id)
references Trailer(id)

```

```

INSERT INTO Card VALUES ('eric', '123456', '21')
INSERT INTO Card VALUES ('eric2', '123456', '22')
INSERT INTO Card VALUES ('eric3', '123456', '23')
INSERT INTO Card VALUES ('eric4', '123456', '4')

INSERT INTO Driver VALUES ('Eric Jacobsen', '0000000', 'eric@eric.eric', '1')
INSERT INTO Driver VALUES ('Eric Jacobsen2', '0000000', 'eric@eric.eric', '2')
INSERT INTO Driver VALUES ('Eric Jacobsen3', '0000000', 'eric@eric.eric', '3')
INSERT INTO Driver VALUES ('Eric Jacobsen4', '0000000', 'eric@eric.eric', '4')

INSERT INTO Truck VALUES ('1', 1000)
INSERT INTO Truck VALUES ('2', 1000)
INSERT INTO Truck VALUES ('3', 1000)
INSERT INTO Truck VALUES ('4', 1000)

INSERT INTO Trailer VALUES ('1', 200, 2500, '1')
INSERT INTO Trailer VALUES ('2', 250, 5500, '2')
INSERT INTO Trailer VALUES ('3', 150, 4500, '3')
INSERT INTO Trailer VALUES ('4', 220, 3500, '4')
INSERT INTO Trailer VALUES ('5', 200, 2500, '1')
INSERT INTO Trailer VALUES ('6', 250, 5500, '2')
INSERT INTO Trailer VALUES ('7', 150, 4500, '3')
INSERT INTO Trailer VALUES ('8', 220, 3500, '4')

INSERT INTO Dock VALUES ('1', 'ChristmasTree', 1)
INSERT INTO Dock VALUES ('2', 'Palletised', 1)
INSERT INTO Dock VALUES ('3', 'ChristmasTree', 0)
INSERT INTO Dock VALUES ('4', 'Boxed', 1)

INSERT INTO ExternalOrder VALUES ('1', 'ChristmasTree', 1000, 6)
INSERT INTO ExternalOrder VALUES ('2', 'ChristmasTree', 400, 4)
INSERT INTO ExternalOrder VALUES ('3', 'Palletised', 7000, 2)
INSERT INTO ExternalOrder VALUES ('4', 'Boxed', 4000, 1)

INSERT INTO SubOrder VALUES ('1', 500, 20, '2016', 1, 1)
INSERT INTO SubOrder VALUES ('2', 500, 20, '2016', 2, 1)
INSERT INTO SubOrder VALUES ('3', 300, 20, '2016', 3, 2)
INSERT INTO SubOrder VALUES ('4', 100, 20, '2016', 4, 2)
INSERT INTO SubOrder VALUES ('5', 5000, 20, '2016', 5, 3)
INSERT INTO SubOrder VALUES ('6', 2000, 20, '2016', 6, 3)
INSERT INTO SubOrder VALUES ('7', 3500, 20, '2016', 7, 4)
INSERT INTO SubOrder VALUES ('8', 500, 20, '2016', 8, 4)

INSERT INTO Schedule VALUES ('1', '1', '1', '1')
INSERT INTO Schedule VALUES ('2', '2', '2', '2')
INSERT INTO Schedule VALUES ('3', '3', '3', '3')
INSERT INTO Schedule VALUES ('4', '4', '4', '4')

```

```

create procedure getCards AS
BEGIN
select * from Card;
END

create procedure getDocks AS

```

```

BEGIN
select * from Dock;
END

create procedure getDrivers AS
BEGIN
select * from Driver;
END

create procedure getOrders AS
BEGIN
select * from ExternalOrder;
END

create procedure getSchedules AS
BEGIN
select * from Schedule;
END

create procedure getSubOrders AS
BEGIN
select * from SubOrder;
END

create procedure getTrailers AS
BEGIN
select * from Trailer;
END

create procedure getTrucks AS
BEGIN
select * from Truck;
END

Exec getCards
Exec getDocks
Exec getDrivers
Exec getOrders
Exec getSchedules
Exec getSubOrders
Exec getTrailers
Exec getTrucks

```

```

CREATE PROCEDURE getTrailersAtDate @date date AS
BEGIN
    select t.* from Schedule s join SubOrder so on s.subOrder_id=so.id join Trailer t on
s.trailer_id=t.id where so.loadingDate=@date;
END

EXEC getTrailersAtDate '2016'

```

```

create procedure createTruck @number varchar(255), @truckWeight decimal as
BEGIN
INSERT INTO Truck VALUES (@number,@truckWeight)
END

```

```

create trigger truck_trigger on Truck AFTER INSERT AS
BEGIN
DECLARE @number as varchar(255)
DECLARE @truckWeight as decimal
SELECT @number=number FROM inserted
SELECT @truckWeight=truckWeight FROM inserted
PRINT @number
PRINT @truckWeight
END

```

Java code

```
package ex1;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class MainApp {
    private static Connection con = null;

    public static void main(String[] args) {
        try {

            String url = "jdbc:sqlserver://localhost\\SQLEXPRESS;databaseName=Project";
            Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
            con = DriverManager.getConnection(url, "sa", "123456");
            printCards();

            System.out.println("-----");
            printDocks();

            System.out.println("-----");
            printDrivers();

            System.out.println("-----");
            printOrders();

            System.out.println("-----");
            printSubOrders();

            System.out.println("-----");
            printSchedules();

            System.out.println("-----");
            printTrailers();

            System.out.println("-----");
            printTrucks();
        } catch (Exception e) {
            e.printStackTrace();
        }

        public static void createTruck(String number, Double truckWeight) throws SQLException {
            PreparedStatement ps = con.prepareStatement("Exec createTruck ?,?");
            ps.setString(1, number);
            ps.setDouble(2, truckWeight);
            ps.executeUpdate();
        }

        public static void printTrucks() throws SQLException {
            Statement s = con.createStatement();
            s.executeQuery("EXEC getTrucks");
            ResultSet rs = s.getResultSet();

            while (rs.next()) {
                System.out.print(rs.getString("number") + " ");
                System.out.println(rs.getDouble("truckWeight"));
            }
        }

        public static void printDrivers() throws SQLException {
            Statement s = con.createStatement();
            s.executeQuery("EXEC getDrivers");
            ResultSet rs = s.getResultSet();
        }
    }
}
```

```

        while (rs.next()) {
            System.out.print(rs.getInt("id") + " ");
            System.out.print(rs.getString("name") + " ");
            System.out.print(rs.getString("phoneNumber") + " ");
            System.out.println(rs.getString("email") + " ");
        }
    }

    public static void printTrailers() throws SQLException {
        Statement s = con.createStatement();
        s.executeQuery("EXEC getTrailers");
        ResultSet rs = s.getResultSet();

        while (rs.next()) {
            System.out.print(rs.getString("id") + " ");
            System.out.print(rs.getDouble("emptyTrailerWeight") + " ");
            System.out.println(rs.getDouble("trailerWeight") + " ");
        }
    }

    public static void printOrders() throws SQLException {
        Statement s = con.createStatement();
        s.executeQuery("EXEC getOrders");
        ResultSet rs = s.getResultSet();

        while (rs.next()) {
            System.out.print(rs.getString("id") + " ");
            System.out.print(rs.getString("productType") + " ");
            System.out.print(rs.getDouble("externalOrderWeight") + " ");
            System.out.println(rs.getDouble("weightMargin"));
        }
    }

    public static void printSchedules() throws SQLException {
        Statement s = con.createStatement();
        s.executeQuery("EXEC getSchedules");
        ResultSet rs = s.getResultSet();

        while (rs.next()) {
            System.out.print(rs.getString("id") + " ");
            System.out.println(rs.getString("dock_number") + " ");
        }
    }

    public static void printSubOrders() throws SQLException{
        Statement s = con.createStatement();
        s.executeQuery("EXEC getSubOrders");
        ResultSet rs = s.getResultSet();

        while (rs.next()) {
            System.out.print(rs.getString("id") + " ");
            System.out.print(rs.getDouble("suborderWeight") + " ");
            System.out.print(rs.getInt("loadingTime") + " ");
            System.out.println(rs.getDate("loadingDate") + " ");
        }
    }

    public static void printDocks() throws SQLException {
        Statement s = con.createStatement();
        s.executeQuery("EXEC getDocks");
        ResultSet rs = s.getResultSet();

        while (rs.next()) {
            System.out.print(rs.getString("number") + " ");
            System.out.print(rs.getString("productType") + " ");
            if (rs.getInt("available") == 1) {
                System.out.println("available");
            } else {
                System.out.println("unavailable");
            }
        }
    }

```

```
    }

    public static void printCards() throws SQLException {
        Statement s = con.createStatement();
        s.executeQuery("EXEC getCards");
        ResultSet rs = s.getResultSet();

        while (rs.next()) {
            System.out.print(rs.getString("username") + " ");
            System.out.println(rs.getString("pass"));
        }
    }
}
```

Appendix 3

Java code

Table of contents

[Model](#)

[Card Class](#)

[Dock Class](#)

[Driver Class](#)

[Order Class](#)

[ProductType Enumeration](#)

[Schedule Class](#)

[SubOrder Class](#)

[Trailer Class](#)

[Truck Class](#)

[Service](#)

[Service Class](#)

[Mail Class](#)

[Storage](#)

[Storage Class](#)

[Test](#)

[ServiceTest JUnit Test](#)

[GUI](#)

[MainApp Class](#)

[DriverPane Class](#)

[EmployeePane Class](#)

[TreeViewService Class](#)

Model

Card Class

```
package model;

import java.io.Serializable;

public class Card implements Serializable {

    private static final long serialVersionUID = -7013923319001924204L;
    private String      username;
    private String      password;

    // Link fields
    private Driver      driver;

    public Card(Driver driver, String username, String password) {
        this.driver = driver;
        this.username = username;
        this.password = password;
    }

    public String getUsername() {
        return username;
    }

    public String getPassword() {
        return password;
    }

    public Driver getDriver() {
        return driver;
    }

    @Override
    public String toString() {
        return username;
    }
}
```

Dock Class

```
package model;

import java.io.Serializable;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.Deque;
import java.util.LinkedList;
import java.util.List;

public class Dock implements Serializable {

    private static final long serialVersionUID = -8404423780682109357L;
    private String          number;
    private ProductType     productType;
    private boolean         available;

    // Link fields
    private Deque<Schedule> schedules      = new LinkedList<>();

    private static LocalDateTime startTime = LocalDateTime.of(8, 0);

    public Dock(String number, ProductType productType) {
        this.number = number;
        this.productType = productType;
    }

    public String getNumber() {
        return number;
    }

    public void setNumber(String number) {
        this.number = number;
    }

    public ProductType getProductType() {
        return productType;
    }

    public void setTransportEquipment(ProductType transportEquipment) {
        this.productType = transportEquipment;
    }

    public List<Schedule> getSchedules() {
        return new ArrayList<Schedule>(schedules);
    }

    public void addSchedule(Schedule schedule) {
        schedule.getSubOrder().setLoadingTime(nextAvailableTime());
        schedules.addLast(schedule);
    }

    public void removeSchedule(Schedule schedule) {
        schedules.remove(schedule);
    }

    public LocalDateTime nextAvailableTime() {
        int totalMinutes = 0;

        for (Schedule s : schedules) {
            totalMinutes += s.getSubOrder().getLoadingDuration() + 120;
        }

        return startTime.plusMinutes(totalMinutes);
    }

    public boolean isAvailable() {
        return available;
    }
}
```

```
public void setAvailable(boolean available) {
    this.available = available;
    if (available == false) {
        while (schedules.size() > 0) {
            schedules.pop().clear();
        }
    }
}

@Override
public String toString() {
    return number;
}
}
```

Driver Class

```
package model;

import java.io.Serializable;

public class Driver implements Serializable {

    private static final long serialVersionUID = 1432176189793004629L;
    private String      name;
    private String      phoneNumber;
    private String      email;

    // Link fields
    private Truck        truck;

    public Driver(String name, String phoneNumber, String email) {
        this.name = name;
        this.phoneNumber = phoneNumber;
        this.email = email;
    }

    public String getName() {
        return name;
    }

    public String getPhoneNumber() {
        return phoneNumber;
    }

    public Truck getTruck() {
        return truck;
    }

    public void setTruck(Truck truck) {
        this.truck = truck;
    }

    public String getEmail() {
        return email;
    }

    public Card createCard(String username, String password) {
        Card card = new Card(this, username, password);
        return card;
    }

    @Override
    public String toString() {
        return name;
    }
}
```

Order Class

```
package model;

import java.io.Serializable;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;

public class Order implements Serializable {

    private static final long serialVersionUID = 6617985724254451386L;
    private String id;
    private double weight;

    // Link fields
    private ProductType typeOfProduct;
    private List<SubOrder> subOrders = new ArrayList<>();

    public Order(String id, ProductType typeOfProduct, double weight) {
        this.id = id;
        this.typeOfProduct = typeOfProduct;
        this.weight = weight;
    }

    public ProductType getTypeOfProduct() {
        return typeOfProduct;
    }

    public void setTypeOfProduct(ProductType typeOfProduct) {
        this.typeOfProduct = typeOfProduct;
    }

    public double getWeight() {
        return weight;
    }

    public void setWeight(double weight) {
        this.weight = weight;
    }

    public List<SubOrder> getSubOrders() {
        return new ArrayList<>(subOrders);
    }

    public SubOrder createSubOrder(Trailer trailer, double weight, double weightMargin,
        LocalDate loadingDate,
        int loadingTime) {
        SubOrder subOrder = new SubOrder(trailer, weight, weightMargin, loadingDate,
        loadingTime);
        subOrder.setOrder(this);
        subOrders.add(subOrder);
        return subOrder;
    }

    public void deleteSubOrder(SubOrder subOrder) {
        subOrder.setOrder(null);
        subOrders.remove(subOrder);
    }

    public String getId() {
        return id;
    }

    @Override
    public String toString() {
        return "Order ID " + id;
    }
}
```

ProductType Enumeration

```
package model;

public enum ProductType {
    ChristmasTree, Boxed, Palletised;
}
```

Schedule Class

```
package model;

import java.io.Serializable;

public class Schedule implements Serializable {

    private static final long serialVersionUID = -43279402907064814L;
    private SubOrder      subOrder;
    private Dock          dock;
    private Trailer       trailer;

    public Schedule(Dock dock, Trailer trailer, SubOrder suborder) {
        this.dock = dock;
        this.trailer = trailer;
        this.subOrder = suborder;
        trailer.addSchedule(this);
        dock.addSchedule(this);
    }

    public SubOrder getSubOrder() {
        return subOrder;
    }

    public Dock getDock() {
        return dock;
    }

    public void setDock(Dock dock) {
        this.dock = dock;
    }

    public Trailer getTrailer() {
        return trailer;
    }

    public void setTrailer(Trailer trailer) {
        this.trailer = trailer;
    }

    public void clear() {
        subOrder.setSchedule(null);
        dock.removeSchedule(this);
        trailer.removeSchedule(this);
        subOrder = null;
        dock = null;
        trailer = null;
    }
}
```

SubOrder Class

```
package model;

import java.io.Serializable;
import java.time.LocalDate;
import java.time.LocalTime;

public class SubOrder implements Serializable {

    private static final long serialVersionUID = -8127897371374870895L;
    private double weight;
    private double weightMargin;
    private LocalDate loadingDate;
    private int loadingDuration;
    private LocalTime loadingTime;

    // Link fields
    private Order order;
    private Trailer trailer;
    private Schedule schedule;

    public SubOrder(Trailer trailer, double weight, double weightMargin, LocalDate loadingDate, int loadingDuration) {
        this.trailer = trailer;
        this.weight = weight;
        this.weightMargin = weightMargin;
        this.loadingDate = loadingDate;
        setLoadingDuration(loadingDuration);
    }

    public double getWeight() {
        return weight;
    }

    public void setWeight(double weight) {
        this.weight = weight;
    }

    public Trailer getTrailer() {
        return trailer;
    }

    public void setTruck(Trailer trailer) {
        this.trailer = trailer;
    }

    public Order getOrder() {
        return order;
    }

    public void setOrder(Order order) {
        this.order = order;
    }

    public Schedule getSchedule() {
        return schedule;
    }

    public void setSchedule(Schedule schedule) {
        this.schedule = schedule;
    }

    public double getWeightMargin() {
        return weightMargin;
    }

    public void setWeightMargin(double weightMargin) {
        this.weightMargin = weightMargin;
    }
}
```



```
public LocalDate getLoadingDate() {  
    return loadingDate;  
}  
  
public LocalTime getLoadingTime() {  
    return loadingTime;  
}  
  
public void setLoadingTime(LocalTime time) {  
    this.loadingTime = time;  
}  
  
public int getLoadingDuration() {  
    return loadingDuration;  
}  
  
public void setLoadingDuration(int loadingDuration) {  
    this.loadingDuration = loadingDuration;  
}  
  
public LocalTime getLoadingEndTime() {  
    return this.loadingTime.plusMinutes(this.loadingDuration);  
}  
}
```

Trailer Class

```
package model;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

public class Trailer implements Serializable {

    private static final long serialVersionUID = -5673100625618639835L;
    private String id;
    private double weight;
    private double trailerWeight;

    // Link fields
    private Truck truck;
    private List<ProductType> equipment = new ArrayList<>();
    private List<Schedule> schedule = new ArrayList<>();

    public Trailer(String id, double trailerWeight) {
        this.id = id;
        this.trailerWeight = trailerWeight;
    }

    public double getWeight() {
        return weight;
    }

    public void setWeight(double weight) {
        this.weight = weight;
    }

    public Truck getTruck() {
        return truck;
    }

    public void setTruck(Truck truck) {
        this.truck = truck;
    }

    public String getId() {
        return id;
    }

    public double getTrailerWeight() {
        return trailerWeight;
    }

    public List<ProductType> getEquipment() {
        return new ArrayList<>(equipment);
    }

    public void addEquipment(ProductType equipment) {
        this.equipment.add(equipment);
    }

    public List<Schedule> getSchedules() {
        return new ArrayList<>(schedule);
    }

    public void addSchedule(Schedule schedule) {
        this.schedule.add(schedule);
    }

    public void removeSchedule(Schedule schedule) {
        this.schedule.remove(schedule);
    }
}
```

Truck Class

```
package model;

import java.io.Serializable;

public class Truck implements Serializable {

    private static final long serialVersionUID = 8391654626870656755L;
    private String      number;

    // Link fields
    private Driver      driver;
    private Trailer     trailer;

    public Truck(String number) {
        this.number = number;
    }

    public Driver getDriver() {
        return driver;
    }

    public void setDriver(Driver driver) {
        this.driver = driver;
    }

    public Trailer getTrailer() {
        return trailer;
    }

    public void setTrailer(Trailer trailer) {
        this.trailer = trailer;
    }

    public String getNumber() {
        return number;
    }

}
```

Service

Service Class

```
package service;

import java.io.EOFException;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Comparator;
import java.util.List;
import java.util.Random;
import java.util.stream.Collectors;

import model.Card;
import model.Dock;
import model.Driver;
import model.Order;
import model.ProductType;
import model.Schedule;
import model.SubOrder;
import model.Trailer;
import model.Truck;
import storage.Storage;

public class Service {

    public static List<Schedule> getSchedules(Driver d) {
        assert d != null;
        return d.getTruck().getTrailer().getSchedules();
    }

    public static List<Schedule> getSchedules(Dock d) {
        assert d != null;
        return d.getSchedules();
    }

    public static Schedule createSchedule(SubOrder subOrder, Dock dock, Trailer trailer) {
        assert subOrder != null;
        assert dock != null;
        assert trailer != null;
        Schedule schedule = new Schedule(dock, trailer, subOrder);
        Storage.saveSchedule(schedule);
        return schedule;
    }

    public static Driver createDriver(String name, String phoneNumber, String email) {
        assert name != null;
        assert phoneNumber != null;
        assert email != null;
        Driver driver = new Driver(name, phoneNumber, email);
        return driver;
    }

    public static Truck createTruck(String number) {
        assert number != null;
        Truck truck = new Truck(number);
        return truck;
    }

    public static Trailer createTrailer(String id, double trailerWeight) {
        assert id != null;
        assert trailerWeight < 0;
        Trailer trailer = new Trailer(id, trailerWeight);
    }
}
```

```

        return trailer;
    }

    /*
     * calls the other main organizeSchedule method
     */
    public static void organizeSchedules() {
        Service.organizeSchedules(Storage.getOrders());
    }

    /*
     * collects all the suborders from all the orders filters the suborders that
     * are from today and that are not scheduled check every type of the docks
     * for every type it gets the dock that are of that type schedules them 1 by
     * 1
     */
    public static void organizeSchedules(List<Order> orders) {
        // Get all suborders
        List<SubOrder> subOrders = new ArrayList<>();
        orders.stream().forEach(order -> subOrders.addAll(order.getSubOrders()));

        // Filter orders for today
        List<SubOrder> today = subOrders.stream().filter(s ->
s.getLoadingDate().equals(LocalDate.now()))
            .filter(s -> s.getSchedule() == null).collect(Collectors.toList());

        List<ProductType> types = Arrays.asList(ProductType.values());

        Comparator<Dock> dockComparator = new Comparator<Dock>() {
            @Override
            public int compare(Dock d1, Dock d2) {
                return d1.nextAvailableTime().compareTo(d2.nextAvailableTime());
            }
        };

        // For every type
        types.stream().forEach(type -> {

            // Get all the appropriate docks
            List<Dock> docks = Storage.getDocks().stream().filter(d -> d.getProductType()
== type)
                .collect(Collectors.toList());

            // Get all suborders of the specific type
            today.stream().filter(s -> s.getOrder().getTypeOfProduct() == type).forEach(so
-> {
                docks.sort(dockComparator);
                Dock earliestDock = docks.get(0);

                Service.createSchedule(so, earliestDock, so.getTrailer());
            });

        });

    }

    public static void initStorage() {

        Driver[] driver = new Driver[15];
        Card[] card = new Card[15];
        Truck[] truck = new Truck[15];
        Trailer[] trailer = new Trailer[15];
        for (int i = 0; i < 15; i++) {
            driver[i] = Service.createDriver("Driver " + i, "1232334" + i, "driver" + i +
"@dc.com");
            card[i] = Service.createCard(driver[i], "driver " + i, "123456");
            truck[i] = Service.createTruck("10" + i, 1);
            trailer[i] = Service.createTrailer("" + i, 150);
            Service.assignDriver(driver[i], truck[i], trailer[i]);
        }
    }

```

```

        Order[] orders = new Order[15];

        for (int i = 0; i < 15; i++) {
            int randomProductType = new Random().nextInt(3);

            int secondRandom = new Random().nextInt(3);
            orders[i] = Service.createOrder("" + i,
                ProductType.values()[randomProductType], 1000);
            orders[i].createSubOrder(trailer[i], 500, 7, LocalDate.now(), (secondRandom +
1) * 10);

            /**
             * first will be trailer 1 Second suborder will be assigned to
             * 39,38... so they will never meet in the middle
             */

            orders[i].createSubOrder(trailer[14 - i], 500, 7, LocalDate.now(),
(randomProductType + 1) * 10);
        }

        Service.createDock("1", ProductType.Boxed);
        Service.createDock("2", ProductType.Boxed);
        Service.createDock("3", ProductType.ChristmasTree);
        Service.createDock("4", ProductType.ChristmasTree);
        Service.createDock("5", ProductType.Palletised);
    }

    public static Dock createDock(String number, ProductType type) {
        assert number != null;
        assert type != null;
        Dock dock = new Dock(number, type);
        Storage.saveDock(dock);
        return dock;
    }

    public static List<Schedule> getAllSchedules() {
        return Storage.getSchedules();
    }

    public static Order createOrder(String id, ProductType type, double weight) {
        assert id != null;
        assert type != null;
        assert weight > 0;
        Order order = new Order(id, type, weight);
        Storage.saveOrder(order);
        return order;
    }

    public static SubOrder createSubOrder(Order order, Trailer trailer, double weight,
double weightMargin,
        LocalDate loadingDate, int loadingTime) {
        assert order != null;
        assert trailer != null;
        return order.createSubOrder(trailer, weight, weightMargin, loadingDate,
loadingTime);
    }

    public static Card createCard(Driver driver, String username, String password) {
        assert driver != null;
        Card card = driver.createCard(username, password);
        Storage.saveCard(card);
        return card;
    }

    public static List<Dock> getDocks() {
        return Storage.getDocks();
    }

    /**
     * checks if the provided username and passowrd are correct in the storage

```

```

    */
    public static Card verifyUser(String username, String password) {

        if (username == null || password == null)
            return null;

        for (Card c : Storage.getCards()) {
            if (username.equals(c.getUsername()) && password.equals(c.getPassword()))
            {
                return c;
            }
        }
        return null;
    }

    public static List<Driver> getDrivers() {
        List<Driver> drivers = new ArrayList<>();

        for (Card c : Storage.getCards()) {
            drivers.add(c.getDriver());
        }
        return drivers;
    }

    /*
    * sets connections between driver, truck and trailer
    */
    public static void assignDriver(Driver driver, Truck truck, Trailer trailer) {
        assert driver != null;
        assert truck != null;
        assert trailer != null;
        driver.setTruck(truck);
        truck.setDriver(driver);

        truck.setTrailer(trailer);
        trailer.setTruck(truck);
    }

    /*
    * removes the schedule that is ready to go on trip
    */
    public static void loaded(Schedule schedule, Dock dock) {
        assert schedule != null;
        assert dock != null;
        dock.removeSchedule(schedule);
    }

    public static void setDockAvailability(Dock dock, boolean availability) {
        assert dock != null;
        dock.setAvailable(availability);
        if (availability == false) {
            Mail.sendMail("petoknm@gmail.com", "Dock Alert", "Dock " + dock + "
availability has changed!");
            organizeSchedules();
        }
    }

    /*
    * puts the storages objects in a file. The specific objects use serializable
    */
    public static void putStorageToFile() {
        try (FileOutputStream streamOut = new FileOutputStream("src/Storage.dat")) {
            ObjectOutputStream objectOutputStream = new ObjectOutputStream(streamOut);
            for (Schedule s : Service.getAllSchedules()) {
                objectOutputStream.writeObject(s);
            }
            for (Card c : Storage.getCards()) {
                objectOutputStream.writeObject(c);
            }
            for (Dock d : Service.getDocks()) {

```

```

        outputStream.writeObject(d);
    }
    for (Order o : Storage.getOrders()) {
        outputStream.writeObject(o);
    }
} catch (Exception e) {
    e.printStackTrace();
}

}

/*
 * reads the storage from file
 */
public static void readStorageFromFile() {
    try (FileInputStream streamIn = new FileInputStream("src/Storage.dat")) {
        ObjectInputStream objectInputStream = new ObjectInputStream(streamIn);
        while (objectInputStream.readObject() != null) {
            System.out.println(objectInputStream.readObject());
        }
    } catch (EOFException ignored) {
        // ignored when the list is finished
    } catch (Exception e) {
        e.printStackTrace();
    }
}

}

/*
 * checks if the weight is out of provided margin
 */
public static boolean outOfMargin(double actualWeight, double targetWeight, double
margin) {

    if (actualWeight <= 0 || targetWeight <= 0 || margin < 0)
        return true;

    if (actualWeight > targetWeight * (margin / 100.0 + 1)) {
        return true;
    } else if (actualWeight < targetWeight * (1 - margin / 100.0)) {
        return true;
    }
    return false;
}

}

```


Mail Class

```
package service;

import java.time.LocalDateTime;
import java.util.Properties;

import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;

public class Mail {
    private String sender;
    private String pass;

    public Mail(String sender, String pass) {
        this.sender = sender;
        this.pass = pass;
    }

    public void sendEmail(String reciever, String title, String messageText) {
        assert reciever != null && title != null && messageText != null;

        // using google host
        String host = "smtp.gmail.com";

        // Get system properties
        Properties properties = System.getProperties();

        // Setup mail server
        try {
            properties.put("mail.smtp.starttls.enable", "true");
            properties.put("mail.smtp.host", host);
            properties.put("mail.smtp.user", sender);
            properties.put("mail.smtp.password", pass);
            properties.put("mail.smtp.port", "587");
            properties.put("mail.smtp.auth", "true");
        } catch (Exception e) {
            System.out.println(e.getLocalizedMessage());
        }

        // Get the default Session object.
        Session session = Session.getDefaultInstance(properties);

        try {
            // Create a default MimeMessage object.
            MimeMessage message = new MimeMessage(session);

            // Set From: header field of the header.
            message.setFrom(new InternetAddress(sender));

            // Set To: header field of the header.
            message.addRecipient(Message.RecipientType.TO, new InternetAddress(reciever));

            // Set Subject: header field
            message.setSubject(title);

            // Now set the actual message
            message.setText(messageText);

            // Send message
            Transport transport = session.getTransport("smtp");
            transport.connect(host, sender, pass);
            transport.sendMessage(message, message.getAllRecipients());

            System.out.println(
                "Message from: " + sender + " to " + reciever + " sent successfully at: "
                + LocalDateTime.now());
        }
    }
}
```

```
        } catch (MessagingException mex) {
            mex.printStackTrace();
        }
    }

    public static void sendMail(String email, String title, String message) {
        Mail mail = new Mail("danishCrownProject@gmail.com", "danishcrown1");
        mail.sendEmail(email, title, message);
    }
}
```

Storage

Storage Class

```
package storage;

import java.util.ArrayList;
import java.util.List;

import model.Card;
import model.Dock;
import model.Order;
import model.Schedule;

public class Storage {

    private static List<Schedule> schedules = new ArrayList<>();
    private static List<Card> cards = new ArrayList<>();
    private static List<Dock> docks = new ArrayList<>();
    private static List<Order> orders = new ArrayList<>();

    public static List<Schedule> getSchedules() {
        return new ArrayList<>(schedules);
    }

    public static void saveSchedule(Schedule schedule) {
        schedules.add(schedule);
    }

    public static List<Card> getCards() {
        return new ArrayList<>(cards);
    }

    public static void saveCard(Card card) {
        cards.add(card);
    }

    public static List<Dock> getDocks() {
        return new ArrayList<>(docks);
    }

    public static void saveDock(Dock dock) {
        docks.add(dock);
    }

    public static List<Order> getOrders() {
        return new ArrayList<>(orders);
    }

    public static void saveOrder(Order order) {
        orders.add(order);
    }
}
```

Test

ServiceTest JUnit Test

```
package test;

import java.time.LocalDate;
import java.time.LocalTime;

import org.junit.Assert;
import org.junit.Test;

import model.Card;
import model.Dock;
import model.Driver;
import model.Order;
import model.ProductType;
import model.Trailer;
import model.Truck;
import service.Service;
import storage.Storage;

public class ServiceTest {

    @Test
    public void testOrganizeSchedules() {
        Driver[] driver = new Driver[3];
        Card[] card = new Card[3];
        Truck[] truck = new Truck[3];
        Trailer[] trailer = new Trailer[3];
        for (int i = 0; i < 3; i++) {
            driver[i] = Service.createDriver("Driver " + i, "1232334" + i, "driver" + i +
"@dc.com");
            card[i] = Service.createCard(driver[i], "driver " + i, "123456");
            truck[i] = Service.createTruck("10" + i, 1);
            trailer[i] = Service.createTrailer("" + i, 150);
            Service.assignDriver(driver[i], truck[i], trailer[i]);
        }

        Order[] orders = new Order[3];
        for (int i = 0; i < 3; i++) {

            orders[i] = Service.createOrder("" + i, ProductType.values()[i], 1000);
            orders[i].createSubOrder(trailer[i], 500, 10, LocalDate.now(), (i + 1) * 10);

            orders[i].createSubOrder(trailer[2 - i], 500, 10, LocalDate.now(), (i + 1) *
10);
        }
        Dock dock1 = Service.createDock("1", ProductType.Boxed);
        Dock dock2 = Service.createDock("2", ProductType.Palletised);
        Dock dock3 = Service.createDock("3", ProductType.ChristmasTree);

        Service.organizeSchedules();

        Assert.assertEquals(6, Storage.getSchedules().size());
        Assert.assertEquals(dock3, Storage.getSchedules().get(0).getDock());
        Assert.assertEquals(dock3, Storage.getSchedules().get(1).getDock());
        Assert.assertEquals(dock1, Storage.getSchedules().get(2).getDock());
        Assert.assertEquals(dock1, Storage.getSchedules().get(3).getDock());
        Assert.assertEquals(dock2, Storage.getSchedules().get(4).getDock());
        Assert.assertEquals(dock2, Storage.getSchedules().get(5).getDock());
        Assert.assertTrue(LocalTime.of(8,
00).equals(Storage.getSchedules().get(0).getSubOrder().getLoadingTime()));
        Assert.assertTrue(LocalTime.of(10,
10).equals(Storage.getSchedules().get(1).getSubOrder().getLoadingTime()));
        Assert.assertTrue(LocalTime.of(8,
00).equals(Storage.getSchedules().get(2).getSubOrder().getLoadingTime()));
        Assert.assertTrue(LocalTime.of(10,
20).equals(Storage.getSchedules().get(3).getSubOrder().getLoadingTime()));
    }
}
```

```

        Assert.assertTrue(LocalTime.of(8,
00).equals(Storage.getSchedules().get(4).getSubOrder().getLoadingTime()));
        Assert.assertTrue(LocalTime.of(10,
30).equals(Storage.getSchedules().get(5).getSubOrder().getLoadingTime()));
        Assert.assertTrue(LocalTime.of(8,
10).equals(Storage.getSchedules().get(0).getSubOrder().getLoadingEndTime()));
        Assert.assertTrue(LocalTime.of(10,
20).equals(Storage.getSchedules().get(1).getSubOrder().getLoadingEndTime()));
        Assert.assertTrue(LocalTime.of(8,
20).equals(Storage.getSchedules().get(2).getSubOrder().getLoadingEndTime()));
        Assert.assertTrue(LocalTime.of(10,
40).equals(Storage.getSchedules().get(3).getSubOrder().getLoadingEndTime()));
        Assert.assertTrue(LocalTime.of(8,
30).equals(Storage.getSchedules().get(4).getSubOrder().getLoadingEndTime()));
        Assert.assertTrue(LocalTime.of(11,
00).equals(Storage.getSchedules().get(5).getSubOrder().getLoadingEndTime()));
    }

    @Test
    public void testVerifyUser() {
        Driver driver = Service.createDriver("Benn", "22336699", "benn@benndc.dk");
        Driver driver1 = Service.createDriver("Mikael", "99336699", "miakel@mikaeldc.dk");

        Card cardBenn = Service.createCard(driver, "benn", "123456");
        Card cardMikael = Service.createCard(driver1, "mikael", "123456");

        Assert.assertEquals(cardBenn, Service.verifyUser("benn", "123456"));
        Assert.assertEquals(cardMikael, Service.verifyUser("mikael", "123456"));
        Assert.assertNull(Service.verifyUser("benn", "1241"));
        Assert.assertNull(Service.verifyUser("mikel", "123456"));
        Assert.assertNull(Service.verifyUser("Benn", "123456"));
        Assert.assertNull(Service.verifyUser(null, "123456"));
        Assert.assertNull(Service.verifyUser("benn", null));
    }

    @Test
    public void testOutOfMargin() {
        Assert.assertFalse(Service.outOfMargin(100, 100, 10));
        Assert.assertFalse(Service.outOfMargin(110, 100, 10));
        Assert.assertFalse(Service.outOfMargin(90, 100, 10));
        Assert.assertFalse(Service.outOfMargin(100, 100, 0));
        Assert.assertTrue(Service.outOfMargin(50, 100, 10));
        Assert.assertTrue(Service.outOfMargin(150, 100, 10));
        Assert.assertTrue(Service.outOfMargin(0, 100, 10));
        Assert.assertTrue(Service.outOfMargin(-100, 100, 10));
        Assert.assertTrue(Service.outOfMargin(-100, -100, 10));
    }
}

```

GUI

MainApp Class

```
package gui;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.ComboBox;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.stage.Stage;
import model.Driver;
import service.Service;

public class MainApp extends Application {

    private Text          txtStatus;
    private TextField      txfPassword;

    private ComboBox<String> userType;
    private ComboBox<Driver> chooseDriver;

    public static void main(String[] args) {
        Service.initStorage();
        Service.organizeSchedules();
        Application.launch(args);
    }

    @Override
    public void start(Stage stage) {
        stage.setTitle("Control Panel");
        GridPane pane = new GridPane();
        initContent(pane);

        Scene scene = new Scene(pane);
        stage.setScene(scene);
        stage.show();
    }

    private void initContent(GridPane pane) {
        // pane.setGridLinesVisible(true);
        pane.setPadding(new Insets(20));
        pane.setHgap(10);
        pane.setVgap(20);

        VBox vbox = new VBox(30);
        pane.add(vbox, 0, 0);
        vbox.setAlignment(Pos.CENTER);

        // header label
        Label lblHeader = new Label("Welcome to DC!");
        vbox.getChildren().add(lblHeader);
        lblHeader.setTextFill(Color.BLACK);
        lblHeader.setFont(Font.font("Calibri", FontWeight.BOLD, 36));

        // comboBox for user selection
        userType = new ComboBox<String>();
        userType.getItems().addAll("Driver", "Employee");
        userType.setValue("Driver");
    }
}
```

```

pane.add(userType, 0, 1);
userType.setOnAction(event -> setDifferentUserType());

// user name and password fields
Label lblUserName = new Label("Username:");
pane.add(lblUserName, 0, 2);

chooseDriver = new ComboBox<Driver>();
chooseDriver.getItems().addAll(Service.getDrivers());
chooseDriver.setValue(Service.getDrivers().get(0));
pane.add(chooseDriver, 1, 2);

Label lblPassword = new Label("Password:");
pane.add(lblPassword, 0, 3);

this.txfPassword = new TextField();
pane.add(txfPassword, 1, 3);
txfPassword.setText("123456");

// Driver Pane Button
Button btnLogIn = new Button("Log In");
pane.add(btnLogIn, 0, 4);
btnLogIn.setOnAction(event -> openPaneAction());

// status text
this.txtStatus = new Text("");
pane.add(this.txtStatus, 0, 5);
this.txtStatus.setFont(Font.font("Calibri", FontWeight.NORMAL, 20));
this.txtStatus.setFill(Color.FIREBRICK);
}

private void setDifferentUserType() {
    if (userType.getSelectionModel().getSelectedIndex() == 1) {
        chooseDriver.setDisable(true);
        txfPassword.setDisable(true);
    } else {
        chooseDriver.setDisable(false);
        txfPassword.setDisable(false);
    }
}

private void openPaneAction() {
    DriverPane driverPane = new
DriverPane(chooseDriver.getSelectionModel().getSelectedItem());
    EmployeePane employeePane = new EmployeePane();

    // Shows pane selected on ComboBox
    if (userType.getSelectionModel().getSelectedIndex() == 1) {
        employeePane.show();
    } else {
        if
(Service.verifyUser(chooseDriver.getSelectionModel().getSelectedItem().getName().toLowerCase(),
txfPassword.getText()) != null) {
            driverPane.show();
            this.txtStatus.setText("You are logged in!");
        } else {
            this.txtStatus.setText("Invalid password or username");
        }
    }
}
}
}

```

DriverPane Class

```
package gui;

import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.control.TreeItem;
import javafx.scene.control.TreeView;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;
import model.Driver;
import model.Schedule;
import model.SubOrder;
import service.Service;

public class DriverPane extends Stage {
    private final Driver driver;
    private TreeView<String> tvSchedules;
    private TextField txWeight;

    public DriverPane(Driver driver) {
        this.driver = driver;

        setTitle("Driver Pane");
        GridPane pane = new GridPane();
        initContent(pane);
        Scene scene = new Scene(pane);
        setScene(scene);
    }

    private void initContent(GridPane pane) {
        // pane.setGridLinesVisible(true);
        pane.setPadding(new Insets(20));
        pane.setHgap(10);
        pane.setVgap(20);

        addScheduleTree();

        Label lblName = new Label("Name: ");
        pane.add(lblName, 0, 0);

        Label lblName1 = new Label(driver.getName());
        pane.add(lblName1, 1, 0);

        Label lblPhoneNum = new Label("Phone #: ");
        pane.add(lblPhoneNum, 0, 1);

        Label lblPhoneNum1 = new Label(driver.getPhoneNumber());
        pane.add(lblPhoneNum1, 1, 1);

        Label lblTruckNum = new Label("Truck #: ");
        pane.add(lblTruckNum, 0, 2);

        Label lblTruckNum1 = new Label(driver.getTruck().getNumber());
        pane.add(lblTruckNum1, 1, 2);

        Label lblTrailerNum = new Label("Trailer #: ");
        pane.add(lblTrailerNum, 0, 3);

        Label lblTrailerNum1 = new Label(driver.getTruck().getTrailer().getId());
        pane.add(lblTrailerNum1, 1, 3);

        Label lblWeight = new Label("Weight: ");
        pane.add(lblWeight, 0, 4);
    }
}
```



```

        this.txWeight = new TextField();
        pane.add(txWeight, 1, 4);

        Button btnNewEntry = new Button("Log Out");
        pane.add(btnNewEntry, 0, 5);
        btnNewEntry.setOnAction(event -> logOutAction());

        pane.add(tvSchedules, 0, 6, 5, 3);
    }

    private void logOutAction() {
        try {
            Schedule s = Service.getSchedules(driver).get(0);
            s.getTrailer().setWeight(Integer.parseInt(txWeight.getText().trim()));

            double actualWeight = Double.parseDouble(this.txWeight.getText().trim());
            SubOrder sOrder =
this.driver.getTruck().getTrailer().getSchedules().get(0).getSubOrder();
            if (Service.outOfMargin(actualWeight, sOrder.getWeight(),
sOrder.getWeightMargin())) {
                Alert alert = new Alert(AlertType.WARNING);
                alert.setTitle("Warning message!");
                alert.setHeaderText("Info");
                alert.setContentText(
                    "Trailer weight is out of margin, please return to the loading area
to fix issue.\nPlease reload the trailer!");
                alert.showAndWait();
            } else {
                s.clear();
                close();
            }
        } catch (Exception e) {
            Alert alert = new Alert(AlertType.WARNING);
            alert.setTitle("Weight!");
            alert.setHeaderText("Info");
            alert.setContentText("Weight is not a number. Type a proper number");
            alert.showAndWait();
        }
    }

    private void addScheduleTree() {
        TreeItem<String> root = new TreeItem<String>();
        for (Schedule s : Service.getSchedules(driver)) {
            root.getChildren().add(TreeViewService.createTreeItemDriver(s));
        }

        this.tvSchedules = new TreeView<String>(root);
        this.tvSchedules.setShowRoot(false);
    }
}

```

EmployeePane Class

```
package gui;

import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.control.Button;
import javafx.scene.control.ComboBox;
import javafx.scene.control.Label;
import javafx.scene.control.TreeItem;
import javafx.scene.control.TreeView;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;
import model.Dock;
import model.Schedule;
import service.Service;

public class EmployeePane extends Stage {

    private TreeView<String> tvSchedules;
    private ComboBox<Dock> chooseDock;

    public EmployeePane() {
        setTitle("Employee Pane");
        GridPane pane = new GridPane();
        initContent(pane);
        Scene scene = new Scene(pane);
        setScene(scene);
    }

    private void initContent(GridPane pane) {
        pane.setPadding(new Insets(20));
        pane.setHgap(10);
        pane.setVgap(20);

        Label lblDock = new Label("Dock #: ");
        pane.add(lblDock, 0, 0);

        chooseDock = new ComboBox<>();
        chooseDock.getItems().addAll(Service.getDocks());
        chooseDock.valueProperty().addListener((o, oldValue, newValue) ->
addScheduleTree(newValue));
        pane.add(chooseDock, 1, 0);

        Button btnLoaded = new Button("Loaded");
        pane.add(btnLoaded, 0, 2);
        btnLoaded.setOnAction(event -> loaded());

        Button btnLogOut = new Button("Log out");
        pane.add(btnLogOut, 1, 2);
        btnLogOut.setOnAction(event -> logout());

        Button btnBreakDown = new Button("Dock broke down");
        pane.add(btnBreakDown, 2, 2);
        btnBreakDown.setOnAction(event -> breakDown());

        this.tvSchedules = new TreeView<String>(new TreeItem<String>());
        this.tvSchedules.setShowRoot(false);
        pane.add(tvSchedules, 0, 3, 4, 3);
    }

    private void breakDown() {
        if (chooseDock.getSelectionModel().getSelectedIndex() < 0) {
            return;
        }
        Dock selectedDock = chooseDock.getSelectionModel().getSelectedItem();
        Service.setDockAvailability(selectedDock, false);
        Service.organizeSchedules();
    }
}
```

```

        Alert alert = new Alert(AlertType.INFORMATION);
        alert.setTitle("Mail been sent");
        alert.setHeaderText("Info");
        String s = "The administartor has been informed that the dock has broked down";
        alert.setContentText(s);
        alert.showAndWait();
    }

    private void logOut() {
        close();
    }

    private void loaded() {
        if (chooseDock.getSelectionModel().getSelectedIndex() < 0) {
            return;
        }
        Alert alert = new Alert(AlertType.INFORMATION);
        alert.setTitle("SMS been sent");
        alert.setHeaderText("Info");
        String s = "Driver will be informed about loading";
        alert.setContentText(s);
        alert.showAndWait();
    }

    private void addScheduleTree(Dock dock) {
        TreeItem<String> root = this.tvSchedules.getRoot();
        root.getChildren().clear();
        for (Schedule s : Service.getSchedules(dock)) {
            root.getChildren().add(TreeViewService.createTreeItemEmployee(s));
        }
    }
}

```

TreeViewService Class

```
package gui;

import javafx.scene.control.TreeItem;
import model.Schedule;

public class TreeViewService {

    public static TreeItem<String> createTreeItemDriver(Schedule s) {

        TreeItem<String> item = new TreeItem<String>("Schedule");
        item.getChildren().add(new TreeItem<String>("Dock num: " + s.getDock()));
        item.getChildren().add(new TreeItem<String>("Date: " +
s.getSubOrder().getLoadingDate()));
        item.getChildren().add(new TreeItem<String>("Weight: " +
s.getSubOrder().getWeight() + "kg"));
        item.getChildren().add(new TreeItem<String>("Loading time start: " +
s.getSubOrder().getLoadingTime()));
        item.getChildren()
            .add(new TreeItem<String>("Loading duration: " +
s.getSubOrder().getLoadingDuration() + "min"));
        item.getChildren().add(new TreeItem<String>("Loading time end: "
+
s.getSubOrder().getLoadingTime().plusMinutes(s.getSubOrder().getLoadingDuration())));

        return item;
    }

    public static TreeItem<String> createTreeItemEmployee(Schedule s) {

        TreeItem<String> schedule = new TreeItem<String>("Schedule " +
s.getSubOrder().getOrder().getId());
        schedule.getChildren().add(new TreeItem<String>("Trailer ID: " +
s.getSubOrder().getTrailer().getId()));
        TreeItem<String> subOrder = new TreeItem<String>("SubOrder No: " +
s.getSubOrder().getOrder().getId());
        schedule.getChildren().add(subOrder);
        subOrder.getChildren().add(new TreeItem<String>("Weight: " +
s.getSubOrder().getWeight()));
        subOrder.getChildren().add(new TreeItem<String>("Loading Time: " +
s.getSubOrder().getLoadingTime()));

        return schedule;
    }
}
```