

Rapport CDIO del 2



Mikkel
Leth
s153073



Sebastian
Hjorth
s153255



Senad
Begovic
s153349



Thomas
Madsen
s154174



Rasmus
Olsen
s153953



Nicki
Rasmussen
s153448

Dato for aflevering: 09-04-2016

Institut: Danmarks Tekniske Universitet

Opgavetype: CDIO projekt

Fag/Retning: 02324 + 02327 + 62576

Vejledere: Ronnie Dalsgaard, Stig Høgh

Timeregnskab

Time-regnskab	Ver. 2015-17-09						
Dato	Deltager	Design	Impl.	Test	Dok.	Andet	I Alt
2015-17-09	Senad	3	2	1	2	0	8
2015-17-09	Rasmus	2	3	1	3	0	9
2015-17-09	Mikkel	3	2	3	3	0	11
2015-17-09	Thomas	3	5	3	3	0	14
2015-17-09	Sebastian	2	8	2	3	0	15
2015-17-09	Nicki	2	2	1	2	0	7

Indholdsfortegnelse

<u>Timeregnskab</u>	<u>2</u>
<u>Indholdsfortegnelse</u>	<u>3</u>
<u>Indledning</u>	<u>4</u>
<u>Course goals</u>	<u>4</u>
<u>Purpose</u>	<u>4</u>
<u>Resumé</u>	<u>4</u>
<u>Specifications:</u>	<u>4</u>
<u>Main Chapter</u>	<u>5</u>
<u>Tokenizer</u>	<u>5</u>
<u>Error messages and error handling</u>	<u>5</u>
<u>Requirements</u>	<u>5</u>
<u>Test</u>	<u>7</u>
<u>Implementation:</u>	<u>9</u>
<u>Commands:</u>	<u>9</u>
<u>RM20 8:</u>	<u>9</u>
<u>Q:</u>	<u>9</u>
<u>B</u>	<u>9</u>
<u>I</u>	<u>10</u>
<u>DW</u>	<u>10</u>
<u>S</u>	<u>10</u>
<u>D</u>	<u>10</u>
<u>P111</u>	<u>10</u>
<u>Display:</u>	<u>10</u>
<u>GRASP:</u>	<u>11</u>
<u>Conclusion</u>	<u>11</u>
<u>Overall conclusion</u>	<u>11</u>
<u>Product Oriented conclusion</u>	<u>12</u>
<u>Use cases</u>	<u>13</u>
<u>Class Diagram</u>	<u>19</u>
<u>System Sequence Diagram</u>	<u>20</u>
<u>Domain model</u>	<u>21</u>

Indledning

Course goals

We are a group of students at the Technical University of Denmark. Who have been given the task to make a login system using our knowledge gained from the course: "Higher Programming". With this at mind, we are going to implement the four terms (CDIO) to this assignment in the best way possible to satisfy our costumer. We will use our skills to analyze, design, implement and test our system, so it will fit the demands and requirements set down by the customer, which will be specified in the section below¹.

Purpose

We've been assigned the task to use an administration module with a simple GUI. The goal is to be able to utilize the program further later in the 3 week period.

Resumé

We started off analyzing the task at hand and understanding the protocol. We received a table that shows the input and output of the weight, as well as a couple of explanation of what the functions should do.

The given program was a simulator of an older version of the protocol. Our task was to update and optimise the program for the new protocol. The old program was constructed by simple if-statements for each command, and we made it faster and easier to comprehend, by making a switch with each command as a case. We split the program into four different classes to divided the responsibility instead of having a single entity with it all. Furthermore the program had no way of handling errors or wrong inputs, now it is able to overcome wrong inputs. Lastly the program can now send error messages so that a user or developer can see and/or fix the problem.

Specifications:

- Java 1.7 eller nyere
- 15 MB ram
- Supporter TCP/protocol
- Telnet klient(preferably PuTTY)

¹ Taken from our previous CDIO project(CDIO2-1)

Main Chapter

Tokenizer

We have created a class which splits the input from the network into tokens. It works by splitting the the string so that each token is either separated by a space or contained within two quotation marks. This enables us to efficiently parse the commands and get the right arguments. It also allows us to quickly add new commands because the parsing is already done for us. There is one flaw with the tokenizer and that is the lack support for multiple spaces. Commands should therefore be carefully entered. In case of a double space the command is not guaranteed to be recognized.

Error messages and error handling

When we first got the program in our hands, we noticed an immediate lack of error handling. So when the base program was done, we started implementing a comprehensive overhaul concerning exceptions and error messages. Now every command and feature have the necessary exception handling and an appropriate error message, so you always have a fair chance at seeing where the program went wrong. All error messages are sent back to the client and prefixed with ES. A couple of examples can be seen in the section with test cases.

Requirements

For this project, we've been asked to implement the following things for our server:

Functional Requirements	Non functional requirements
The program must use port 8000 by default	The weight simulator must have a console user interface.
Porten skal kunne ændres fra kommandolinjen	The program have to emulate a "Mettler BBK" weight.
The command "B 1.123 kg" should change the weights gross limit.	
The program should be able to execute the 6 tasks that the customer wants to have implemented.	
Typing "Q" should close the program.	
Giving the command "tara" should reset the weight counter.	

One should be able to type a response after giving the "RM20 8" command.	
Alle resources should close properly when the program is told to close.	

After writing down what the requirements for the interface, we then decided to priorities the requirements. Since the list of requirements was small, and the program was a minor part of a bigger project, we agreed that all the requirements was a "must have" to get a fully functional project going later on.

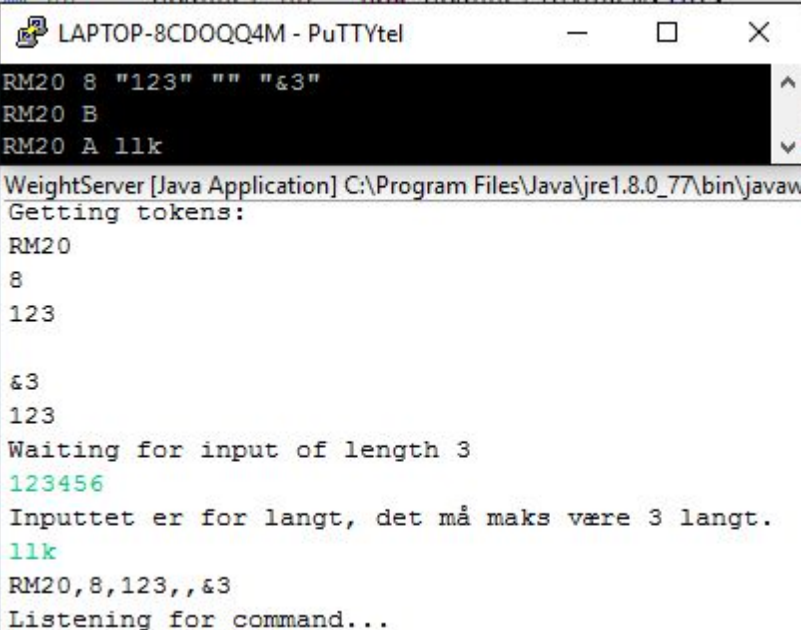
After finishing up the requirements, we started on designing the program. First off, we started out by making use cases for the all the commands that were listed in the requirement. We did this to get an easier overview of the program's flow and also to get an agreement on how the program should work. Down below you will see one of our use cases as an example, while the rest of the use cases will be in the appendix:

Use case description: "Weighing"
ID: 1
Brief description: When you want to measure an object
Primary actors: Player
Secondary actors: None
Preconditions: When the weight is on
Main flow: <ol style="list-style-type: none"> 1. Place an object on the weight 2. Input "S" in the weight display 3. The weight now return the weight of the object
Post conditions: None
Alternative flows: None

This use case is simulated by our program by having the user simulate placing an object on the weight by using the command B, which then sets the weight on the server, then S is called to print the netto which in this case is brutto-0 as T(tara) wasn't used.

Test

In the following we tested the RM20 8 command. The idea was to first test the error handling, to see if any unexpected input would be treated correctly. Afterwards, a test of an expected input was set up, to see if the command works properly. As you can see on the picture down below the operator asked for a message to 123 of length 3. First the server answered with a string of length 6, and the program correctly responded that the message can only be 3 characters long. The server then responds with another message "llk" of length 3, which is properly sent back to the display.



```
LAPTOP-8CDOQQ4M - PuTTYtel
RM20 8 "123" "" "&3"
RM20 B
RM20 A llk
WeightServer [Java Application] C:\Program Files\Java\jre1.8.0_77\bin\javaw.
Getting tokens:
RM20
8
123
&3
123
Waiting for input of length 3
123456
Inputtet er for langt, det må maks være 3 langt.
llk
RM20,8,123,,&3
Listening for command...
```

Now we wanted to test the B, T and S commands. First we wanted to weigh the emballage, we did that by typing in B and right after here, the weight of the emballage. Now we wanted to tara that weight, doing it by typing T. After this we needed to weigh the actual "product". This is the netto weight, but because of this program being a weight simulator the result of using S, to print the netto weight, we will get the last entered B-value minus the saved tara. This causes the netto weight to be negative if tara is greater than the entered netto weight. In our test we start by typing in B 70, and then T. Afterwards we wrote B 5, and then S, to print the netto value. This is expected to be -65. If we want to get the actual netto weight in this simulator, we need to remember the tara weight and add it to the netto weight we enter last.

All this is more deeply explained in the "commands" section.

```

LAPTOP-8CDOQQ4M - PuTTYtel
B 70
DB
T
T S 70.0 kg
B 5
DB
S
S S -65.0 kg
WeightServer [Java Application] C:\Program Files\Java\jre1.8.0_77\bin\javaw.exe (0)

```

```

*****
Netto: -65.0 kg
Instruktionsdisplay:
Sekundørt instruktionsdisplay:
*****

```

Now we moved on test D and P111. D should be able to write a message no longer than 7 characters. The same applies for P, but that message can be of length 30.

```

LAPTOP-8CDOQQ4M - PuTTYtel
D "1234567"
D A
D "12345678"
D A
P111 "1234567890123456789012345678901"
P111 A
P111 "123456789012345678901234567890"
P111 A

```

The first command should be able to write the whole string.

The second command should be able to write the first seven characters, leaving "8" out of the message sent to the server.

The third command should be able to write everything but the last "1".

And the last command should be able to write the whole string.

Down below is the output in our console in eclipse in ordered sequence.

```

*****
Netto: 0.0 kg
Instruktionsdisplay: 1234567
Sekundørt instruktionsdisplay:
*****

```

Output for the first two commands as expected.

```

*****
Netto: 0.0 kg
Instruktionsdisplay: 123456789012345678901234567890
Sekundørt instruktionsdisplay:
*****

```

And this is the output for the third and fourth command.

Implementation:

Commands:

This section contains an overview of all the commands and what we considered when implementing each.

RM20 8:

Description:

Shows *message* to the client and prompts for a response of max length *len*

Format:

RM20 8 "message" "" "&len"

Implementation:

When prompting the user a Scanner is created which then gets the user input until a input less than *len* is entered. When done the scanner is closed. Before user input the command sends RM20 B to the client and RM20 A "response" when the user has entered a valid *response* string into the console.

Q:

Description:

Terminates the program and clean up any resources used by the server.

Format:

Q

Implementation:

All buffers and streams are closed and the program is terminated.

B

Description:

Sets the brutto value of the weight

Format:

B weight

Implementation:

Sets the brutto value of the weight. This is used to simulate adding weight to a scale. This is non-cumulative and hence should stimulate additional weight by adding it to previous weight. Etc: If you call B 20 and then B 50 you won't have a brutto weight of 70, but rather 50. The correct use should be B 20, then B 70 to simulate the added weight. Sends D B to the client if entered correctly.

T

Description:

Sets the current brutto as the tara

Format:

T

Implementation:

Sends T S *tara* to the client where *tara* is the new tara value.

DW

Description:

Sets the display to show nothing

Format:

DW

Implementation:

Sends DW A to the client.

S

Description:

Sends a message to the console containing the netto weight

Format:

S

Implementation:

Sends S S *netto* kg to the console

D

Description:

Prints a *message* of maximum 7 characters to the console.

Format:

D *message*

P111

Description:

Prints a *message* of maximum 30 characters to the console.

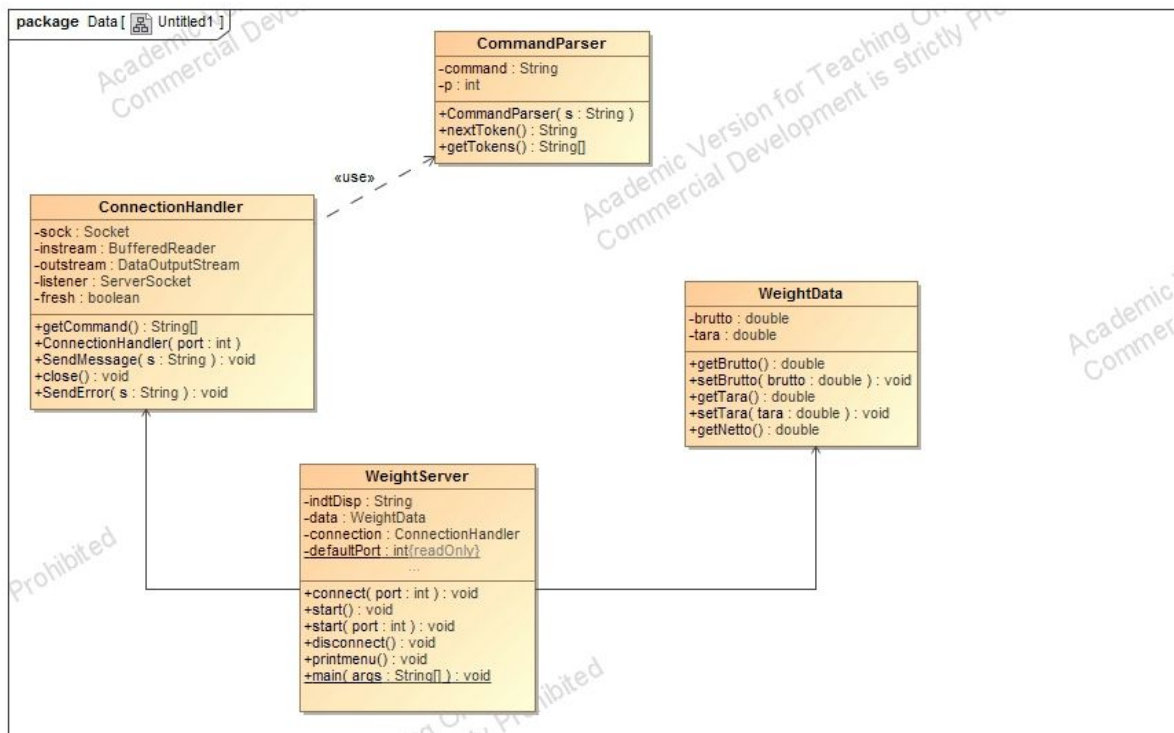
Format:

P111 *message*

Display:

The physical weight has two displays whereas our simulator only has one(the console), hence we have implemented the “different displays” to be one string, but truncating the string to be the length of the display on the weight.

GRASP:



We have split the code into 4 classes(as seen in the UML diagram), each with their own responsibility.

ConnectionHandler handles setting up the TCP port and sending or receiving data from the port. The default port is set to 8000, but it can be modified through the constructor. It makes use of the tokenizer to return the input as a string array of tokens.

The command parser parses each arguments from a string and aids in getting the right arguments with easy. See the CommandParser section for more.

The WeightServer class is the entry point of the application. It makes uses of the ConnectionHandler to get input from the client and then runs the desired commands. Each command is encased in a case statements. We considered concealing each command in their own commands and access them using a hashmap and an ICommand interface, but further considerations deemed that excessive. To sum it up, we have tried our best to keep a high cohesion and low coupling in our program.

Conclusion

Overall conclusion

We are very satisfied with our overall work. We've manage to finish the interface along with all the requirements needed. Our overall group work have been great as always, and we keep improving by making useful decision together, which makes our work easier in the future. A great example was to redesign the code that we got from the beginning which was

mentioned in the main section. Down below you will find the product oriented conclusion, which will describe our interface a little more.

Product Oriented conclusion

The product fulfills all the needs that the customer asked for. Furthermore the secondary display has been removed temporarily as per ordered. As a last feature we've added error messages and error handling. We also simplified supporting new commands by implementing the CommandParser.

In the future following improvements can be made:

- Allowing spaces in the CommandParser
- Printing errors and stacktraces to a file instead of the console, this will improve readability and help debugging.
- Add a GUI to support a primary and secondary display.

Use cases

Use case description: “Afvejning”
ID: 1
Brief description: When you want to measure an object
Primary actors: Player
Secondary actors: None
Preconditions: When the weight is on
Main flow: <ol style="list-style-type: none">1. Place an object on the weight2. Input “S” in the weight display3. The weight now return the weight of the object
Post conditions: None
Alternative flows: None

Use case description: "Tarer vægten"
ID: 2
Brief description: When you want to find a net weight by removing the tara weight
Primary actors: Player
Secondary actors: None
Preconditions: When the weight is on
Main flow: <ol style="list-style-type: none">1. Place an object on the weight2. Input "T" in the weight display3. The weight now return the net weight
Post conditions: None
Alternative flows: None

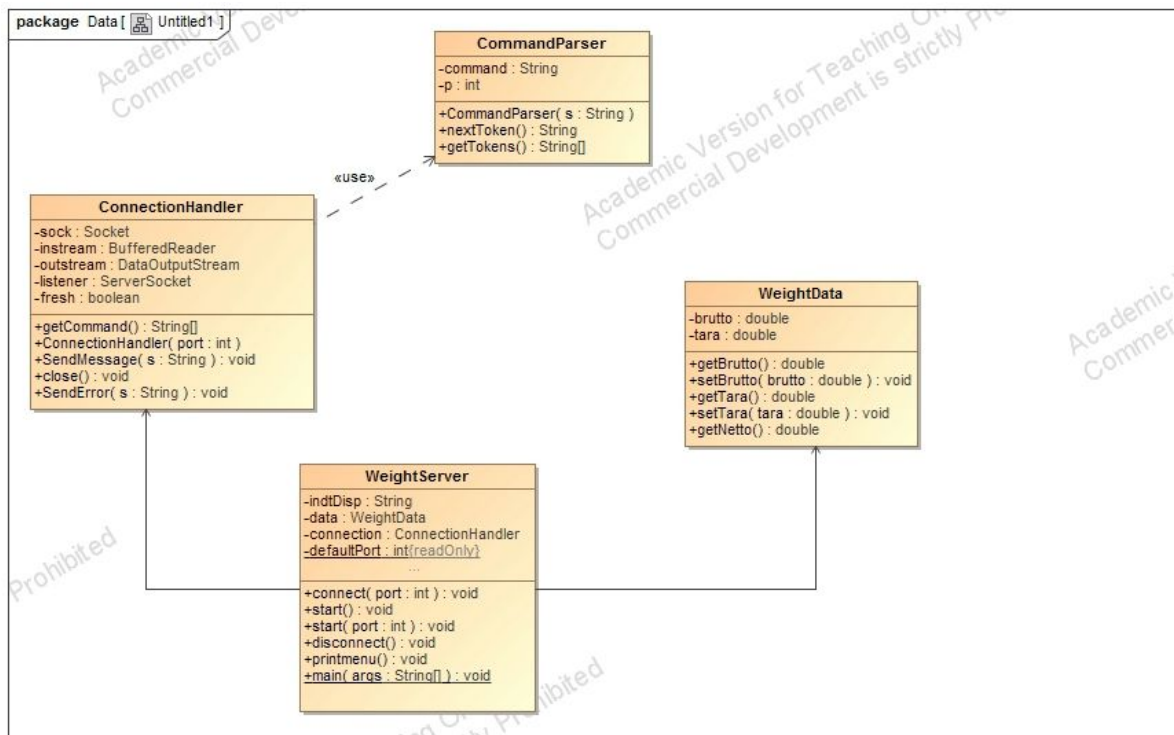
Use case description: “Skriv I vægtens display”
ID: 3
Brief description: When you want to write a text on the weight’s display
Primary actors: Player
Secondary actors: None
Preconditions: When the weight is on
Main flow: 1. Input “D” in the weight’s display followed by a text of maximum 7 characters 2. The weight now returns the text
Post conditions: None
Alternative flows: None

Use case description: “Slet vægtens display”
ID: 4
Brief description: When you want to delete a weights display
Primary actors: Player
Secondary actors: None
Preconditions: When the weight is on
Main flow: <ol style="list-style-type: none">1. Input “DW” in the weight display2. The weight will now clear the display3. It will return the deleted display
Post conditions: None
Alternative flows: None

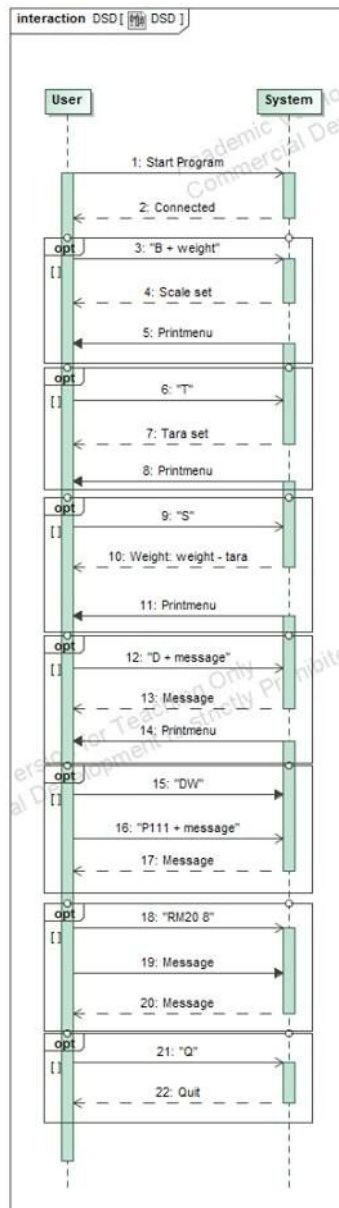
Use case description: “Ændring i brutto belastningen”
ID: 5
Brief description: When you want to change the gross load of an object
Primary actors: Player
Secondary actors: None
Preconditions: When the weight is on
Main flow: <ol style="list-style-type: none">1. Place an object on the weight2. Input “B” followed by the new weight3. The weight will now return the new gross load of the object
Post conditions: None
Alternative flows: None

Use case description: “Quit”
ID: 6
Brief description: When you want to close the program
Primary actors: Player
Secondary actors: None
Preconditions: When the weight is on
Main flow: <ol style="list-style-type: none">1. Input “Q” in on the weight display2. The weight will stop everything and shutdown
Post conditions: None
Alternative flows: None

Class Diagram



System Sequence Diagram



Domain model

