13_CDIO_2

'Vægt simulator med grafisk brugergrænseflade'

Videregående programmering 02324, Softwareteknologi

Gruppe Nr.: 13

Afleveringsfrist: Lørdag 18/03-2016 05:00

Institut: DTU Compute

Vejleder: Ronnie Dalsgaard og Christian Budtz

Denne rapport er afleveret via Campusnet (der skrives ikke under)

Denne rapport indeholder 28 sider eks. denne sider.

Jesper Bang, s144211

Bertram Christian Henning, s153538

Jia Hao Johnny Chen, s165543

Jonathan Yngve Friis, s165213

Christopher David Carlson Chytræus, s165230

Thomas Kristian Lorentzen, s154424

Contents

Abstract

The project documented in this report is a simulation of a weight using a graphical user interface. To develop our simulation, we have based our program on a skeleton of a project handed to us in java. The program we have coded is developed using that skeleton. The main focus in terms of coding on our part has primarily been centered around sockets and thread programming. Because a larger part of the code was handed to us beforehand by our professors than in earlier projects, we haven't relied as much on UML in terms of visualising patterns for a more effective development process. Likewise, we haven't spent as much time on unit testing with this project as we have done in previous projects for the same reasons.

Lastly, we can conclude that this has been a successful project that we've managed to finish and met the requirements set by the customer.

Timeregnskab

CDIO-D2						
Timeregnskab	**Uge 10**					
Deltager	**Design**	**Impl.**	**Test**	**Dok.**	**Andet**	**I Alt**
Bertram	1	3		1	1	6
Christopher	2	1		4		7
Johnny	1	2		3		6
Jonathan	2	2		3		7
Thomas	1	1		4		6
Jesper	1	3		2	1	7
Sum	8	12		17	2	39

CDIO-D2						
Timeregnskab	**Uge 11**					
Deltager	**Design**	**Impl.**	**Test**	**Dok.**	**Andet**	**I Alt**
Bertram	3	9	5	4		21
Christopher	5	5	5	4		19
Johnny	4	7	4	5		20
Jonathan	4	6	5	6		21
Thomas	4	4	4	7		19
Jesper	2	9	3	6		20
Sum	22	40	26	32		120

Formål

Formålet med denne rapport er at dokumentere udviklingsprocessen for vores projekt (hvor vores hovedfokus har været at benytte trådet programmering og sockets), dvs. hvad vi har gjort for at få en mere effektiv process, hvordan vi har testet den kode vi er kommet frem med, samt vise hvordan vores kode er blevet testet. Dette inkluderer også til en hvis grad UML diagrammer. Samtidigt er formålet med denne rapport i en mindre grad også at forklare på teoretisk vis hvilke fremgangsmåder vi har valgt, samt begrundelse for dette.

Kravspecifikation

Functional:

1. Systemet skal kunne modtage kommandoer over TCP.
2. Systemet skal kunne sende kommandoer over TCP (se bilag 1).
3. Systemet skal kunne vise modtaget inputs på GUI'en.
4. Systemet skal kunne modtage flere inputs samtidig.
5. Systemet skal have port 8000 som default værdi.
6. System skal have en bruger der hedder "Anders And" med 12 som bruger ID.
7. Systemet skal ved opstart kunne overskrive portens default værdi fra kommandolinjen.
8. Systemet skal kunne følge kundens afvejnings procedure (se bilag 2).

Implementation:

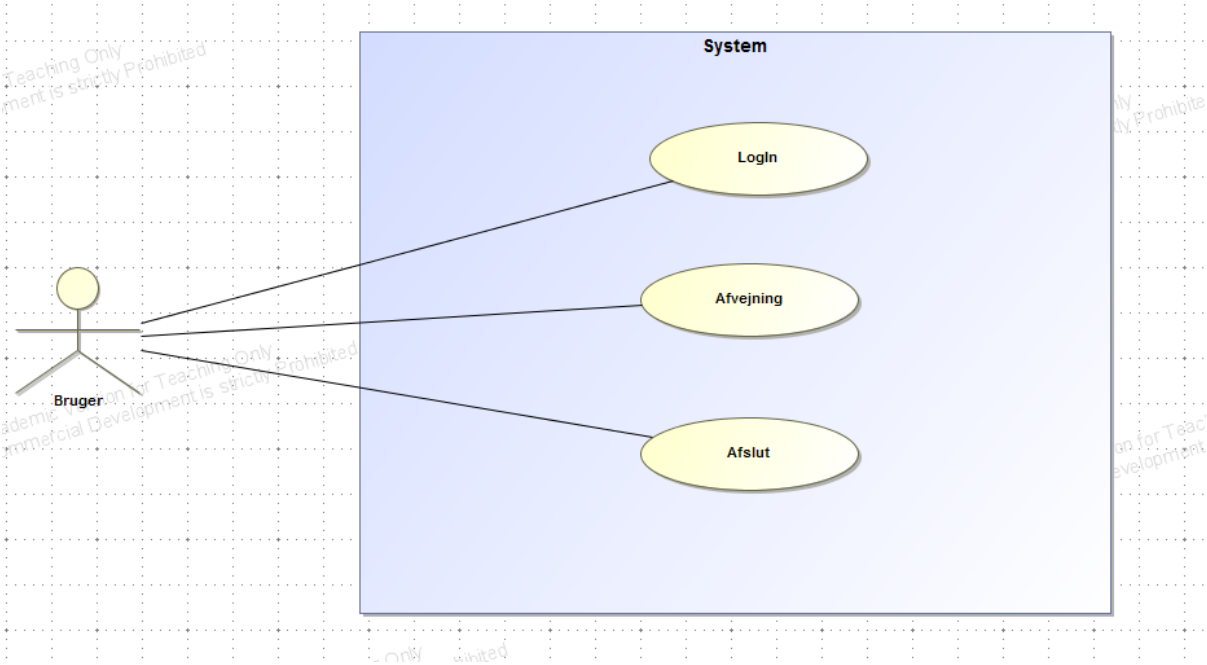
9. Systemet skal være testet og fungere på en af DTU's databarer
10. Systemet skal udvikles i Eclipse
11. Systemet skal kodes i Java
12. Systemet skal bruge UTF-8 som tegnsætning

Hovedafsnit

Observer pattern:

I vores projekt bruger vi observer pattern. Dette pattern bruger vi fordi at vores program er et event handling system, med andre ord så gør vores program noget ud fra en bestemt handling f. eks et klik i GUIen. Vores program fungerer på den måde at de forskellige klasser notificerer hinanden om en handling i GUIen og derfor kan de siges at vores program følger observer pattern.

Use Cases:



Ovenfor ses use case diagram for vores program. I vores program har brugeren mulighed for at logge ind, afveje og afslutte programmet. Nedenfor ses use case's:

Use Case: Log ind
ID: 01

Brief Description: Brugeren logger ind på vægten
Primary Actors: Brugeren
Preconditions: Der er ikke nogen bruger der er logget ind på vægtens bruger grænseflade
Main Flow: 1. Brugeren taster sit navn ind på vægten 2. Maskinen sender besked til brugeren 3. Brugeren accepterer
Postconditions: Brugeren er logget ind på vægtens grænseflade

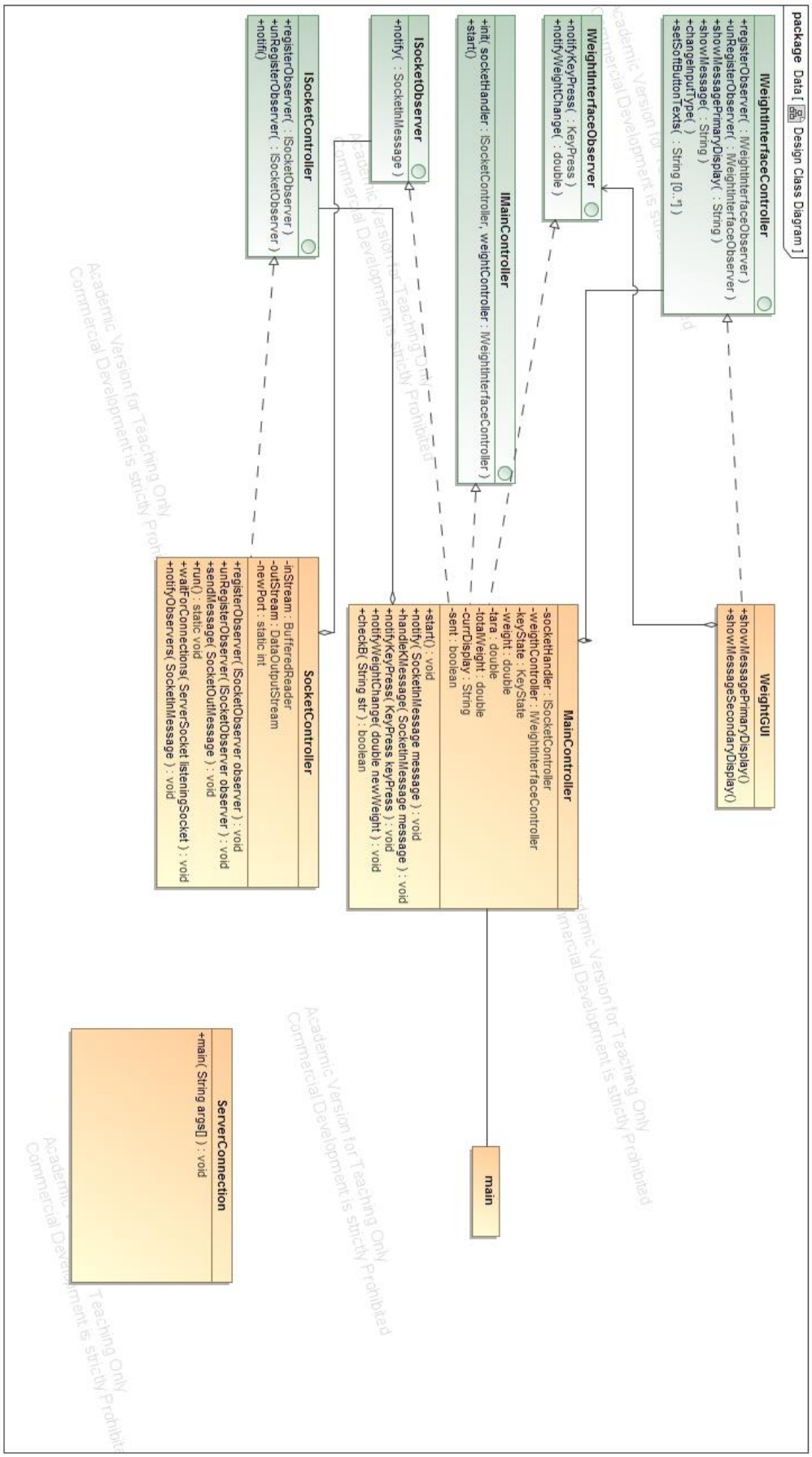
Use Case: Vejning
ID: 02
Brief Description: Brugeren følger afvejningsproceduren
Primary Actors: Brugeren
Preconditions: Brugeren er korrekt logget ind
Main Flow: 1. Brugeren indtaster batchnummer 2. Brugeren placerer tom beholder på vægten(tara) 3. Brugeren hælder pulveret i den tomme beholder(netto) 4. Brugeren fjerner beholder med indhold fra vægten

Post Conditions:

Use Case: Afslut
ID: 03
Brief Description: Brugeren lukker ned for simulatoren
Primary Actors: Brugeren
Preconditions: Brugeren er logget ind
Main Flow: 1. Brugeren taster “q” i interfacet
Post Conditions: Brugeren har afsluttet programmet

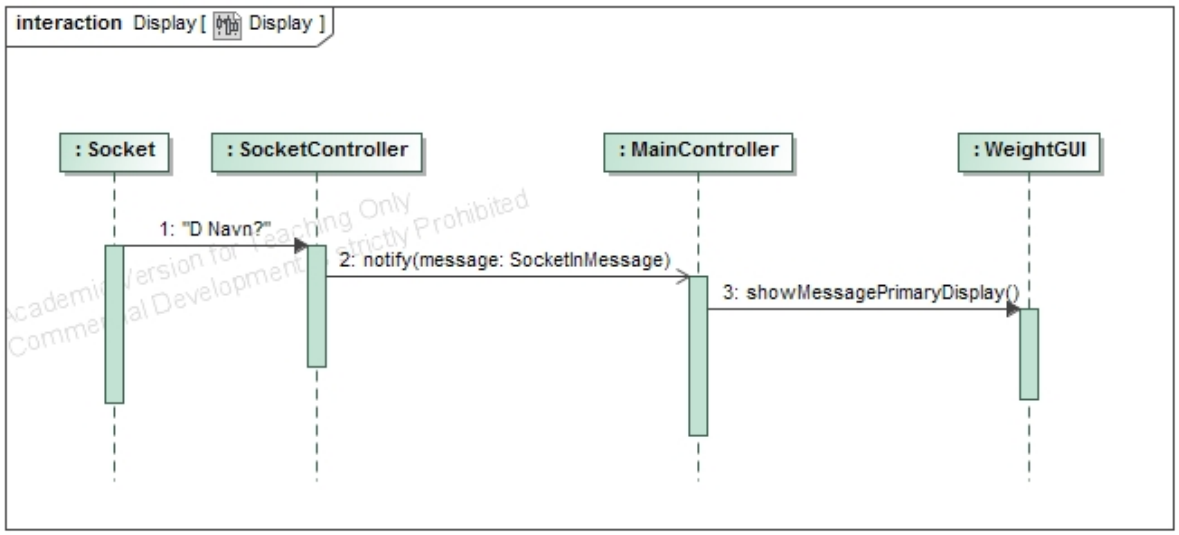
Ovenover ses formelle brugsscenarier for de 3 overordnede scenarier i vores program. Formålet med disse er, at give en mere dybdegående forklaring af hvad der sker, hvilke forbehold det sker under, og hvem der er de primære aktører.

Design klassediagram



Ovenover ses design klassediagrammet for vores program. Vi har en del af programmet der står for at oprette "vægten", dette er alt undtagen ServerConnection klassen. Vi har så ServerConnection klassen, som er et program der sender kommandoer til vægten.

Sekvensdiagram



Ovenover ses et sekvens digram for vores program. Dette sekvensdiagram beskriver sendelsen af en besked fra en socket til vores SocketController, derefter notificer SocketController MainControlleren om at den har modtaget en besked og hvad beskeden er. Derefter sender MainController en besked til WeightGUI om at vise denne besked i "vægten".

Code snippets

```
try {
    sock = new Socket(hostName, portNumber);
    out = new PrintWriter(sock.getOutputStream(), true);
    in = new BufferedReader(new InputStreamReader(sock.getInputStream()));
} catch (IOException e) {
    e.printStackTrace();
}
```

Ovenover ses kode, som opretter en socket og forbinder denne socket til hosten "hostName" og til portnummeret "portNumber". Derefter oprettes en PrintWriter out, som vi senere bruger til at senere fra denne socket til den som vi vælger at connecte den til. Nu oprettes der så en BufferedReader in, som vi senere bruger til at modtage fra en socket. Der er sat en try catch på denne del af koden for at fange eventuelle IOExeptions. Dette er vores klient socket.

```
Socket activeSocket = listeningSocket.accept(); //Blocking call
inStream = new BufferedReader(new InputStreamReader(activeSocket.getInputStream()));
outStream = new DataOutputStream(activeSocket.getOutputStream());
```

Ovenover ses koder, som opretter en socket. Denne socket er vores server socket.

```
public boolean checkB(String str) {
    try {
        if (Double.parseDouble(str) <= 6 && str.length() <= 6 && str.length() >= 1) {
            if (str.contains(".") && str.length() >= 3) {
                return true;
            }
        }
        return false;
    } catch (NumberFormatException e) {
        return false;
    }
}
```

checkB bliver brugt til at undersøge hvorvidt at inputtet er en double og om det opfylder visse krav. Som eksempel at det indtastede tal ikke kan være højere vægt end vægten tillader og man skal skrive et decimal på hvis man benytter sig af komma ("1." - er ulovligt, men "1.0" - er ok). Vi tjekker også at den double som bliver indtastet ved brug af B kommandoen ikke har flere decimaler end displayet kan vise.

```
case B:
    //Checking if command is valid.
    if (checkB(message.getMessage())){
        //Changing weight's screen to match new value.
        String tempB = message.getMessage() + " kg";
        weightController.showMessagePrimaryDisplay(tempB);
        System.out.println(tempB);
        this.notifyWeightChange(Double.parseDouble(message.getMessage()));
        socketHandler.sendMessage(new SocketOutMessage("B A \r\n"));

    } else{
        // If command is not a double value ES will be returned.
        socketHandler.sendMessage(new SocketOutMessage("ES\r\n"));
}
    break;
```

Case B sætter en ny bruttovægt på display'et hvis en tilladt kommando indtastes og svarer tilbage med en bekræftelse.

Implementerede kommandoer

- Vi har ikke lavet knappen C da selve funktionen mangler i den udleverede kode. Der sker derfor ikke noget når man trykker på C i GUI'en.

DW:

```
case DW:
    weightController.showMessagePrimaryDisplay(df.format(totalWeight)
            .toString().replace(",", ".") + " kg");
    socketHandler.sendMessage(new SocketOutMessage("DW A \r\n"));
    break;
```

Sletter teksten i det primære display, sender en bekræftelse tilbage og viser vejeresultatet igen.

T:

```
case T:
    tara = weight;
    weight = 0.0000;
    weightController.showMessagePrimaryDisplay(df.format(weight).toString().replace(",", ".") + " kg");
    socketHandler.sendMessage(new SocketOutMessage("T S " + tara + " kg\r\n"));
    totalWeight = totalWeight + tara;
    break;
```

Tarér vægten og gemmer den taréde vægt. Først sættes tara lig med vægten og efterfølgende sættes vægten lig med 0. Vægten bliver printet ud i GUI'en og den taréde vægt kvitteres. Herefter bliver totalvægt lagt sammen med den taréde vægt.

S:

```
case S:
    socketHandler.sendMessage(new SocketOutMessage("S S " + weight.toString()+" kg \r\n"));
    break;
```

Henter den nuværende vægt på den primære display.

P111:

```
case P111:
    if(message.getMessage().length() > 30){
        socketHandler.sendMessage(new SocketOutMessage("ES\r\n"));
        break;
    }
    weightController.showMessageSecondaryDisplay(message.getMessage()+"");
    socketHandler.sendMessage(new SocketOutMessage("P111 A\r\n"));


    break;
```

Sender en tekst til det sekundære display og kvitterer.

Q:

```
case Q:
    socketHandler.sendMessage(new SocketOutMessage("Closing...\r\n"));
    System.exit(0);
    break;
```

Afslutter programmet.

Test

Testcase ID	01
Summary	The system shall receive commands with TCP and shows them
Requirements	R01 and R03
Preconditions	1. Started the weight program
Postconditions	The weight program successfully received commands.
Test Procedure	1. Start cmd and connect to IP address 127.0.0.1 and port 8000 2. Send commands to the program 3. See if the GUI updates accordingly to the command.
Test Data	Command: B 123
Expected Result	The GUI shows 1.23 kg in the primary display
Actual Result	The GUI showed successfully showed 1.23 kg in the primary display.
Status	Passed
Tested by	Jia Hao Johnny Chen and Jonathan Yngve Friis

Date	17-03-2017
Data environment	Eclipse jee neon 2.0 on windows 10

Testcase ID	02
Summary	The system is able to send command over TCP
Requirements	R02
Preconditions	1. Started the weight program
Postconditions	The system successfully sends data to the cmd.
Test Procedure	1. Open cmd and connect to IP address 127.0.0.1 and port 8000 2. Enter numbers into GUI via the numpad. 3. Press send in the GUI 4. See if the cmd prints the number that was entered
Test Data	Number: 14
Expected Result	The cmd prints the number 14
Actual Result	The cmd successfully prints 14

Status	Passed
Tested by	Jia Hao Johnny Chen and Jonathan Yngve Friis
Date	17-03-2017
Data environment	Java 1.8 in Eclipse

Testcase ID	03
Summary	The system shall be able to take multiple inputs at the same time.
Requirements	R04
Preconditions	1. The weight program is started 2. The serverConnection is started
Postconditions	
Test Procedure	1. Start a cmd on a different pc. 2. Connect to IP adress 127.0.0.1 and port 8000 with cmd. 3. Type commands in the cmd while also typing a number in the GUI.
Test Data	Multiple values

Expected Result	The GUI shows both inputs. For instance if the user on the other pc writes the command to update the primary display and the user on the pc with the program gives the command to update the secondary display, the GUI updates both at the same time.
Actual Result	The GUI successfully updated both displays at the same time.
Status	Passed
Tested by	Jia Hao Johnny Chen and Jonathan Yngve Friis
Date	17-03-2017
Data environment	Java 1.8 in Eclipse

Testcase ID	04
Summary	We test if the program connects to port no. 8000 by default
Requirements	R05
Preconditions	Our program has been started, and our GUI

	is running
Postconditions	Connection established to port 8000
Test Procedure	1.Try to connect via localhost and port 8000 2. Check if a connection has been established
Test Data	telnet 127.0.0.1 8000
Expected Result	“connected to localhost” message in cmd
Actual Result	“connected to localhost” message in cmd
Status	Passed
Tested by	Thomas Kristian Lorentzen, Christopher David Carlson Chytræus, Bertram Christian Henning
Date	17-03-2017
Data environment	Java 1.8 in Eclipse

Testcase ID	05
Summary	We test if the system has a user called

	Anders And with the ID no. 12
Requirements	R06
Preconditions	Both programs has been started
Postconditions	The system successfully logs in with the user Anders And
Test Procedure	1. Start both programs 2. Type 12 on GUI 3. Acknowledge you are Anders And
Test Data	UserID: 12
Expected Result	Program lets us past login
Actual Result	Program let us past login
Status	Passed
Tested by	Thomas Kristian Lorentzen, Christopher David Carlson Chytræus, Bertram Christian Henning
Date	17-03-2017
Data environment	Java 1.8 in Eclipse

Testcase ID	06
Summary	We test if the system can overwrite the port number with launch arguments.
Requirements	R07
Preconditions	None
Postconditions	A connection is established on another port than 8000
Test Procedure	1. Set launch argument to 8001 2. Start program 3. Connect to localhost on port 8001
Test Data	arg = 8001 telnet 127.0.0.1 8000
Expected Result	“connected to localhost” message in cmd
Actual Result	“connected to localhost” message in cmd
Status	Passed
Tested by	Thomas Kristian Lorentzen, Christopher David Carlson Chytræus, Bertram Christian Henning
Date	17-03-2017

Data environment	Java 1.8 in Eclipse

Testcase ID	07
Summary	We test whether the system successfully follows the weighing procedure specified by the client.
Requirements	R08
Preconditions	The system and GUI is running.
Postconditions	The weighing worked accordingly to the weighing procedure.
Test Procedure	1. The system asks for a user ID 2. The user types in their user ID and confirms that it is the correct name 3. The system asks for a batch ID 4. The user types in batch ID 5. The system asks the user to confirm the weighing item 6. The system asks the user to tare the weight 7. The system asks the user to place the tare weight 8. The system asks the user to place the netto weight

	9. The system asks the user to remove the gross weight 10. Weighing successful
Test Data	User ID: 12 Batch ID: 1234 User presses “send” to confirm after doing each task
Expected Result	The wieght prints “OK”
Actual Result	The wieght prints “OK”
Status	Passed
Tested by	Thomas Kristian Lorentzen, Christopher David Carlson Chytræus, Bertram Christian Henning
Date	17-03-2017
Data environment	Java 1.8 in Eclipse

Testcase ID	08
Summary	The system must work on the databar computers on DTU campus
Requirements	R09

Preconditions	Download the files into eclipse.
Postconditions	none
Test Procedure	1. Start the program on the databar computers. 2. See if the program works correctly on the computer.
Test Data	N/A
Expected Result	The program runs successfully on the databar computers.
Actual Result	The program ran successfully on the databar computers.
Status	Passed
Tested by	Jia Hao Johnny Chen and Jonathan Yngve Friis
Date	17-03-2017
Data environment	Java 1.8 in Eclipse

Traceability Matrix

	TC 01	TC 02	TC 03	TC 04	TC 05	TC 06	TC 07	TC 08
R01	**x**							
R02		**x**						
R03	**x**							
R04			**x**					
R05				**x**				
R06					**x**			
R07						**x**		
R08							**x**	
R09								**x**

Kongfiguration

Programmet kræver en computer med java 1.7 eller nyere installeret og en IDE.

For at køre programmet kræves det at man kører begge main metoder og sørger for at begge mains har samme arg. Begge mains har per default 8000 som argument. Ønskes det at bruge en anden port kræves det at man ændrer argumenterne i begge mains til det samme.

Guide til importering af git repository:

1. Start browser

2. Gå ind på siden: https://github.com/BertramHenning/13_CDIO_2

3. Klik på “clone or download” og kopier derefter linket

4. Åben eclipse

5. Klik på “file” og vælg “import”

6. Vælg “Git-> Projects from Git”

7. Tryk “next”

8. Klik på “clone URL”

9. Klik på “next”

10. Klik på “next” igen

11. Vælg sti at gemme det på

12. Klik på “next”

13. Vælg “import existing eclipse projects” og tryk “next”

14. Sørg for at project mappen har et flueben ud for sig eller klik i den tomme kasse

15. Tryk “finish”

16. Åben projektet i eclipse

17. Åben “src”

18. Åben pakken “controller”

19. Åben Main.java

20. Klik på den grønne knap med den hvide pil i.

21. Åben pakken “Main”

22. Åben ServerConnection.java

23. Klik på den grønne knap med den hvide pil i.

Konklusion

Vores vægt simulator er blevet færdigudviklet med en grafisk brugergrænseflade. Kravspecifikationerne er testet og blevet opfyldt samt kommandoerne, og vægt simulatoren simulerer en Mettler BKK vægt. Dermed kan vi konkludere at vores program har fyldt afleveringsopgaven.

Litteraturliste

- CDIO-D2 Vægt Simulatoren:

  https://docs.google.com/document/d/1MPiPVtAjLK0Geio2UEvLhrufU08Zqtzb-0iFCHpTHjk/edit#

- CDIO-D2 Afvejning Processen:
- https://docs.google.com/document/d/1Tji1TduZNTFgxBp9-xEWIAVCdO4z4imr9Q7NUDijNdA/edit

Bilag

Bilag 1: Liste og forklaring af kommandoer

1. S: Send stabil afvejning - Vægten svarer over netværket med nettovægten (svt. den vægt, der står på displayet).
2. T: Tarér vægten - Nulstil vægten så nuværende belastning svarer til 0.000 kg (displayet viser 0.000). Vægten svarer med den nuværende belastning.
3. D: Skriv i vægtens display - Vægten udskriver den sendte besked i displayet og svarer med en bekræftelse. Er allerede implementeret.
4. DW: Slet vægtens display - Vægtens display ryddes og vægten svarer med en bekræftelse.
5. P111: Skriv max. 30 tegn i sekundært display - Vægtens sekundære display viser den sendte besked. Der svares med en bekræftelse.
6. RM20 8: Skriv i display, afvent indtastning - Vægten viser en besked til brugen og afventer brugerens input på vægten. Der sendes en bekræftelse. Når brugeres input er afsluttet sendes dette. NB: der er **to svar** fra vægten.
7. K - Skifter vægtens knap-tilstand. Når der trykkes på funktionstaster (Tara, Zero, [-> (Send)) afhænger resultatet af vægtens tilstand. Knappens funktion bliver enten udført på vægten eller ej, og funktionskoden bliver enten sendt over netværket eller ej - i alt 4 muligheder:
   K 1: Funktionen udføres, men funktionskoden sendes ikke (standardindstillingen).
   K 2: Funktionen udføres ikke og funktionskoden sendes ikke (knapperne er inaktive).
   K 3: Funktionen udføres ikke, men funktionskoden sendes.
   K 4: Funktionen udføres og funktionskoden sendes.
   Trykkes eks. på Tara eller [-> (Send) sendes K A 1 hhv K A 3 over netværket. NB: Denne kommando er allerede implementeret. K4 kan eks. bruges til at bede brugeren om at sætte et emne på vægten og trykke på [->.
8. B - Sæt ny bruttovægt (Svarende til at der placeres et emne med en given vægt på den fysiske vægt). B 1.123 ændrer vægtens brutto-belastning til 1.123 kg.
9. Q - Afslut simuleringen (Slukker vægten).

Bilag 2: Afvejningsprocedure

Kunden ønsker følgende afvejningsprocedure:

Start: Vægten beder om, at der indtastes operatørnummer →

← Operatøren indtaster sit brugernummer (område 11-99)

Operatørens navn findes i databasen og sendes til vægten →

← Operatøren kvitterer for at navnet er korrekt.

Vægten beder om, at der indtastes batch nummer (område 1000-9999) →

Operatøren instrueres om, at vægten skal være ubelastet →

← Operatøren kvitterer

Vægten tareres →

Operatøren instrueres om, at placere tara (tom beholder) på vægten →

← Operatøren kvitterer

Tara's vægt registreres →

Vægten tareres →

Operatøren instrueres om at placerer netto (pulver hældes i skål) på vægten →

← Operatøren kvitterer

Nettovægt registreres →

Vægten tareres →

Operatøren instrueres om at fjerne brutto fra vægten →

Bruttovægt registreres (negativ) →

Vægten Tareres →

Der udskrives OK eller kasseret på vægten →

← Operatøren kvitterer

--

Der fortsættes ved start med en ny afvejning.

****