Projektopgave forår 2013 - Juni

02324 Videregående programmering.

CDIO del 2

Gruppe nr: 13. Afleveringsfrist: Mandag den 25/03 2013 Kl. 05:00

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

Denne rapport indeholder 13 sider incl. denne side.

s113730, Østergaard, Jesper 

 s123694, Celik ,Veli

s123849, Walbom, Oliver 

 s122966, Schmidt, Christopher

s123123, Kok, Mathias 

 s114211, Hviid, Nicklas

s123658 , Bengtsson , Martine 

Videregående programmering - CDIO 2							
Time-regnskab							
Dato	**Deltager**	**Design**	**Impl.**	**Test**	**Dok.**	**Andet**	**Ialt**
04-03-2013	Jesper	1				1	2
-	Oliver	1				1	2
-	Martine	1				1	2
	Veli	1				1	2
	Mathias	1				1	2
	Chrisopher	1				1	2
11-03-2013							
	Jesper		2				2
	Oliver			2			2
	Martine			2			2
	Veli		2				2
	Mathias		2				2
	Chrisopher		2				2
	Nicklas		2				2
18-03-2013							
	Jesper				2		2
	Oliver				2		2
	Martine				2		2
	Veli				2		2
	Mathias				2		2
	Chrisopher				2		2
	Nicklas				2		2
21-03-2013							
	Jesper				1		1
	Oliver			1			1
	Martine				1		1
	Veli				1		1
	Chrisopher				1		1
	Nicklas				1		1
	Martine			1			1
24-03-2013							
	Jesper				1		1
	Oliver				1		1
	Martine				1		1
	Veli				1		1
	Chrisopher				1		1
	Nicklas				1		1

	Martine				1		1
	Sum	6	10	6	26	6	54

Indhold

Forord 3
Indledning 3
Problemformulering 3
Usecase diagram 4
Løsnings domæne 5
Sekvensdiagrammer 5
Klassediagram 7
Implementering 8
Test 11
Afsluttende kommentar / konklusion 12

Forord

Dette er gruppe 13's rapport til CDIO del 2 i kursus 02324. I denne rapport vil der være beskrivelser af de forskellige aspekter af projektet, inklusiv forklaringer af programmets opbygning vha. UML diagrammer og forklarende tekst.

Indledning

Til CDIO opgave del 2, vil der blive lavet et program der simulerer en "Mettler BBK" vægt. Dette program vil både kunne lytte på kommandoer fra en netværksforbindelse og på det "fysiske" input fra "vægtens" allerede eksisterende funktioner. Dette program er dog stand alone, og er endnu ikke sat sammen med programmet lavet i CDIO del 1. Programmet mangler også stadig en database der kan huske individuelle afvejninger samt recepter, samt en browser UI.

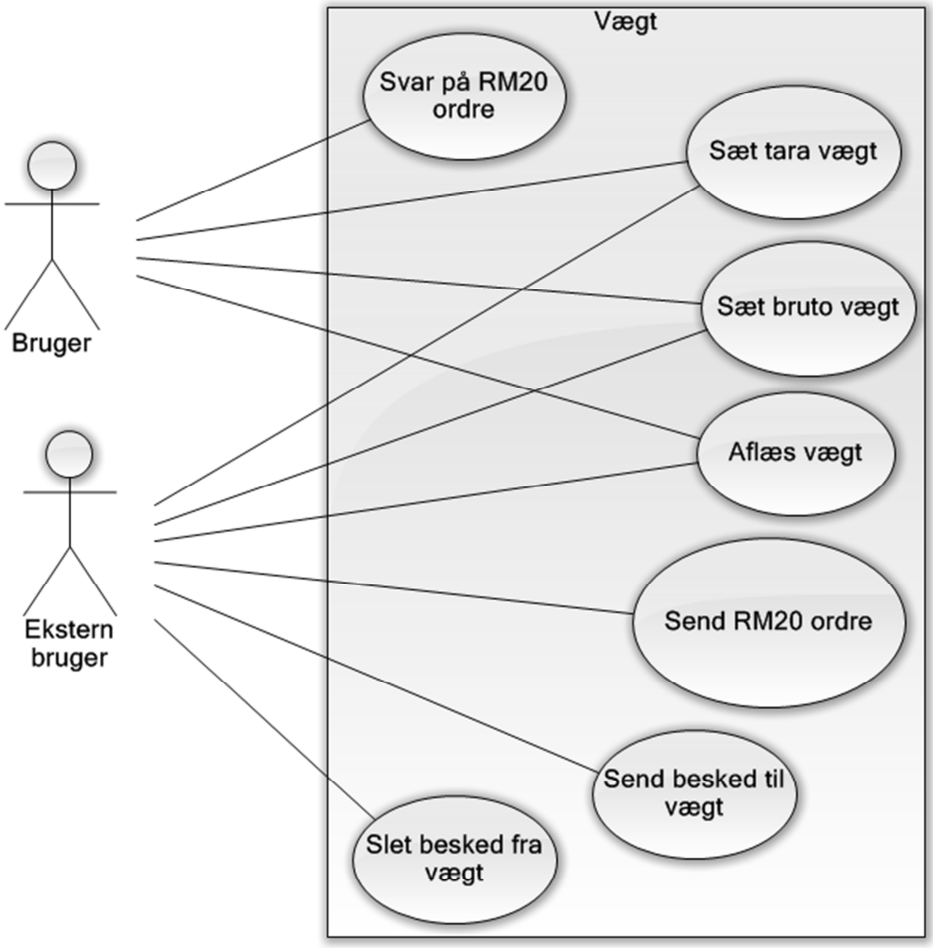
Problemformulering

Til denne CDIO opgave skal der laves et vægt simuleringsprogram. Dette program skal kunne håndtere at få sendt kommandoer over en netværksforbindelse (programmet skal som default lytte på port 8000) dette kan gøres via. telnet, hyperterminal eller et andet vilkårligt program, udføre kommandoen samt sende det rette svar. Ud over dette skal programmet også kunne simulere den fysiske del af vægten, i

form af tarering af vægt, nedlukning af program, samt sætte en bruttovægt. Det er også muligt at implementere en GUI der kan repræsentere den fysiske vægt.

Usecase diagram

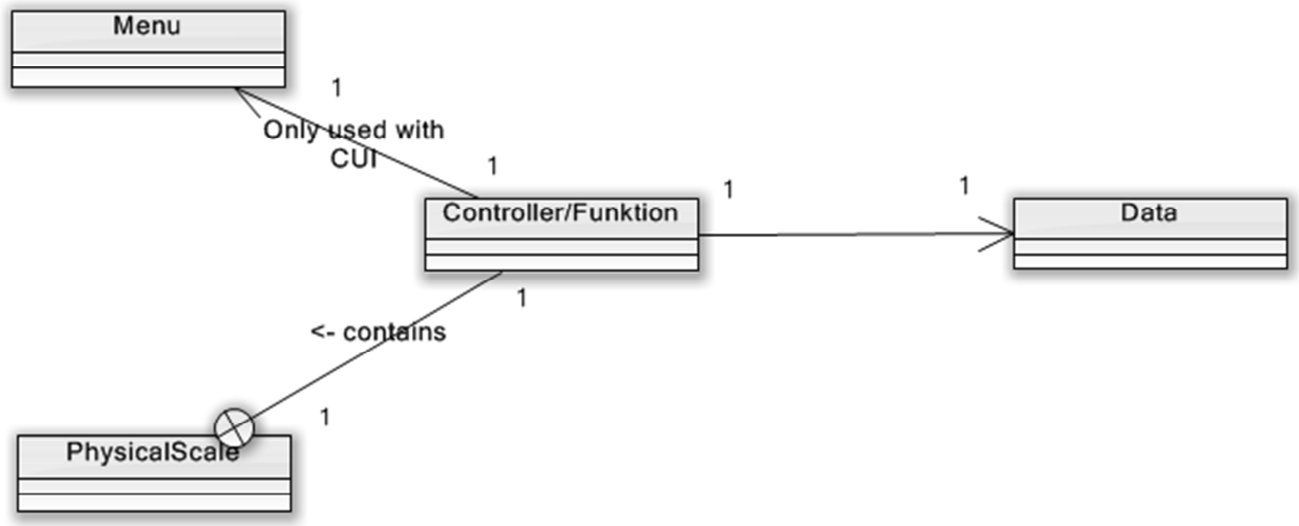
Den overordnede use case til vægt simulatoren er "brug vægt". Men for at få et bedre overblik over hvilke funktioner simulatoren skal have, har vi delt hoved use case'n "brug vægt" op i mindre sub use cases. Dette har vi illustreret i følgende use case diagram. Der er 2 aktører til simulatoren, en ekstern aktør, som bruger simulatoren via netværket, samt den normale aktør, som bruger de fysiske knapper på vægten.



Ifølge opgavebeskrivelsen, er det første skridt altid, at der skal oprettes forbindelse til vores vægt simulator via. telnet eller lignende før vægtens fysiske knapper, kan udnyttes. En precondition er derfor, at der skal være oprettet forbindelse til en klient via netværket. Nu har både den eksterne bruger samt den normale bruger mulighed for at udnytte vægtens funktioner.

Før den normale bruger kan svare på en RM20 ordre, skal der selvfølgelig være afsendt en RM20 ordre fra den eksterne bruger. Dette er derfor en precondition for at brugerne kan svare på RM20 ordren.

Løsnings domæne



Programmet består af 5 klasser, en data klasse, en controller/funktions klasse, en UI (boundary) klasse, en klasse der skal repræsentere den fysiske vægt samt en klasse, der indeholder de forskellige menuer, der er til rådighed i programmet.

Data klassens primære formål er at indeholde information om netværksforbindelsen i form af sockets og listeners. Controller/Funktions klassens formål er at håndtere input fra forbindelsen og udføre de korrekte funktioner. PhysicalWeight klasen skal virke som selve vægtenheden, hvor vægt ligges på og bliver taget af, samt udfører de andre essentielle funktioner, som der er på den virkelige vægt. Menu klassen bruges til at opbevare de forskellige menuer programmet kan vise (konsol baseret).

Sekvensdiagrammer

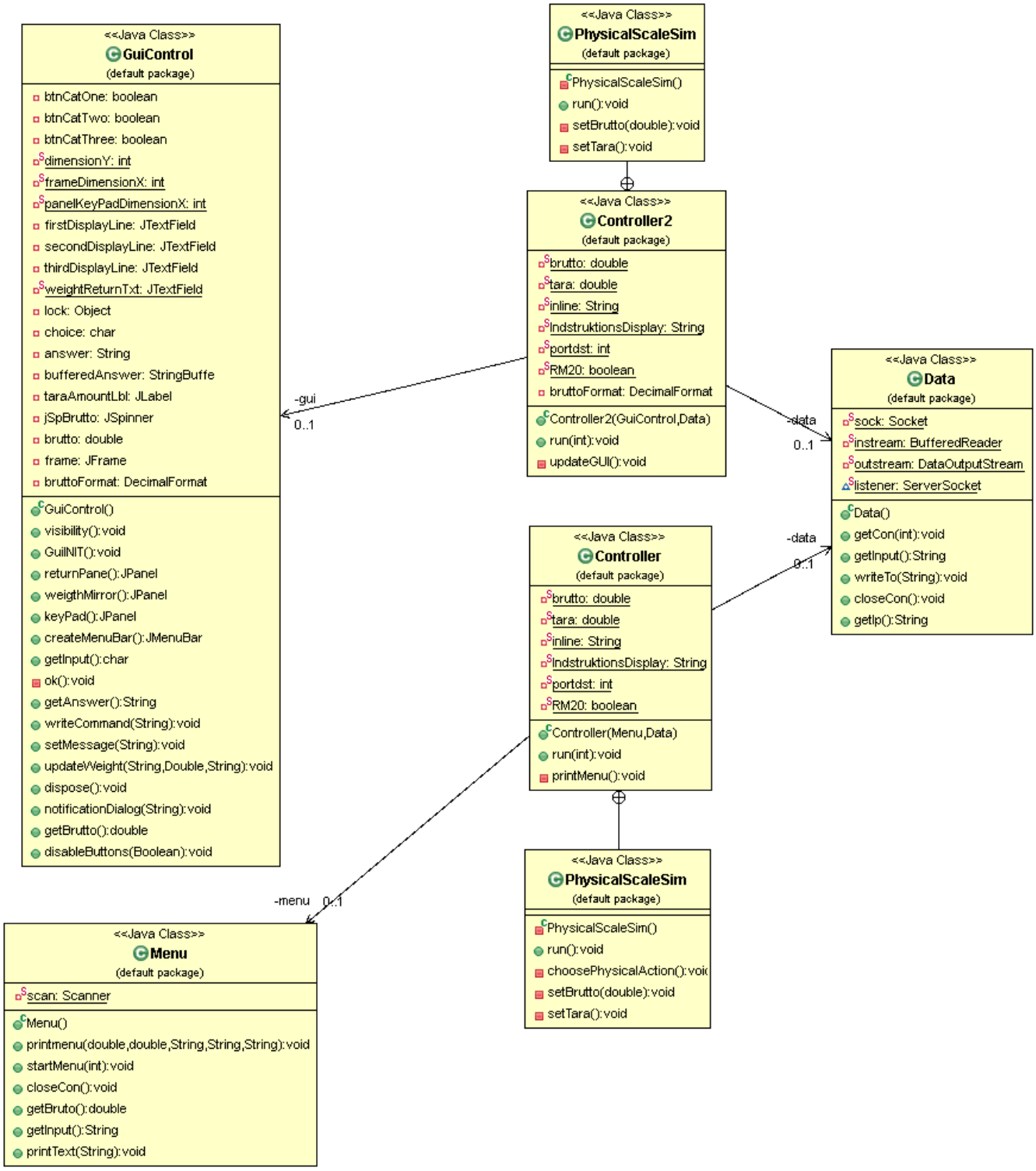
Et Sekvens diagrams hovedopgave er at beskrive et projekts metoders interaktion enten med sig selv eller med andre metoder. Dette gøres ved hjælp af en nøje beskrivelse af de enkelte instanser af metoderne og de mulige variabler de måtte medbringe i den beskrevet kommunikation. Ligeledes klarligger Sekvens diagrammet også mulige forgreninger i programmet såvel som betingelses afhængige valg.

Ved ethvert større og komplekst projekt må et Sekvens diagram ses som værende en nødvendighed for i hvert fald de essentielle metoder der bære program logikken, dette kan dog udelades for mindre simple metoder hvis overskuelighed er til at overkomme. Så set fra Sekvens Diagrammets logik til CDIO projekt 2's strukturelle omfang gør opgavens ret simple programstruktur ret hurtigt et Sekvens Diagram til en overflødig men ret ressourcekrævende del. Diagrammet ville blotlægge at de enkelte metoder som

befinder sig i programmet sjældent kalder andre end sig selv og at ”hovedformålet” med programmet ikke kræver meget andet end at læse og vurdere keyboard input eller ligeledes input fra en socket.

Det er dog tydeligt at Vægt emulatoren skal benyttes videre hen i forløbet og man kan hurtigt forestille sig langt flere krævende opgave som den kunne blive kaldt til at opfylde og på daværende tidspunkt da ville opgavernes stigende kompleksitet muligvis retfærdiggøre ”cost/benefit” forholdet for fremstillingen af diagrammet.

Klassediagram



Vi valgte ikke at bruge interfaces, da programmet var meget småt. Da vi skulle tilføje GUI'en fandt vi dog ud af, at interfaces ville have været en stor fordel. Vi har i denne opgave arbejdet udfra BCE modellen. Vi valgte BCE frem for 3-lagsmodellen, eftersom den eneste entitet, vi har, er Data, (vores server-side socket), og denne ikke har brug for et funktionslag da funktionaliteten er ret begrænset i programmet.. Inde i vores controller, har vi en inner class, PhysicalScaleSim, som implementerer interfacet Runnable. Denne klasse bruger vi til at oprette en tråd, som lytter på og behandler alt, der indtastes i consollen på den fysiske vægt, mens den anden tråd lytter på og behandler alt, der sendes til socket'en. Vi har lavet PhysicalScaleSim som en inner class, fordi vi kun bruger den inde i Controller klassen. Som det ser ud nu, er vores variabler statiske. Dette ville være et problem, hvis vi skulle have flere instancer af vægten, eftersom disse så skulle dele fx. netto værdi. Da det er meget sandsynligt, at vi på et tidspunkt skal have flere brugere tilkoblet vægten på en gang og dermed også flere instancer af Controlleren, er dette noget, vi vil ændre inden næste delprojekt.

Implementering

Vi har implementeret vores program således at det har to displays. Det skal dog bemærkes at disse er implementeret i et konsolmiljø. Det ene display viser netto vægten og det andet eventuelle instruktioner til brugeren. Til at kommunikere med brugere anvendes fire forskellige klasser, henholdsvis Socket, ServerSocket, BufferedReader og DataOutputStream. I starten af programkørslen udskrives noget tekst om at der ventes på en forbindelse og hvilken port der lyttes på. Som standard er denne port sat til port 8000, det er dog muligt at ændre denne ved at angive et portnummer som argument til programmet når det startes. Først oprettes en ServerSocket på denne port, og der ventes på at en klient opretter forbindelse til ServerSocket. Forbindelsen accepteres og der oprettes en Socket til forbindelsen. For at kunne læse fra og skrive til forbindelsen oprettes henholdsvis en BufferedReader og en DataOutputStream.

Vores vægtsimulator er således klar til anvendelse. Derfor udskrives en menu med informationer om nettovægten, eventuelle instruktioner, debugging info og understøttede kommandoer.

Herefter oprettes tråden choosePhysicalAction og denne tråd startes. Denne tråd simulerer den fysiske "del" af vægten. Denne tråd tager sig af de "fysiske" knappetryk. Da vi ikke har fysiske knapper kommer dette input i stedet fra konsollen. Løkken som lytter på input ligger i en uendelig løkke. Hvis vægten er i RM20 mode (altså hvis der ventes på svar til en RM20 ordre), sættes inputOk lig false og returværdien lig 0. Mens inputOk er lig false tjekkes om inputtets længde er mindre eller lig 8. Hvis det er det laves input strengen om til en int og inputOk sættes lig true. Hvis input strengen ikke kan konverteres til en int (fx. hvis man har skrevet bogstaver) eller hvis inputtets længde er større end 8, kastes en NumberFormatException. Denne exception fanges og der udskrives i konsollen en fejlmeddelse og der tages imod et nyt input. Når brugeren taster et korrekt input forsøges der at sendes en acceptmeddelse

til klienten som indeholder inputtet. Hvis forbindelsen til klienten ikke længere fungerer kastes en IOException. Herefter sættes RM20 mode til false og menuen udskrives.

```
char choise = input.charAt(0);
switch(choise)
{
case 'T':
    setTara();
    printMenu();
    break;
case 'B':
    double newBruto = menu.getBruto();
    setBrutto(newBruto);
    printMenu();
    break;
case 'Q':
    menu.closeCon();
    data.closeCon();
    System.exit(0);
default:
    System.out.println("Ugyldigt input");
    printMenu();

}
```

Hvis ikke vægten er i RM20 mode switches på første karakter af inputtet fra klienten. Hvis inputtet er "T" tareres vægten og udskrives menuen. Ellers hvis inputtet er "B", kaldes metoden getBruto() på Menu klassen som sørger for at tage imod en ny korrekt brutto vægt fra brugeren af vægten. Denne bruttoværdi returneres af getBruto() metoden, og metoden setBrutto() kaldes med den nye korrekte bruttoværdi som parameter, således at bruttovægten på vægten opdateres. Ellers hvis inputtet er "Q" udskrives til brugeren (på vægten/konsollen) at programmet er stoppet, forbindelsen til klienten lukkes ned og programmet terminerer. Alt andet input end T, B og Q medfører at udskrives "ugyldigt input" og at menuen udskrives.

Når programstart-metoden bliver brugt i Controller, bruger den porten som der er angivet. Den skriver en menu ud og lytter for forbindelser på den givne port. Hvis forbindelsen bliver oprettet udskrives Printermenuen og der startes en tråd til at lytte på input fra programmet.

Efter dette bliver en try sat i gang, den indeholder en while (true) lykke, som igen indeholder en try. Den prøver så at hente input fra den tilhørende klient.

```
public synchronized void programstart(int port)
{

    portdst = port;


    try
    {
        //Udskriver velkomst menu
        menu.startMenu(portdst);
        //lytter for forbindelser på portdst
        data.getCon(portdst);
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
    //Forbindelse er nu oprettet. Printermenuen udskrives.
    printMenu();

    //Der oprettes en ny tråd som skal lytte på input fra selve programmet
    new Thread(new PhysicalScaleSim()).start();

    try{

        while (true){
            try
            {
                //Henter input fra den tilsluttede klient
                inline = data.getInput().toUpperCase();
            }
```

Det næste er at der bliver tjekket om RM20 er i gang og den skal vente på svar. Hvis ikke fjerner den tekst fra displayet. Der bliver også tjekket om der skal skrives til displayet.

Ellers tjekker den om den skal sætte tara på vægten (nede t.v). Næst tjekkes der om der skal ændres i netto og den skal skrives ud, eller om der skal Sættes brutto og om brutto er skrevet så den kan forstå det, hvis ikke bliver der skrevet ES ud (nede t.h).

```
catch(NullPointerException e)
{
    //Hvis forbindelsen er tabt, skal programmet l
    inline = "Q";
}
//Hvis programmet afventer svar fra RM20 ordre
if(RM20)
{
    data.writeTo("RM20 I\r\n");
}
//Fjerner tekst fra display
else if (inline.startsWith("DW")){
    IndstruktionsDisplay="";
    printMenu();
    data.writeTo("DW"+"\r\n");
}
//Skriver tekst til display
else if (inline.startsWith("D")){
    if (inline.equals("D"))
        IndstruktionsDisplay="";
    else
        IndstruktionsDisplay=(inline.substring(2,
                inline.length()));
    printMenu();
    data.writeTo("DB"+"\r\n");
}
// Sætter tara vægt
else if (inline.startsWith("T")){
    data.writeTo("T " + (tara) + " kg "+"\r\n");
    tara = brutto;
    printMenu();

}
```

```
//Henter netto vaegt
else if (inline.startsWith("S")){
    printMenu();
    data.writeTo("S " + (brutto-tara)+ " kg " +"\r\n");
}
//saetter ny brutto baegt.
else if (inline.startsWith("B")){
    try
    {
        char c = inline.charAt(1);
        if(c == ' ')
        {
            String temp= inline.substring(2,inline.length());
            brutto = Double.parseDouble(temp);
            printMenu();
            data.writeTo("DB"+"\r\n");
        }
        else
            data.writeTo("ES"+"\r\n");
    }
    catch(IndexOutOfBoundsException e) {
        data.writeTo("ES"+"\r\n");
    }
    catch(NumberFormatException e)
    {
        data.writeTo("ES"+"\r\n");
    }
}
```

Hvis at der modtages en RM20 ordre laver den et array kaldt temp, og en count der sættes til 0. Den kører derefter den *inline* igennem som den fik. For hver " tæller den 1 op i count.
Hvis count kommer op på 6, bliver temp arrayet fyldt med den forskellige dele af den *inline* som den modtog. Hvis temp[1], som er beskeden der skal udskrives, er længere end 30, bliver der skrevet ud at det er en dårlig parameter, da den ikke skal kunne vise mere end 30 i længde. Hvis længden er okay bliver det skrevet ud, RM20 bliver sat til true, og menuen printes med besked om at den vente på RM20 8 ordre. Hvis count ikke er 6, bliver det også skrevet ud at den har modtaget en dårlig parameter. Ellers hvis den modtager Q, skriver menuen at den lukker, og data lukker forbindelserne.
Hvis den ellers modtager noget den ikke forstår skriver den "ES" ud.

```
//Modtager RM20 8 ordre
else if ((inline.startsWith("RM20 8"))){
    String[] temp;
    int count = 0;
    for (int i=0; i<inline.length(); i++)
    {
        if(inline.charAt(i)=='\"')
        {
            count++;
        }

    }
    if (count==6)
    {
        String delimiter = "\"";
        temp = inline.split(delimiter);

        if(temp[1].length()>30)
        {
            data.writeTo("RM20 L\r\n");
        }
        else
        {
            data.writeTo("RM20 B\r\n");
            RM20 = true;
            printMenu();
            menu.printText("Venter på RM20 8 ordre: "+temp[1]);
        }
    }
    else
        data.writeTo("RM20 L\r\n");
}
else if ((inline.startsWith("Q"))){
    menu.closeCon();
    data.closeCon();
    System.exit(0);
```

```
        else
        {
            printMenu();
            data.writeTo("ES"+"\r\n");
        }
    }
}
catch (Exception e){
    e.printStackTrace();
    //menu.printText("Forbindelse til klient tabt, programmet lukkes");

}
```

Når man laver metoder, som skal kunne køres af flere tråde, skal man være opmærksom på, at der kan opstå problemer, hvis to tråde "samtidigt" ændrer på en variabels værdi. Derfor har vi gjort metoder, hvor der ændres på ikke-lokale variabler syncronized, så de kun kan køres af en tråd ad gangen.

Test

Vi prøvede først at forbinde den vægt klient, vi kodede i datakommunikation til den rigtige fysiske vægt for at se, hvordan denne reagerede på forskelligt input, samt hvad de forskellige knapper gjorde og hvor mange cifre hvert display kunne indeholde. Derefter tjekkede vi, om vores vægt gav de samme resultater ved tilsvarende tests. Bemærk at vægten skal tilgås fra en client, som overholder telnet protokollen. Tilgår man vægten fra Putty, vil denne i forbindelse med et handshake sende en string til vægten. Når server socket'en i vægten læses første gang vil der altså både ligge stringen fra handshaket og første kommando sendt fra klienten. Første kommando vil derfor altid blive vurderet som ugyldig, når vægten tilgås fra Putty.

I løbet af vores test fandt vi en række fejl, som kan opdeles i følgende kategorier:

Forbindelse mellem klient og vægt

- Hvis en klient afbryder forbindelsen til vægten fås en null pointer exception på vægten.
- Hvis klienten afbryder forbindelsen til vægten for derefter at genoprette forbindelsen, modtager vægten ikke længere de kommandoer, klienten sender. Vægten virker dog stadig lokalt. (Fejlen opstår kun, hvis vægten ikke har været lukket ned fra forbindelsen blev afbrudt til den blev forsøgt genoprettet.)

Håndtering af ugyldigt input og utilsigtede situationer

- Input som: rm20 8 "(hvaddunuvilskrive), accepteres selvom det er forkert syntax.
- Efter RM20 er blevet sendt fra klienten, afventer vægten et input på den fysiske vægt simulator. Hvis dette input ikke er et tal, fås der en NumberFormatException. Dette skete fordi programmet prøvede at parse inputtet direkte til en int uden try/catch exception handling.
- Efter RM20 er blevet sendt fra klienten, afventer vægten et input på den fysiske vægt simulator. Hvis der sendes en anden kommando fra klinenten, før RM20 forespørgslen er blevet opfyldt, forsvinder teksten fra RM20 på den fysiske vægt simulator. Hvis brugeren nu indtaster en kommando på den fysiske vægt simulator, fås der en NumberFormatException hvis inputtet ikke er et tal. Dette sker, da den fysiske vægt simulator stadig afventer inputtet fra brugeren.

- Indtastes B 'tal' som kommando, registreres tallet for brutto korrekt. Hvis der indtastes B'tal', som er forkert syntax, så registreres fejlen ikke, men tallet sættes til brutto manglede det først indtastede ciffer. Dette skete, fordi der blev antaget, at når der var indtastet B, ville tallet for den indtastede brutto begynde 2 tegn efter b'et. Derfor blev der ikke taget højde for, hvis mellemrummet blev glemt.
- Indtastes B og intet andet fås en StringIndexOutOfBoundsException. Dette er fordi som i ovenstående tilfælde, at der blev antaget så snart et input startede med B ville brutto input stå 2 tegn efter B, i dette tilfælde står der ikke mere end B. Dermed vil programmet prøve at tilgå et sted i denne string som ikke findes.

"Fysiske" begrænsninger

- Brugeren kan skrive max 10 cifre som svar efter en klient har sendt RM20 til vægten, men der kan kun være 8 cifre på det display, brugeren skal skrive på.
- Teksten, der sendes med RM20 til vægten, kan være på over 30 tegn, dvs. det vil være muligt at sende en tekst, som ikke kan stå på displayet.

Alle ovenstående fejl blev rettet. Derudover kørte vi en automatisk test, hvori vi satte en klient til at sende en række kommandoer til vægten. Tiden der gik mellem kommandoerne blev sendt satte vi til at være et tilfældigt antal millisekunder mellem 0 og 1000. Vi testede så, om klienten fik svar tilbage, samt om disse svar stod i den rigtige rækkefølge. Begge dele var tilfældet. Siden alle fundne fejl er rettet, har vi heller ikke oplevet, at tråde (pga. manglende synkronisering) har ført til forkerte værdier af variable.

Afsluttende kommentar / konklusion

Vi har udviklet et program der simulerer en Mettler BBK vægt, vores simulator simulererer den fysiske del af vægten som tarering, sætning af brutto osv. Desuden har simulatoren også muligheden for at modtage kommandoer sendt over en netværksforbindelse. En af mulighederne der kun kan bruges via netværks kommandoerne, er muligheden for at kræve et input fra den fysiske bruger. Der kan kun være en klient forbundet til vægten ad gangen. Vægten blev først implementeret, så den kunne køre i consollen, og siden fik vi lavet en GUI til. Her ville det have været en fordel at have brugt interfaces, så overgangen fra consol til GUI havde gået hurtigere. Under udviklingen af vores program har vi både løbende men især til sidst testet, om det levede op til de stillede krav, hvilket vi vil mene, at det nu gør.