CDIO del 2

02324 Advanced programming

Deltagere:

Christin Holt, s120561	s073653	Emil Eriksen	
Jens Nielsen	s123115	Khan Noori	s122997
Kim Rylund	s123667	Magnus Vestergaard s113727	
Thomas Mortensen	s110795	Mathias Larsen	s113734

Lærer:
Mads Nyborg

16_02324										Controls hidden. F
Time-regnskab	Ver.2013									
Dato	Deltager	Design	Impl.	Test	Dok.	Andet	Ialt			
12/3/2013	Emil	1	0	1	0	0	2		Sum of Ialt	
-	Khan	2	0	0	0	0	2		Deltager	Total
-	Mathias	2	0	0	0	0	2		Christin	12
	Kim	1	0	0	0	0	1		Emil	11
	Jens	2	0	0	0	0	2		Jens	15
	Magnus	2	0	0	0	0	2		Khan	12
	Christin	2	0	0	0	0	2		Kim	10
18/03/2013	Emil	1	1	1	0	0	3		Magnus	11
-	Khan	2	1	0	0	0	3		Mathias	11
-	Mathias	1	2	0	0	0	3			
	Kim	1	1	0	0	0	2			
	Jens	2	2	0	0	0	4		Grand Total	82
	Magnus	2	2	0	0	0	4			
	Christin	2	1	0	0	1	4			
19/03/2013	Emil	0	1	1	0	0	2			
-	Khan	0	1	1	0	1	3			
-	Mathias	0	1	1	0	1	3			
	Kim	0	0	2	0	1	3			
	Jens	0	3	0	0	1	4			
	Magnus	0	0	2	0	0	2			
	Christin	0	0	0	1	1	2			
22/03/2013	Emil	0	0	2	1	1	4			
-	Khan	0	0	2	1	1	4			
-	Mathias	0	0	2	1	0	3			
	Kim	0	0	2	1	1	4			
	Jens	0	0	2	2	1	5			
	Magnus	0	0	2	1	0	3			
	Christin	0	0	2	1	1	4			
							0			
	Sum	23	16	23	9	11				

1. Indledning

Vi skal i denne opgave lave et interface, som en bruger kan benytte for at kommunikere med vægten. Vi har valgt at lave en GUI, som forhåbentlig er med til at gøre programmet mere overskueligt. Vi har i kurset 02325 allerede lavet en klient som kommunikere med simulatoren. Derfor skal vi have lavet programmet som kan simulere vægten. Vi kan derfor binde de to projekter sammen, ved at få dem til at snakke med hinanden.

2. Krav

Der ønskes designet en vægtsimulator med console interface.(Det sorte dos vindue.) Programmet skal simulerer en Mettler BBK vægt som ved øvelsen i 02325 i uge 2. Det vedlagte programskellet for vægtsimulatoren kan modtage ordre fra er styreprogram telnet, hyperterminal eller det program som i selv har skrevet i 02325.

1. Simulatoren skal lytte på port 8000, ved opstart fra Kommandolinjen skal denne default værdi kunne overskrives med en vilkårlig port nummer.

2. Tilføje funktionalitet så man på vægtensimulatoren kan taste tara.

3. Tilføje funktionalitet så man kan ændre belastning på vægten (brutto).

4. Implementer mulighed for at taste indtaste svar efter at vægten har modtaget ”RM20 8” ordren.

5. Overvej hvordan man sikrer at alle resurser lukker korrekt ned når programmet afsluttes.

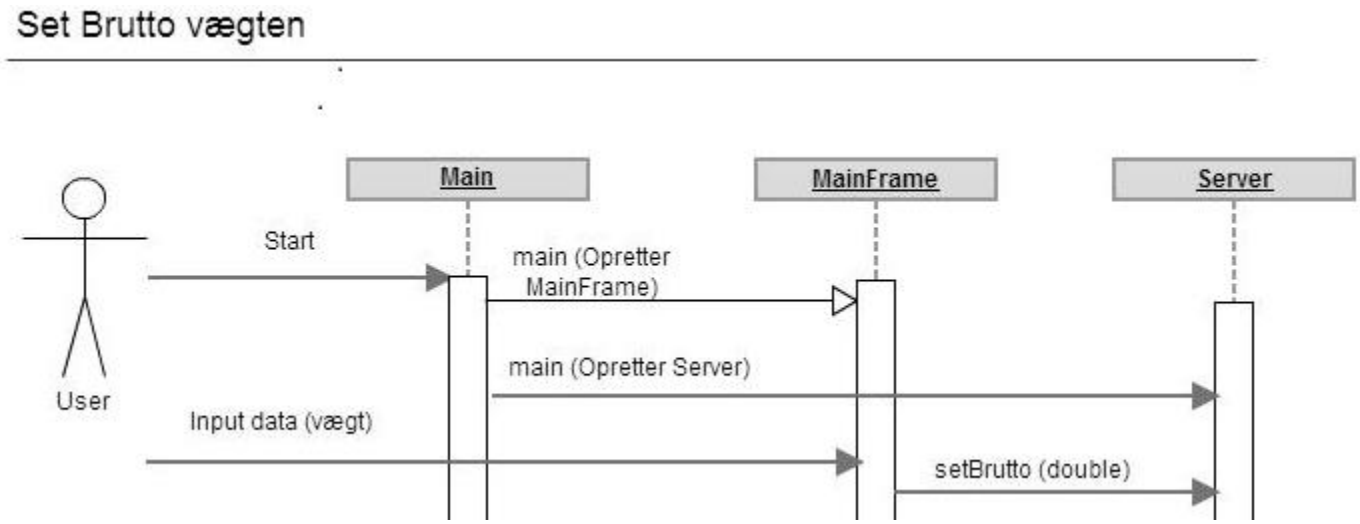
Her er en tabel der viser de kommandoer som simulatoren skal understøtte, hver kommando samt svarene sendes som en enkelt tekst linje afsluttet med \r\n.

Kommando der sendes til simulatoren	Svar fra simulatoren	Kommentar
S	S S x kg	Anmoder om den nuværende netto vægt. x erstattes med vægten i kilo, hvor punktum anvendes som decimal tegn.
T	T S x kg	Tarere vægten og retunere den nye tara. x erstattes med vægten i kilo, hvor punktum anvendes som decimal tegn.
DW	DW A	Fjern tekstmeddelelsen fra displayet og vend tilbage til visning af nettovægten.
D x	D A	Viser en tekstmeddelelse på vægtens display. x skal erstattes med den tekst der ønskes vist på

		displayet.
RM20 8 “w” “x” “y”	RM20 B RM20 A “z”	Viser tre tekstmeddelelser på vægtens display og returnere den indtastede værdi fra vægtens tastatur. w, x og y skal erstattes med de tre tekster der ønskes vist på displayet. z erstattes med værdien indtastet på vægtens tastatur. Det første svar “RM20 B” sendes øjeblikkeligt efter kommandoen er modtaget, det andet svar “RM20 A “z”” sendes efter successfuld indtastning på vægten.

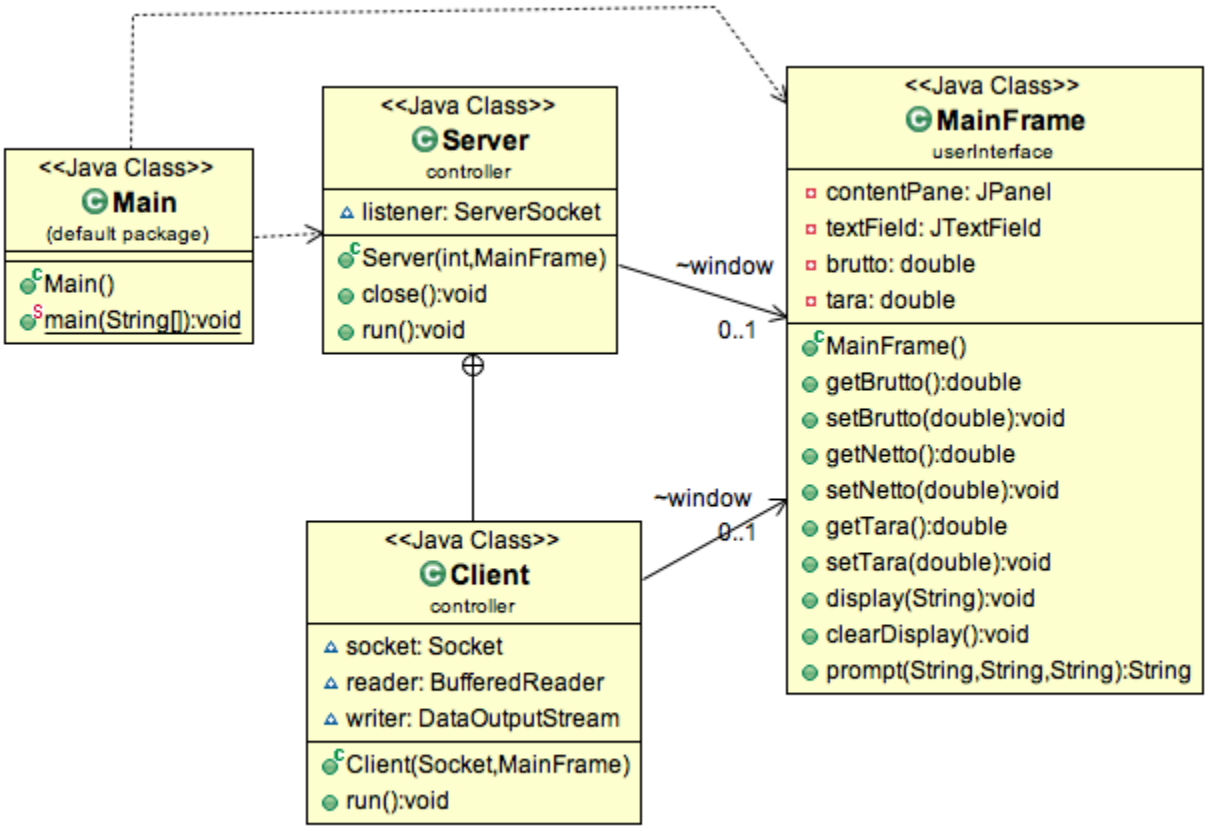
3. Design

Sekvens diagram.



I dette sekvensdiagram Ser vi hvordan en bruger starter programmet op og indtaster den bruttovægt den gerne vil simulere. Ydermere ser vi de forskellige metodekald der går igennem programmet for at den indtastede værdi bliver gemt på serveren. Da vi har et grafisk user interface ligger mange af metoderne inde i MainFrame men vi kan se ud fra diagrammet at Main først skal oprette både MainFrame og Server, før brugeren får lov til at indtaste sit input

Design klassediagram.



Design klassediagrams beskrivelse.

Som det kan ses, viser vores design klassediagram koblingen mellem vores klasser og hvordan de kommunikerer med hinanden. Programmet starter fra Main i Main server og Mainframe. Main server kører på port nr. 8000, som vi er blevet bedt om og det er muligt at ændre port nr. ved hjælp af et argument, når programmet starter. Klassen Serveren lytter til det port nr. vi har valgt og opretter en instans af klassen Klient

Klassen MainFrame er en brugeroverflade klasse og den skal simulere vægten og den håndterer interaktionen mellem brugere og programmet.
Klassen Client er en inner klasse og den er lavet private, fordi det skal kun serveren kende til.

4. Implementering

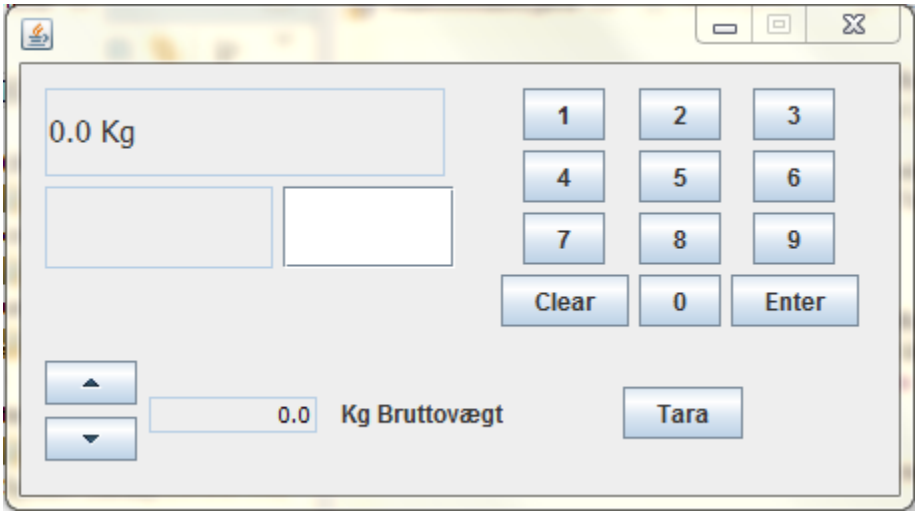
Design af GUI

Generelt

Til layoutet af vores GUI har vi benyttet 2 forskellige layout managers. Vi benytter GroupLayout som vores overordnede manager, og derudover benytter vi GridBagLayout, til organiseringen af det numeriske tastatur og nogle af tekstboksene. Grunden til at vi vælger at bruge to forskellige layoutmanagers, er at GroupLayout er utrolig fleksibel og kan bruges til at organisere den overordnede struktur, hurtigt og nemt. GridBagLayout er baseret på et gitter, hvilket giver en meget fast struktur, hvilket var godt til vores numeriske tastatur.

Problemer med datatyper

Ved ”pil-justeringen” af bruttolasten, løb vi ind i nogle problemer med datatyperne float og double. Disse to datatyper er designet efter nogle hastighedsprincipper, som gør at de nogle gange er en smule upræcise, når der regnes med kommatal. Dette medførte ind imellem nogle grimme tal. Vi prøvede derudover også at bruge Klassen `BigDecimal`, som er en del af `math`-pakken. Dette gav dog de samme problemer. Vi ender med at pilene på interfacet, justerer bruttolasten med +-1 kg.



Server

Server klassens konstruktør tager imod et portnummer samt det vindue der blev instantieret af main klassen. Server klassen starter en TCP server på den angivne port og begynder at lytte efter indkommende forbindelser i en baggrunds tråd. Når en klient tilsluttes instantieres der en client klasse med den indkommende forbindelse, hvorefter serveren fortsætter med at lytte efter nye klienter. Det er muligt for flere klienter at være tilsluttet samtidigt.

Her ses koden for server klassens konstruktør.

```
public Server(int port, MainWindow window) throws IOException {
    if (window == null) { //Tjekker om window eksisterer.
        throw new IllegalArgumentException("Window is undefined.");
    }
    listener = new ServerSocket(port); //Hvor den lytter.
    this.window = window;
    start();
}
```

Her ses koden der acceptere indkommende forbindelser.

```
public void run() {
    try {
        while (true) {
            new Client(listener.accept(), window);
        }
    } catch (IOException e) {
    }
}
```

Client klassen åbner en reader samt en writer for streamen, hvorefter den venter på at modtage en kommando fra klienten i en baggrunds tråd.

Her ses koden for client klassens konstruktør.

```
public Client(Socket socket, MainWindow window) {
    System.out.println(" " + socket.getInetAddress().getHostAddress() + ": Connected.");
    this.socket = socket;
    this.window = window;
    try {
        reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        writer = new DataOutputStream(socket.getOutputStream());
        start();
    } catch (IOException e) {
    }
}
```

Et udpluk af koden der behandler kommandoerne ses her.

```
public void run() {
    Pattern patternRM20 = Pattern.compile("^RM20 8 \"([^\"]*)\" \"([^\"]*)\" \"([^\"]*)\"$");
    try {
        while (true) {
            String line = reader.readLine();
            if (line == null) {
                break;
            }
            System.out.println(" " + socket.getInetAddress().getHostAddress() + ": Received: " + line + ".");
            Matcher matcherRM20 = patternRM20.matcher(line);
            if (line.equals("S")) {

                // Retuner netto vægt.
                writer.writeBytes("S S " + ("" + window.getNetto()).replace(",", ".") + " kg\r\n");

            } else if (...) {

                ... se resterende kode i kildekoden ...

            }
        }
    } catch (IOException e) {
    }
    System.out.println(" " + socket.getInetAddress().getHostAddress() + ": Disconnected.");
}
```

RM20:
Vi har et regulært udtryk (en regex) i vores projekt, som vi bruger til at modtage de 3 beskeder fra RM20 kommandoen.
Vores redex ser sådan her ud (Linje 75 i Server.java):
^RM20 8 \"([^\"])\" \"([^\"]*)\" \"([^\"]*)\"$*
Det første tegn ^ og den efterfølgende tekst *RM20 8 \”* tjekker om starten på det den skal tjekke matcher RM20 8 “” “” “” og dollar-symbolet tjekker at der ikke er mere efter det.
Mellem gåseøjnene er der tegnene *([^\"]*)*, () gemmer en tekst, mellem det mellem paranteserne specificerer hvad paranteserne gemmer. [^\”] betyder at der bliver gemt et hvilket som helst tegn, så længe dette tegn ikke er et gåseøje, og stjernen bagefter gør at det ikke bare er et tegn som () gemmer, men at der bliver gemt mellem 0 og uendelige tegn. Det stjernen helt præcist gør, er at den gentager den forrige så mange gange det er muligt, dvs. indtil der kommer nogle gåseøjne.

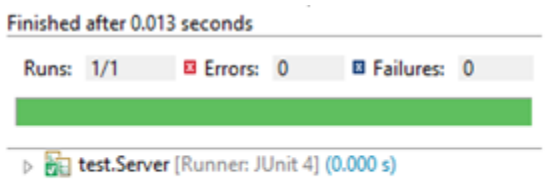
Server.Client.run():
Starter med at lave det regex mønster som der står noget om i forrige afsnit.
Derefter går koden ind i en try-block, som catcher en IOException.
Inde i try-blokken starter en evig while-loop, som starter med at læse klientens besked (den klient som er forbundet til simulatoren over netværket).

Hvis klientens besked ikke indeholder noget, dvs. = null, så kommer man ud af loopet og den lytter ikke længere til klientens input, grunden til dette er, at det eneste tilfælde hvor klienten vil sende et null, er hvis han disconnector fra simulatoren.
Efter der er tjekket om klienten er blevet disconnected, så bliver der lavet en match over det som klienten har sendt i forhold til regex-mønsteret.
Derefter bliver der tjekket om det klienten har sendt er: “S”, "T", “DW”, “D ” efterfulgt af en besked, eller “RM20 8 ” efterfulgt af 3 strenge som hver især er indkapslet i gåseøjne..
Hvis “S” så returneres “S S ” efterfulgt af vægten og “ kg”.
Hvis “T” så bliver tara på vægten sat lig med den nuværende brutto på vægten, og der bliver returneret “T S” til klienten.
Hvis “D ” efterfulgt af en streng, så vil strengen blive vist på vægtens display, og “D A” bliver returneret til klienten.
Hvis “DW” så bliver der fjernet en besked fra vægten, og vægten selv viser netto-vægten. “DW A” bliver retuneret til klienten.
Hvis “RM20 8 ” efterfulgt af 3 strenge, så vil der dukke en besked op på vægten i form af et spørgsmål, og vægten vil returnere et tal til klienten, som en person har stået og indtastet på vægten.

Design af GUI

5. Test

Programmet er testet både ved hjælp af black-box testing og ved udvalgte J-Unit test. Black-box testingen er foregået på normal vis, med vurdering af output i forhold til indput. J-unit testene er lavet på udvalgte metoder, som kan sikre at metoderne virker efter hensigten.
Den første J-unit test er lavet som en test på server-klassen og tjekker for forbindelser fra forskellige porte, vha. de metoder server-klassen indeholder. Der bliver bl.a. prøvet både med højeste (65535) og laveste (0) port, som forventes at fejle, da den som stadard er sat til port 8000:

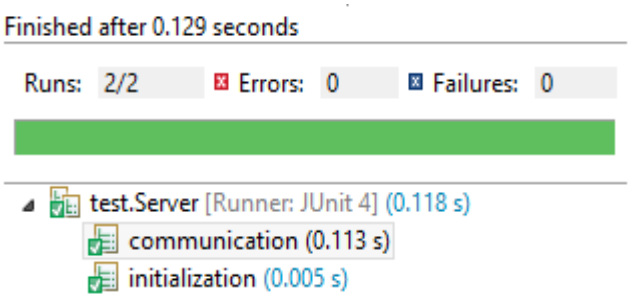



```
public class Server {

    @Test
    public void newServer() {
        controller.Server server = null;

        // For lav
        try {
            new controller.Server(-1).close();
```

Testen giver ingen fejl, og vi kan derfor regne med at server-klassen fungere, som vi forventer.

En anden test vi har oprettet, er en test, som afprøver kommunikationen mellem serveren og en klient. Testen forløber uden nogle problemer, og al kommunikationen foregår som forventet.

Mere indgående oprettes der en ny mainframe og en ny server. Derefter gives der de forskellige kommandoer til "vægten", for at se om om den udregner brutto, netto og tara rigtigt. Vi opnår ingen fejl og vi kan derfor også mere sikkert sige, at kommandoerne vi giver til serveren virker.



6. Konklusion

Vi skulle lave en server, som lyttede på port 8000, men som kunne overskrives ved starten af programmet. Programmet har ikke været det sværeste at sætte op, mens sammenspillet mellem GUI'en og selve serveren var lidt sværere. Det GUI'en står for er, at modtage indputs og sende dem videre til selve serveren. Serveren lytter på port 8000 ved hjælp af Treads, hvilket vil sige at den hele tiden venter på svar, fra en klient, som prøver at komme i kontakt med port 8000.