PROSJEKTOPPGAVE INFORMATIKK

2012

Emil Niclas Hofseth | Informatikk | November 12, 2012

INNHOLDSFORTEGNELSE

INTRODUKSJON

Denne rapporten er et vedlegg til kildekoden pakket i ZIP-filen «2012-11-12 (U) Prosjektoppgave Hofseth v.1.0».

Hensikten med rapporten er å gi et innblikk i fremgangsmåten for programmeringen og en beskrivelse av informasjonsflyten i programmet.

BAKGRUNN

Denne høstens emne «Informatikk» har et prosjektarbeid som avsluttende oppgave. Oppgaven var som følger:

> «Vi skal lage et system som simulerer overvåkning av en prosess. Vi skal registrere temperatur og fyllingsgrad i en tank. I tillegg kan systemet generere alarmer som skal presenteres for brukeren.»

I tillegg til dette ble det gitt noen nærmere retningslinjer for utførelsen av server og klient, samt forslag til protokoll mellom server og klient.

Java er som kjent et programmeringsspråk som bygger på «C». Da undertegnende har en noe utradisjonell rekkefølge på emnene i studiet, har jeg enda ikke tatt «Digitale Systemer 2», hvor «C» blir undervist. Det nærmeste jeg på forhånd har kommet programmering er sporadisk «C++» -ekvivalent i Arduino-programmering, samt nylig påbegynt «Assembly» -programmering i emnet «Digitale Systemer 1». På bakgrunn av denne mangelen på erfaring og kunnskap, har fremdriften i programmeringen vært svært lite strømlinjeformet og jeg har ved flere anledninger måtte «gå tilbake til scratch» for å starte på nytt. Læringskurven har vært svært bratt, og hver gang jeg har lært noe nytt har jeg gått tilbake til kildekoden for å implementere dette. Oppsummert har dette ført til at kildekoden først fant sin nåværende form kun få dager før innlevering.

Dette prosjektet har virkelig økt min kompetanse innen Java. Jeg har lært mye nytt, og det har blitt stadig artigere jo mer jeg har lært og fått til. Det skal nå bli greit å levere inn, og jeg tror samboeren min syntes det blir hyggelig å se meg igjen også.

FREMGANGSMÅTE

Metoden jeg valgte for design og uttenking av prosjektet var i svært liten grad bærekraftig: Jeg hev meg rett ut i kodingen og møtte stadig veggen av lav kompetanse. Dette gjorde at jeg mang en gang var nødt til å snu, for å gå tilbake til start og prøve på nytt. Mange timer og dager har jeg i stedet for å programmere, lest og sett undervisning-videoer på internett, for å søke kunnskap som jeg kan benytte meg av i prosjektet. Og jo mer jeg har lært jo raskere og bedre (etter mitt skjønn) har kodingen blitt. De siste par dagene har det gått i (relativt) høyt tempo, og jeg venter på at Google eller Microsoft snart ringer på døren …

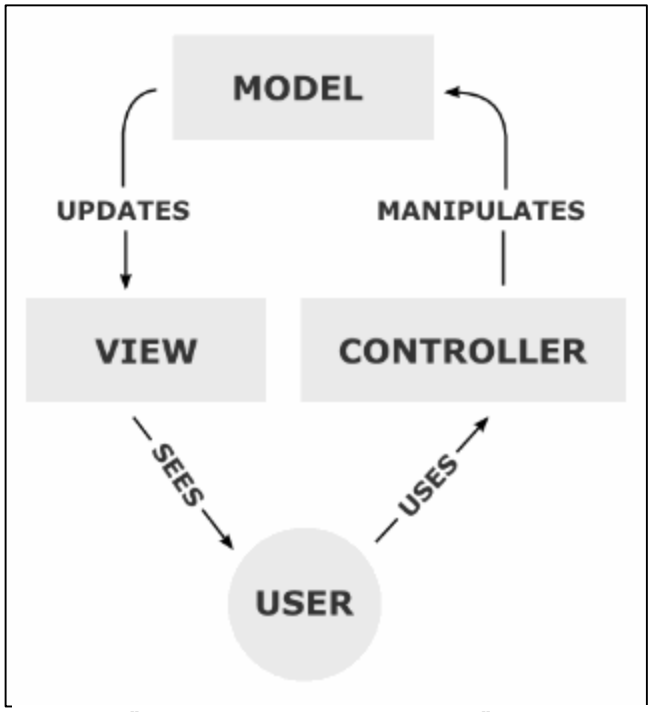
Spøk til side. «Lesson learned» er uansett at en god plan må til; hvor kodingen tar form på norsk, uten Java-syntaks, hvor man setter opp alt som må gjøres og hvordan det gjennomføres. Først når dette er klart kan en begynne på selve programmeringen. Denne idéen fikk jeg ikke bare basert på mine feil, men det er også noe vi nettopp lærte i «Digitale Systemer 1». Denne kunnskapen tar jeg med videre til mitt neste prosjekt, og får trøste meg med at dette prosjektet kom seg i land til slutt på tross av mangelfull planlegging.

Jeg har for å starte en god vane, programmert på engelsk, og vil også i denne rapporten benytte engelske fag-utrykk. Jeg har også forsøkt å kommentere så mye som mulig, vel vitende om at det ikke kan kommenteres for mye når dagen kommer at en skal gjennomgå koden på nytt. Metodenavn og variabler er også valgt for å gjøre lesingen av koden så enkel som mulig.

PROGRAM DESIGN

MODEL VIEW CONTROLLER

I starten var oppbygningen min av programmet svært rotete, og objekter ble laget og sendt frem og tilbake mellom klassene. Det var i det hele svært uoversiktlig. Etter mye lesing kom jeg frem til at jeg ville prøve meg på et oppsett basert på «MVC – Model View Controller»(Figur 1). Dette oppsettet deler opp programmet, slik at den viktige koden(«Model») er skilt fra det mindre viktige brukergrensesnittet «View». For å kontrollere samhandlingen mellom disse benytter jeg meg av en «Controller». Denne modellen gjør at objekter kun trenger å lages en gang og ikke er avhengig av hverandre i hverandres konstruktører når de blir til.



FIGUR 1 - "MODEL VIEW CONTROLLER"

LYTTERE OG EVENTS

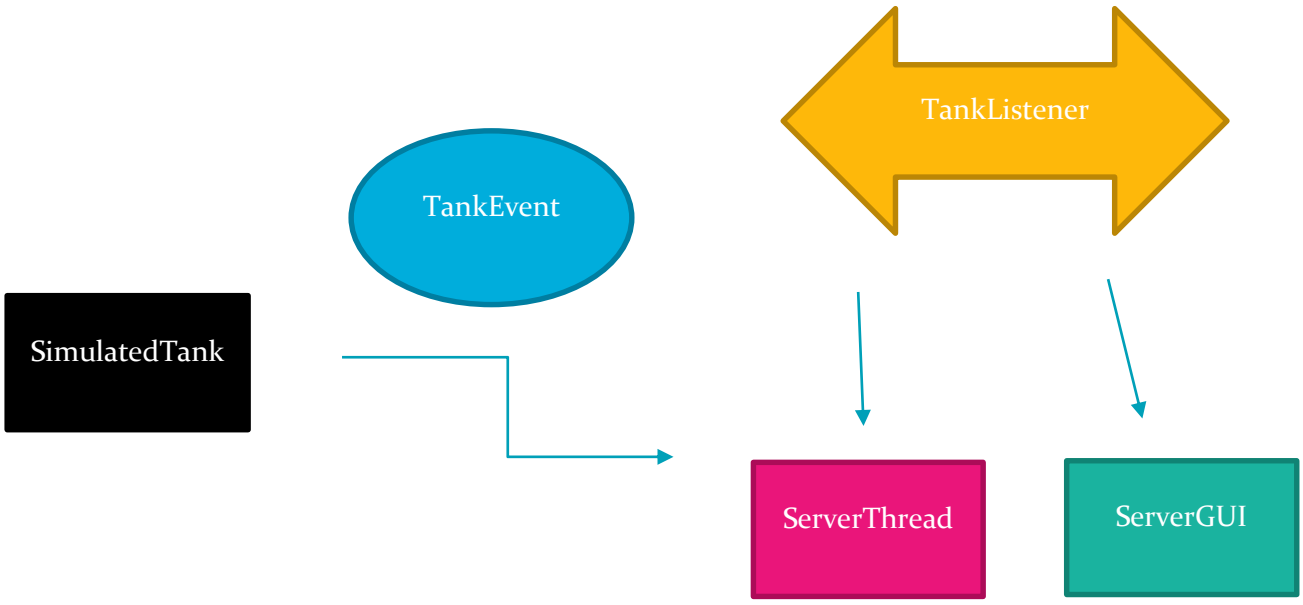
I programmet har jeg utstrakt bruk av egenproduserte «events» og lyttere. Disse «events» er arvet av «EventObjects» og er spesialtilpasset sine formål. I eksempelet under ser vi en «TankEvent» laget for å transportere data fra den simulerte tanken/målerne.

```
private List<LogEntry> log;
private double tankLevel;
private double tankTemp;
private Timestamp dtg;
private int id;

public TankEvent(Object source, List<LogEntry> log, int id, Timestamp dtg,
        double tankLevel, double tankTemp) {
    super(source);
    this.log = log;
    this.dtg = dtg;
    this.tankLevel = tankLevel;
    this.tankTemp = tankTemp;
    this.id = id;

}
```

FIGUR 2 - "TANK EVENT"

Lytterne er «interfaces» som er utformet for å transportere events. Når f.eks. den simulerte tanken lager en ny TankEvent, vil et «interface» kalt «TankListener» «sende» denne eventen til de som måtte ønske. I dette eksempelet vil eventen bli send til serverens grafiske brukergrensesnitt og til server-tråden, som igjen tar ansvar for å sende den til klientene.

«TankEventen» har en rekke «getters» som andre klasser kan benytte seg av for å trekke informasjon ut av eventen. De kan f.eks. bruke «.getLog()» om de ønsker å få ut objektet «log» fra eventen.



Programmet er oppdelt i fire pakker, to for klienten og to for serveren. Der igjen er pakkene oppdelt i «model» og «gui». Det er hhv. ServerApp og ClientApp som inneholder main-metodene. Det er også disse *App-klassene som fungerer som kontrollere, med unntak av noen ekstra kontroller-funksjoner i enkelte klasser, på grunn av simulering osv.

Jeg har ikke klart å gjennomføre «MVC» til det fulle, men prøver å etterstrebe dette så langt min kunnskap gjør det mulig.

BRUKERGRENSESNITT

FELLES

I begge brukergrensesnittene har jeg benyttet «GridBagLayout», denne tok seg tid å lære, men ihht. Det jeg har lært er det også blant de mer kraftige og tilpasningsdyktige layout-ene til SWING.

Jeg har også lagt til «mnenonics», eller tastatur-snarveier, slik at det skal gå raskere å navigere i programmet.

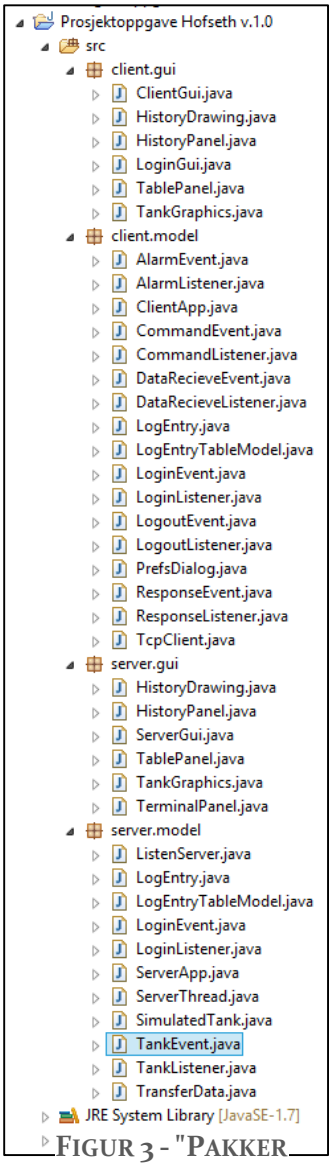
SERVER

Serverens brukergrensesnitt er av ren informativ art. En serveradministrator vil kunne overvåke tankens nivå og temperatur, grafisk og i tabell-form. Server-terminalen inneholder også en oversikt over hvilke klienter som har logget seg på, eventuelt om noen har forsøkt å logge seg på med et brukernavn som eksisterer, eller om de har benyttet feil passord. Se Figur 4.

KLIENT

Klientens brukergrensesnitt (se Figur 5) er i tillegg til å opplyse om tankens tilstand i form av grafisk og tabellarisk form, programmert til å kunne endre enkelte verdier. Den har også en fil-meny og innstillings-vindu (se Figur 6).

FUNKSJONER



FIGUR 3 - "PAKKER OG FIL-OVERSIKT"

«STAY ALIVE»

For at ikke brukere skal overbelaste serveren ved å glemme å logge av, vil serveren etter en periode (60 sekunder, pga. demonstrasjonen) sende en forespørsel til klienten om brukeren fortsatt vil være pålogget. Svarer ikke brukeren på denne i løpet av 60 sekunder, vil brukeren automatisk logges av.

ALARMER

Serveren kan programmeres til å sende alarmer. For demonstrasjonsøyemed er det kun satt én alarm, og det på tanktemperaturen. Når denne overstiger 48 grader det sendes en alarm til klienten og det vil sprette opp en varsel-boks som brukeren selv må lukke.

INTERVALLER OG BATCH

Brukeren kan i innstillings-vinduet velge hvor ofte serveren skal sende tankmålinger. Dette påvirker så klart kun denne klienten og ikke andre som måtte være tilknyttet serveren. Ønsker klienten å motta en oppsamling av målinger, kan dette gjøres ved å velge «batch» og deretter spesifisere lengden på tidsperioden mellom hver sending. Grunnet demonstrasjonen så er alle tidsintervaller satt til lave tall.

BRUKER-ADMINISTRASJON

Ved innlogging vil brukernavn og passord bli sjekket opp mot «databasen» på serveren. Brukeren vil bli informert om brukernavn er for kort eller ikke funnet, samt om passordet er for kort eller ikke funnet tilknyttet brukernavnet. Ved utlogging vil klienten gi beskjed til serveren, slik at den kobler ned på korrekt måte.
Dette vil også skje om brukeren lukker klient-vinduet uten å logge ut, klienten vil, takket være en «WindowListener» sende «close» i det vinduet lukkes.

Til informasjon er gyldige innloggingskombinasjoner «emilniclas:emilniclas123», «guest:guest123» og «admin:admin123» .

SIMULERT TANK

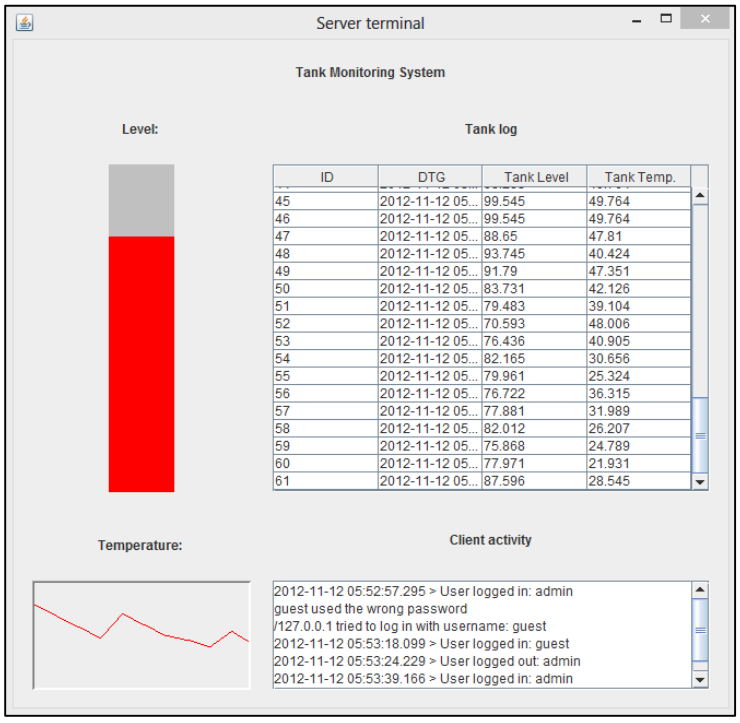
Den simulerte tanken lager verdier for id, nivå og temperatur, samt et tidsstempel for når målingen blir tatt. Det er dette tidsstempelet og id-en som følger målingene hos både server og klient. Dette er en fordel slik at man har full oversikt over målingene.

For øvrig er tanken full ved verdien 100, tom ved verdien 0, og temperaturen kan variere fra 0 til 50. «Sensoren» leser av disse verdiene hvert sekund.

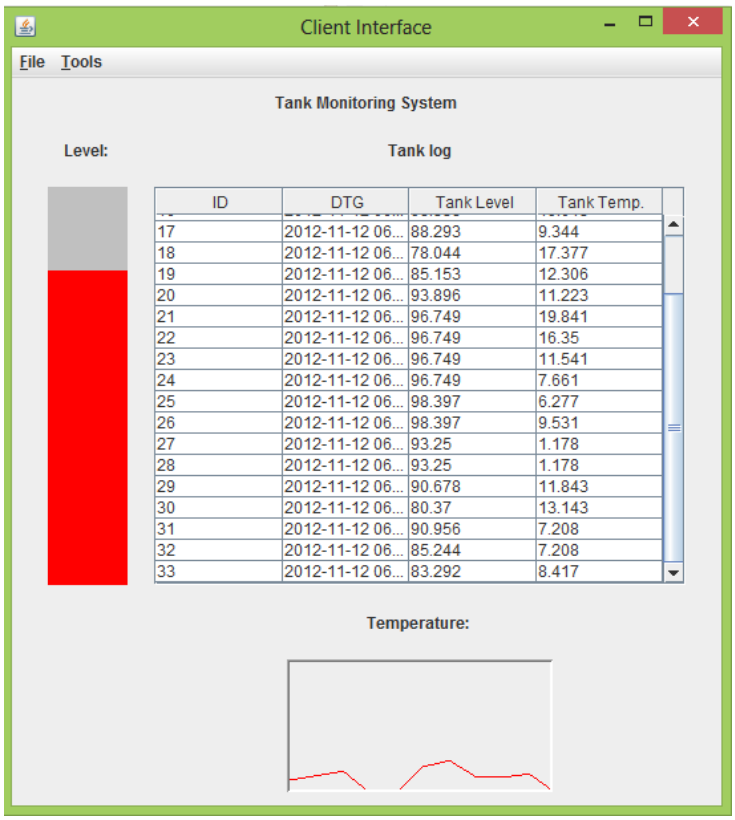
DATABASE

Målingene, blir lagret som en «LogEntry» og deretter lagt i en «LinkedList». Dette var opprinnelig lagret i en mySQL-database satt opp på en lokal Ubuntu-server hjemme hos undertegnende. Microsoft Access-database ble forsøkt, med hell, helt til jeg oppdaterte til Windows 8, og kombinasjonen 64-bits OS og 32-bits ODBC-driver ble for mye (selv om Windows 7 64-bit klarte det!). Java-SQL-kombinasjonen virket derimot upåklagelig etter en dags konfigurering. Ikke helt Informatikk-pensum, men jeg har i alle fall lært mye om java-SQL-interaksjon. For å redusere risikoen for «stryk», valgte jeg å ikke gå for denne løsningen på innleveringen – da det blir for mange risikofaktorer ved VPN, NAT og/eller en åpen mySQL-database mot internett.

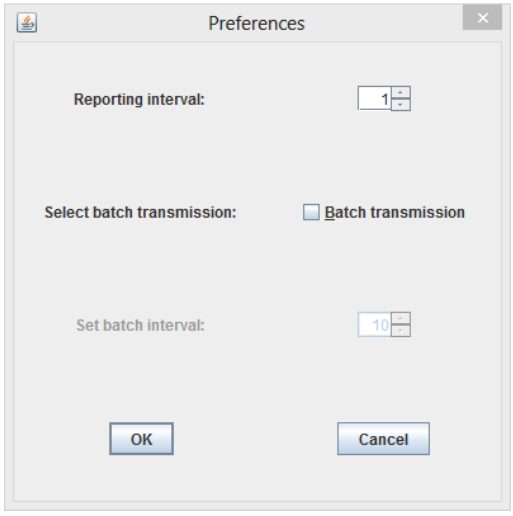
YTTERLIGERE FIGURER



FIGUR 4 - "SERVERENS BRUKERGRENSESNITT"



FIGUR 5 - "KLIENTENS BRUKERGRENSESNITT"



FIGUR 6 - "KLIENT-INNSTILLINGER"