NORGES TEKNISK-NATURVITENSKAPELIGE UNIVERSITET
INSTITUTT FOR DATATEKNIKK OG INFORMASJONSVITENSKAP

Faglig kontakt under eksamen:
Dag Svanæs, Tlf: 73 59 18 42

EKSAMEN I FAG
TDT4180 - MMI
Lørdag 26. mai 2012
Tid: kl. 0900-1300

Bokmål

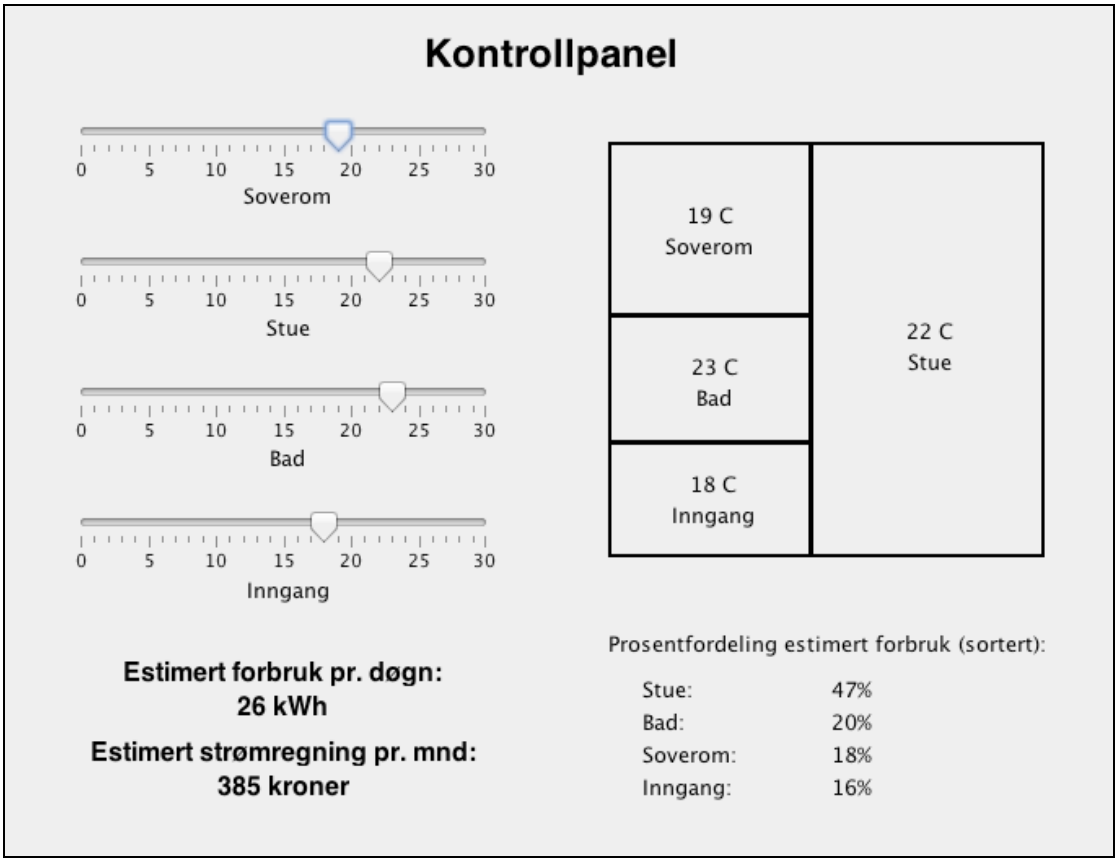
Sensuren faller 18. Juni

Løsningsforslag

Hjelpemiddelkode: **D** Ingen trykte eller håndskrevne hjelpemidler tillatt.
Bestemt enkel kalkulator tillatt.

Kvalitetssikret: Hallvard Trætteberg

Oppgave 1 (30%) Grensesnittdesign



Du er ansatt som konsultent i et IT-firma som har fått i oppdrag å utvikle brukergrensesnittet for klimastyring i en bygård med utleieleiligheter. Alle leilighetene er like. Oppdragsgiver er eier av bygården. I hver leilighet skal det være et kontrollpanel på en liten berøringsfølsom skjerm som styrer termostatene i den leiligheten. Det er et uttalt ønske fra huseier at systemet skal gjøre leieboerne bevisste sitt strømforbruk.

Bildet over viser en skisse til skjermbilde som er laget når du kommer inn i prosjektet. Leiligheten har fire rom: Soverom, Stue, Bad og Inngang. En glidebryter (”slider”) brukes til å sette ønsket temperaturen i hvert rom. Basert på ønsket temperatur så beregner programmet automatisk estimert forbruk pr. døgn og estimert strømregning pr. mnd. for leiligheten. I tillegg så lister den prosentvis fordeling av strømforbruket på de fire rommene, sortert på prosentandel.

Estimatet er basert på følgende data:

Gjennomsnittlig utetemperatur:	10 C	(Utetemp)
Forbruk pr. dag pr. m^2 pr. grad	0,05 kWh	(Forbruksfaktor)
Strømpris:	0,50 Kr/kWh	(Pris)

Soverom:	10 m^2
Stue:	20 m^2
Bad:	8 m^2
Inngang	18 m^2

Forbruksfaktoren angir altså gjennomsnittlig forbruk i kWh pr. dag for 1 m^2 boareal med en temperaturforskjell på en grad mellom innetemperatur og utetemperatur.

Estimert forbruk for et rom pr. dag er da gitt av:
(Innetemp – Utetemp) * Kvadratmeter * Forbruksfaktor.

Dersom Innetemp <= Utetemp så settes forbruket til null.

Dette gir for eksempel for stuen med en innetemperatur på 22 grader:
(22 – 10) * 20 * 0,05 = 12 kWh pr. dag.

a. "Affordance"
Forklar begrepet "affordance" slik det brukes av bl.a. Don Norman og i læreboka. Hvorfor er begrepet nyttig i interaksjonsdesign? Diskuter mulige svakheter i brukskvaliteten (brukervennligheten) i skissen til kontrollpanel som kan belyses ved begrepet "affordance". Foreslå eventuelle forbedringer.

Affordance er den bruk et objekt signaliserer gjennom sin form og farge. Det finnes forskjellig typer affordance: gitt av menneskekroppen, kulturelt, og kontekstuelt.

Relevansen for interaksjonsdesign er at det er viktig visuelt signalisere hva som kan trykkes på i et grafisk brukergrensesnitt. Spesielt er dette viktig når man ikke har en 3D form som for eksempel i produktdesign (for eksempel Cola-flasken). I en skjermflate så må man skille "forgrunn" (det som kan trykkes på) fra "bakgrunn" (det som ikke kan trykkes på) v.h.a. form, farge og tekst. Problemer med manglende "affordance" er veldig vanlige i skjermdesign, og er en av grunnene til at det er viktig å bli enige om standardelementer i GUI.

I forhold til forslaget til kontrollpanel så finnes det konvensjon om hvordan slidere skal se ut. Dette gir god affordance for disse. I oversiktskartet over de enkelte rommene så er det et tall ved hvert rom. Konvensjonen sier at dette kun er output, men det kan lett finnes brukere som kan finne på å prøve å trykke på disse for å sette temperaturen v.h.a. tallverdier. Det er ikke lett å vite hvordan man kan unngår dette v.h.a visuelle hjelpemidler. Den enkleste løsningen er antagelig å implementere denne funksjonaliteten slik at brukergrensesnittet oppfører seg slik en del brukere vil forvente at det gjør.

Vurdering:
D: Kun forklaring av affordance uten eksempel.
C: Forklarer affordance og diskuterer det i forhold til designcaset.
B: Typer av affordance. Forgrunn/bakgrunn (trykkbart/ikke trykkbart) i skjermbildet. Foreslår løsninger.
A: En veldig god diskusjon.

b. Konseptuell modell

Forklar begrepet konseptuell modell (”conceptual modell”). Hvorfor er begrepet nyttig i interaksjonsdesign? Diskuter mulige svakheter i brukskvaliteten i skissen til kontrollpanel som kan belyses ved begrepet konseptuell modell. Foreslå eventuelle forbedringer.

En konseptuell modell er den mentale modellen vi ønsker at brukeren skal ha av virkemåten og strukturen til et produkt. Dette begrepet er relevant fordi det er viktig å designe hvordan brukeren forstår strukturen og virkemåten til det vi lager.

Eksempel: En termostat.
Brukeren må forstå den grunnleggende virkemåten, ellers så vil vi få adferd som er uønsket. En termostat kan forstås av brukeren på flere måter. Den korrekte måten, d.v.s. den som passer med designerens konseptuelle modell, er at man setter ønsket temperatur og så slår varmekilden av eller på avhengig av om målt temperatur er over eller under ønsket temperatur. Forenklet så kan man si at man setter ønsket temperatur og så blir det den temperaturen. En alternativ mental modell er at man vrir på en knapp, og jo høyere man vrir den jo mer varme får man, altså som på en komfyr. Dersom er bruker har feil mental modell så kan det for eksempel føre til at man kommer hjem i et kaldt hus og setter termostaten på 40 grader fordi man ønsker å få det varmt så fort som mulig. Hvis man da drar bort en halv dag så kommer man hjem til 40 grader, og ikke 20 som man egentlig ønsket. Feil mental modell kan føre til uønsket adferd. Den ”riktige” mentale modellen kalles for konseptuell modell. I utformingen av brukergrensesnitt er det viktig å formidle den konseptuelle modellen på en slik måte at brukeren får den riktige mentale modell.

I skissen til kontrollpanel så er det ikke kommunisert hvordan estimatene beregnes. Dette er skjult for brukeren, og kan lett oppleves som ”magisk”. Den eneste måten brukeren kan få en forståelse av virkemåten er gjennom å eksperimentere med forskjellige verdier av ønsket temperatur. For mange brukere er dette antagelig ok, men et mulig forslag til forbedring er å ha en ”avansert” knapp der brukere som ønsker det kan ”titte under panseret” og få forklart for eksempel at estimatet er basert på en gjennomsnitts utetemperatur på 10 grader C.

Vurdering:
D: Kun forklaring uten eksempel. Dårlig analyse
C: Et eksempel som illustrerer det. Forklarer relevans for design.
B: Får fram at det er modell av virkemåten, evt. sammenligner med mental modell. Godt forslag til forbedret design.
A: Ekstra dybde i analyse og eksempel i forhold til å vise relevans.

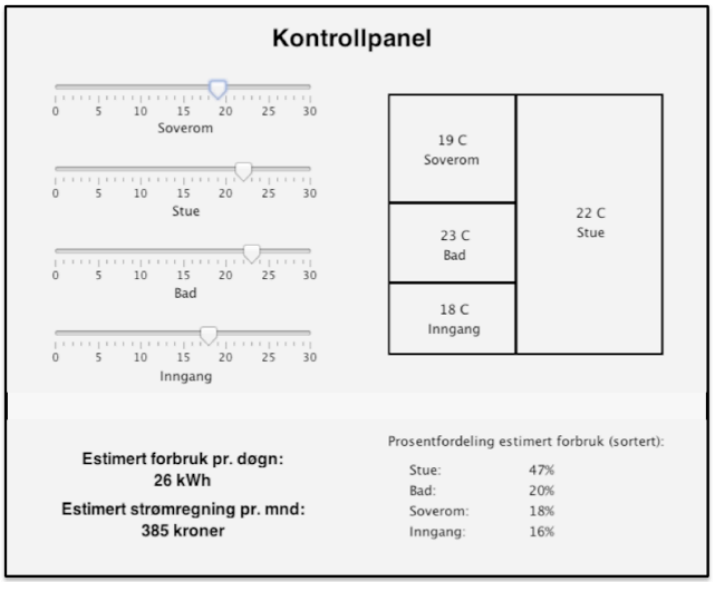
c. Gestaltprinsippet om nærhet

Forklar gestaltprinsippet om nærhet. Hvorfor er dette nyttig i interaksjonsdesign? Diskuter mulige svakheter i brukskvaliteten i skissen til kontrollpanel som kan belyses ved gestaltprinsippet om nærhet. Foreslå eventuelle forbedringer.

Gestaltprinsippene stammer fra psykologisk forskning tidlig i det 20de århundre. Psykologene fant ut at synsinntrykk blir behandlet før de når bevisstheten, slik at vi ikke bare ser enkeltheter men også helheter. Det gjør f.eks. at ting som ligger i nærheten av hverandre former er gruppe, og ting som ligger etter hverandre former en linje.

Dette er viktig for design av grafiske brukergrensesnitt i forhold til layout av skjermelementene fordi brukeren ikke bare tolker mening ut i fra enkelte skjermelementene (knapper, tekster, ikoner,,), men også ut i fra deres plassering i forhold til hverandre. Dersom f.eks. tre knapper ligger nær hverandre, men langt unna andre knapper, så blir det tolket som at de danner en gruppe og hører sammen. Det er da viktig at de faktisk hører sammen, og ikke bare tilfeldigvis har havnet i det som ser ut som en gruppe på skjermen.

I skissen til kontrollpanel er det fire grupper av elementer: (1) Sliderne, (2) ”kartet” over leiligheten, (3) estimatene for forbruk for dag og mnd., og (4) prosentfordeling av forbruk. Disse fire gruppene er lagt slik at det lett kan sees som om sliderne hører sammen med dag/mnd. estimatene og romkartet hører sammen med prosentfordelingen. En bedre måte kunne være å lage større avstand mellom sliderene og romkartet øverst, og estimatene nederst som illustrert under:



Vi ser at litt mer ”luft” endrer opplevelsen av gruppering. Dette forklares godt av gestaltprinsippet om nærhet/gruppe (*proximity*).

Vurdering:
D: Kun forklaring uten eksempel. Dårlig analyse
C: Et eksempel som illustrerer det. Forklarer relevans for designet.
B: Viser hvordan prinsippet kan brukes i redesign.
A: Ekstra dybde i analyse og eksempel i forhold til å vise relevans.

Oppgave 2 (30%) Designprosessen og evaluering

ISO 9241-11 definerer brukskvalitet (usability) som følger:

> *"The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use."*
>
> *Context of use* defineres videre som:
>
> *"Users, tasks, equipment (hardware, software and materials), and the physical and social environment in which a product is used."*

Begrepene *"effectiveness, efficiency and satisfaction"* oversettes på norsk til "anvendbarhet, effektivitet og tilfredsstillelse".

a. En konsekvens av denne ISO-definisjonene er at brukskvalitet er en kontekstavhengig egenskap ved et produkt. Den er avhengig av "context of use" (brukskontekst). Hvilke konsekvenser har dette generelt for brukbarhetstesting?

Standarden sier at brukervennlighet skal måles for spesifikke brukere, med spesifikke mål, i spesifikke bruksomgivelser. Dette gjør definisjonen kontektavhengig. Det gir altså ikke mening å snakke om et produkts brukervennlighet uten å ha spesifisert for hvem, hva, og hvor. Dette har konsekvenser for gjennomføring av brukbarhetstester:

- Spesifikke brukere: Det er viktig å kartlegge relevante brukergrupper, og sørge for at testpersonene er et representativs utvalg av brukergruppen. En mobiltelefon for svaksynte må f.eks. testes på svaksynte. En nettside for ungdom må testes på ungdom etc.

- Spesifikke mål: Produktet må testes i forhold til de oppgavene det faktisk skal brukes til. Dette krever at det utarbeides scenarier og oppgaver som er realistiske. Dersom en mobiltelefon skal brukes til å ringe med, sende SMS, og ta bilder, så må det lages realistiske oppgaver for disse funksjonene.

- I spesifikke omgivelser: En konsekvens av standarden er at vi må gjenskape de omgivelsene som produktet skal brukes i. Det er ikke alltid mulig å gjenskape dette 100%, men det er viktig å gjøre så godt man kan.

b. ISO-definisjonen sier at det som skal måles er anvendbarhet, effektivitet og tilfredsstillelse. Hva er de vanligste måtene å måle disse tre faktorene i en brukbarhetstest?

Definisjonen sier noe om hva som skal måles (anvendbarhet, effektivitet og subjektiv tilfredstillelse):

- Anvendbarhet sier noe om i hvilken grad brukeren er i stand til å få utført de oppgavene som produktet skal tilby. Dette måles ofte som gjennomføringsgrad. (task completion), d.v.s. antall deloppgaver som brukeren har fått til i en brukbarhetstest. Vi måler også opplevde problemer, fordi dette sier noe om ting ved brukergrensesnittet som kan føre til nedsatt anvendbarhet.

- Effektivitet sier noe om hvor mye ressurser som brukes for å få utført oppgavene. Dette måles ofte som medgått tid til oppgavene. Ved å måle hvor lang tid hver oppgave tar så får man en ide om systemets effektivitet.

- Subjektiv tilfredstillelse sier noe om brukerens subjektive opplevelse og vurdering av produktet. Det finnes skjemaer, bl.a. SUS, som stiller brukeren spørsmål om dette. SUS gir f.eks. et tall mellom 0 og 100 på om brukeren liker produktet etter å ha utført en brukbarhetstest. Et intervju etter testen vil i tillegg kunne gi verdifull informasjon om brukerens subjektive vurdering.

c. Du skal planlegge en brukbarhetstest for kontrollpanelet i oppgave 1. Med utgangspunkt i ISO-standarden og svarene dine på oppgave 2a og 2b, hva vil du gjøre for å få til en best mulig test?

ISO-standarden kan brukes som en sjekkliste i forberedelsene. I dette tilfellet så er brukerne beboere i en leiegård, deres mål er å kontrollere temperatur, og omgivelsene er leiligheten der kontrollpanelet skal stå.

Forberedelser:
Jeg ville begynne forberedelsene med å diskutere med utviklingsteamet og oppdragsgiver om hva som er kravene til brukervennlighet for dette produktet. Hva er akseptable nivåer av anvendbarhet, effektivitet og subjektiv tilfredstillelse?

Hvem er brukerne?
Jeg vil så gjøre research i forhold til hvem som skal bo i disse leielighetene. Jeg ville pratet med huseier og intervjuet leieboere i lignende områder. Dette ville gitt meg info til å kunne utvikle personas. Viktige ting å se etter er hos brukerne er bl.a. alder, kjønn, IT-kompetanse, leseferdighet, nasjonalitet/kultur, funksjonshemmede (syn, bevegelse,,).

Hva er deres mål med å bruke produktet?
Oppgavene må gjenspeile typiske brukssituasjoner for kontrollpanelet. Også gjerne noen utypiske situasjoner. Det er viktig at testpersonene kjenner seg igjen i oppgavene.

Fysisk brukskonteks og utstyr:
Jeg ville ha fått tak i en berøringsfølsom skjerm av den størrelse som er planlagt, og plassert den i brukbarhetslabben på den måten interiørarkitektene har forslått.

Sosial brukskontekst:
Intervjuene vil avdekke om dette er en skjerm som brukes i en sammenheng der flere deltar, eller om det kun er én bruker involvert. Dersom det ofte er en veldig sosial situasjon så bør man vurdere å rekruttere testpersoner for eksempel i par der den ene forteller den andre hva han/hun skal gjøre.

Jeg ville så ha rekruttert brukere til testen. Det er her viktig å få et så representativt utvalg som mulig, og det må settes av penger/midler til å betale de som deltar.

Gjennomføring
For å kunne få gode data i en akseptansetest så trengs minst 8-10 personer. Under testen så ville jeg anvende standard testmetodikk. Etter testen så ville jeg ha gjennomført et intervju og bedt testpersonene om å fylle ut et SUS skjema.

Analyse
Jeg ville ha analysert testene i forhold til oppgavegjennomføring, opplevde problemer, tid, SUS skjema og data fra intervjuene. Oppgavegjennomføring ville være i hvilken grad testpersonene faktisk fikk til å bruke kontrollpanelet til å kontrollere sette ønsket temperatur. Dette kan utrykkes kvantitativt som f.eks. prosent gjennomførte oppgaver. Jeg ville ha målt medgått tid. Dette vil si noe om effektivitet.
Jeg ville så ha summert opp SUS skjemaene og sett igjennom intervjuene. Jeg ville så ha regnet ut et snitt for SUS, og oppsummert de viktigste tingene som kom fram i intervjuene. Summene av disse funnene ville jeg så ha sammenlignet med de kravene som var satt til brukervennlighet.
Dersom et av målene med kontrollpanelet er at leieboerne skal bli seg bevisst sitt eget strømforbruk, så er det viktig at det lages realistiske tester som måler dette. Det krever også at det stilles spørsmål etter testen som får fram om testpersonene har forstått den feedbacken som gis i forhold til estimert strømforbruk.

Vurdering:
D:. Store mangler
C: Har fått med brukere, oppgaver, og omgivelser.
B: Viser kobling mot standarden..
A: Ekstra dybde og refleksjon.

Oppgave 3 (40%) Grensesnittkonstruksjon

Du skal i denne oppgaven vise bruk av Swing-komponenter og MVC-arkitektur for kontrollpanelet i figuren i oppgave 1. (Altså uten de eventuelle forslagene til forbedring som du har angitt i oppgave 1).

Du skal i løsningen ikke gjøre rede for mekanismene for layout. Fokus i oppgaven er på informasjonsflyt og datastrukturer.

Besvarelsen skal inneholde følgende:

- En forklaring av hvordan du vil gjøre bruk av MVC prinsippet i din løsning.
- Tegn opp de klassene du definerer og deres metoder og relasjoner. Beskriv deres rolle i MVC-arkitekturen.
- Hvor ligger informasjonen om temperaturene?
- Hvor beregnes estimatene?
- Tegn opp sekvensdiagrammer for følgende:
  - Initielt når objektene skapes og kontrollpanelet vises første gang.
  - Når brukeren forandrer ønsket temperatur på ett av rommene.

Klassen *JLabel* behandler i utgangspunktet ikke tall, men konvertering mellom *String* og *int* kan gjøres enkelt i Java v.h.a innebygde metoder i klassen *Integer.*

static String toString(int i)
Returns a String object representing the specified integer.

Eksempel: s = Integer.toString(i);

For hver av komponentklassene så er relevante metoder og hjelpeklasser listet i appendikset bakerst. **Merk: Det forlanges ikke at du skal anvende metoder eller klasser som ikke er listet i appendikset.**

- En forklaring av hvordan du vil gjøre bruk av MVC prinsippet i din løsning.

Jeg ville ha samlet de fire temperatursettingene i en modell TempModel. Denne modellklassen ville jeg også ha utstyrt med metoder for å beregne estimert forbruk for hvert av de fire rommene, gitt en utetemperatur. Jeg ville også latt modellen holde gitt utetemperatur (10 grader C i eksempelet).

Jeg ville så ha laget fire JPanel subklasser som views for å holde på de fire gruppene av skjermelementer: SliderView, FloorPlanView, EstimatesView og SortedEstimatesView. Disse befinner seg i et "samleview" ControlPanel (JPanel).

Jeg ville så ha koblet alle de fire views mot den underliggende TempModel modellen. Endringer i en slider vil da forplante seg via modellen til alle de tre andre views. Dette vil også være en utvidbar arkitektur dersom man for eksempel ønsket at brukeren kan sette temperatur direkte i FloorPlanView.

- Tegn opp de klassene du definerer og deres metoder og relasjoner. Beskriv deres rolle i MVC-arkitekturen

Rolle i MVC-arkitekturen er beskrevet over.

Class TempModel (arver ikke fra noen).
public void setTemp(int temperature, int roomNo)
public int getTemp(int roomNo)
public void setOutTemp(int temperature)
public int getOutTemp()
public int getEstimatePerDay(int roomNo)
public void addPropertyChangeListener(PropertyChangeListener listener)

Class SliderView extends JPanel implements PropertyChangeListener, ChangeListener
public void setModel(TempModel theModel)
public void propertyChange(PropertyChangeEvent evt)
public void stateChanged(ChangeEvent e)

Class FloorPlanView extends JPanel implements PropertyChangeListener
public void setModel(TempModel theModel)
public void propertyChange(PropertyChangeEvent evt)

Class EstimatesView extends JPanel implements PropertyChangeListener
public void setModel(TempModel theModel)
public void propertyChange(PropertyChangeEvent evt)

Class SortedEstimatesView extends JPanel implements PropertyChangeListener
public void setModel(TempModel theModel)
public void propertyChange(PropertyChangeEvent evt)
private void sortRoomEstimates(String[] rooms, int[] values);

Alternativ løsning:

Vi ser her at alle fire views har samme kode for setModel og propertyChange. Vi kunne derfor ha laget en abstrakt superklasse TempView som de fire view-klassene arvet fra:

Abstract Class TempView extends JPanel implements PropertyChangeListener
public void setModel(TempModel theModel)
public void propertyChange(PropertyChangeEvent evt)

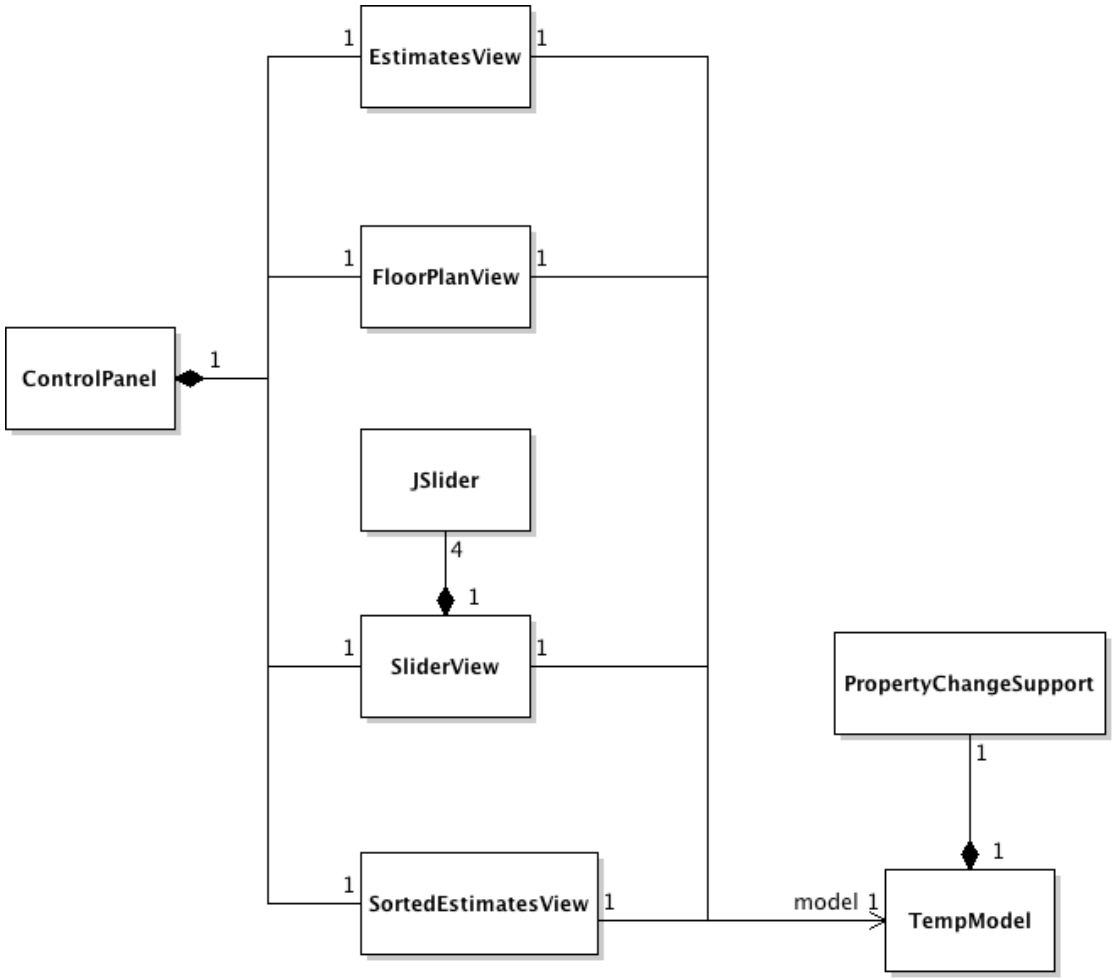
SetModel vil implementere koden for å koble mot modellen, mens propertyChange ville være abstrakt og følgelig tom og måtte implementeres i subklassen.

SliderView FloorPlanView ville da for eksempel bli slik:

Class SliderView extends TempView implements ChangeListener
public void stateChanged(ChangeEvent e)

Class FloorPlanView extends TempView

Sammenheng mellom klassene:



I tillegg så har EstimatesView , FloorPlanView of SortedEstimatesView en rekke JLabel og JPanel som ikke er listet i figuren.

- Hvor ligger informasjonen om temperaturene?
- Hvor beregnes estimatene?

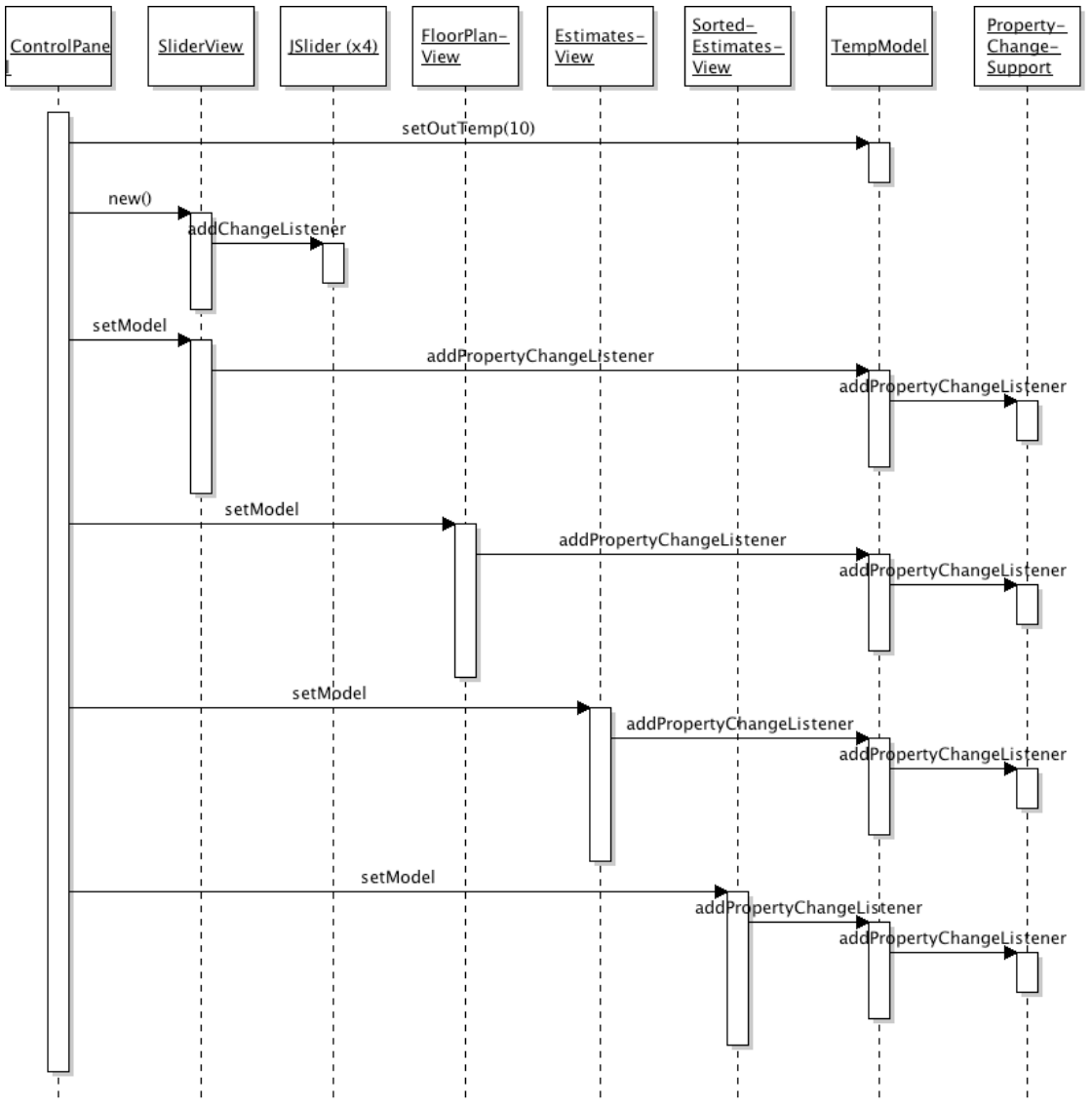
Informasjonen om temperaturene ligger i modellen (TempModel).

Beregningen av estimatet for en dag og pr. rom gjøres i TempModel. (metoden **getEstimatePerDay(int roomNo)**

Estimatet pr. måned gjøres i EstimatesView basert på summen av svarene fra TempModel.

Den prosentvise presentasjonen og sorteringen gjøres av SortedEstimatesView basert på svarene fra TempModel.

- Tegn opp sekvensdiagrammer for følgende:
  - Initielt når objektene skapes og kontrollpanelet vises første gang.

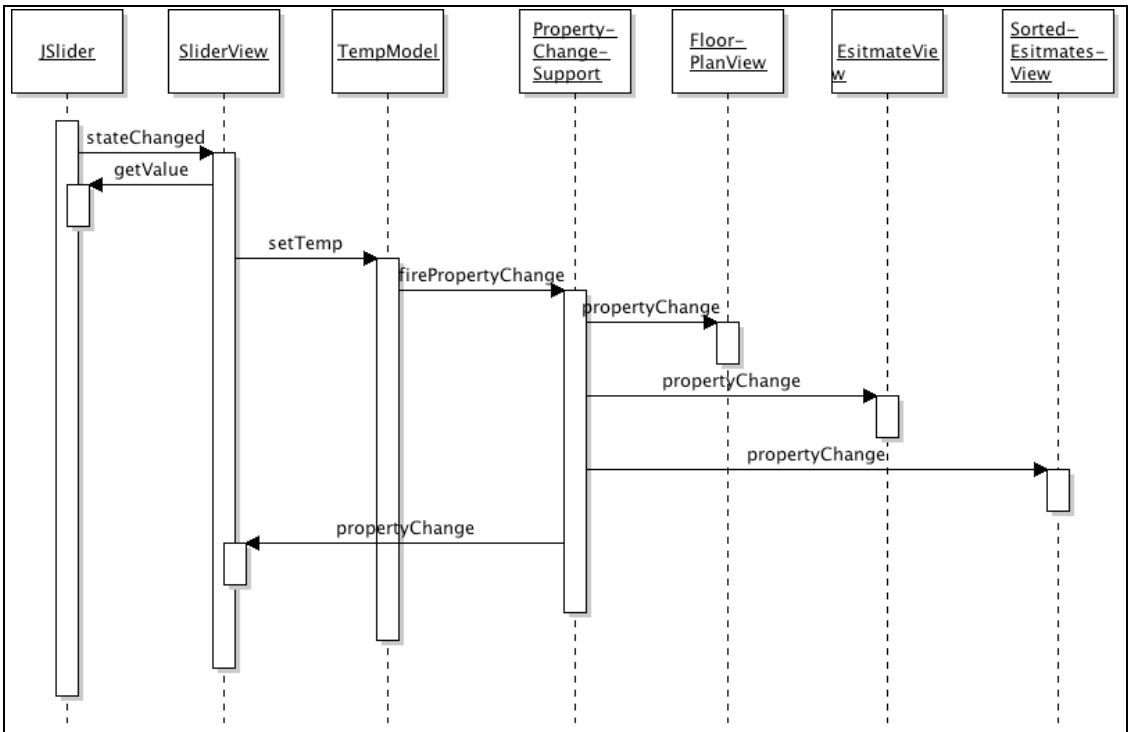


Vi ser i figuren over sekvensdiagrammet ved oppstart. ControlPanel starter med å opprette de enkelte objektene. Kun SliderView er vise her ettersom den setter eventhandler for sine fire JSliders til å være seg selv. Endringshendelser i de fire JSliders vil følgene havne som et kall til SliderView.

Modellen skapes, og utetemp settes til en fast verdi (10 grader).

ControlPanel kaller deretter hver av de fire store views for å koble de opp til modellen. Modellen sender videre til sitt PropertyChangeSupport objekt som vil ta seg av å kringkaste de faktiske endringshendelsene når TempModel enderer en av sine temperaturer (metoden setTemp).

- Når brukeren forandrer ønsket temperatur på ett av rommene.



I figuren over ser vi hva som skjer når brukeren endrer verdi i en slider. Sliderens SliderView får beskjed om endring (stateChanged). Den spør så den kallende JSlider om hva som er ny verdi. Modellen kalles så med denne verdien for den JSlider den gjelder (nr. 1,2,3 eller 4). Modellen detter verdien internt, og kaller deretter sitt PropertyChangeSupport objekt for å kringkaste endringen. PropertyChangeSupport objektet sender så beskjed til alle de fire hovedviews som har TempModel som s modell.

Det er her ikke tegnet opp det som skjer videre i hvert view:

Floorplan view vil spørre modellen om ny verdi og sende endringsmelding til den JLabel som skal ha ny verdi.

EstimateView henter alle fire romestimater, legger disse sammen og får samlet verdi pr. dag. Den ganger så dette tallet med 30 og får estimatet pr. mnd. Disse to verdiene skrives til de respektive Labels.

SortedEstimateView henter også alle fire estimatene. Beregner samlet verdi, prosentvis fordeling, og skriver dette ut sortert til de fire JLabels for rommene og de fire JLabels for verdiene.

SliderView får også en endringsmelding, men behøver ikke gjøre noe med denne.

Vurdering:
D: Har laget en modell, men store mangler.
C: Har laget en modell og koblet den opp slik at den fungerer.
B: Forklarer godt hvor ting skjer, og er klar på detaljene.
A: Godt nok til å kunne implementere.

Vedlegg: En del nyttige klasser og grensesnitt for oppgave 3.

Det følgende er klippet og limt fra Java definisjonen.

class PropertyChangeSupport

This is a utility class that can be used by objects that support bound properties. You can use an instance of this class as a variable in your object and delegate various work to it.

Methods:

public void addPropertyChangeListener(PropertyChangeListener listener)

Add a PropertyChangeListener to the listener list. The listener is registered for all properties.

Parameters:
listener - The PropertyChangeListener to be added

public void firePropertyChange(String propertyName,
Object oldValue,
Object newValue)

Report a bound property update to any registered listeners. No event is fired if old and new are equal and non-null.

Parameters:
propertyName - The programmatic name of the property that was changed.
oldValue - The old value of the property.
newValue - The new value of the property.

interface PropertyChangeListener

A "PropertyChange" event gets fired whenever an object changes a "bound" property. You can register a PropertyChangeListener with a source object so as to be notified of any bound property updates.

Methods:

public void propertyChange(PropertyChangeEvent evt)

This method gets called when a bound property is changed.

Parameters:
evt - A PropertyChangeEvent object describing the event source and the property that has changed.

class PropertyChangeEvent

A "PropertyChange" event gets delivered whenever an object changes a "bound" or "constrained" property. A PropertyChangeEvent object is sent as an argument to the PropertyChangeListener method.

Normally PropertyChangeEvents are accompanied by the name and the old and new value of the changed property. Null values may be provided for the old and the new values if their true values are not known.

An event source may send a null object as the name to indicate that an arbitrary set of if its properties have changed. In this case the old and new values should also be null.

Methods:

public String getPropertyName()

Gets the programmatic name of the property that was changed.

Returns:
The programmatic name of the property that was changed. May be null if multiple properties have changed.

public Object getNewValue()

Gets the new value for the property, expressed as an Object.

Returns:
The new value for the property, expressed as an Object. May be null if multiple properties have changed.

public Object getOldValue()

Gets the old value for the property, expressed as an Object.

Returns:
The old value for the property, expressed as an Object. May be null if multiple properties have changed.

class JPanel

JPanel is a generic lightweight container.

Methods:

public Component add(Component comp)

Appends the specified component to the end of this container.

class JLabel

A display area for a short text string or an image, or both. A label does not react to input events. A JLabel object can display either text, an image, or both.

Methods:

public void setText(String text)

Defines the single line of text this component will display.

Class JSlider

A component that lets the user graphically select a value by sliding a knob within a bounded interval.

Methods:

public int getValue()

Returns the sliders value.

public void setValue(int n)

Sets the sliders current value.

public void addChangeListener(ChangeListener l)

Adds a ChangeListener to the slider.

Interface ChangeListener

Defines an object which listens for ChangeEvents.

public void stateChanged(ChangeEvent e)

Invoked when the target of the listener has changed its state.

Class ChangeEvent

ChangeEvent is used to notify interested parties that state has changed in the event source.

Methods:

public Object getSource()

The object on which the Event initially occurred.

Returns:
The object on which the Event initially occurred.
