NORGES TEKNISK-NATURVITENSKAPELIGE UNIVERSITET
INSTITUTT FOR DATATEKNIKK OG INFORMASJONSVITENSKAP

Faglig kontakt under eksamen:
Hallvard Trætteberg, Tlf: 73 59 34 43

Løsningsforslag

EKSAMEN I FAG
TDT4180 MMI
Onsdag 28. mai 2008
Tid: kl. 0900-1300

Bokmål

Sensuren faller 18. juni 2008

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

Oppgave 1 (30%) Grensesnittdesign

a) *Jacob Nielsen har utarbeidet en 10 punkts liste for gode brukergrensesnitt. Hans første regel er "Visibility of system status" (synlighet av systemets status). Hva menes med dette? Gi to eksempler.*

Med dette menes at brukeren aldri skal være i tvil om hvilken tilstand datamaskinen og de kjørende programmene er i. Dette gjelder både i forhold til oppgaver som det tar tid å utføre og viktige tilstander til et program. Det gjelder også slike ting som at man skal se klart hvilket program man er inne i til en hver tid.
Synlighet er viktig for grafiske brukergrensessnitt fordi brukeren ikke har noen annen måte å vite hvilken tilstand maskinen er i. Dette er forskjellig fra f.eks. enkle fysiske objekter som er Swiss Army Knife. Du ser her alltid direkte hvilket verktøy (kniv, saks,,) som er aktivt, og trenger ingen ekstra tilbakemelding. En skjerm er i utgangspunktet bare en rekke piksler som man må bruke til å kommunisere med brukeren.

- Eksempel 1: De fleste programmer har en eller annen måte å vise at noe foregår som brukeren må vente på. Det vanligste er at markøren blir en klokke og en s.k. "progress bar". Det motsatte er f.eks. i Unix tekst input der man bare får tilbake ">".
- Eksempel 2: I nettbutikker så legger man ofte varer i en "handlevogn". Det er viktig at brukeren hele tiden får en oppdatert oversikt over hva som befinner seg i handlevognen.

b) *Et annet av hans punkter er "Recognition rather than recall" (gjenkjennelse heller enn å huske). Hva mener han med dette? Gi to eksempler.*

Utgangpunktet for denne regelen er at vi mennesker er flinkere til å gjenkjenne ting enn å huske nøyaktig. Det er derfor ikke lurt å basere et brukergrensesnitt på at brukeren skal måtte huske en rekke kommandoer. Det blir mye mindre feil dersom brukeren kan velge mellom en "meny" av forskjelllige kommandoer eller objekter. Don Norman kaller dette "kunnskap i verden" vs. "kunnskap i hodet". Med "kunnskap i verden" mener han alt vi gjør for å hjelpe på hukommelsen v.h.a ting "i verden", som f.eks. en "meny" av mulige kommandoer.

- Eksempel 1: Ikonlisten i MS Word gir deg bl.a. ikoner for stilling av tekst (venstre, midt, høyre). Dette baserer seg på "recognition". Når vi har brukt dette en gang så gjenkjenner vi det neste gang, og behøver ikke huske noen kommandoer.
- Eksempel 2: Unix kommandogrensesnitt baserer seg på at brukeren må huske en mengde kommandoer. Det fungerer greit for avanserte brukere som bruker det nesten hver dag, men dårlig for brukere som primært holder på med andre ting enn datamaskiner (d.v.s. flesteparten av brukerne).

c) *Gestaltprinsippene fra psykologien sier noe om menneskelig persepsjon. Hvorfor er denne teorien viktig for design av grafiske brukergrensesnitt? Gi to eksempler.*

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.

- Eksempel 1: Knappene for stilling av tekst i MS Word ligger sammen på en rekke, og adskilt fra andre knapper. De danner derfor en gruppe. Dette er intensjonelt ettersom de faktisk hører sammen.
- Eksempel 2: På de fleste mobiltelefoner så danner de 12 knappene for tall (1..9,0,*,#) en 3x4 firkant. Ved å legge de slik så dannes en gruppe som er annerledes enn andre knapper på mobiltelefonen.

Oppgave 2 (30%) Designprosessen

Du arbeider for et IT-firma som skal lage en ny løsning for håndholdte billettlesere for konduktører på tog. I det nye systemet så skal alle togbilletter ha strekkoder som kan leses av en integrert strekkodeleser i en håndholdt enhet.

Figuren over viser eksempel på en slik håndholdt enhet og en overfylt togvogn. Enheten har strekkodeleser integrert i toppen, trykkfølsom skjerm og noen ekstra knapper. Konduktørene skal kunne ha denne i lommen, og bruke den ved kontroll av billetter.

Første versjon av systemet skal leveres om et år. Du har fått i oppdrag å planlegge en siste brukbarhetstest av systemet som skal gjøres før systemet skal settes i drift hos togselskapet som har bestilt løsningen.

ISO 9241-11 definerer brukervennlighet/brukskvalitet (usability) som "anvendbarhet, effektivitet og subjektiv tilfredstillelse for spesifikke brukere, med spesifikke mål, i spesifikke omgivelser".

Bruk denne definisjonen som utgangspunkt for planleggingen av testen.

a) *Hvilke konsekvenser har denne definisjonen av brukervennlighet generelt for hva brukbarhetstester skal måle, og for hvordan de skal gjennomføres?*

Definisjonen sier noe om hva som skal måles (anvendbarhet, effektivitet og subjektiv tilfredstillelse) og om brukssammenhengen (for spesifikke brukere, med spesifikke mål, i spesifikke omgivelser).

- 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. 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.

Standarden sier at brukervennlighet skal måles for spesifikke brukere, med spesifikke mål, i bestemte omgivelser. 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. Skal man teste en mobiltelefon som skal kunne brukes i dårlige lysforhold, så må man senke lyset når testen gjennomføres.

b) *Beskriv hvordan du vil forberede, gjennomføre og analysere brukbarhetstesten i dette konkrete tilfellet.*

I dette tilfellet så er brukerne konduktører, deres mål er å kontrollere billetter, og omgivelsene er togvogner som kan være i bevegelse og fulle av folk.

Forberedelse:
Jeg ville begynne forberedelsene ved 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? Hvor mange feil er akseptabelt, hvor lang tid pr. billett er akseptabelt, og hvor fornøyde bør konduktørene være med produktet? Dette gir meg målbare brukbarhetskrav.

Jeg vil så gjøre litt research i forhold til hvordan konduktører jobber. Jeg ville blitt med noen konduktører på jobb en dag, og pratet med dem om det å kontrollere billetter. Dette ville gitt meg info til å kunne utvikle et scenarie og noen oppgaver.

Jeg ville ha snakket med utviklingsteamet for å sørge for å få laget noen billetter som kan testes, og for å få klarhet i hva som skal til rent teknisk for å få gjennomført testen.

Jeg ville så ha prøvd å få tak i en togvogn som kunne brukes under testen. Alternativt ville jeg ha sørget for å innrede et stort rom med stoler som gjenskaper en del av en togvogn.

Jeg ville måtte finne ut hvor mange statister jeg trenger. Kanskje det holder med 5-6 personer med billetter for å få testet det viktigste. Disse må så rekrutteres.

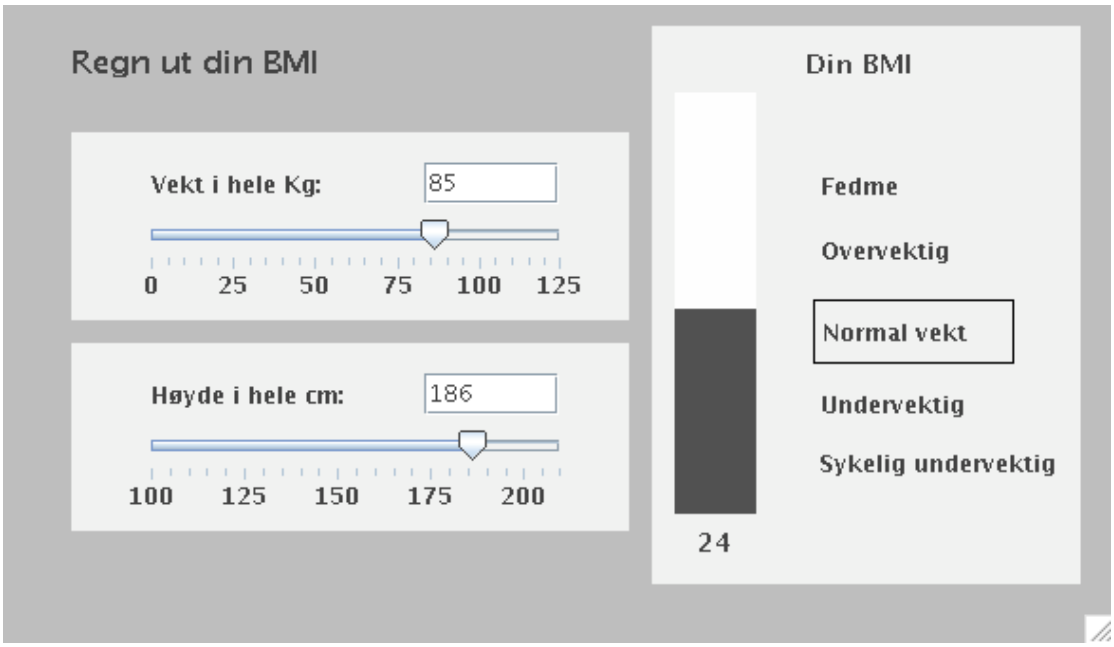
Dersom testen skulle foregå i en togvogn så ville jeg måtte ha rigget den med videoutstyr og mikrofoner for å kunne få med det som skjedde.

Gjennomføring
For å kunne få gode data i en akseptansetest så trengs minst 5-6 tester. Jeg ville lagt opp til 8 tester. Jeg antar da at jeg allerede har rekruttert konduktører.
Under testen så ville jeg anvende standard testmetodikk. I dette tilfelle så ville bruk av ”tenke høyt” være vanskelig, så jeg ville droppet det. Før testen så ville jeg satt konduktøren inn i scenariet og forklart kort om den nye håndholdte terminalen.
Jeg ville også ha instruert statistene i deres oppgaver.
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 konduktørene faktisk fikk til å bruke terminalen til å kontrollere billetter. Dette kan utrykkes kvantitativt som f.eks. prosent gjennomførte oppgaver.
Ved å se igjennom videoene så vil jeg kunne finne ut hvor konduktørene opplevde problemer med brukergrensesnittet. Jeg ville ha samlet opp problemene, og sett etter problemer som dukker opp hos flere testpersoner.
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.

Oppgave 3 (40%) Grensesnittkonstruksjon



Du har fått i oppgave å implementere en Java/Swing applet for å regne ut BMI (Body Mass Index) på en webside for slanking. Bildet over viser hvordan BMI-kalkulatoren skal se ut. Brukeren skal kunne legge inn sin vekt og høyde og automatisk få regnet ut sin BMI.

For både vekt og høyde så kan brukeren enten dra i en skyvespak (”slider”) eller taste inn verdien direkte i et tekstfelt. I begge tilfelle vil skjermbildet oppdateres automatisk med de riktige verdiene.

Formelen for BMI er:

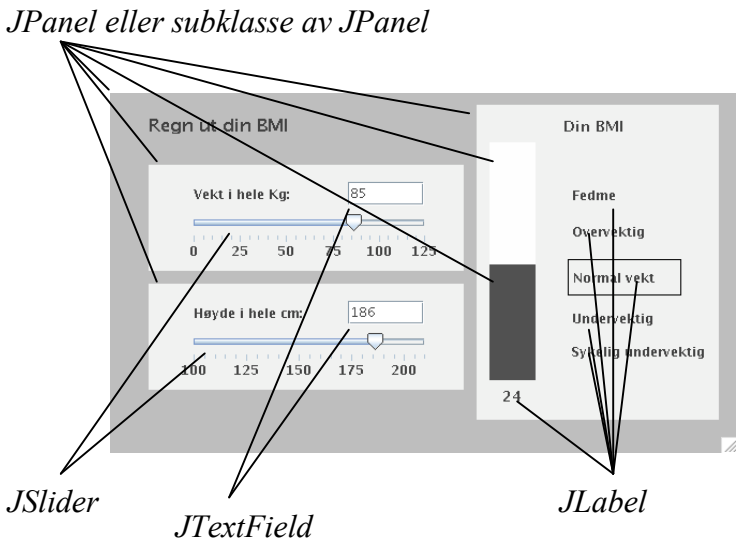
$$\mathrm{BMI} = \frac{vekt\ (\mathrm{kg})}{hoyde^2 (\mathrm{m})}$$

Eksempel

$$19{,}6 = \frac{60\,(\mathrm{kg})}{1{,}75^2\,(\mathrm{m})}$$

* *BMI på under 16,5 = sykelig undervektig*
* *BMI på mellom 16,5 og 18,5 = undervektig.*
* *BMI på mellom 18,5 og 24,9 = normal vekt.*
* *BMI på mellom 25 og 29,9 = overvektig.*
* *BMI på 30 og over = fedme.*

Figuren under viser bruk av SWING-komponenter i implementasjonen. I tillegg til det som er angitt på figuren så er de faste tekstene implementert som JLabel.



Klassen JTextField behandler i utgangspunktet ikke tall, men konvertering mellom String og int kan gjøres enkelt i Java v.h.a innebygde metoder i klassen Integer. (Du trenger ikke forholde deg til feilsituasjoner ved at brukeren skriver inn tekst som ikke er gyldige heltall)

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

Eksempel: s = Integer.toString(i);

static int parseInt(String s)
Parses the string argument as a signed decimal integer.

Eksempel: i = Integer.parseInt(s);

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.***

Søylen til høyre er laget v.h.a. en JPanel som forandrer høyde. Angivelse av BMI-kategori (”Fedme”,,) gjøres v.h.a. skifte av rammefarge rundt teksten. Du trenger ikke gå i detalj om hvordan disse grensesnittelementene implementeres.

a) *Hvordan vil du bruke Model-View-Controller til å skille presentasjon og data i din løsning av BMI-kalkulatoren? Forklar hovedtrekkene i din løsning i forhold til valg av modell/modeller.*

Jeg ville ha laget en BMIModel som inneholdt høyde og vekt, og som regnet ut BMI. Denne vil ha følgende API for å sette og lese data:

- void setHeight(int height)
- int getHeight()
- void setWeight(int height)
- int getWeight()
- int getBMI()

Metoden getBMI regner ut og returnerer BMI basert på innholdet av feltene height (i cm) og weight (i kg). BMIModel trenger ikke ha et eget felt for BMI ettersom dette kan regnes ut hver gang det er behov for det.

I tillegg vil BMIModel ha en instans av klassen PropertyChangeSupport som tar vare på alle views som har denne modellen som modell. I forhold til å legge til nye views til modellen så ville jeg i APIet også ha en metode:

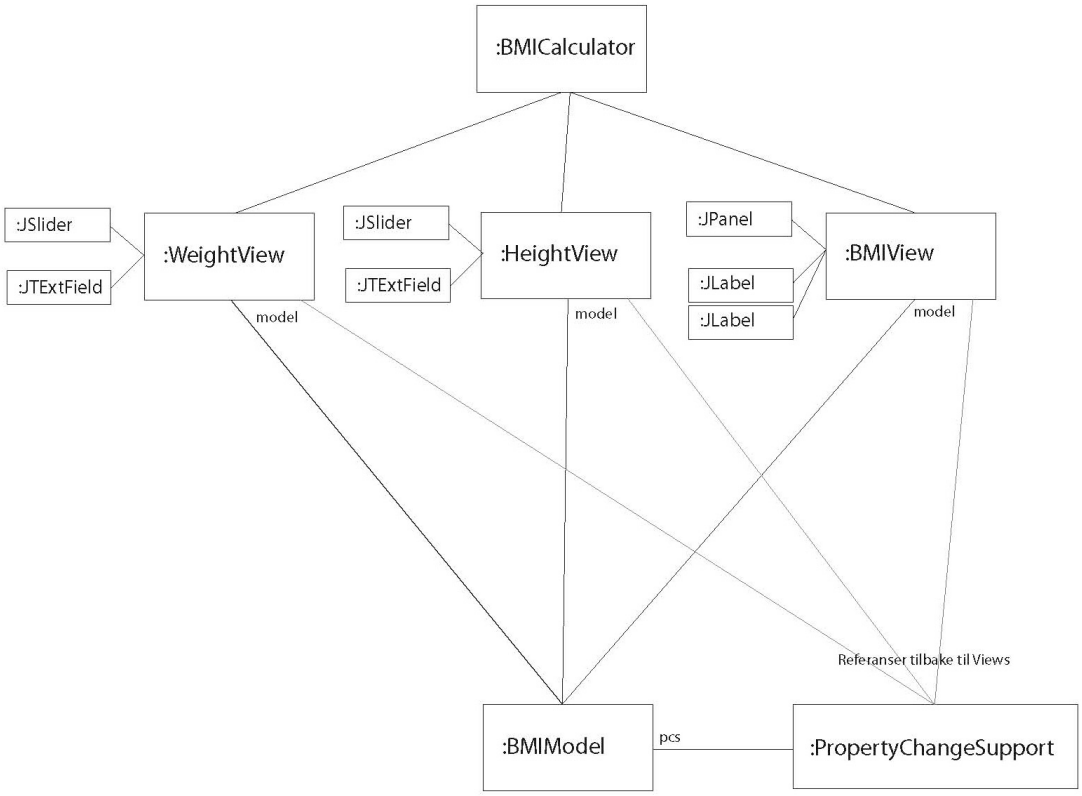
- void addPropertyChangeListener(PropertyChangeListener listener)
  Denne metoden sender kallet videre til PropertyChangeSupport objektet.

Jeg ville så ha laget tre views for h.h.v. vekt, høyde og BMI. La oss kalle de WeightView, HeightView og BMIView. Disse vil være subklasser av JPanel og implementere interfacet PropertyChangeListener. Ved å implementere PropertyChangeListener så kan de ha BMIModel som modell og motta hendelser om endring. De må da implementere metoden propertyChange. De vil også ha et felt model som holder en referanse til BMIModel objektet.
De tre viewene må ha en metode for å koble opp modellen ”utenfra”, f.eks.:

- void setModel(BMIModel model)

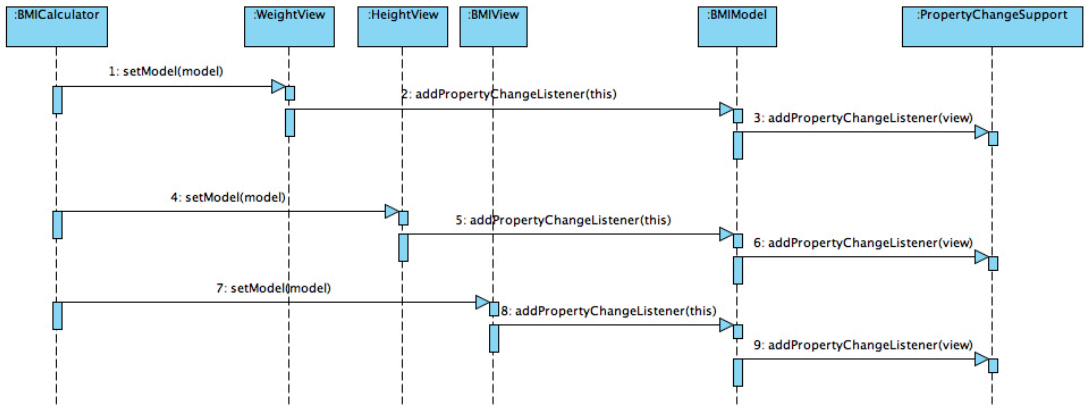
”Utenpå” de tre viewene så ville jeg ha hatt et samle-view (JPanel), som er selve applikasjonen eller appleten. La oss kalle den BMICalculator. Innen de tre viewene så vil jeg ha Swing-elementer for slidere, tekstfelt og labels. Jeg ville så ha koblet opp disse slik at endringer i heightView eller weightView fører til at det sendes riktige set kall til modellen. Modellen vil da sende beskjed til alle views om at det har skjedd en endring, og de enkelte views vil spørre modellen om ny verdi og gjøre oppdateringer av sine Swing-elementer.

b) *Tegn opp hvilke instanser/objekter som eksisterer når BMI-kalkulatoren kjører. Hva er deres relasjoner/referanser til hverandre? Bruk enkle firkanter for å angi instanser/objekter og piler for å angi relasjoner/referanser.*

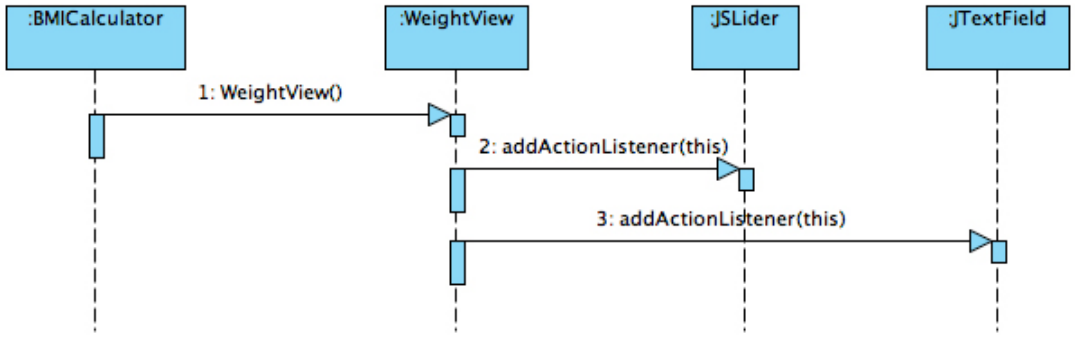


Her er det viktig å ha med et BMIModel objekt og vise at det har et PropertyChangeSupport objekt knyttet til seg. Så er det viktig å vise de tre viewene og at de har Swing komponenter. Viewene må ha BMIModel objektet som modell, ved at deres model felt refererer til modellen.

c) *Tegn opp sekvensdiagrammet når BMI-kalkulatoren startes opp. Forklar.*

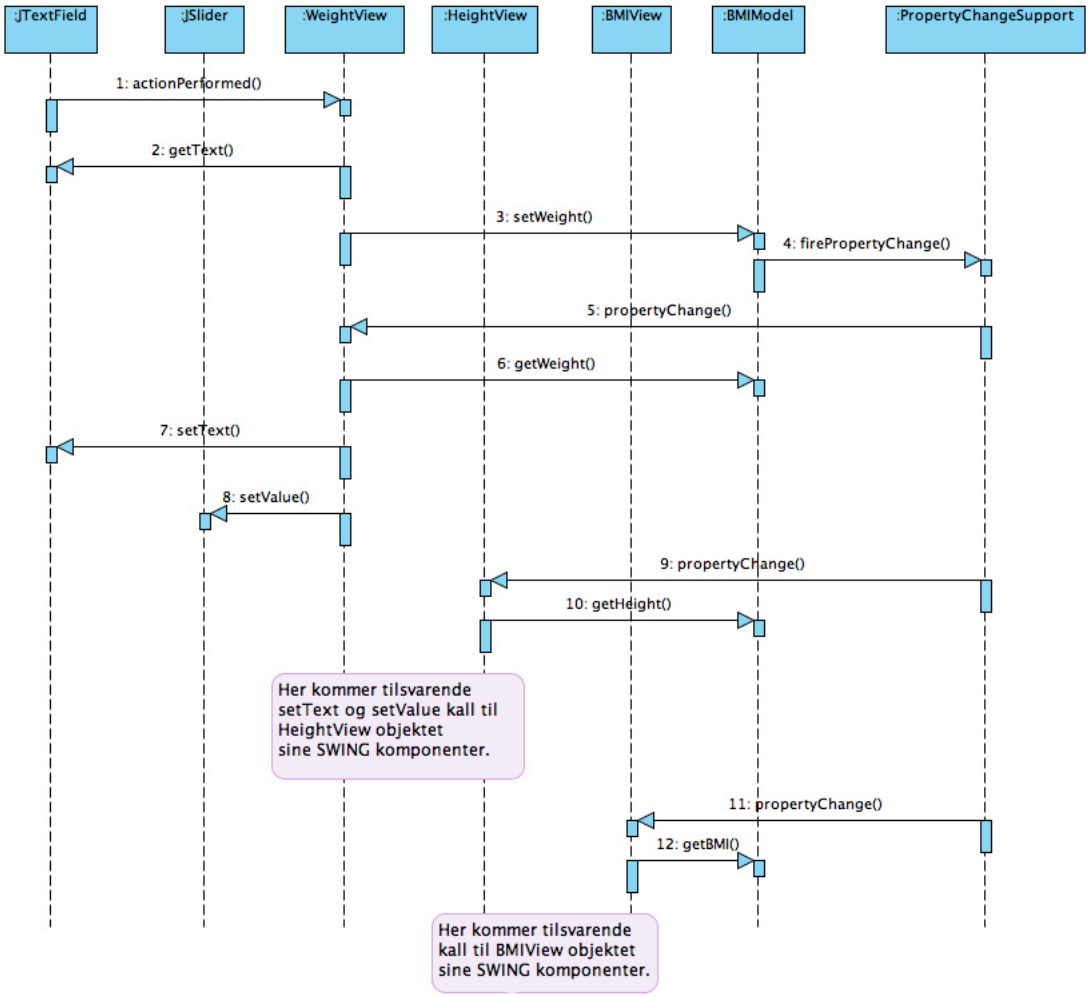


Modellen skapes fra BMICalculator. Den kobles så opp mot viewene ved at BMICalculator kaller deres setModel metoder. Dette forplanter seg så til de enkelte viewene som kaller addPropertyChangeListener til modellen. BMIModel objektet sender disse kallene videre til sitt PropertyChangeSupport objekt.



For hver av viewene så må de enkelte SWING komponentene kobles opp med addActionListener(this) for at endringshendelser skal havne i riktig view.

d) *Tegn opp sekvensdiagrammet når brukeren legger inn en ny verdi i tekstfeltet for vekt. Forklar.*



Her er det viktig å vise at endringen i tekstfeltet forplanter seg til WeightView, som henter ut ny verdi av feltet og kaller sin modell (BMIModel) med metoden setWeight med den nye verdien. Modellen kaller så sitt PropertyChangeSupport objekt med metoden firePropertyChange som sørger for at det sendes endringsmeldinger (propertyChange) til alle de tre views. Hvert view vil så kalle modellen for å lese sin oppdaterte verdi og gjøre tilsvarende endringer i alle sine Swing-elementer.

Appendiks: Relevante klasser, interface og tilhørende metoder

Det følgende er klippet og limt fra den offisielle dokumentasjonen på java.sun.com.

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 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 name of the property that was changed. May be null.

public Object getNewValue()

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

Returns:
The new value for the property. May be null.

public Object getOldValue()

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

Returns:
The old value for the property. May be null.

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 JTextField

JTextField is a lightweight component that allows the editing of a single line of text.

Methods:

public String getText()

Returns the text contained in this JTextField

public void setText(String t)

Sets the text of this JTextField to the specified text.

public void addActionListener(ActionListener l)

Adds the specified action listener to receive action events from this JTextField.

interface ActionListener

The listener interface for receiving action events. The class that is interested in processing an action event implements this interface, and the object created with that class is registered with a component, using the component's addActionListener method. When the action event occurs, that object's actionPerformed method is invoked.

Methods:

public void actionPerformed(ActionEvent e)

Invoked when an action occurs.

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 JLabel

A display area for a short text string. A label does not react to input events.

Methods:

public void setText(String text)

Defines the single line of text this component will display.

MERK: Det er ikke nødvendig å benytte ActionEvent eller ChangeEvent i implementasjonen.