NTNU
Norges teknisk-naturvitenskapelige universitet

Fakultet for informasjonsteknologi, matematikk og elektroteknikk

Institutt for datateknikk og informasjonsvitenskap

BOKMÅL

EKSAMEN I FAG
TDT4180 Menneske-maskin-interaksjon

Lørdag 27. juni 2006
Kl. 09.00 – 13.00

Faglig kontakt under eksamen:
Gry Seland, tlf. 404 76098

Ingen tillatte hjelpemidler.

Formatert: Punktmerking og nummerering

Sensurdato:
17. juni 2006. Resultater gjøres kjent på http://studweb.ntnu.no/ og sensurtelefon 81 54 80 14.

Prosentsatser viser hvor mye hver oppgave teller innen settet.

Lykke til!

Oppgave 1 – Brukersentrert systemutvikling (30 %)

Stortinget har bestemt at det skal settes i gang et utredningsarbeid for å undersøke muligheten til å gi alle stemmeberettigede mulighet til å stemme via internett ved neste stortingsvalg. Du er leder for dette utredningsarbeidet, og ønsker i samarbeid med designteamet ditt på tre personer å utvikle og prøve ut en prototyp av systemet som en del av dette arbeidet. Denne prototypen skal presenteres for Stortinget i midten av oktober 2006.

a) I forbindelse med utviklingsarbeidet forslår du at designteamet utvikler low- fidelity og high-fidelity prototyper. Hva er fordelene og ulempende med henholdsvis low-fidelity og high-fidelity prototyper, og hvordan kan de anvendes i dette prosjektet?

10 poeng

Hensikt: Studentene skal vite foreskjellen på low- og hi fidelity prototyper, og klare å se hvordan de ulike typene prototyper kan anvendes i et spesifikt prosjekt.

Low fidelity prototyp (4 poeng): Enkle prototyper uten mye detaljer verken grafikk eller interaksjon, ofte laget av papir. Skisseaktig, ser uferdig ut.

Fordel: Uferdighet gjør det lett å gi tilbakemeldinger, tar kort tid å lage, billig teknologi (papir), trenger ikke være programmerer for å lage

Trekker 1 poeng dersom det ikke er nevnt at uferdighet/skisseaktighet gjør det lett å gi tilbakemeldinger.

Ulempe: Mangler detaljrikdom
Kan ikke brukes til å simulere bevegelser, animasjoner, lyder og lignende. Testpersonene må spille rollespill og late som om papirprogrammet er et dataprogram. Dette krever evne til innlevelse fra brukerens side.

Trekker 1 poeng dersom det ikke er nevnt at low-fi prototyper krever evne til innlevelse.

High fidelity prototyp (4 poeng): Komplekse prototyper med mer detaljer. Laget på PC, ser ut som et ferdig program.

Fordel: Kan teste ut brukergrensesnittet uten å programmere det fullstendig.

Ulempe: Det at prototypen ser ferdig ut, gjør det vanskeligere for testpersoner å gi tilbakemelding på skjermdesignet. Det tør ikke fordi programmet ser så ferdig ut.
Koster mye å lage: Lages ofte v.h.a. prototypingsverktøy.

Dette prosjektet (2 poeng): Det finnes sannsynligvis flere muligheter, men studentene bør ha med et forslag til bruk av begge typer prototyper.

- Low fidelity prototyp for å få tilbakemelding på tidlige ideer fra brukere
- Low fidelity prototyp for å vise fram for Stortinget og overbevise om at dette er et system de bør satse på

Har gitt 1 poeng dersom man sier at low-fi prototyper brukes tidlig i prosessen, og high-fi prototyper brukes sent, selv om studenten ikke har relatert det direkte til valgsystemprosjektet.

b) Siden systemet skal kunne benyttes av alle stemmeberettigete i Norge, kommer temaet ”universelt design” opp tidlig i designprosessen. Hva betyr universelt design, og hvordan bør designteamet ta hensyn til dette når de skal utvikle brukergrensesnittet for systemet?

Hensikt: Studentene skal vite hva som ligger i begrepet universelt design, og klare å relatere begrepet til et spesifikt prosjekt.

10 poeng

Universelt design (5 poeng):

I begrepet universelt design ligger det at systemet skal fungere for de fleste typer mennesker, uavhengig av fysiske, kognitive, kulturelle egenskaper. Det er mao designet for mange ulike typer brukergrupper.

Dette prosjektet (5 poeng) – andre faktorer er også mulige:

- Systemet skal brukes av folk i alle aldre fra 18 år til ca. 90.
- Syn: Synet varierer og skrift, knapper og lignende må være synlige for alle
- Varierende dataferdigheter: Både noviser og eksperter skal kunne bruke systemet. Det betyr at det ikke må være noen unødvendig funksjonalitet i programmet, og at skjermdesignet må være veldig tydelig på hvordan det fungerer (affordance)
- Fargeblinde: Farger må brukes slik at fargeblinde ikke diskriminert (nesten 10 % av befolkningen)

En del studenter har tolket ” hvordan bør designteamet ta hensyn til dette når de skal utvikle brukergrensesnittet for systemet” som at man bør inkludere og teste et vidt spekter av brukere i prosessen, og det gies også poeng for dette.

c) I tillegg til å presentere prototypen for Stortinget ønsker du å gjennomføre en evaluering av prototypen før den blir pilottestet i et av fylkene. Angi konkrete evalueringsmål for anvendbarhet, effektivitet og brukertilfredshet (efficiency, effectiveness og user satisfaction) for dette systemet.

Hensikt: *Studentene skal kunne sette opp relevante brukbarhetsmål for et spesifikt prosjekt. For å kunne gjøre dette må de ha en forståelse for hva som ligger i begrepene anvendbarhet, effektivitet og brukertilfredshet, men de trenger ikke å forklare dette eksplisitt.*

10 poeng

Dersom noen har forstått hva som ligger i begrepene anvendbarhet, effektivitet og brukertilfredshet, men ikke har svart på oppgaven gies det 4 poeng.

Trekker opptil 4 poeng dersom svarene (brukbarhetsmålene) ikke er spesifikke nok.

Eksempel (andre er også mulig):

Anvendbarhet: 90 % av alle stemmeberettige klarer å gi sin stemme til det partiet de ønsker å stemme på via systemet.

Effektivitet: 75 % av alle stemmeberettige klarer å gi sin stemme til det partiet de ønsker å stemme på via systemet på mindre enn fem minutter

Brukertilfredshet: 70 % av alle som bruker systemet til å avgi sin stemme krysser av for ”nokså tilfreds” eller ”svært tilfreds” på en skala fra ”ikke tilfreds” til ”svært tilfreds” etter den første utprøvingen av systemet i et av fylkene.

---

d) ISO 13407 *Human centred design processes for interactive systems* recommends an iterative design process. What are the four phases of this iterative process?

Ved en feiltakelse ble ikke denne oppgaven fjernet fra engelsk og nynorsk versjon. Denne oppgaven rettes derfor for de studentene som har gjort den.

Hensikt: Studentene skal kunne skissere hovedtrekkene i ISO 13407 og forstå betydningen av begrepene ”forstå og spesifisere brukskontekst”, ”spesifisere bruker- og organisasjonskrav”, ”produsere designløsninger” og ”evaluere løsningene mot brukbarhetskrav”.

10 poeng?

Skisse 2 poeng?

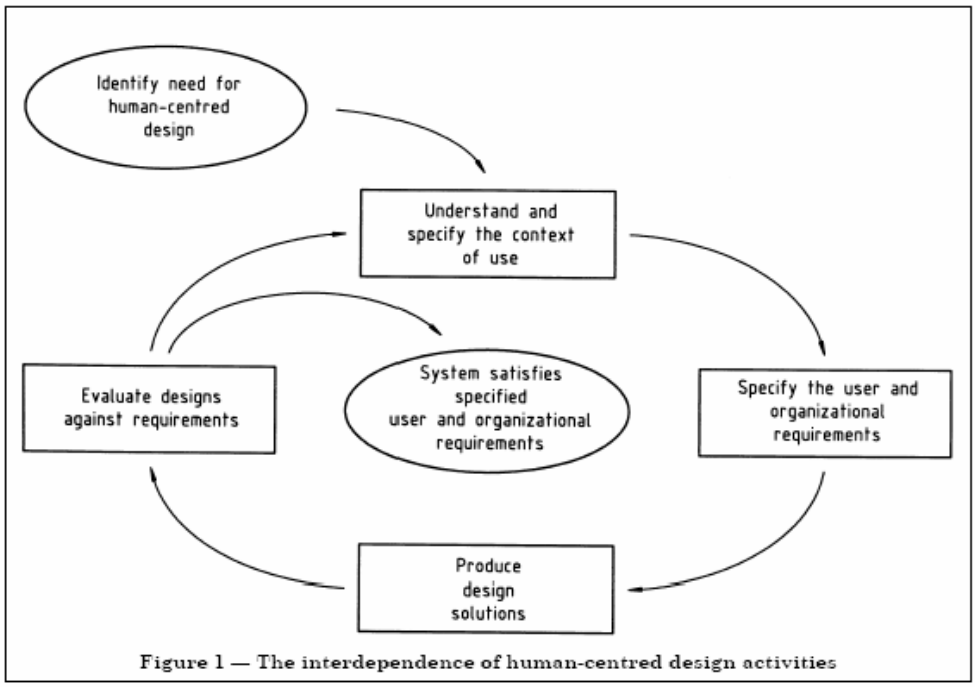


Figure 1 — The interdependence of human-centred design activities

Forstå og spesifisere brukskontekst (2 poeng?):

- Karakterisere brukere
- Beskrive brukernes oppgaver
- Beskrive brukernes miljø (inkl. annen hardware og software)

Spesifisere bruker- og organisasjonskrav: (Denne delen er litt uklart beskrevet i iso13407, må vise litt skjønn i forhold til retting). (2 poeng?)

- Relateres til beskrivelsen av brukskonteksten

- Identifisere relevante brukere (range of relevant users)
- Klare menneske-sentrerte mål og målbare kriterier
- Godkjennes av brukere/brukerrepresentanter

Produsere designløsninger (2 poeng?):

- Utvikle konkrete prototyper
- Presentere designløsningene for brukere og la de teste de
- Endre design på bakgrunn av tilbakemeldinger fra brukere
- Hold orden på iterasjonen
- Utnytte flerfaglig input for å lage forslag til løsninger

Evaluere designforslag i forhold til krav (2 poeng?):

- Ha en evalueringsplan
- Bruke evalueringer for å gi input i designprosessen
- Undersøk om målene er nådd
- Gjennomfør feltevalueringer og måling av langtidsbruk

Oppgave 2 – Skjermdesign (30 %)

a) Direkte manipulasjon brukes i mange programmer i dag. Hva er fordelene og ulempene ved bruk av direkte manipulasjon som interaksjonsstil? Illustrer med et eksempel.

Hensikt: Vite hva som er fordelene og ulempene ved direkte manipulasjon som interaksjonsstil

10 poeng

Fordeler – flere er mulige: (4 poeng)
- Basert på recognition-hukommelse (gjenkjennelses-hukommelse)
- Mulighetene er synlige
- Lett(ere) å lære
- Ikke farlig å prøve og feile…
  → signaliserer en positiv holdning til brukeren (jobber konkret, ikke abstrakt)

Ulemper – flere er mulige: (4 poeng)
- ikke alle tilstander (og -endringer) er lette å visualisere
- krevende å programmere
- problematisk når direkte manipulasjon ikke fungerer som forventet
- ekspertbrukere foretrekker hurtigere måter å jobbe på

For eksempel: ”Tar tak i” et dokument for å flytte det fra en katalog til en annen. (2 poeng)

10 poeng dersom studenten tydelig har forstått hva direkte manipulasjon er inkl.. fordeler og ulemper

b) I tilfelle en handling ikke kan uføres må brukeren av systemet få beskjed om hva som er feil. Hva kjennetegner en god feilmelding? Illustrer med et eksempel.

Hensikt: Studenten skal kunne gjengi hva som kjennetegner en god feilmelding. Eksempelet må gi mening. Fordel om eksempelet inneholder både en god og en dårlig feilmelding.

God feilmelding (6 poeng):
Spesifikk, konkret
Ikke truende (positiv tone)
Hjelper brukeren til å forstå det som er feil, og gjøre det riktige neste gang
Bruker brukerens språk

Eksempel – mange mulige løsninger (4 poeng); god feilmelding (positiv tone som en dialog, forteller hva som er galt, og også andre ting man bør unngå).

Dårlig feilmelding: Negativ tone (varseltegn), forteller bare at den ene karakteren er ikke er lov (neste gang gjør brukeren en ny feil og får en tilsvarende beskjed), dataspråk – ikke brukerens språk (Calender Identifier).

c) Don Normans begrep om ”affordance” er nyttig i forhold til skjermdesign. Hva betyr affordance? Illustrer med et eksempel.

Hensikt: Forstå hva som ligger i begrepet affordance og hvorfor det er relevant for design av brukergrensesnitt.

Forklaring (7 poeng)
Ting ”tillater”visse handllinger
Utformingen sier noe om hvordan tingen kan brukes
Relasjon mellom objekt og aktørs fysiske egenskaper
Fysisk: Gitt av kroppens og objektets anatomi (hammeren,cola-flasken,...)
Kulturelt: Konvensjoner som læres (”Slik gjør vi det her”: dørhåndtak som vris eller trykkes ned)
Gitt av konteksten/omgivelsene(”Nei, nei, på Mac’en gjør en det slik”)

Affordance er relevant for design av brukergrensesnitt fordi brukergrensesnittet er den eneste muligheten designeren har til å kommunisere mulighetene og funksjonaliteten i programmet. Ved å bruke skjermelementer som umiddelbart signaliserer hvordan de skal brukes (affordance) forstår brukeren lettere hvordan programmet skal brukes.

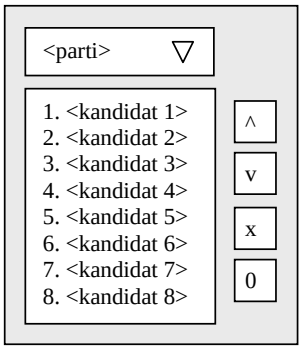
Eksempler – mange andre er også mulige (3 poeng):
Dørhåndtaket sier:”Trykk meg ned”
Hammeren sier:”Ta tak i meg”
Cola-flasken sier:”Drikk meg”
Lego-klossensier”Sett oss sammen”

10 poeng gies dersom studenten tydelig har forstått hva begrepet affordance går ut på (inkl. forskjell mellom universell og kulturelt betinget affordance), og hvorfor det er relevant for skjermdesign.

Noen poeng trekkes dersom forklaringen er tynn
1-2 poeng legges til dersom relevansen forklares.

Oppgave 3 – Konstruksjon av Swing-GUI med MVC-arkitektur (40 %)

Du skal konstruere et Swing-GUI for stemmegivning, med funksjonalitet for å kunne *stemme* på et parti og *stryke* og *endre rekkefølgen* på kandidatene til dette partiet. Designet er vist til høyre og brukskravene er som følger:



1) Brukeren skal kunne velge ett og kun ett av et sett partier ved hjelp av en nedtrekksmeny (JComboBox).
2) Når et parti er valgt, skal brukeren se listen (JList) av alle kandidatene til partiet.
3) Brukeren skal ved å velge kandidat i lista og trykke ”^”- eller ”v”-knappen (JButton) kunne flytte kandidater hhv. opp eller ned i lista.
4) Brukeren skal kunne markere kandidater som strøket og visningen i lista skal indikere om kandidaten er strøket eller ikke, f.eks. med en annen skriftstype eller farge. Når en kandidat i lista er valgt, skal ”x”-knappen (JToggleButton) indikere om kandidaten er strøket og kunne brukes til endre denne tilstanden.
5) Brukeren skal ved å trykke på ”0”-knappen (JButton) kunne nullstille valg av parti, stille listen av kandidater tilbake til den opprinnelige rekkefølgen og fjerne alle strykninger.

Din jobb er kke å vurdere om dette er et godt design, sett fra brukerens side, men å beskrive konstruksjonen av et Swing-GUI med MVC-arkitektur. Husk at vi ikke er ute etter kodingsdetaljer, men strukturen av klasser og samhandling mellom objekter.

a) *10 poeng*. Hvilke krav stilles generelt til en modell i MVC-arkitekturen? Definer et eller flere grensesnitt (Java interface) for modellen som gir tilgang til dataene som vises og manipuleres av GUI’et vist i figuren, slik at modellen oppfyller disse kravene.

Det generelle kravet er observerbarhet, dvs. at alle relevante data kan hentes ut og evt. endres og at en kan registrerer lyttere og få beskjed når dataene endres. I praksis betyr dette at en må ha et komplett sett metoder for å hente ut data, relevante metoder for å endre dem og metoder for å registrere lyttere. *2 poeng gis for begrepet observerbarhet og 2 for å forklare hva det betyr i praksis.*

I dette tilfellet er det naturlig å definere tre-fire grensesnitt: Ett for seddelen som helhet, med lese-metoder for parti og kandidater og endringsmetoder for valgt part og kandidatrekkefølge, ett for kandidaten, med get/set-metoder for strykningstilstanden, og ett eller to grensesnitt for lytting på disse strukturene. Vårt forslag til disse vist under. Strukturen kan variere en god del, men det må finnes metoder for (våre metoder i parentes) 1) å lese partilista (getPartyCount, getParty), 2) å lese kandidatene (getCandidateCount, getCandidateVote, getState), 3) lese og sette valgt parti (getSelectedParty/setSelectedParty), 4) å endre rekkefølgen på kandidatene (swapCandidates) og sette strykningstilstanden til en kandidat (setState). I tillegg kommer metoder for å registrere lyttere (5). Lyttergrensesnittet må ha metoder for hver type endring (valgt parti, endring i kandidatrekkefølge og strykninger), evt. informasjon om dette i et eget hendelsesobjekt 6). *Det gis 1 poeng for hver av disse elementene. Siden dette kan struktureres på mange ulike måter, er det viktig å se etter metoder som har ønsker bruk og effekt, ikke etter navnene.*

Vi har valgt å ha en hovedmodell (ElectionModel) for partier og kandidater, og et eget grensesnitt (CandidateVote) for detaljene om hver kandidat og strykningstilstanden. Vi har valgt å definere ett lyttergrensesnitt, som knyttes til ElectionModel, hvilket betyr at endringer i CandidateVote må kringkastes gjennom ElectionModel.

```
public interface ElectionModel {
    public int getPartyCount();
    public String getParty(int i);
    public int getParty(String name);
    public int getSelectedParty();
    public void setSelectedParty(int i);

    public int getCandidateCount(int party);
    public CandidateVote getCandidateVote(int party, int i);

    public void swapCandidates(int party, int candidate1, int candidate2);

    public void addElectionListener(ElectionListener listener);
    public void removeElectionListener(ElectionListener listener);
}

public interface CandidateVote {
    public String getName();
    public boolean getState();
    public void setState(boolean state);
}

public interface ElectionListener {
    public void selectedPartyChanged(ElectionModel model);
    public void candidatesChanged(ElectionModel model, int party, int candidate);
    public void candidateStateChanged(ElectionModel model, int party, int candidate);
}
```

b) *15 poeng*. Forklar hvordan modellen beskrevet i a) brukes av GUI'et, dvs. hvordan modellen er knyttet til relevante Swing-komponenter og leses og endres av disse. Beskriv spesielt evt. ekstra klasser, f.eks. adapter-klasser, som trengs for å gjøre konstruksjonen komplett iht. MVC-arkitekturen. Bruk gjerne diagrammer, f.eks. UML klasse- og samhandlingsdiagrammer, ved siden av tekst.

Dersom du ikke behersker MVC-arkitekturen, så beskriv likevel hvordan du ville konstruert GUI'et uten MVC-arkitektur, slik at du får vist din kunnskap om Swing-basert konstruksjon.

Hver av komponentene bruker modellen på sin måte. JCombo-boksen for partivalg bruker ElectionModel som sin ComboBoxModel, med en adapterklasse som "oversetter" ComboBoxModel sine metoder-kall til

ElectionModel sine og ElectionModel sin(e) hendelse(r) til ComboBoxModel sine (getSize, getElementAt og get/setSelectedItem til hhv. getPartyCount, getParty og get/setSelectedParty, og selectedPartyChanged til contentsChanged). Det sentrale her er ikke detaljene men at ElectionModel knyttes til JComboBox'en gjennom en adapterklasse og at den "oversetter" meldinger frem og tilbake.

Kandidatlista sin modell følger samme logikk, men trenger en adapterklasse mot andre ElectionModel-metoder (getCandidateCount og getCandidateVote) og reagerer på de to andre hendelsene (candidatesChanged og candidateStateChanged).

For visning av strykningstilstanden til kandidater må en bruke getState og dette gjøres typisk vha. en ListCellRenderer.

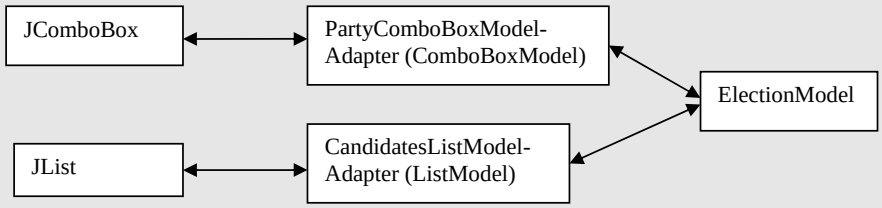
Flytt opp og ned vil bruke metoden for å endre rekkefølge (swapCandidates), stryk-knappen vil bruke setState. Nullstillknappen er spesiell, da den best implementeres ved å bytte ut hele modellen, istedenfor å endre den. Det er fullt mulig å lage modeller for knappene også (ButtonModel), men det har vi ikke lagt opp til (ikke noe dokumentasjon), da det er unødvendig knotete og vi ikke har øvd på det tidligere.

Det gis 3 (1) poeng for JComboBox'en, 3 (1) for JList'a, 4 (2) poeng for flytt opp- og ned-knappene, 3 (2) poeng for strykningknappen og 2 poeng for nullstillknappen (ikke-MVC-varianten i parentes).

c) *15 poeng*. Beskriv *kort* hvordan hvert av punktene 1-5 i kravlisten kan implementeres og hva som skjer (meldingsflyten) for hver brukeraksjon. Husk at detaljene ikke er så viktig, så lenge rollen til klassene og hvordan de samhandler kommer tydelig frem.

Standardkonstruksjonen er å ha en overordnet JPanel-subklasse, som inneholder og lytter på relevante komponenter, i dette tilfellet seleksjon i JList'a (ListSelectionListener) og trykk på knappene (ActionListener). JPanel-subklassen har i tillegg en modell, i vårt tilfelle av typen ElectionModel.

1) JComboBox'en er knyttet til ElectionModel gjennom et adapter (PartyComboBoxModel-Adapter). Når brukeren velger et nytt parti, så kalles setSelectedItem på adapteret, som kaller setSelectedParty på ElectionModel. Dette genererer en hendelse, dvs. et kall til selectedPartyChanged på adapteret (som lytter på sin ElectionModel) som i sin tur sier fra til sine ListDataListener'e. Ikke-MVC-alternativet er å fylle JComboBox'en med partidata og reagere på actionPerformed-hendelser.
   3 *poeng for MVC, 1 uten.*
2) JList'a er knyttet til ElectionModel gjennom et (annet) adapter (CandidatesListModel-Adapter). Når ElectionModel sitt valgte parti endres, må dette adapteret generere en endringshendelse for kandidatlista, dvs. si fra til sine ListDataListener'e om at hele lista er endret. Dette illustrerer at samme underliggende hendelse kan formidles videre som to ulike hendelser, én for JComboBox'en og én for JList'a. I tillegg må endringer i rekkefølgen, formidlet som candidatesChanged-hendelser oversettes til contentsChanged-hendelser av adapteret. Ikke-MVC-alternativet er å lytte på JComboBox'en sin action-hendelse og fylle JList'a med nye kandidatelementer.
   3 poeng for MVC, 1 uten.



3) Valg i JList'a må håndteres med en eksplisitt ListSelectionListener. Trykk på aksjonsknappene håndteres ved å registrere ActionListener-lyttere, som kaller relevant endringsmetode på ElectionModel-objektet som ligger under både JComboBox'en og JLista. Sistnevnte oppdateres gjennom candidatesChanged-hendelser oversatt til contentsChanged-hendelser.
   4 poeng, 2 uten.
4) Spesiell visning av strykningstilstanden til kandidater håndteres av en egen ListCellRenderer (arv gjerne fra DefaultListCellRenderer), som setter skriftstype og/eller farge som en del av getListCellRendererComponent-metoden. Når seleksjonen i JList'a endres, må selected-tilstanden til strykningsknappen settes tilsvarende. I tillegg må metoden for å invertere strykningstilstanden kalles når strykningsknappen trykkes. Dette resulterer i endringshendelser (candidateStateChanged) som

oversettes av adapteret til contentsChanged-hendelser.
3 poeng.

5) Nullstilling håndteres best ved å bytte ut hele modellen, f.eks. ved å innføre en setModel på hovedvinduet, heller enn å endre eksisterende modell.
2 poeng.

Utdrag fra Swing sin dokumentasjon

```
// The model of a JList. Use JList.setModel to customize the model for a particular JList.
public interface ListModel {
  // Returns the length of the list.
  int getSize();
  // Returns the value at the specified index.
  Object getElementAt(int index);
  // Adds a listener to the list that's notified each time a change to the data model occurs.
  void addListDataListener(ListDataListener l);
  // Removes a listener from the list that's notified each time a change occurs.
  void removeListDataListener(ListDataListener l);
}

// The model of a JComboBox
public interface ComboBoxModel extends ListModel {
  // Set the selected item
  void setSelectedItem(Object anItem);
  // Returns the selected item
  Object getSelectedItem();
}

// The listener interface for ListModel changes
public interface ListDataListener extends EventListener {
    // Sent after the indices in the index0,index1 interval have been inserted.
    void intervalAdded(ListDataEvent e);
    // Sent after the indices in the index0,index1 interval have been removed.
    void intervalRemoved(ListDataEvent e);
    // Sent when the contents of the list has changed in a way
    // that's too complex to characterize with the previous methods
    void contentsChanged(ListDataEvent e);
}

// Useful superclass for ListModel implementations.
// Implements ListDataListener handling and provides useful fireXXX methods
public abstract class AbstractListModel implements ListModel {
    // implements ListDataListener handling
    public void addListDataListener(ListDataListener l) { … }
    public void removeListDataListener(ListDataListener l) { … }

    // Subclasses must call this method after one or more elements of the list change.
    protected void fireContentsChanged(Object source, int index0, int index1) { … }
    //Subclasses must call this method after one or more elements are added to the model.
    protected void fireIntervalAdded(Object source, int index0, int index1) { … }
    // Subclasses must call this method after one or more elements are removed from the model.
    protected void fireIntervalRemoved(Object source, int index0, int index1) { … }
}

public interface ListSelectionListener extends EventListener
{
  // Called whenever the value of the selection changes.
  void valueChanged(ListSelectionEvent e);
}

// Interface for the object responsible for drawing each element in a JList.
// Use JList.setCellRenderer to customize the renderer for a particular JList.
public interface ListCellRenderer {
    // Return a component that has been configured to display the specified value.
    Component getListCellRendererComponent(JList list, Object value, int index,
        boolean isSelected, boolean cellHasFocus);
}

// Implementation of ListCellRenderer that uses a JLabel
// Subclasses may extend it, override getListCellRendererComponent and
// utilize the fact that it returns a JLabel.
public class DefaultListCellRenderer extends JLabel implements ListCellRenderer { … }

// Superclass for the various button classes, including JButton and JToggleButton
public abstract class AbstractButton {
```

```
    // Enables (or disables) the button.
    public void setEnabled(boolean b) { … }
    // Sets the state of the button.
    public void setSelected(boolean b) { … }
    // Sets the Action of the button.
    public void setAction(Action a) { … }
}

// interface for listening to action events on an AbstractButton and a JComboBox
public interface ActionListener extends EventListener {
    // Invoked when an action occurs.
    public void actionPerformed(ActionEvent e);
}

public interface Action extends ActionListener {
    // Sets the enabled state of the Action.
    public void setEnabled(boolean b);
}

// Useful superclass for Actions. Subclasses need only implement actionPerformed(ActionEvent)
public abstract class AbstractAction implements Action {
    // Defines an Action object with the specified description string and a default icon.
    public AbstractAction(String name) { … }
    // Enables or disables the action.
    public void setEnabled(boolean newValue) { … }
}
```

Merknad: Ved endelig karaktersetting ble designdelen (oppgave 1 og 2) vektet med 65 % og teknikkdelen (oppgave 3) vektet med 35 %. Så nært som alle studentene gjorde det bedre på designdelen enn på teknikkdelen, så denne vektingen ble sett på som mer rettferdig enn 60/40.