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 21. august 2010
Tid: kl. 0900-1300

Bokmål

Sensuren faller 13. september 2010

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

Oppgave 1 (25%) Grensesnittdesign

Når man skal designe et brukergrensesnitt så står man ofte overfor valg av interaksjonsstil (eng.: interaction style).

a) På generelt grunnlag, hva er fordeler og ulemper med hver av de 3 interaksjonsstilene (I) direktemanipulasjon, (II) menybasert og (III) tekstlig kommandospråk?

Anta at du skal lage et brukergrensesnitt til et PC-program som rektorer i barneskoler skal bruke til å lage ukeplaner for alle klassene på skolen sin. En rektor beskrev denne planleggingsoppgaven som et stort puslespill der timer, lærere, klasser og klasserom skal passe sammen best mulig.

b) Hvilken interaksjonsform tror du vil være best for dette PC-programmet? Begrunn svaret.

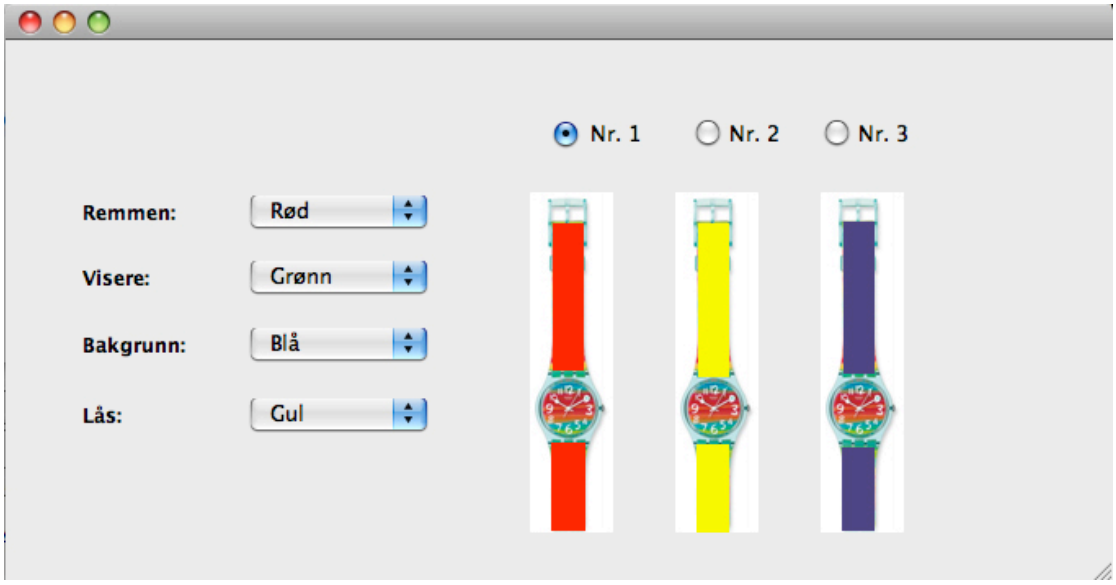
Oppgave 2 (35%) Designprosessen

Du er ansatt på HCI-ekspert i et IT-firma som skal planlegge et lite utviklingsprosjekt for helsevesenet. Din jobb er å sørge for nødvendig brukerkontakt i alle faser av prosjektet. Det skal utvikles en web-basert løsning for familier til barn som er under opptrening etter trafikkulykker. Systemet skal fungere som kommunikasjonskanal og en slags ”min side” mellom familien og alle involverte i det offentlige (for eksempel lege, fysioterapeut, NAV, apotek, sykesøster på skolen).

Dere har begrenset med ressurser i prosjektet, og har kun råd til tre brukernære aktiviteter. (Eksempel på slike aktiviteter er feltstudie, intervjuer, fokusgruppe, designworkshop, papirprototyping m/brukbarhetstesting, logging av bruk).

Hvilke tre aktiviteter vil du velge, og hvor vil du legge dem i utviklingsløpet (tidlig, midt, sent) ? Begrunn svaret.

Oppgave 3 (40%) Grensesnittkonstruksjon



På bildet over ser man en skisse til et skjermbilde for en Java/Swing applet der brukeren skal kunne komponere sitt eget armbåndsur. Brukeren skal kunne se tre klokker samtidig, og for hver av dem kunne velge farge på remmen, viserne, bakgrunn og låsen. I eksempelet over er klokke nr. 1 valgt. Brukeren skifter farge på valgt armbåndsur v.h.a. combobokser. Når brukeren for eksempel endrer farge på remmen, så vil fargen automatisk oppdateres på valgt armbåndsur. Anta at det kun finnes fire farger: rød, grønn, blå og gul. Når brukeren skifter valgt armbåndsur, så vil comboboksene oppdateres med valgte farger for dette uret.

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.**

a) Hvordan vil du bruke Model-View-Controller (MVC) til å skille presentasjon og data i din løsning av dette programmet? Forklar hovedtrekkene i din løsning i forhold til valg av modell/modeller.
b) Beskriv modellen/modellene i detalj.
c) Tegn opp hvilke instanser/objekter som eksisterer når programmet kjører. Hva er deres relasjoner/referanser til hverandre? Bruk enkle firkanter for å angi instanser/objekter og piler for å angi relasjoner/referanser.
d) Tegn opp sekvensdiagrammet når det hele startes opp. Forklar.
e) Tegn opp sekvensdiagrammet når klokke nr. 1 er valgt og brukeren skifter farge på remmen til grønn. Forklar.
f) Tegn opp sekvensdiagrammet når klokke nr. 3 velges. Alle combobokser skal da oppdateres til de verdier som klokke nr. 3 har.

Du trenger ikke forholde deg til layout, eller hvordan klokkene tegnes ut.

Hint: Et JPanel eller en subklasse av JPanel kan godt være usynlig (ha samme farge og rammefarge som bakgrunnen) og brukes til å samle flere Swing-komponenter.

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 removePropertyChangeListener(PropertyChangeListener listener)

Remove a PropertyChangeListener from the listener list.

Parameters:
listener - The PropertyChangeListener to be removed

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 JComboBox

A component that combines a button or editable field and a drop-down list. The user can select a value from the drop-down list, which appears at the user's request.

Methods:

public void addActionListener(ActionListener l)

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

public Object getSelectedItem()

Returns the current selected item.

public void setSelectedItem(Object anObject)

Sets the selected item in the combo box display area to the object in the argument.

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 ActionEvent

A semantic event which indicates that a component-defined action occured. This high-level event is generated by a component (such as a Button) when the component-specific action occurs (such as being pressed). The event is passed to every ActionListener object that registered to receive such events using the component's addActionListener method.

The object that implements the ActionListener interface gets this ActionEvent when the event occurs. The listener is therefore spared the details of processing individual mouse movements and mouse clicks, and can instead process a "meaningful" (semantic) event like "button pressed".

Methods:

public Object getSource()

The object on which the Event initially occurred.

Returns:
The object on which the Event initially occurred.

Class JRadioButton

An implementation of a radio button -- an item that can be selected or deselected, and which displays its state to the user. Used with a ButtonGroup object to create a group of buttons in which only one button at a time can be selected. (Create a ButtonGroup object and use its add method to include the JRadioButton objects in the group.)

Methods:

public void addActionListener(ActionListener l)

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

public boolean isSelected()

Returns the state of the button. True if the toggle button is selected, false if it's not.

Returns:
true if the radiobutton is selected, otherwise false

public void setSelected(boolean b)

Sets the state of the radiobutton. Note that this method does not trigger an actionEvent.

Parameters:
b - true if the button is selected, otherwise false

class ButtonGroup

This class is used to create a multiple-exclusion scope for a set of buttons. Creating a set of buttons with the same ButtonGroup object means that turning "on" one of those buttons turns off all other buttons in the group. A ButtonGroup can be used with any set of objects that inherit from AbstractButton. Typically a button group contains instances of JRadioButton, JRadioButtonMenuItem, or JToggleButton. Initially, all buttons in the group are unselected.

Methods:

public void add(AbstractButton b) *(for example JRadioButton)*

Adds the button to the group.

Parameters:
b - the button to be added

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.