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

Torsdag 27. mai 2010
Tid: kl. 0900-1300

Bokmål

Sensuren faller 17. juni 2010

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

Kvalitetssikret: Hallvard Trætteberg

Oppgave 1 (25%) Grensesnittdesign

Læreboka lister opp 8 prinsipper (”golden rules”). Tre av prinsippene er:

- Lag konsistente grensesnitt (”Strive for consistency”)
- Tillat brukeren å angre (”Permit easy reversal of actions”)
- La brukeren ha kontrollen (”Support internal locus of control”)

For hver av disse tre prinsippene, svar på følgende:

- Forklar hva prinsippet går ut på, gjerne med et eksempel.
- Hvorfor er dette er et viktig prinsipp?

Oppgave 2 (35%) Designprosessen

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

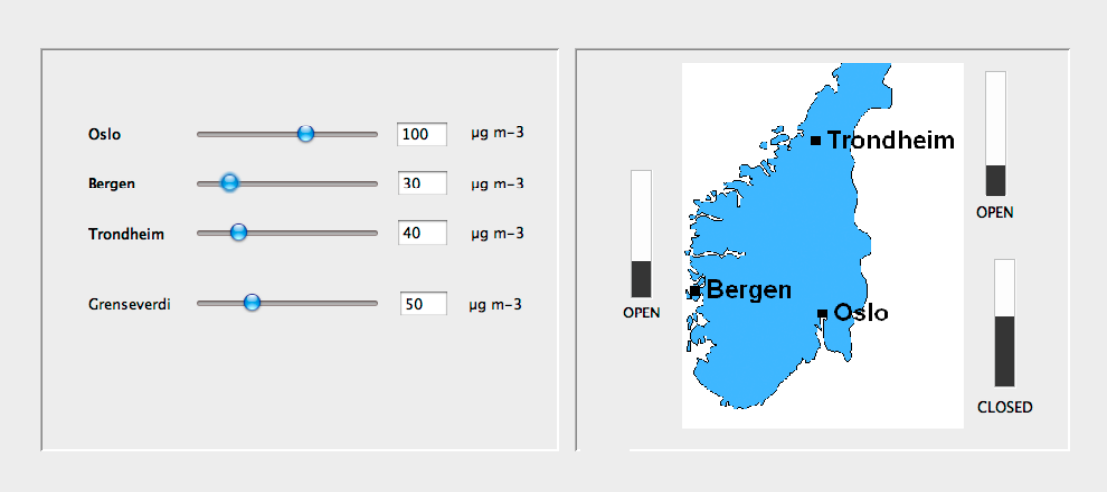
a) Hvordan måler man typisk hver av de tre faktorene anvendbarhet, effektivitet og subjektiv tilfredsstillelse i en brukbarhetstest?

b) Standarden angir at de tre faktorene måles for ”*spesifikke brukere, med spesifikke mål, i spesifikke omgivelser”.* Hvilke konsekvenser har dette for planleggingen og utførelsen av en brukbarhetstest?

Oppgave 3 (40%) Grensesnittkonstruksjon

I 2010 har europeisk luftfart blitt forstyrret av aske fra vulkanutbruddet på Island. I den forbindelse så ønsker avisen Morgenposten å tilby sine lesere en daglig oversikt over tilstanden på landets tre viktigste flyplasser i forhold til askeinnhold i luften. Dette vil de gjøre ved å lage et program i Java/Swing der journalisten kan oppdatere måledata og få fram en grafisk kartframstilling som de så kan klippe og lime inn i avisen. Det å klippe og lime er ikke en del av denne oppgaven.

På figuren under ser vi en skisse av hvordan programmet skal se ut.



I ruten til venstre setter journalisten inn askedata og gjeldende grenseverdi. Bildet til høyre viser så automatisk fram søyler for hver by, og teksten ”OPEN” eller ”CLOSED” avhengig om verdien for denne byen er under eller over grenseverdien.

De statiske tekstene (”Oslo”,”Bergen”, ”OPEN”,”CLOSED”, etc.) er realistert v.h.a. *JLabel*. Input-feltene for tall er realisert v.h.a. *JTextField*. Sliderene er realisert v.h.a *JSlider*. Brukeren kan endre askeinnholdet for f.eks. Oslo enten ved å taste inn et nytt tall eller ved å dra i slideren. I begge tilfeller så oppdateres alle relevante deler av grensesnittet automatisk.

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.**

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 brukeren legger inn et nytt tall i tekstfeltet for ”Grenseverdi”. Forklar.

Du trenger ikke forholde deg til layout, figuren av kartet eller hvordan denne 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 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 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 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.

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.