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

Mandag 4. august 2008
Tid: kl. 0900-1300

Bokmål

Sensuren faller 25. august

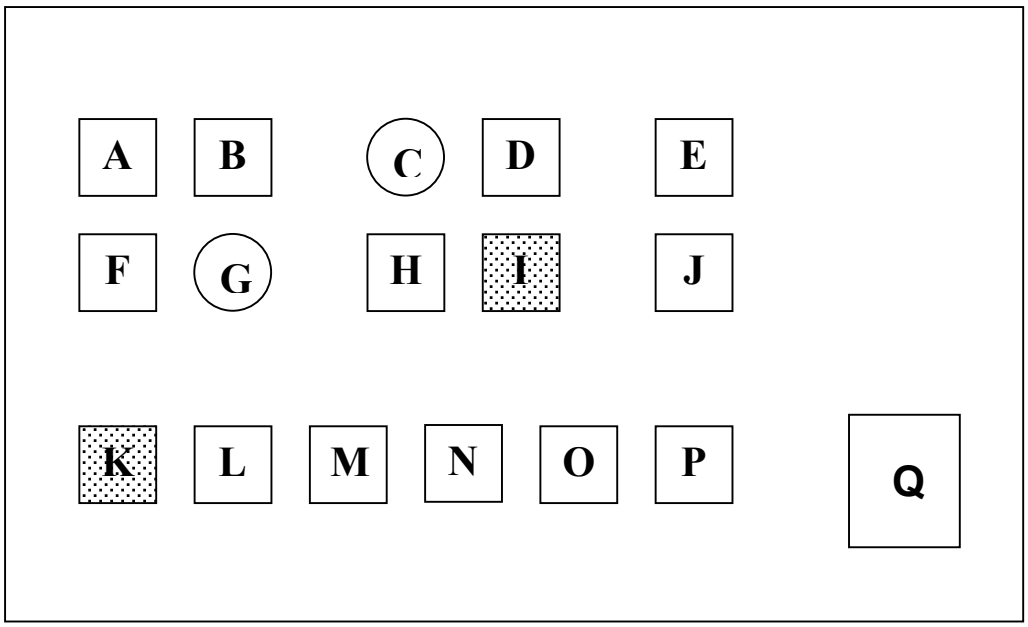
Hjelpemiddelkode: **D** Ingen trykte eller håndskrevne hjelpemidler tillatt.
Bestemt enkel kalkulator tillatt.

Oppgave 1 (30%) Grensesnittdesign

a. "Affordance"
Forklar begrepet "affordance" slik det brukes av bl.a. Don Norman og i læreboka.
Gi minst to eksempler fra datasystemer, og to fra andre produkter.
Hvorfor er begrepet nyttig i interaksjonsdesign?

b. Gestaltpsykologi
Forklar de 3-4 viktigste gestaltprinsippene relatert til visuell komposisjon.
Hvorfor er det nyttig å ha kjennskap til disse prinsippene når man skal komponere et skjermbilde?

Hvilke implisitte sammenhenger mellom elementene kan leses ut av layoutet under.
Relater dette til gestaltprinsippene. Anta at mønsteret i elementene I og K er en farge.



Oppgave 2 (30%) Designprosessen og evaluering

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

> ... *"the effectiveness, efficiency, and satisfaction with which specified users achieve specified goals in particular environments"*

På norsk: "Anvendbarhet, effektivitet og tilfredsstillelse for bestemte brukere med bestemte mål i bestemte omgivelser".

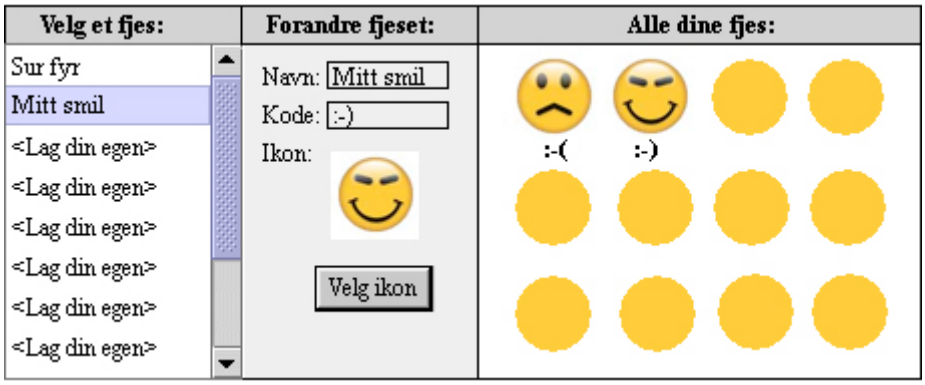
a. En konsekvens av denne definisjonene er at brukskvalitet er en kontekstavhengig egenskap ved et produkt. Hvilke konsekvenser har dette for utviklingen av brukervennlige produkter?
b. Definisjonen sier at det som skal måles er anvendbarhet, effektivitet og tilfredsstillelse. Hvordan måler man dette i en brukbarhetstest? Hvilke av disse målene er kvantitative og hvilke er kvalitative?

Oppgave 3 (40%) Grensesnittkonstruksjon

Du skal i denne oppgaven vise bruk av SWING-komponenter og MVC-arkitektur for et gitt eksempel.

Eksempelet som er skissert i figur 1 og figur 2 er hentet fra et tenkt chatteprogram for barn. Brukeren skal kunne definere seg "fjes" for tilhørende bokstav-kombinasjoner. For enkelthets skyld er antall fjes satt fast til 12 stk.. For hvert fjes som brukeren ønsker skriver han/hun inn et navn, en bokstavkombinasjon (kode) og velger seg et ikon fra en fast meny av 35 slike ved å trykke på knappen "Velg ikon".

Til venstre i figur 1 er en liste med de 12 fjesene. Programmet starter med at ingen fjes er definert. Når brukeren velger et fjes kan han/hun editere på definisjonen i området i midten. Området til høyre viser alltid en totaloversikt over alle de 12 fjesene med tilhørende koder (dersom definert). Fjes som ikke er definert vises som en tom sirkel uten kode. Området til høyre er passivt, d.v.s. at ingenting skjer når disse ikonene blir trykket på. Området til høyre oppdateres automatisk når det skjer forandinger i definisjonen av fjesene.



Figur 1, Grensesnittet for å definere egne fjes.

Figur 2. Popup for å velge et ikon.

Figur 2 viser dialogboksen som kommer opp for å velge ikon blant de 35 mulige. Denne dialogboksen skal ikke beskrives i løsningen. Figur 2 er kun med for å vise hva brukeren ser. Det er kun grensesnittet i figur 1 som skal beskrives i løsningen.

Du kan anta at ikonene ligger som ikonfiler med navn ”ikon1.jpg”, ”ikon2.jpg” o.s.v. Ikonet for den tomme sirkelen heter ”sirkel.jpg”. Du kan videre anta at du kan bruke SWING-klassen JLabel for å tegne ut ikoner og ikoner med tekst. (Se vedlegget for detaljer). Det betyr at du aldri behøver å lagre selve ikonene, kun stringer som angir hvilke filer de skal hentes fra.

Du skal i løsningen ikke gjøre rede for mekanismene for layout. Fokus i oppgaven er på informasjonsflyt og datastrukturer. Du skal heller ikke beskrive mekanisme for scrolling i listen.

Besvarelsen skal inneholde følgende:

- En forklaring av hvordan du vil gjøre bruk av MVC prinsippet i din løsning.
- Hvor ligger informasjonen om de 12 fjesene?
- Tegn opp de klassene du definerer, og deres metoder og relasjoner.
- Tegn opp objektene som vil eksistere under kjøring og deres relasjoner.
- Tegn opp sekvensdiagrammer for følgende:
  - Initielt når objektene skapes og vinduet vises første gang.
  - Når brukeren velger et fjes i listen til venstre.
  - Når brukeren forandrer bokstavkode for valgte fjes ved å taste inn en ny kode i feltet for ”kode” i midten.

I vedlegget ligger nødvendig definisjon av en del nyttige klasser og grensesnitt.

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 Jlist

JList is a component that allows the user to select one or more objects from a list. A separate model, ListModel, represents the contents of the list. The selection state is managed by a separate delegate object, an instance of ListSelectionModel.

The contents of a JList can be dynamic, in other words, the list elements can change value and the size of the list can change after the JList has been created. The JList observes changes in its model. A correct implementation of ListModel notifies it's listeners each time a change occurs. Simple dynamic-content JList applications can use the DefaultListModel class to store list elements. This class implements the ListModel interface and provides the java.util.Vector API as well.

Methods:

public void setModel(ListModel model)

Sets the model that represents the contents or "value" of the list.

public void addListSelectionListener(ListSelectionListener listener)

Adds a listener to the list that's notified each time a change to the selection occurs.

public void setSelectionMode(int selectionMode)

Determines whether single-item or multiple-item selections are allowed. ListSelectionModel.SINGLE_SELECTION: Only one list index can be selected at a time.

class DefaultListModel implements ListModel

A data model that provides a List with its contents (a ListModel).

Methods:

public void addElement(Object obj)

Adds the specified component to the end of this list.

public Object getElementAt(int index)

Returns the component at the specified index.

public void setElementAt(Object obj, int index)

Sets the component at the specified index of this list to be the specified object.

interface ListSelectionListener

The listener that's notified when a lists selection value changes.

Methods:

public void valueChanged(ListSelectionEvent e)

Called whenever the value of the selection changes.

class ListSelectionEvent

An event that characterizes a change in the current selection. The change is limited to a row interval. ListSelectionListeners will generally query the source of the event for the new selected status of each potentially changed row.

Methods:

public int getFirstIndex()

Returns the index of the first row whose selection may have changed. If the selection mode is ListSelectionModel.SINGLE_SELECTION, this method returns the selected row starting at row 0.

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 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()

Returns:
The object on which the Event initially occurred.

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.

public void setIcon(Icon icon)

Defines the icon this component will display.

Merk: Selve ikonet blir laget v.h.a. hjelpeklassen ImageIcon, slik at kallet for å hente et ikon for en JLabel fra filen "myImage.jpg" blir:

myJLabel.setIcon(new ImageIcon("myImage.jpg"))

class JButton

An implementation of a "push" button.

Methods:

void addActionListener(ActionListener l)

Adds an ActionListener to the button.

void setText(String text)

Sets the button's text.