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 15. august 2011
Tid: kl. 0900-1300

Bokmål

Sensuren faller 5. september

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

Oppgave 1 (25%) 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. Gi eksempler relatert til design av brukergrensesnitt. Hvorfor er det nyttig å ha kjennskap til disse prinsippene når man skal komponere et skjermbilde?

Oppgave 2 (35%) 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".

En konsekvens av denne definisjonene er at brukskvalitet er en kontekstavhengig egenskap ved et produkt. Hva menes med dette? Hvilke konsekvenser har dette for planlegging og gjennomføring av brukbarhetstester?

Oppgave 3 (40%) Grensesnittkonstruksjon

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

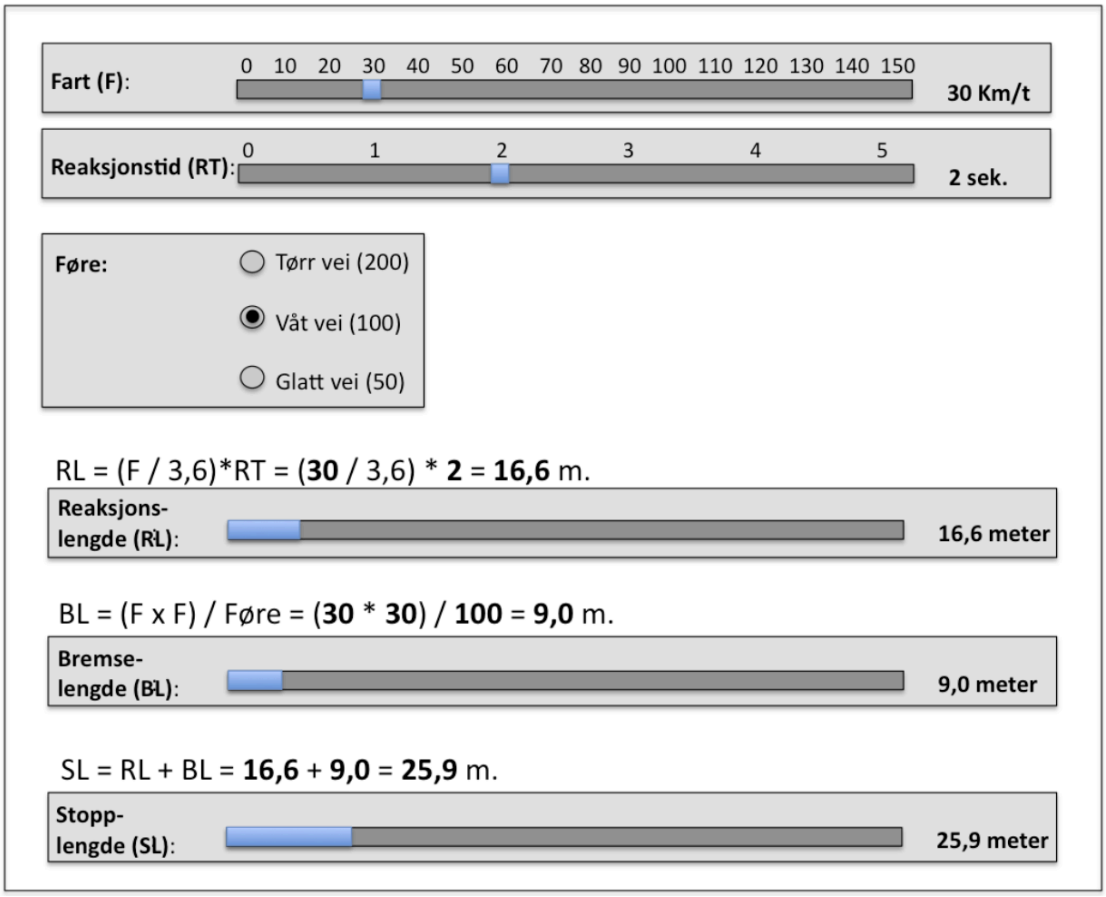
Du har fått i oppdrag å lage en del av et interaktivt spill for å lære kjøreskoleelever om reaksjonslengde, bremselengde og stopplengde

Stopplengden er den totale lengden bilen beveger seg fra føreren blir klar over en fare til bilen stopper. Stopplengden er summen av reaksjonslengden og bremselengden. Reaksjonslengden er lengden bilen beveger seg fra føreren blir klar over en fare til bremsepedalen trykkes inn. Bremselengden er lengden bilen kjører fra bremsepedalen trykkes til bilen står stille.

Følgende formler gjelder:
Reaksjonslengde = (Fart / 3,6) * Reaksjonstid
Bremselengde = (Fart x Fart) / Føre
Stopplengde = Reaksjonslengde + Bremselengde

Føre:

Tørr vei:	200
Våt vei:	100
Glatt vei:	50



Figuren over er en skisse av hvordan kunden ønsker at det skal se ut for brukeren.

Fart og reaksjonstid kan endres med en slider. Føre kan velges med radioknapper. Dette er de eneste tre tingene brukeren kan gjøre.

Systemet regner så ut reaksjonslengde, bremselengde og stopplengde, og viser dette som søylediagrammer og tekst. Tekstene og søylene skal oppdateres automatisk hver gang noe endres. Husk også at teksten til høyre for slidere og søyler skal oppdateres. Søylediagrammer kan lages i SWING v.h.a. en JPanel (for eksempel blå) i en annen JPanel (for eksempel grå).

Du skal i løsningen ikke gjøre rede for mekanismene for layout eller selve utregningen av formlene.

Besvarelsen skal inneholde følgende:

- En forklaring av hvordan du vil gjøre bruk av MVC prinsippet i din løsning.
- Hvor ligger informasjonen, og hvordan gjøres beregningene. Beskriv modellklassen(e) for din løsning?
- Tegn opp de klassene du definerer, og deres metoder og relasjoner.
- Tegn opp sekvensdiagrammer for følgende:
  - Initielt når objektene skapes og vinduet vises første gang.
  - Når brukeren endrer farten.
  - Når brukeren endrer føre.

Klassen *JLabel* 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);

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.**

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 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.

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 JRadioButton(String text)
Creates an unselected radio button with the specified text.
Parameters:
`text` - the string displayed on the radio button

public void addActionListener(ActionListener l)
Adds the specified action listener to receive action events from this JToggleButton.

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

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 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