NORGES TEKNISK-NATURVITENSKAPELIGE UNIVERSITET
INSTITUTT FOR DATATEKNIKK OG INFORMASJONSVITENSKAP

Faglig kontakt under eksamen:

Dag Svanæs, Tlf: 73 59 18 42

LØSNINGSFORSLAG (LF)

TDT4180 MMI

Fredag 10. juni 2011

Tid: kl. 1500-1900

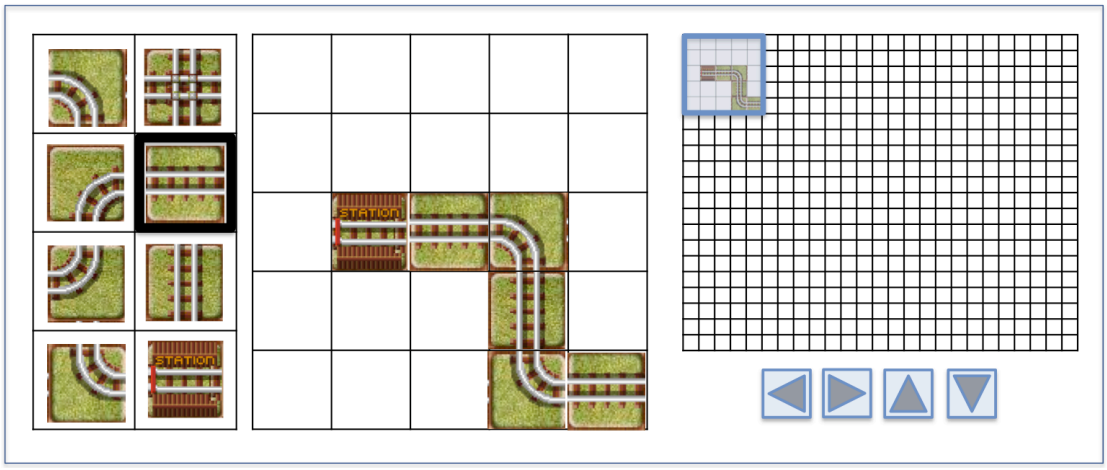
Bokmål

Sensuren faller 1. juli 2011

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

Oppgave 1. Grensesnittdesign (30%)

Figuren under er en skisse til en del av et nettbasert jernbanespill som skal lages som en Java applet.



Målgruppen for spillet er barn fra 6 år. Spillet skal gjøre det mulig å kjøre tog i selvlagede baner. Den delen av spillet som er avbildet i figuren over skal brukes av spillerne til å lage sine egne baner. Det er bare denne delen av spillet vi skal holde på med her.

Til venstre ser vi 8 ikoner for forskjellige baneelementer. De kan plasseres ut på en 25x20 matrise. Brukeren velger hvilket ikon som skal være aktivt tegneelement ved å trykke på baneelementet. I figuren over er for eksempel rett horisontalt baneelement valgt. Når aktivt baneelement er valgt, så kan brukeren plassere dette ut på 5x5 utsnittet i midten ved å trykke på den ruten som valgt baneelement skal tegnes i. Brukeren "tegner" med dette baneelement inntil et annet baneelement velges.

Til høyre kan brukeren se hele banen (25x20 ruter). Brukeren kan ikke "tegne" direkte i denne, men v.h.a. pilknappene under kan han/hun skifte utsnitt. Knappene flytter utsnittet 5 ruter av gangen, venstre, høyre, opp eller ned. Når utsnittet flyttes, så tegnes også riktig utsnitt ut i 5x5 matrisen, og det er dette utsnittet brukeren "tegner" i.

a. *I eksempelet over så er det gjort en del valg i forhold til dialogform/interaksjonsteknikk. Beskriv hvilke valg som er gjort, og diskuter hvilke fordeler og ulemper du ser ved dette valget.*

b. *Beskriv tre andre dialogformer/interaksjonsteknikker som kunne ha vært brukt i dette tilfelle. Gjerne noen som er åpenbart uegnede. Tegn en skisse av brukergrensesnittet, og diskuter fordeler og ulemper ved hver av disse i forhold til brukergruppen.*

Løsningsforslag oppg. 1.

1a.

Det er valgt å bygge opp en meny av ikoner, der man "tegner" med valgt ikon. I tillegg så er det valgt en løsning med et vindussutsnitt som kan flyttes v.h.a. piltaster på skjermen.

Fordeler i forhold til målgruppe:

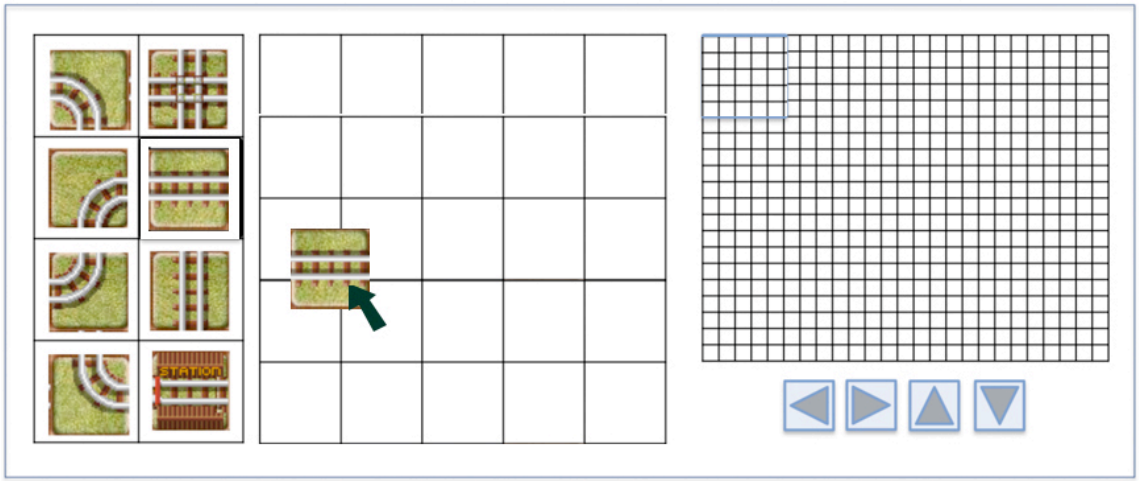
- Dette er en enkel løsning, der alle valg er gjort synlig. Altså "Knowledge in the world", minst mulig "Knowledge in the head".
- Dette er også i forhold til at pilknappene for å flytte vinduet er gjort synlige på skjermen. De kunne ha vært gjort direkte knyttet til pilene på tastaturet, men da måtte kanskje noen fortelle barna at de skulle bruke piltastene.

Ulemper:

- Vil barn på 6 år forstå et vindusutsnitt? Kanskje. Det må nok testes.
- Det mangler muligheter for å angre. Det er jo et minus.
- Er det intuitivt, eller må det læres? Vanskelig å si.

1b.

Alternativ 1: Drag & Drop



Med drag & drop så kan brukeren dra ikoner ut på "banen".

Fordeler:

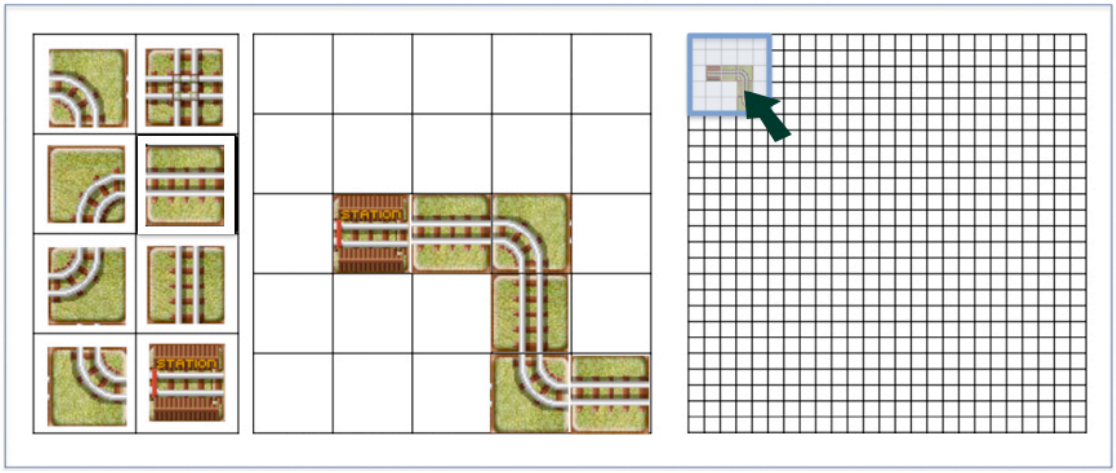
- Det kan oppleves som mer intuitivt, og mer morsomt.

Ulemper:

- Det må kanskje læres. Det er ikke umiddelbart intuitivt at ikoner kan dras.

- Det tar litt lenger tid dersom man skal ”tegne” mange like ikoner. Med den opprinnelige løsningen så gikk det fort når man først hadde valgt en ”tegnefarge”.

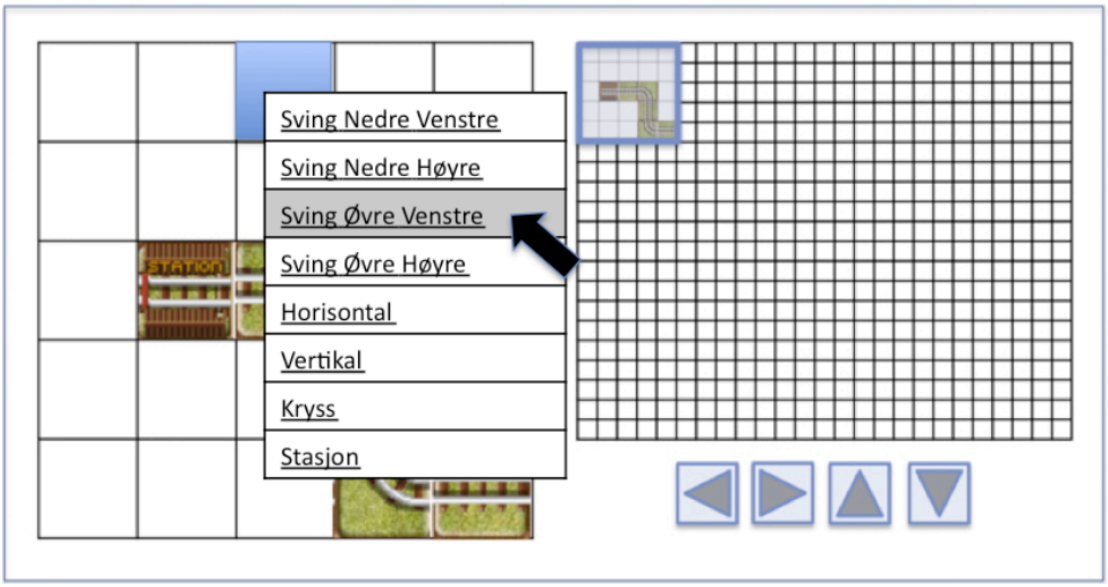
Også flytting av utsnittsvindu kan gjøres direkte med mus, som illustrert under:



Fordelen med dette er at man får mer skjermplass, mens ulempen igjen er at det må kunnes eller vites.

Man kunne i tillegg ha tastaturpilene som hurtigtaster, noe som er en fordel for erfarne brukere; noe 6-åringene jo fort blir ☺.

Alternativ 2: Popup meny



Med denne løsningen så får brukeren opp en popup meny ved å trykke høyreknappen over en rute som skal tegnes. Brukeren velger så fra en liste med tekstlige beskrivelser av ikoner.

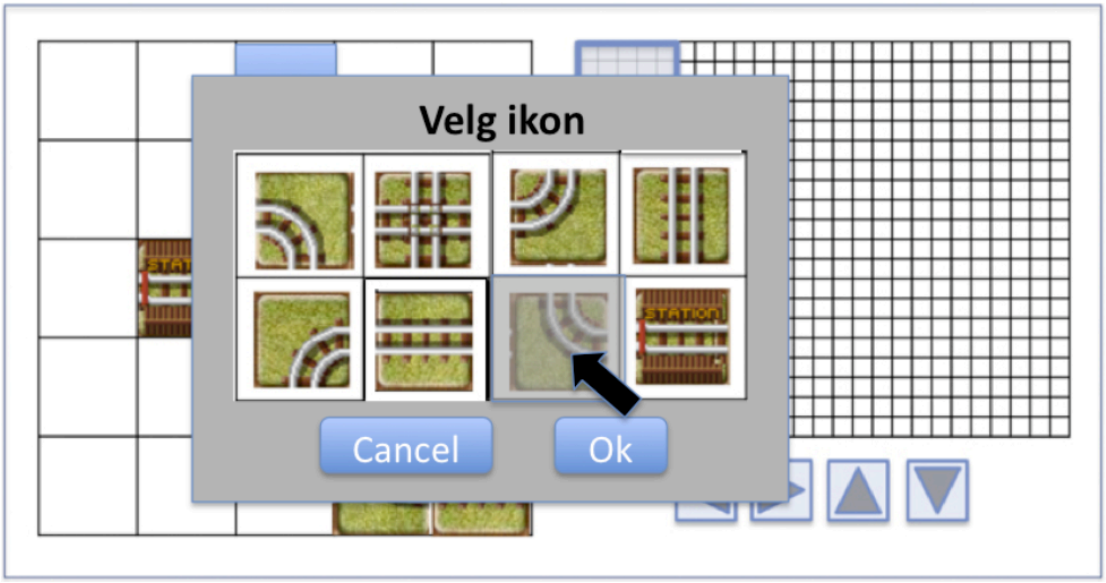
Fordel:

- Det frigjøres skjermplass. Ikke lenger behov for ikoner på skjermen.

Ulemper:

- Jeg synes ikke dette er en god løsning for 6-åringer. De må lære å bruke høyreknapp, og de må lese tekst.
- Forstår de venstre/høyre/øvre/nedre? Neppe.
- Dette er mer en løsning for voksne.

Alternativ 3: Dialogboks



Med denne løsningen så får brukeren opp en dialogboks ved å trykke museknappen over en rute som skal tegnes. Brukeren velger så fra en liste med ikoner.

Fordel:

- Det frigjøres skjermplass. Ikke lenger behov for ikoner på skjermen.

Ulemper:

- Jeg synes ikke dette er en god løsning for 6-åringer., kanskje ikke egentlig for noen.
- Det tar mye tid, og dialogboksen skjuler det man allerede har laget.

Av de tre alternativene så er det bare drag&drop som jeg tror kanskje kan bli like brukervennlig som forslaget i oppgaveteksten.

Karakterguide for sensor, 1a + 1b:

Av andre løsninger som er gjennomgått i faget kan nevnes:

*** Tekstinput som i Unix med eget inputspråk, for eksempel ">TEGN B,5,VERT" for å tegne et vertikalt ikon i rute B5 på et sjakkbrett.**

*** Tvungen dialog, som i et installasjonsprogram. Det ville da bli en rekke dialogbokser som spurte for eksempel "Velg målrute", "Velg ikon",,,.**

A: En god gjennomgang som i LF, med tegninger og argumentasjon i forhold til målgruppen. Ting som at ikke alle 6-åringer kan lese er viktig.

B: En god gjennomgang som i LF, med tegninger og argumentasjon. Mindre fokus på målgruppen, og kanskje litt mindre argumentasjon.

C: Det er tegnet opp listet opp minst tre alternativer, og det er minst en god og en dårlig ting ved hver av de. Ikke nødvendigvis tenkt på målgruppe.

D: Færre illustrasjoner, og dårlig argumentasjon.

E: Enda mindre..

F: Ikke noe…

Oppgave 2. Designmetodikk (30%)

Du har fått i oppgave å designe og utvikle den delen av spillet i oppgave 1 som er beskrevet over. Det legges opp til en brukernær utviklingsprosess.

Målgruppen for spillet er som nevnt barn fra 6 år og oppover. Du har fått en avtale med en barneskole at du kan få lov til å komme på besøk i en 1. klasse gruppe med fire elever i en dobbelttime en gang i uken i tre etterfølgende uker.

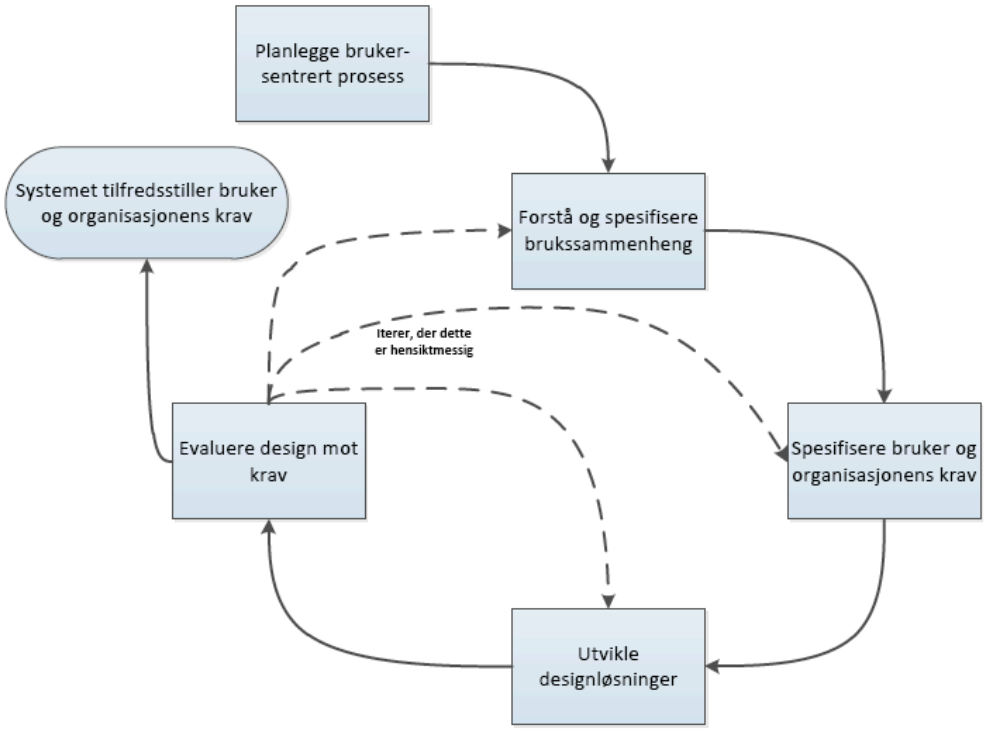
Du har altså tilgjengelig 4 representanter for målgruppen to timer tre ganger, med en ukes mellomrom. Det er to uker til første gang du skal møte elevene. En uke etter siste besøk på skolen skal du ha klart et design som kan programmeres av noen andre enn deg selv.

- *Målet er å gjøre best mulig nytte av den tiden du har fått tildelt. Planlegg hvilke brukernære aktiviteter du vil gjøre hver av de tre gangene du er på besøk på skolen, og i tiden før, mellom og etter besøkene. Diskuter de valgene du har gjort i forhold til andre valg du kunne ha gjort. Du kan anta at du kan ta med deg en eller flere medarbeidere fra din arbeidsplass som behersker brukernær utvikling. Du har også budsjett til å kjøpe inn diverse materialer og/eller datautstyr.*

Løsningsforslag oppg. 2.

Det er altså syv perioder som det skal planlegges for:

1. To uker fra nå til første møte med brukerne.
2. Første møte
3. Mellom første og andre møte
4. Andre møte
5. Mellom andre og tredje møte
6. Tredje møte
7. Etter tredje møte, før overlevering av design.



Det finnes mange måter å gjøre dette på. En løsning er å ta utgangspunkt i ISO-standarden for brukersentrert utvikling (13407 / 9241-210). Den sier at man først skal planlegge. Deretter skal man forstå brukeren og brukskonkteksten, spesifisere krav, lage prototyper og evaluere. Så skal man iterere.

Første uke som man nå er i skal brukes til å planlegge prosessen. Deretter må man planlegge første besøket på skolen.

Jeg forstår oppgaven slik at det er gitt at det skal lages et spill for å kunne lage egne jernbaner og leke med de banene man har laget. Det er ikke sagt noe om hva hensikten egentlig er, og jeg regner derfor med at det hele rett og slett er et spill for å ha det artig. Det kan allikevel være lurt å bruke litt tid på å forstå brukerne og deres brukssammenheng.

Mye av oppgaven vil da være å lage et godt brukergrensesnitt. D.v.s at mye av aktivitetene skal sirkulere rundt prototyp-bygging og evaluering.

Første møte:

I det første møtet så vil jeg allikevel begynne med å finne ut hvor mye data som ungene behersker. Det vil jeg finne ut ved å ta med en PC, en mobil, en iPad og en XBox og se hva ungene får til i en del spill.

Jeg vil deretter sette de fire barna i en sirkel og spørre de litt om hvor mye de spiller, hva de spiller, hva de spiller på (PC, XBox, mobil,,), hvor de spiller, om de spiller sammen med andre barn, og om hva foreldrene mener om spillingen deres.

Deretter vil jeg har gjort klar to papirprototyp som jeg vil teste på elevene, en som i oppgaveteksten, og en med drag&drop. Jeg ville ha med tre medhjelpere

slik at jeg kunne kjøre to tester i parallelt. Teste med par av elever. Et guttepar, og et jentepar for å få begge kjønn sine perspektiv med.

Jeg ville teste begge papirprototypene. Begynne veldig åpen med å fortelle hva den skulle brukes til, og så spørre de hva de tror de må gjøre for å få det til.

Det hele ville jeg ha tatt opp på video.

Mellom møte 1 og 2:

Mellom møtene så ville jeg ha laget en powerpoint prototyp, basert på det de fortalte oss. Den skal kunne fake en del av interaksjonene, men være ganske komplett mh.t. layout. Jeg ville så samtidig ha begynt på en Flash eller SWING prototyp med mer interaktivitet som skulle brukes i det siste møtet.

Jeg ville også ha sammenfattet det jeg lærte om spillvaner og teknologikomptanse for å ta med videre i prosjektet. Det kunne kanskje være utgangspunkt for et par personas og noen scenarier som skal følge med designet til programmerne.

Andre møte

Nå vil jeg la ungene teste ut powerpoint prototypen, og jeg ville etterpå ha brukt en projektor til å få opp skjermbildet på lerret. Så ville jeg ha kjørt en slags fokusgruppe der vi gikk igjennom alle delene av skjermbildet. Dette kan nok fort bli litt kjedelig for ungene, så det er viktig å ha noe annet å gjøre innimellom. Spilling, at de kan tegne togbaner på papir. Dette må vi tenke ut mer i detalj etter første møte.

Mellom møte 2 og 3

Nå blir det hektisk med å oppsummere hva vi har lært, og få det inn i en Flash eller SWING prototyp. Mye programmering og skjermdesign denne uken.

Tredje møte

Nå har vi med en kjørende prototyp som vi tror er bra. Den må ungene få prøve seg på. Gjerne på to PCer i parallell. De må få lage baner selv, og vi må følge nøye med på hva de får til og hva de har problemer med. Viktig at de får med seg utskrifter på papir av banene de har laget. Det skal føre til noe.

Ta god tid til å avslutte. Få ungene til å kommentere på om dette er enkelt. Om det er morsomt etc… Ha med noen gaver som takk for strevet. Også til SFO-personalet.

Etter tredje møte

Det er nå bare en uke til designdelen av prosjektet er over. Oppsummer funnene fra siste møtet. Lag en enkel rapport med skjermbilder og forklarende tekst. Legge ved kjørende Flash eller SWING prototyp. Hvis tid, også lage to personas (en gutt og en jente), og plasser de inn et ett eller to hverdagsscenarier der de lager togbaner.

Prøver å få til et overleveringsmøte på en time eller to med programmererne der det blir mulighet for demo, gjennomgang av skjermbilder, personas og scenarier. Det er viktig å formidle til programmerne at det endelig resultatet er basert på grundig dialog med brukerne, og at de ikke skal endre på ting uten en veldig god teknisk grunn.

Karakterguide for sensor, oppg. 2:

A: En detaljert plan som den over som tar utgangspunkt i iterative prosesser med brukersentrert design. Viktig at det anbefales en kombinasjon av metoder, og at de er tilpasset prosjektet og målgruppen. Konkrete og begrunnede forslag til aktiviteter med brukere og aktiviteter mellom møtene.

B: Litt mindre begrunnede og konkrete forslag. Må ha med noe om iterative prosesser. Må forholde seg til caset.

C: Viser at det krever iterative prosesser. Ikke helt tilpasset caset, men allikevel en plan som kan følges.

D: Kun stikkordmessig og ganske abstrakt om metodene. Ikke relatert til caset.

E: Enda mindre..

F: Ikke noe…

Oppgave 3. Konstruksjon (40%)

a. *Hva mener vi med en Model-View-Controller (MVC) arkitektur, og hvordan er MVC realisert i Java/SWING?*

b. *Spillet i oppgave 1 skal realiseres i Java/SWING v.h.a. en MVC-arkitektur. Beskriv modellen/modellene du vil lage. Hva er de private variablene, og hva er get/set metodene?*

c. *Beskriv hvordan du vil realisere brukergrensenittet v.h.a SWING elementer. I vedlegget finnes en oversikt over en del relevante SWING elementer.*

d. *Tegn opp klassediagrammet for de klassene du vil lage.*

e. *Lag sekvensdiagrammer for følgende tre situasjoner:*

- *Brukeren trykker på høyreknappen på skjermen, slik at valgt utsnitt flyttes 5 ruter til høyre.*
- *Brukeren velger et nytt aktivt baneelement-ikon, for eksempel vertikalt baneelement, ved å trykke på dette.*
- *Brukeren plasserer valgte baneelement i en av rutene i 5x5 matrisen ved å trykke i denne ruten. Oppdatering skal da automatisk skje i 25x20 matrisen.*

Løsningsforslag oppg. 3.

3a.

Model-View-Controller (MVC) er en software arkitektur som muliggjør å skille datalaget fra presentasjonslaget i en applikasjon. I Java/Swing er View og Controller ofte sydd sammen i ferdige GUI komponenter som for eksempel JButton. Datene/tilstandene legges i modell-objekter. Disse kobles til View-Controller. Når en verdi endres i et View så er det viewets ansvar å oppdatere modellen. Endringer i modellen propageres så til alle Views som har denne modellen som sin modell. På denne måten vil tilstanden til alle Views til enhver tid være oppdatert fordi dataene kun ligger ett sted, nemlig i modellen.

Karakterguide for sensor, oppg. 3a:

A: En besvarelse som har med seg alle elementene over.

B: Noen små mangler, men får fram hovedvirkemåten.

C: Får fram hovedvirkemåten.

D: Veldig kort.

E: Nesten ingenting

F: Tomt.

3b.

Jeg kaller modellen RC_Model (RC for Rail Constructor).

Jeg vil her lagre en 20x25 matrise av ikoner. Ikonene kan typisk være en tallkode pr. stk., for eksempel 1-8, og null for blankt felt.

De private dataene er altså en 20x25 matrise av integer.

For å aksessere disse vil jeg ha en set og en get metode:

- void setIcon(int x, int y, int iconID)
- int getIcon(int x, int y)

  Alle modeller må kunne legge til lyttere for endringshendelser. Jeg velger å bruke samme navn på denne metoden som i PropertyChangeListener.

Modellen må da altså ha en privat variable for å holde på sin PropertyChangeListener. Metoden for å legge til lytter blir da:

- public void addPropertyChangeListener(PropertyChangeListener listener)

Det betyr av Views som vil lytte må implementere interfacet PropertyChangeListener, altså metoden propertyChange.

Kall til setIcon fører til at endringshendelser sendes til alle lyttere som er lagt til v.h.a. addPropertyChangeListener.

I tillegg så trengs det en modellmekanisme for å håndtere flytting av utsnittsvindu. Når utsnittsvinduet flyttes i vinduet til høyre så må ting oppdateres i vinduet i midten. Dette kan enten gjøres som en egen modell, eller som en integrert del av modellen RC_Model. Jeg velger her for enkelhets skyld siste løsning.

RC_Model må da ha private data for å holde på et origo. Altså øvre venstre hjørne av utsnittsvinduet. Dette er en x og en y, for eksempel origoX og origoY.

Det må så lages set og get metoder for disse:

- void setOrigoX(int x)
- void setOrigoY(int y)
- int getOrigoX()
- int getOrigoY()

setOrigoX og setOrigoY må så på samme måte som setIcon føre til endringshendelser.

Karakterguide for sensor, oppg. 3b:

A: En besvarelse som har med seg alle elementene på et arkitekturnivå. Har med modell både for ikonene og for utsnittsvinduet.

B: Kun modell for ikonene.

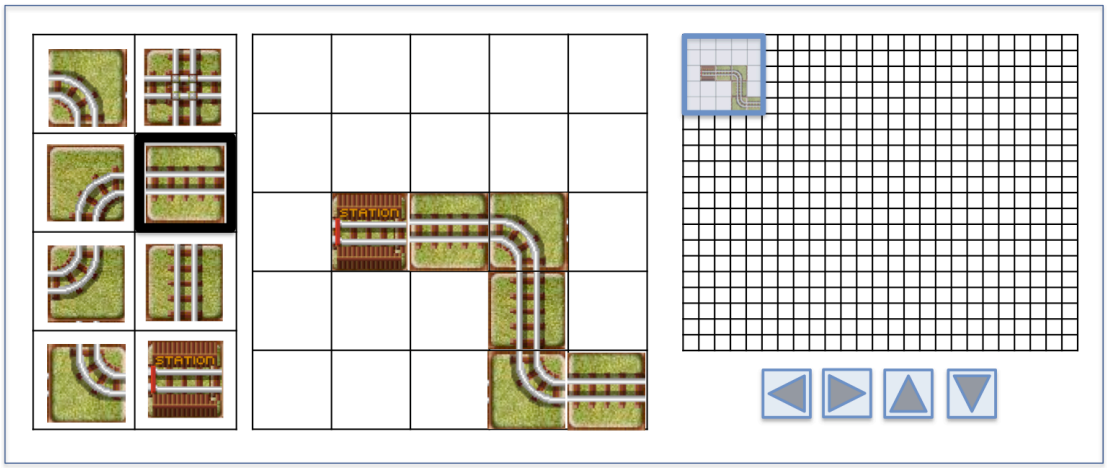
C: Noen svakheter.

D: Veldig kort.

E: Nesten ingenting

F: Tomt.

3c.



Jeg ville ha tre store JPanel i dette vinduet:

- RC_IconView som inneholder de 8 ikonene å velge fra.
- RC_SmallView for 5x5 matrisen.
- RC_LargeView for 20x25 matrisen og de fire knappene.

I RC_IconView ville jeg ha 8 JToggleButton som var koblet sammen som en ButtonGroup for å gi de "radiobutton" oppførsel (kun en valgt av gangen).

I RC_SmallView kunne jeg også ha brukt JToggleButton, eller den enklere varianten JButton som ikke har toggle oppførsel. Jeg ville skape 25 slike, som var lagt til JPanel RC_SmallView.

I RC_LargeView kan jeg bruke 20x25 JToggleButton som er disabled, eller JLabel. I tillegg 4 knapper som JButton. Det flyttbare vinduet kan ordnes v.h.a. rammer. Jeg går ikke så mye i detalj her.

Både RC_SmallView og RC_LargeView har en RC_Model som modell. Det gjør at endringer forplanter seg mellom de.

Når en knapp i RC_SmallView klikkes så spør den RC_IconView om hva som er aktivt "tegneikon". Den setter så tilsvarende verdi i RC_Model v.h.a. setIcon(x,y,verdi). Det fører så til korrekt opptegning både i RC_SmallView og RC_LargeView.

Når brukeren tykker på en a piltastene i RC_LargeView så sendes det endringsmeldinger til RC_Model med ny origoX og origoY. Det fører så til endringshendelser både i RC_SmallView og i RC_LargeView (nytt utsnitt tegnes ut i RC_SmallView og utsnittmarkeringen flyttes i RC_LargeView).

Karakterguide for sensor, oppg. 3c:

A: En besvarelse som har med seg alle elementene på et arkitekturnivå.

B: Har gjort valg av SWING-komponenter for alle skjermelementer. Forklarer hovedvirkemåten.

C: Har gjort valg av SWING-komponenter for alle skjermelementer. Kortfattet på virkemåte.

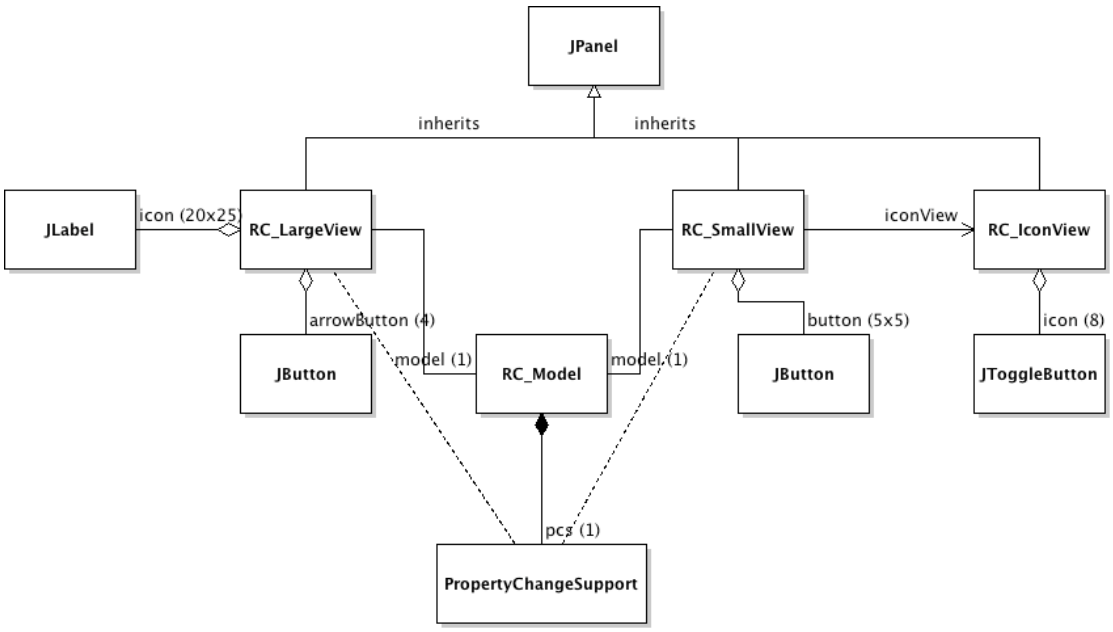
D: Veldig kort.

E: Nesten ingenting

F: Tomt.

3d.

Klassehierarkiet gjenspeiler det som er beskrevet i 3c:



Vi ser at både RC_LargeView og RC_SmallView har en referenanse til en RC_Model. RC_SmallView vet om sin RC_IconView for å kunne spørre om valgt tegneikon. Alle tre views har så sine knapper og labels.

RC_Model har et PropertyChangeSupport objekt. Det er satt på en stiplet linje tilbake til RC_LargeView og RC_SmallView for å indikere at disse blir referert til når RC_Model er blitt kalt med addPropertyChangeListener for disse to.

<u>Karakterguide for sensor, oppg. 3d:</u>

A: En besvarelse som har med seg alle elementene på et arkitekturnivå.

B: Viser kobling til modell, men kanskje ikke så omfattende ellers.

C: Er i samsvar med oppgave 3c.

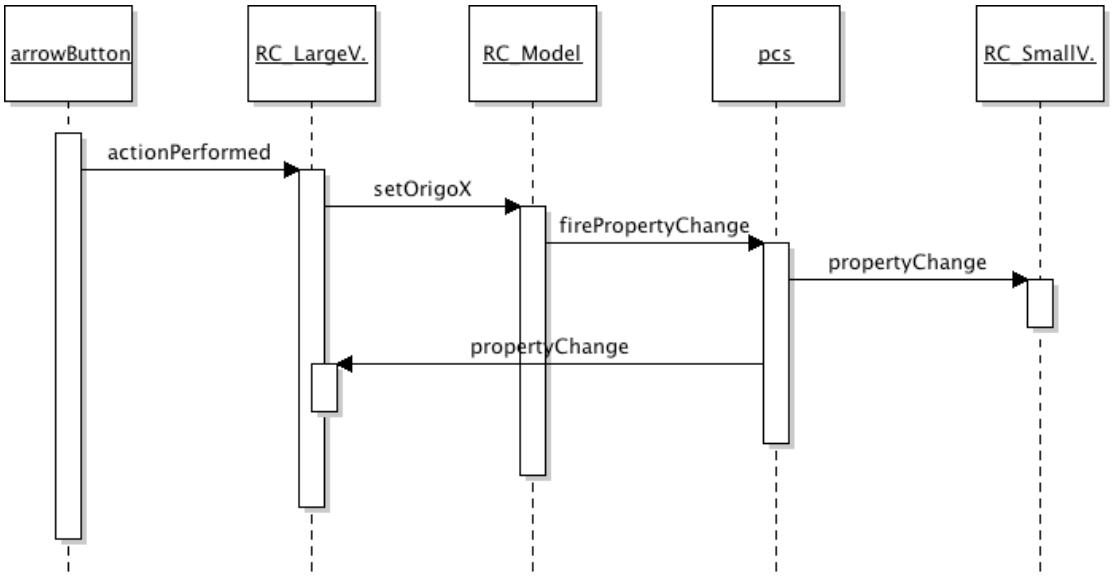
D: Veldig kort.

E: Nesten ingenting

F: Tomt.

3e.

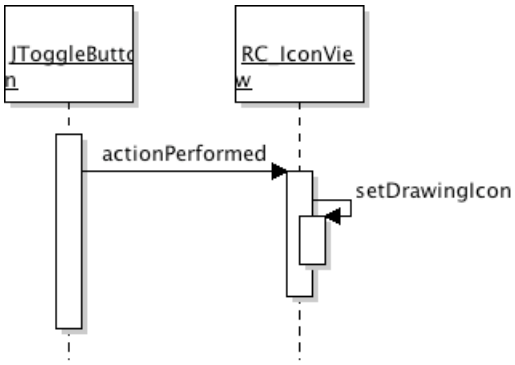
(I) Brukeren trykker på høyreknappen på skjermen, slik at valgt utsnitt flyttes 5 ruter til høyre.



Høyrepil knappen er satt opp med sin RC_LargeView som sin actionhandler. Den sender da beskjed til sin modell om å flytte origo 5 hakk mot høyre. Modellen sender da endringshendelse til sitt pcs-objekt. Det sender endringmeldinger til RC_LargeView og RC_SmallView.

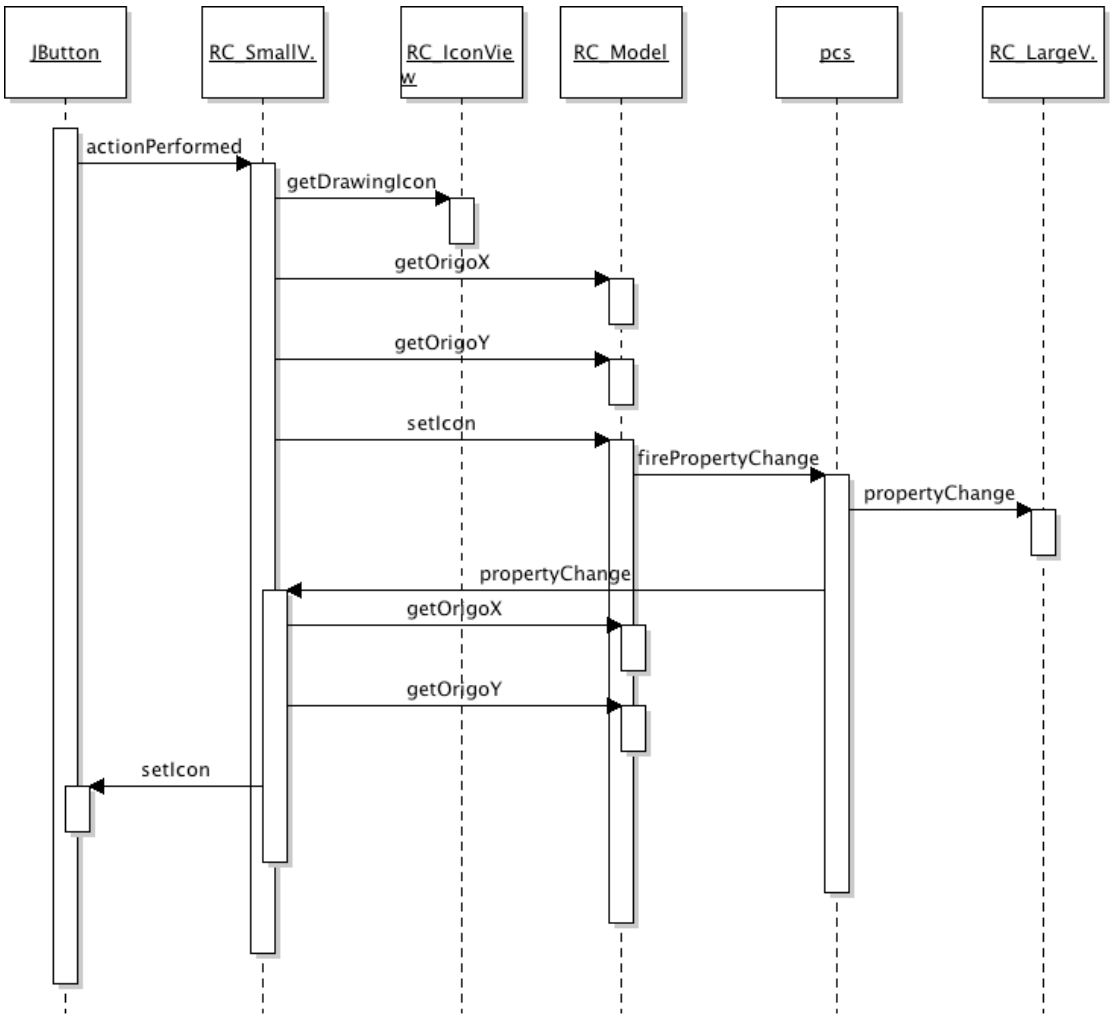
Disse vil så igjen gjøre oppdateringer ved å spørre RC_Modell om nytt origo og innhold av riktige elementer i matrisen.

(II) Brukeren velger et nytt aktivt baneelement-ikon, for eksempel vertikalt baneelement, ved å trykke på dette.



Ikonet har sin actionHandler i sitt RC_IconView. Det eneste denne gjør er å huske på hvilket ikon som er valgt ikon. Ved å kalle seg selg ”setDrawingIcon”.

(III) Brukeren plasserer valgte baneelement i en av rutene i 5x5 matrisen ved å trykke i denne ruten. Oppdatering skal da automatisk skje i 25x20 matrisen.



Gangen er som følger:

En JButton-knapp i RC_SmallView blir trykket. Det fører til at denne får en melding ettersom den er satt opp som lytter på alle knapper.

RC_SmallView spør så RC_IconView om hva som er valgt tegnefarge.

RC_SmallView vet hvilken knapp i sitt vindu som er trykket (f.eks. 3,2), men for å vite hvilket banelement i modellen som skal endres så må det vite hvilket utsnitt det representerer. Det spør derfor modellen om hva som er origo for utsnittet. Det kan for eksempel få (4,4) som svar.

RC_SmallView setter så det riktige baneelementet i modellen ved å legge til (origoX,origoY), for eksempel (3+4,2+4), i koordinatet.

Når RC_Model objektet har fått et setIcon-kall så sender det beskjed om endring til sitt pcs-objekt.

Pcs-objketet sender så endringshendelser til de to lyttende vinduene av klasse RC_SmallView og RC_LargeVIew.

Her er det vist hva som skjer når RC_SmallView får en endringshendelse. Den må da spørre modellen om origo for utsnittet for å kunne beregne riktig lokalt element.

Den trekker fra (orgioX,origoY) og setter ikonet i den samme JButton som ble trykket.

Tilsvarende endringehendelse til RC_LargeView fører til at riktig JLabel der får endret sitt ikon.

Karakterguide for sensor, oppg. 3e:

A: En besvarelse som viser oppdateringsmekanisme som i LF. Får også med seg flytting av origo.

B: Mekanismer som oppdaterer ikonene.

C: En del mangler, men hovertrekkene i MVC er realisert.

D: Veldig kort.

E: Nesten ingenting

F: Tomt.

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 void addMouseListener(MouseListener l)

Adds the specified mouse listener to receive mouse events from this component.

Parameters:

l - the mouse listener

interface MouseListener

The listener interface for receiving "interesting" mouse events (press, release, click, enter, and exit) on a component.

The class that is interested in processing a mouse event implements this interfac).

The listener object created from that class is then registered with a component using the component's addMouseListener method. A mouse event is generated when the mouse is pressed, released clicked (pressed and released). When a mouse event occurs, the relevant method in the listener object is invoked, and the MouseEvent is passed to it.

Methods:

public void mouseClicked(MouseEvent e)

Invoked when the mouse button has been clicked (pressed and released) on a component.

class MouseEvent

An event which indicates that a mouse action occurred in a component.

Methods:

public Object getSource()

The object on which the Event initially occurred.

Returns:

The object on which the Event initially occurred.

class JButton

An implementation of a push button

Methods:

public JButton(Icon icon)

Creates a button with an icon.

Parameters:

icon - the image that the button should display

public void addActionListener(ActionListener l)

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

class JToggleButton

An implementation of a two-state 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 JToggleButton objects in the group.)

Methods:

public JToggleButton(Icon icon)

Creates an initially unselected toggle button with the specified image but no text.

Parameters:

icon - the image that the button should display

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