Lathund för GUI-design

Fokus på testbara modeller, frikopplade från där de används

# Så du vill ha en dialog med Swing-komponenter...

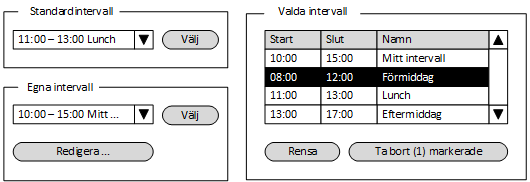
I normalfallet vill du kunna skicka in data med värden som säger något om det tillstånd applikationen befinner sig i till den nya dialogen, för att sedan ta emot data ifrån dialogen i ett senare moment. Du behöver avgöra om din dialog är modal eller ej – vilket i sin tur får följande designkonsekvenser:

* Modal (stannar all annan aktivitet i din applikation tills användaren väljer OK eller Avbryt)
  + Du förväntas inte påverka data i omgivningen innan ett OK
  + Vid ett avbryt ska inget alls ha påverkats.
  + I det här fallet vill du skicka in en modell som inte blir lyssnad på av något utanför dialogen och du behöver se till att valet användaren gör innan dialogen stängs, sparas till modellen så att du kan ta emot valet efter att dialogen stängts.
* Icke-modal
  + Dialogen är öppen och tillgänglig tills användaren stänger/döljer fönstret.
  + Förändringar i dialogen ska i realtid påverka den källa som startade den.
  + I det här fallet kan du skapa modellen som ovan, men istället för att kolla vad användaren valt i samband med att dialogen stängs, vill du registrera lyssnare på modellen från andra delar av systemet redan innan dialogen öppnas.
  + Lyssnarna blir notifierade när data i modellen förändras och de kan då anropa modellen för att hämta data och sedan anpassa sig till det nya tillståndet
    - Pattern: observer-subscriber

Vi kommer att gå igenom både en modal och en icke-modal lösning för dialogen ovan, men låt oss först kika på hur dess underliggande modell behöver se ut för att fungera ovasett dialogtyp.

# Börja med en designskiss

Rita med papper och penna, eller med valfritt designverktyg – men undvik att rita i ditt IDE eftersom det är lätt att låsa sig till genererad kod och betydligt lättare att ändra en skiss i det här läget. Därmed undviker du också frestelsen att skapa alla delar i en och samma dialog – det enda det leder till är en oöverskådlig soppa som i sin tur ofta blir ännu värre när du senare inser att olika delar av gränssnittet ska påverkas av andra delar i samma dialog.



Redan på pappret (dvs i designläget) kan du förenkla både för dig och för dina användare genom att dela upp dialogen i olika delar. Det gör både dialogens och de olika panelernas kod mer kompakt, lättläst och enklare att underhålla.

# Analys och design av dialogmodellen

Av skissen ser vi att vi kommer behöva ha någon form av värdeobjekt som identifierar ett givet tidsintervall och att det intervallet kan komma från två olika källor. Vi kan samtidigt anta att det finns ett för varje källa unikt ID per givet intervall. Det ger oss följande element-modell:

## TidsintervallVo

package com.golcher.tidsintervall.komponenter.data;  
  
public class TidsintervallVo  
{  
 private final String \_id;  
 private final TidsintervallKalla \_kalla;  
 private final String \_start;  
 private final String \_slut;  
 private final String \_namn;  
  
 public TidsintervallVo(  
 String \_id,  
 TidsintervallKalla \_kalla,  
 String \_start,  
 String \_slut,  
 String \_namn  
 )  
 {  
 this.\_id = \_id;  
 this.\_kalla = \_kalla;  
 this.\_start = \_start;  
 this.\_slut = \_slut;  
 this.\_namn = \_namn;  
 }  
  
 public String getId()  
 {  
 return \_id;  
 }  
  
 public TidsintervallKalla getKalla()  
 {  
 return \_kalla;  
 }  
  
 public String getStart()  
 {  
 return \_start;  
 }  
  
 public String getSlut()  
 {  
 return \_slut;  
 }  
  
 public String getNamn()  
 {  
 return \_namn;  
 }  
  
 @Override  
 public String toString()  
 {  
 return \_start + " - " + \_slut + ((\_namn.length() > 0) ? "(" + \_namn + ")" : "");  
 }  
}

## TidsintervallKalla

package com.golcher.tidsintervall.komponenter.data;  
  
public enum TidsintervallKalla  
{  
 *Standard*,  
 *Egen*}

Vidare kan vi av skissen se att dialogens komponenter har det gemensamt att de behöver tillgång till varsin modell som utgörs av en lista av värdeobjekt. Här är det lätt att göra misstaget att designa dialogens modell direkt utifrån komponenternas behov, genom att skapa tre listor. Låt bli det. Det leder till kod som blir både svårare att utöka på sikt och jobbigare att få en överblick över.

Gör hellre en analys utifrån helheten – det vi kan kalla dialogens datamodell. Dialogen som helhet ska kunna hantera alla valbara och valda element och kan hålla isär dem med deras unika id (kombinationen och \_id och \_kalla i TidsintervallVo). Det enda vi skulle behöva lägga till är en wrapper för att dessutom kunna hålla reda på om ett visst element är valt eller inte.

## TidsintervallWrapper

package com.golcher.tidsintervall.komponenter.data;  
  
public class TidsintervallWrapper  
{  
 private final TidsintervallNyckel \_nyckel; // Immutable key att använda i HashMap  
 private TidsintervallVo \_varde;  
 private boolean \_arVald;  
  
 public TidsintervallWrapper(TidsintervallVo vo)  
 {  
 \_nyckel = new TidsintervallNyckel(vo);  
 \_varde = vo;  
 \_arVald = false;  
 }  
  
 public void sattStatus(boolean arVald)  
 {  
 \_arVald = arVald;  
 }  
  
 public TidsintervallNyckel nyckel()  
 {  
 return \_nyckel;  
 }  
  
 public TidsintervallVo varde()  
 {  
 return \_varde;  
 }  
  
 public boolean arVald()  
 {  
 return \_arVald;  
 }  
  
 @Override  
 public String toString()  
 {  
 return \_varde.toString();  
 }  
  
 public void sattVarde(TidsintervallVo vo)  
 {  
 \_varde = vo;  
 }  
}

När vi har lagt till en sådan wrapper kan vi hantera modellen som en HashMap och sedan enkelt avgöra vilken komponent som ska ha ett visst element med en enkel regel – är elementet valt ska det visas med övriga valda oavsett källan, annars ska det visas i den komponent som utger sig för att vara för element av den källan.

För att kunna hantera element i en HashMap korrekt krävs en immutable (oföränderlig) nyckel. Vi vet redan att vi ska bygga vår av respektive elements ID och dess källa, men för att säkerställa att ingen missar hur det ska göras, bryter vi ut det till en egen klass istället för att använda String som nyckel.

## TidsintervallNyckel

package com.golcher.tidsintervall.komponenter.data;  
  
public class TidsintervallNyckel  
{  
 private final String \_nyckel; // Immutable key (id + kalla) i HashMap  
 public TidsintervallNyckel(TidsintervallVo vo)  
 {  
 \_nyckel = vo.getId() + "-" + vo.getKalla().toString();  
 }  
  
 @Override  
 public String toString()  
 {  
 return \_nyckel;  
 }  
}

De data som ska visas för en komponent ska alltid listas sorterade på samma sätt, vilket i sin tur gör att vi inte alls behöver uppdatera listorna för respektive komponent när ett element blir valt eller bortvalt, utan allt vi behöver göra är att be varje komponent att på nytt hämta data från dialogens modell. Det löser vi med hjälp av konceptet med lyssnare.

package com.golcher.tidsintervall.dialoger;  
  
public interface ITidsintervallModellLyssnare  
{  
 void modellenHarUppdaterats();  
}

Dialogmodellen behöver två metoder (valj och valjBort) som tar en sammansatt nyckel och använder den för att flagga om det elementet är valt eller inte. Välj-knapparna i de två panelerna med valbara element kan nu enkelt använda samma metod för att flagga sitt markerade element som valt och valda-komponenten använder valjBort antingen för ett markerat element (Ta bort markerad) eller för samtliga (Rensa). Vidare behöver modellen en möjlighet att låta lyssnare registrera sig som intresserade av att bli notifierade när förändringar inträffar. Dessutom behöver den tillhandahålla en metod för att låta komponenterna hämta de data de ska visa, både när de initieras och när modellen uppdateras. Slutligen behöver den också stöd för att uppdatera egna intervall när vi får nya data från en ”redigerare”.

package com.golcher.tidsintervall.komponenter.data;  
  
public enum KomponentFilter  
{  
 *Standard*,  
 *Egen*,  
 *Valda*}

package com.golcher.tidsintervall.dialoger;  
  
import com.golcher.tidsintervall.komponenter.data.KomponentFilter;  
import com.golcher.tidsintervall.komponenter.data.TidsintervallNyckel;  
import com.golcher.tidsintervall.komponenter.data.TidsintervallVo;  
import com.golcher.tidsintervall.komponenter.data.TidsintervallWrapper;  
  
import java.util.ArrayList;  
  
public interface ITidsintervallModellFasad  
{  
 ArrayList<TidsintervallWrapper> hamtaModellFor(KomponentFilter filter);  
 void registreraLyssnare(ITidsintervallModellLyssnare lyssnare);  
 void valj(TidsintervallNyckel nyckel);  
 void valjBort(TidsintervallNyckel nyckel);  
 void uppdateraEgnaIntervallFranRedigerare(ArrayList<TidsintervallVo> egnaIntervall);  
}

För att kunna utveckla dialogkoden fristående från den applikation i vilket den ska användas (vilket i sig är lämpligt för att den ska bli så återanvändbar som möjligt) så behöver vår dialogmodell förses med data genom instanser av ett interface, låt oss kalla dem dataproviders. Det låter oss skriva mockupkod som ger oss dummydata här och nu och låter de olika projekt i vilken vår grafiska komponent ska användas skriva sina egna applikationsspecifika implementationer. Därmed kan vi även skriva tester för vår kod, utan att vi ens vet vilka våra framtida datakällor kommer att vara.

package com.golcher.tidsintervall.dbproviders;  
  
import com.golcher.tidsintervall.komponenter.data.TidsintervallVo;  
import java.util.ArrayList;  
  
public interface ITidsIntervallDbProvider  
{  
 ArrayList<TidsintervallVo> hamtaData();  
}

package com.golcher.tidsintervall.dbproviders;  
  
import com.golcher.tidsintervall.komponenter.data.KomponentFilter;  
import com.golcher.tidsintervall.komponenter.data.TidsintervallKalla;  
import com.golcher.tidsintervall.komponenter.data.TidsintervallVo;  
  
import java.util.ArrayList;  
  
public class MockupProvider implements ITidsIntervallDbProvider  
{  
 private final ArrayList<TidsintervallVo> \_fejkadData;  
  
 public MockupProvider(KomponentFilter fejkaDatakallaFor)  
 {  
 \_fejkadData = new ArrayList<>();  
 switch (fejkaDatakallaFor)  
 {  
 case *Standard*: fejkaStandard(); break;  
 case *Egen*: fejkaEgen(); break;  
 case *Valda*: fejkaValda(); break;  
 }  
 }  
  
 private void fejkaValda()  
 {  
 \_fejkadData.add(new TidsintervallVo("1", TidsintervallKalla.*Standard*, "08:00", "12:00", "Förmiddag"));  
 \_fejkadData.add(new TidsintervallVo("3", TidsintervallKalla.*Egen*, "13:00", "15:00", ""));  
 }  
  
 private void fejkaEgen()  
 {  
 \_fejkadData.add(new TidsintervallVo("1", TidsintervallKalla.*Egen*, "08:00", "10:00", ""));  
 \_fejkadData.add(new TidsintervallVo("2", TidsintervallKalla.*Egen*, "10:00", "12:00", ""));  
 \_fejkadData.add(new TidsintervallVo("3", TidsintervallKalla.*Egen*, "13:00", "15:00", ""));  
 }  
  
 private void fejkaStandard()  
 {  
 \_fejkadData.add(new TidsintervallVo("1", TidsintervallKalla.*Standard*, "08:00", "12:00", "Förmiddag"));  
 \_fejkadData.add(new TidsintervallVo("2", TidsintervallKalla.*Standard*, "13:00", "17:00", "Eftermiddag"));  
 \_fejkadData.add(new TidsintervallVo("3", TidsintervallKalla.*Standard*, "18:00", "22:00", "Kväll"));  
 \_fejkadData.add(new TidsintervallVo("4", TidsintervallKalla.*Standard*, "08:00", "22:00", "Heldag"));  
 }  
  
 @Override  
 public ArrayList<TidsintervallVo> hamtaData()  
 {  
 return \_fejkadData;  
 }  
}

På samma sätt vet vi väldigt lite om hur andra system väljer att lösa redigeringen av egendefinierade tider (vi vet varken hur de sparas eller vilken användare som ska få läsa vad från vilket underliggande system). Det vi vet är att vi vill att de ska implementera ett interface som gör det möjligt för oss att ropa på en ”redigerare” och från den få en ny lista av egendefinierade element.

package com.golcher.tidsintervall.redigerare;  
  
import com.golcher.tidsintervall.komponenter.data.TidsintervallVo;  
  
import java.util.ArrayList;  
  
public interface IEgnaIntervallRedigerare  
{  
 ArrayList<TidsintervallVo> visaRedigeringsDialog();  
}

Min erfarenhet säger mig att det är betydligt bättre att utgå ifrån en dialoggemensam grundmodell och sedan låta varje komponent få tillgång till en skräddarsydd komponentmodell, som lyssnar på och hämtar data ifrån dialogmodellen, än att skriva flera olika listor och skyffla data mellan dem.

Mitt bevis är att vi med ovanstående design kan komplettera vår grundmodell med fler datakällor vid behov utan att vi behöver skriva om en enda regel. Hade vi hanterat varje enskild komponents egen modell som facit, så hade vi behövt skriva ny kod om någon om ett tag kommer på att vi ska lägga till ytterligare en typ av tidsintervall i dialogen. Nu räcker det med att vi kompletterar våra två enums; TidsintervallKalla och KomponentFilter.

# Nu ska vi designa våra tre komponenter – de har en hel del gemensamt vilket vi lägger i en basklass

package com.golcher.tidsintervall.komponenter;  
  
import com.golcher.tidsintervall.dialoger.ITidsintervallModellFasad;  
import com.golcher.tidsintervall.dialoger.ITidsintervallModellLyssnare;  
import com.golcher.tidsintervall.komponenter.data.KomponentFilter;  
import com.golcher.tidsintervall.komponenter.data.TidsintervallNyckel;  
import com.golcher.tidsintervall.komponenter.data.TidsintervallVo;  
import com.golcher.tidsintervall.komponenter.data.TidsintervallWrapper;  
import com.golcher.tidsintervall.komponenter.util.TidsintervallBreddRadgivare;  
  
import javax.swing.\*;  
import java.util.ArrayList;  
  
public abstract class AbstraktTidsintervallDialogKomponent extends JPanel implements ITidsintervallModellLyssnare  
{  
 private final ITidsintervallModellFasad \_modellFasad;  
 private final KomponentFilter \_komponentFilter;  
 private TidsintervallBreddRadgivare \_breddRadgivare;  
  
 public AbstraktTidsintervallDialogKomponent  
 (  
 ITidsintervallModellFasad modellFasad,  
 KomponentFilter filter  
 )  
 {  
 \_modellFasad = modellFasad;  
 \_komponentFilter = filter;  
 \_breddRadgivare = new TidsintervallBreddRadgivare(\_modellFasad);  
  
 // Registrera oss som lyssnare på dialogmodellen  
 \_modellFasad.registreraLyssnare(this);  
 }  
  
 protected ArrayList<TidsintervallWrapper> hamtaKomponentDataFranDialogModell()  
 {  
 return \_modellFasad.hamtaModellFor(\_komponentFilter);  
 }  
  
 protected void valj(TidsintervallNyckel nyckel)  
 {  
 \_modellFasad.valj(nyckel);  
 }  
  
 protected void valjBort(TidsintervallNyckel nyckel)  
 {  
 \_modellFasad.valjBort(nyckel);  
 }  
  
 protected void uppdateraEgnaIntervallFranRedigerare(ArrayList<TidsintervallVo> egnaIntervall)  
 {  
 \_modellFasad.uppdateraEgnaIntervallFranRedigerare(egnaIntervall);  
 \_breddRadgivare = new TidsintervallBreddRadgivare(\_modellFasad);  
 }  
  
 protected int hamtaLampligBredd()  
 {  
 return \_breddRadgivare.hamtaBredd();  
 }  
  
}

De behöver alla anpassa sin bredd efter det bredaste elementet i dialogmodellen, därav den utbrutna klassen TidsintervallBreddRadgivare.

package com.golcher.tidsintervall.komponenter.util;  
  
import com.golcher.tidsintervall.dialoger.ITidsintervallModellFasad;  
import com.golcher.tidsintervall.komponenter.data.KomponentFilter;  
import com.golcher.tidsintervall.komponenter.data.TidsintervallWrapper;  
  
import javax.swing.\*;  
import java.util.ArrayList;  
  
public class TidsintervallBreddRadgivare  
{  
 private int \_bredd = 0;  
  
 public TidsintervallBreddRadgivare(ITidsintervallModellFasad modell)  
 {  
 int storstaBredd = 0;  
 ArrayList<TidsintervallWrapper> allaElement = new ArrayList<>();  
 allaElement.addAll(modell.hamtaModellFor(KomponentFilter.*Standard*));  
 allaElement.addAll(modell.hamtaModellFor(KomponentFilter.*Egen*));  
 allaElement.addAll(modell.hamtaModellFor(KomponentFilter.*Valda*));  
 for(TidsintervallWrapper wrapper : allaElement)  
 {  
 JLabel tmp = new JLabel(wrapper.toString());  
 int bredd = tmp.getPreferredSize().width + 20;  
 if(bredd > storstaBredd) storstaBredd = bredd;  
 }  
 \_bredd = storstaBredd;  
 }  
  
 public int hamtaBredd()  
 {  
 return \_bredd;  
 }  
}

# Vi kan nu börja designa valda-panelen ovanpå basklassen

## Först behöver vi skapa den ListModel för den JList som vi ska använda

package com.golcher.tidsintervall.komponenter.valda;  
  
import com.golcher.tidsintervall.komponenter.data.TidsintervallWrapper;  
  
import javax.swing.\*;  
import javax.swing.event.ListDataEvent;  
import javax.swing.event.ListDataListener;  
import java.util.ArrayList;  
  
public class ValdaIntervallListModell implements ListModel<TidsintervallWrapper>  
{  
 private ArrayList<TidsintervallWrapper> \_lista;  
 private ArrayList<ListDataListener> \_lyssnare;  
  
 public ValdaIntervallListModell()  
 {  
 \_lista = new ArrayList<>();  
 \_lyssnare = new ArrayList<>();  
 }  
  
 public void uppdateraModellen(ArrayList<TidsintervallWrapper> nyLista)  
 {  
 \_lista = nyLista;  
 for(ListDataListener l : \_lyssnare)  
 {  
 l.contentsChanged(new ListDataEvent(this, ListDataEvent.*CONTENTS\_CHANGED*, 0, \_lista.size()-1));  
 }  
 }  
  
 @Override  
 public int getSize()  
 {  
 return \_lista.size();  
 }  
  
 @Override  
 public TidsintervallWrapper getElementAt(int index)  
 {  
 if(index < 0 || index > \_lista.size()-1) throw new ArrayIndexOutOfBoundsException(index);  
 return \_lista.get(index);  
 }  
  
 @Override  
 public void addListDataListener(ListDataListener l)  
 {  
 \_lyssnare.add(l);  
 }  
  
 @Override  
 public void removeListDataListener(ListDataListener l)  
 {  
 \_lyssnare.remove(l);  
 }  
}

Notera att modellen, när den skapas i konstruktorn är tom. Det är först vid anropet uppdateraModellen som den tilldelas data. Detta är med avsikt, eftersom vi ska låta panelen med vår JList lyssna direkt på vår dialogmodell och vid uppdateringar förmedla informationen till komponentmodellen.

## Nu kan vi bygga själva panelen.

package com.golcher.tidsintervall.komponenter.valda;  
  
import com.golcher.tidsintervall.dialoger.ITidsintervallModellFasad;  
import com.golcher.tidsintervall.komponenter.AbstraktTidsintervallDialogKomponent;  
import com.golcher.tidsintervall.komponenter.data.KomponentFilter;  
import com.golcher.tidsintervall.komponenter.data.TidsintervallWrapper;  
  
import javax.swing.\*;  
import javax.swing.border.TitledBorder;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.util.ArrayList;  
  
public class ValdaIntervallPanel extends AbstraktTidsintervallDialogKomponent  
{  
 private static final String *ramRubrik* = "Valda intervall";  
 private static final String *rensaRubrik* = "Rensa";  
 private static final String *taBortRubrik* = "Ta bort markerade";  
  
 private ValdaIntervallListModell \_komponentModell;  
 private JList<TidsintervallWrapper> \_valda;  
  
 public ValdaIntervallPanel  
 (  
 ITidsintervallModellFasad modellFasad  
 )  
 {  
 super(modellFasad, KomponentFilter.*Valda*);  
 this.setBorder  
 (  
 BorderFactory.*createTitledBorder* (  
 BorderFactory.*createEtchedBorder*(),  
 *ramRubrik*,  
 TitledBorder.*LEFT*,  
 TitledBorder.*TOP* )  
 );  
  
 this.setLayout(new BorderLayout());  
  
 \_komponentModell = new ValdaIntervallListModell();  
 \_valda = new JList(\_komponentModell);  
 \_valda.setSelectionMode(ListSelectionModel.*SINGLE\_SELECTION*);  
 \_valda.setPreferredSize(new Dimension(hamtaLampligBredd(), 100));  
 ScrollPane scrollPane = new ScrollPane();  
 scrollPane.add(\_valda);  
 scrollPane.setPreferredSize(new Dimension(hamtaLampligBredd(), 100));  
 this.add(scrollPane, BorderLayout.*CENTER*);  
  
 byggKnappPanel();  
  
 uppdateraLokalModell();  
 }  
  
 private void byggKnappPanel()  
 {  
 JButton rensaKnapp = new JButton(*rensaRubrik*);  
 rensaKnapp.addActionListener(new AbstractAction()  
 {  
 @Override  
 public void actionPerformed(ActionEvent e)  
 {  
 ValdaIntervallPanel.this.rensa();  
 }  
 });  
  
 JButton taBortKnapp = new JButton(*taBortRubrik*);  
 taBortKnapp.addActionListener(new AbstractAction()  
 {  
 @Override  
 public void actionPerformed(ActionEvent e)  
 {  
 ValdaIntervallPanel.this.taBort();  
 }  
 });  
  
 JPanel knappPanel = new JPanel(new FlowLayout(FlowLayout.*RIGHT*));  
 knappPanel.add(rensaKnapp);  
 knappPanel.add(taBortKnapp);  
  
 this.add(knappPanel, BorderLayout.*SOUTH*);  
 }  
  
 private void uppdateraLokalModell()  
 {  
 ArrayList<TidsintervallWrapper> sorteradLista = hamtaKomponentDataFranDialogModell();  
 \_komponentModell.uppdateraModellen(sorteradLista);  
 \_valda.setPreferredSize(new Dimension(hamtaLampligBredd(), 100));  
 \_valda.invalidate();  
 }  
  
 public void rensa()  
 {  
 // Vi kommer uppdatera modellen direkt när vi kör metoden valjBort, så vi måste först lägga över  
 // de element som finns i listan i en egen array  
 ArrayList<TidsintervallWrapper> elementAttTaBort = new ArrayList<>();  
 for(int index=0; index < \_komponentModell.getSize(); index++)  
 {  
 TidsintervallWrapper wrapper = \_komponentModell.getElementAt(index);  
 elementAttTaBort.add(wrapper);  
 }  
  
 // Nu kan vi anropa modellen och förändra, utan att vi snubblar  
 // på index som ändras medan vi itererar över innehållet  
 for(TidsintervallWrapper wrapper : elementAttTaBort)  
 {  
 valjBort(wrapper.nyckel());  
 }  
 }  
  
 public void taBort()  
 {  
 int index = \_valda.getSelectedIndex();  
 if(index>=0)  
 {  
 TidsintervallWrapper wrapper = \_komponentModell.getElementAt(index);  
 valjBort(wrapper.nyckel());  
 }  
 }  
  
 @Override  
 public void modellenHarUppdaterats()  
 {  
 uppdateraLokalModell();  
 }  
}

# Dags för våra valbara-komponenter – de har också en del gemensamt (både en basklass och en modell)

## Här är den delade modellen för våra valbara – vilka båda behöver en ComboBoxModel

Modellen ska i sin tur lyssna på dialogmodellen och men vet inte mer än att den får data via uppdateraModellen, vilket precis som för valda-komponenten kommer ifrån panelen som agerar lyssnare på dialogmodellen.

package com.golcher.tidsintervall.komponenter.valbara;  
  
import com.golcher.tidsintervall.komponenter.data.TidsintervallWrapper;  
  
import javax.swing.\*;  
import javax.swing.event.ListDataEvent;  
import javax.swing.event.ListDataListener;  
import java.util.ArrayList;  
  
public class TidsintervallValbarModell implements ComboBoxModel  
{  
 private ArrayList<TidsintervallWrapper> \_lista;  
 private ArrayList<ListDataListener> \_lyssnare;  
 private TidsintervallWrapper \_selectedItem;  
  
 public TidsintervallValbarModell()  
 {  
 \_lista = new ArrayList<>();  
 \_lyssnare = new ArrayList<>();  
 \_selectedItem = null;  
 }  
  
 public void uppdateraModellen(ArrayList<TidsintervallWrapper> nyLista)  
 {  
 \_lista = nyLista;  
  
 TidsintervallWrapper currentItem = null;  
 if(\_lista.size() > 0)  
 {  
 currentItem = \_lista.get(0);  
 }  
 \_selectedItem = currentItem;  
  
 for(ListDataListener l : \_lyssnare)  
 {  
 l.contentsChanged(new ListDataEvent(this, ListDataEvent.*CONTENTS\_CHANGED*, 0, \_lista.size()-1));  
 }  
 }  
  
 @Override  
 public void setSelectedItem(Object anItem)  
 {  
 if(null!=anItem && anItem instanceof TidsintervallWrapper)  
 {  
 \_selectedItem = (TidsintervallWrapper) anItem;  
 }  
 }  
  
 @Override  
 public Object getSelectedItem()  
 {  
 return \_selectedItem;  
 }  
  
 @Override  
 public int getSize()  
 {  
 return \_lista.size();  
 }  
  
 @Override  
 public Object getElementAt(int index)  
 {  
 if(index < 0 || index > \_lista.size()-1) throw new ArrayIndexOutOfBoundsException(index);  
 return \_lista.get(index);  
 }  
  
 @Override  
 public void addListDataListener(ListDataListener l)  
 {  
 \_lyssnare.add(l);  
 }  
  
 @Override  
 public void removeListDataListener(ListDataListener l)  
 {  
 \_lyssnare.remove(l);  
 }  
}

## Här är den extra basklassen som de båda valbara-komponenterna delar på

package com.golcher.tidsintervall.komponenter.valbara;  
  
import com.golcher.tidsintervall.dialoger.ITidsintervallModellFasad;  
import com.golcher.tidsintervall.komponenter.AbstraktTidsintervallDialogKomponent;  
import com.golcher.tidsintervall.komponenter.data.KomponentFilter;  
import com.golcher.tidsintervall.komponenter.data.TidsintervallNyckel;  
import com.golcher.tidsintervall.komponenter.data.TidsintervallWrapper;  
  
import javax.swing.\*;  
import javax.swing.border.TitledBorder;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.util.ArrayList;  
  
public abstract class AbstraktTidsintervallValbarKomponent extends AbstraktTidsintervallDialogKomponent  
{  
 private final JComboBox \_valbara;  
 private TidsintervallValbarModell \_komponentModell;  
  
 public AbstraktTidsintervallValbarKomponent  
 (  
 String ramRubrik,  
 String knappRubrik,  
 ITidsintervallModellFasad modellFasad,  
 KomponentFilter filter  
 )  
 {  
 super(modellFasad, filter);  
 this.setBorder  
 (  
 BorderFactory.*createTitledBorder* (  
 BorderFactory.*createEtchedBorder*(),  
 ramRubrik,  
 TitledBorder.*LEFT*,  
 TitledBorder.*TOP* )  
 );  
 this.setLayout(new FlowLayout(FlowLayout.*LEFT*));  
  
 \_komponentModell = new TidsintervallValbarModell();  
 \_valbara = new JComboBox<>(\_komponentModell);  
 \_valbara.setPreferredSize(new Dimension(hamtaLampligBredd(), 20));  
 this.add(\_valbara);  
  
 JButton valjKnapp = new JButton(knappRubrik);  
 this.add(valjKnapp);  
  
 valjKnapp.addActionListener(new AbstractAction()  
 {  
 @Override  
 public void actionPerformed(ActionEvent e)  
 {  
 if(AbstraktTidsintervallValbarKomponent.this.\_valbara.getItemCount() > 0)  
 {  
 TidsintervallWrapper wrapper = (TidsintervallWrapper) AbstraktTidsintervallValbarKomponent.this.\_valbara.getSelectedItem();  
 if(null!=wrapper)  
 {  
 TidsintervallNyckel nyckel = wrapper.nyckel();  
 valj(nyckel);  
 }  
 }  
 }  
 });  
  
 uppdateraLokalModell();  
 }  
  
 private void uppdateraLokalModell()  
 {  
 ArrayList<TidsintervallWrapper> sorteradLista = hamtaKomponentDataFranDialogModell();  
 \_komponentModell.uppdateraModellen(sorteradLista);  
 \_valbara.setPreferredSize(new Dimension(hamtaLampligBredd(),20));  
 \_valbara.invalidate();  
 }  
  
 @Override  
 public void modellenHarUppdaterats()  
 {  
 uppdateraLokalModell();  
 }  
  
  
}

Slutligen kan vi skapa våra två paneler – den ena i princip direkt från den delade basklassen och den andra med lite mer kod för redigering av intervall.

### StandardIntervallPanel

package com.golcher.tidsintervall.komponenter.valbara;  
  
import com.golcher.tidsintervall.dialoger.ITidsintervallModellFasad;  
import com.golcher.tidsintervall.komponenter.data.KomponentFilter;  
  
public class StandardIntervallPanel extends AbstraktTidsintervallValbarKomponent  
{  
 public StandardIntervallPanel  
 (  
 ITidsintervallModellFasad modellFasad  
 )  
 {  
 super("Standardintervall", "Välj", modellFasad, KomponentFilter.*Standard*);  
 }  
}

### EgnaIntervallPanel

package com.golcher.tidsintervall.komponenter.valbara;  
  
import com.golcher.tidsintervall.dialoger.ITidsintervallModellFasad;  
import com.golcher.tidsintervall.komponenter.data.KomponentFilter;  
import com.golcher.tidsintervall.komponenter.data.TidsintervallVo;  
import com.golcher.tidsintervall.redigerare.IEgnaIntervallRedigerare;  
  
import javax.swing.\*;  
import java.awt.event.ActionEvent;  
import java.util.ArrayList;  
  
public class EgnaIntervallPanel extends AbstraktTidsintervallValbarKomponent  
{  
 private final IEgnaIntervallRedigerare \_redigerare;  
 public EgnaIntervallPanel  
 (  
 ITidsintervallModellFasad modellFasad,  
 IEgnaIntervallRedigerare redigerare  
 )  
 {  
 super("Egna intervall", "Välj", modellFasad, KomponentFilter.*Egen*);  
 \_redigerare = redigerare;  
 if(null!=\_redigerare) this.add(redigeringsKnapp());  
 }  
  
 private JButton redigeringsKnapp()  
 {  
 JButton knapp = new JButton("Redigera");  
 knapp.addActionListener(new AbstractAction()  
 {  
 @Override  
 public void actionPerformed(ActionEvent e)  
 {  
 ArrayList<TidsintervallVo> egnaIntervall = \_redigerare.visaRedigeringsDialog();  
 uppdateraEgnaIntervallFranRedigerare(egnaIntervall);  
 }  
 });  
 return knapp;  
 }  
  
}

# Nu äntligen är det dags att skriva koden för den modala dialogen

## IckeModalDialogDemo

package com.golcher.tidsintervall.dialoger.ickemodal;  
  
import com.golcher.tidsintervall.dbproviders.ITidsIntervallDbProvider;  
import com.golcher.tidsintervall.dbproviders.MockupProvider;  
import com.golcher.tidsintervall.dialoger.TidsintervallDialogModell;  
import com.golcher.tidsintervall.dialoger.TidsintervallDialogModellFabrik;  
import com.golcher.tidsintervall.dialoger.TidsintervallDialogPanel;  
import com.golcher.tidsintervall.komponenter.data.KomponentFilter;  
import com.golcher.tidsintervall.komponenter.data.TidsintervallVo;  
import com.golcher.tidsintervall.komponenter.data.TidsintervallWrapper;  
import com.golcher.tidsintervall.redigerare.IEgnaIntervallRedigerare;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.util.ArrayList;  
  
public class IckeModalDialogDemo extends JDialog  
{  
 private TidsintervallDialogPanel \_panel;  
 private TidsintervallDialogModell \_modell;  
  
 public IckeModalDialogDemo(TidsintervallDialogModell modell, IEgnaIntervallRedigerare redigerare)  
 {  
 \_modell = modell;  
 \_panel = new TidsintervallDialogPanel(\_modell, redigerare);  
  
 this.setModal(false);  
 this.setDefaultCloseOperation(WindowConstants.*DISPOSE\_ON\_CLOSE*);  
 this.setLayout(new BorderLayout());  
 this.add(\_panel, BorderLayout.*CENTER*);  
 this.setVisible(true);  
 }  
}

## AnvandarensVal

package com.golcher.tidsintervall.dialoger.modal;  
  
public enum AnvandarensVal  
{  
 *IngetValGjort*,  
 *Avbryt*,  
 *Spara*}

## DialogUtdata

package com.golcher.tidsintervall.dialoger.modal;  
  
import com.golcher.tidsintervall.komponenter.data.TidsintervallVo;  
  
import java.util.ArrayList;  
  
public class DialogUtdata  
{  
 private final AnvandarensVal \_anvandarensVal;  
 private final ArrayList<TidsintervallVo> \_utdata;  
  
 public DialogUtdata(AnvandarensVal anvandarensVal)  
 {  
 \_anvandarensVal = anvandarensVal;  
 \_utdata = new ArrayList<>();  
 }  
  
 public void laggTillPost(TidsintervallVo post)  
 {  
 \_utdata.add(post);  
 }  
  
 public ArrayList<TidsintervallVo> hamtaUtdata()  
 {  
 return \_utdata;  
 }  
  
 public AnvandarensVal hamtaAnvandarensVal()  
 {  
 return \_anvandarensVal;  
 }  
  
}

## ModalDialogDemo

package com.golcher.tidsintervall.dialoger.modal;  
  
import com.golcher.tidsintervall.dbproviders.ITidsIntervallDbProvider;  
import com.golcher.tidsintervall.dbproviders.MockupProvider;  
import com.golcher.tidsintervall.dialoger.TidsintervallDialogModell;  
import com.golcher.tidsintervall.dialoger.TidsintervallDialogModellFabrik;  
import com.golcher.tidsintervall.dialoger.TidsintervallDialogPanel;  
import com.golcher.tidsintervall.komponenter.data.KomponentFilter;  
import com.golcher.tidsintervall.komponenter.data.TidsintervallWrapper;  
import com.golcher.tidsintervall.redigerare.IEgnaIntervallRedigerare;  
  
import javax.swing.\*;  
import javax.swing.border.CompoundBorder;  
import javax.swing.border.EmptyBorder;  
import javax.swing.border.LineBorder;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ComponentAdapter;  
import java.awt.event.ComponentEvent;  
  
public class ModalDialogDemo extends JDialog  
{  
 private TidsintervallDialogPanel \_panel;  
 private TidsintervallDialogModell \_modell;  
 private AnvandarensVal \_anvandarensVal;  
  
  
 public ModalDialogDemo(  
 ITidsIntervallDbProvider standardProvider,  
 ITidsIntervallDbProvider egenProvider,  
 ITidsIntervallDbProvider valdaProvider,  
 IEgnaIntervallRedigerare redigerare  
 )  
 {  
 \_anvandarensVal = AnvandarensVal.*IngetValGjort*;  
 \_modell = TidsintervallDialogModellFabrik.*skapaModell*(standardProvider, egenProvider, valdaProvider);  
 \_panel = new TidsintervallDialogPanel(\_modell, redigerare);  
  
 this.setModal(true);  
 this.setDefaultCloseOperation(WindowConstants.*DISPOSE\_ON\_CLOSE*);  
 this.setLayout(new BorderLayout());  
  
 // this.add(fixaStatusRad(), BorderLayout.NORTH);  
 this.add(\_panel, BorderLayout.*CENTER*);  
 this.add(fixaKnappar(), BorderLayout.*SOUTH*);  
  
 this.setSize(530, 250);  
  
 this.setVisible(false);  
 }  
  
 private JPanel fixaStatusRad()  
 {  
 JPanel statusBar = new JPanel(new FlowLayout(FlowLayout.*LEFT*));  
 statusBar.setBorder(new CompoundBorder(new LineBorder(Color.*DARK\_GRAY*),  
 new EmptyBorder(4, 4, 4, 4)));  
 final JLabel status = new JLabel();  
 statusBar.add(status);  
  
 addComponentListener(new ComponentAdapter()  
 {  
 @Override  
 public void componentResized(ComponentEvent e)  
 {  
 status.setText(ModalDialogDemo.this.getWidth() + "x" + ModalDialogDemo.this.getHeight());  
 }  
 });  
  
 return statusBar;  
 }  
  
 private JPanel fixaKnappar()  
 {  
 JButton avbryt = new JButton("Avbryt");  
 avbryt.addActionListener(new AbstractAction()  
 {  
 @Override  
 public void actionPerformed(ActionEvent e)  
 {  
 avbryt();  
 }  
 });  
  
 JButton spara = new JButton("Spara");  
 spara.addActionListener(new AbstractAction()  
 {  
 @Override  
 public void actionPerformed(ActionEvent e)  
 {  
 spara();  
 }  
 });  
  
 JPanel knappar = new JPanel(new FlowLayout(FlowLayout.*RIGHT*));  
 knappar.setBackground(Color.*DARK\_GRAY*);  
 knappar.setOpaque(true);  
 knappar.add(avbryt);  
 knappar.add(spara);  
 return knappar;  
 }  
  
 private void avbryt()  
 {  
 \_anvandarensVal = AnvandarensVal.*Avbryt*;  
 this.setVisible(false);  
 this.dispose();  
 }  
  
 private void spara()  
 {  
 \_anvandarensVal = AnvandarensVal.*Spara*;  
 this.setVisible(false);  
 this.dispose();  
 }  
  
 public DialogUtdata visaDialog()  
 {  
 this.setVisible(true); // Pausar tills användaren gör något som stänger dialogen (avbryt, spara, stänger fönstret)  
  
 DialogUtdata utdata = new DialogUtdata(\_anvandarensVal);  
 if (\_anvandarensVal.equals(AnvandarensVal.*Spara*))  
 {  
 for (TidsintervallWrapper wrapper : \_modell.hamtaModellFor(KomponentFilter.*Valda*))  
 {  
 utdata.laggTillPost(wrapper.varde());  
 }  
 }  
 return utdata;  
 }  
}

# Avslutar med en icke-modal demo

## TestbaddIckeModalDialogDemo

package com.golcher.tidsintervall.demo;  
import com.golcher.tidsintervall.dbproviders.ITidsIntervallDbProvider;  
import com.golcher.tidsintervall.dbproviders.MockupProvider;  
import com.golcher.tidsintervall.dialoger.ITidsintervallModellLyssnare;  
import com.golcher.tidsintervall.dialoger.TidsintervallDialogModell;  
import com.golcher.tidsintervall.dialoger.TidsintervallDialogModellFabrik;  
import com.golcher.tidsintervall.dialoger.ickemodal.IckeModalDialogDemo;  
import com.golcher.tidsintervall.komponenter.data.KomponentFilter;  
import com.golcher.tidsintervall.komponenter.data.TidsintervallWrapper;  
import com.golcher.tidsintervall.redigerare.IEgnaIntervallRedigerare;  
  
public class TestbaddIckeModalDialogDemo implements ITidsintervallModellLyssnare  
{  
 public static void main(String[] args) {  
 new TestbaddIckeModalDialogDemo();  
 }  
 private final TidsintervallDialogModell \_modell;  
 public TestbaddIckeModalDialogDemo() {  
 IEgnaIntervallRedigerare redigerare = null;  
 ITidsIntervallDbProvider standardProvider = new MockupProvider(KomponentFilter.*Standard*);  
 ITidsIntervallDbProvider egenProvider = new MockupProvider(KomponentFilter.*Egen*);  
 ITidsIntervallDbProvider valdaProvider = new MockupProvider(KomponentFilter.*Valda*);  
 \_modell = TidsintervallDialogModellFabrik.*skapaModell*(standardProvider, egenProvider, valdaProvider);  
 \_modell.registreraLyssnare(this);  
 IckeModalDialogDemo ickeModalDialog = new IckeModalDialogDemo(\_modell, redigerare);  
 }  
  
 @Override  
 public void modellenHarUppdaterats() {  
 System.*out*.println("Modellen har uppdaterats:");  
 for(TidsintervallWrapper wrapper : \_modell.hamtaModellFor(KomponentFilter.*Valda*)) {  
 System.*out*.println(wrapper.toString());  
 }  
 }  
}

## IckeModalDialogDemo

package com.golcher.tidsintervall.dialoger.ickemodal;  
  
import com.golcher.tidsintervall.dbproviders.ITidsIntervallDbProvider;  
import com.golcher.tidsintervall.dbproviders.MockupProvider;  
import com.golcher.tidsintervall.dialoger.TidsintervallDialogModell;  
import com.golcher.tidsintervall.dialoger.TidsintervallDialogModellFabrik;  
import com.golcher.tidsintervall.dialoger.TidsintervallDialogPanel;  
import com.golcher.tidsintervall.komponenter.data.KomponentFilter;  
import com.golcher.tidsintervall.komponenter.data.TidsintervallVo;  
import com.golcher.tidsintervall.komponenter.data.TidsintervallWrapper;  
import com.golcher.tidsintervall.redigerare.IEgnaIntervallRedigerare;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.util.ArrayList;  
  
public class IckeModalDialogDemo extends JDialog  
{  
 private TidsintervallDialogPanel \_panel;  
 private TidsintervallDialogModell \_modell;  
  
 public IckeModalDialogDemo(TidsintervallDialogModell modell, IEgnaIntervallRedigerare redigerare)  
 {  
 \_modell = modell;  
 \_panel = new TidsintervallDialogPanel(\_modell, redigerare);  
  
 this.setModal(false);  
 this.setDefaultCloseOperation(WindowConstants.*DISPOSE\_ON\_CLOSE*);  
 this.setLayout(new BorderLayout());  
 this.add(\_panel, BorderLayout.*CENTER*);  
 this.setVisible(true);  
 }  
}