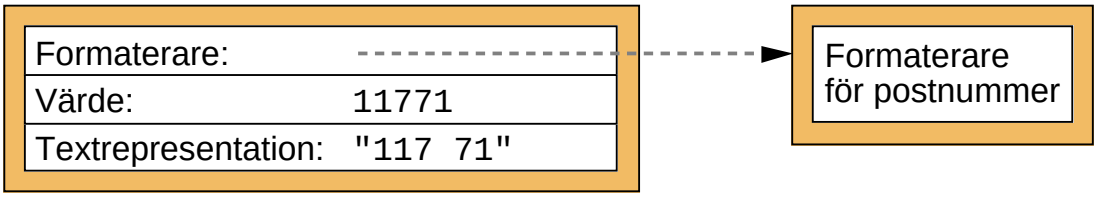
3 Textfält (provavsnitt)

Komponenter i Swing

Klassen JFormattedTextField – enradiga formaterade fält



- Varje enradigt formaterat fält har ett *värde* och en *textrepresentation*
- Programmet interagerar med värdet
- Användaren interagerar med textrepresentationen
- Omvandlingen mellan de båda utförs av en *formaterare*

Enradiga formaterade fält är objekt av klassen `JFormattedTextField`, som är en underklass till klassen `JTextField`. Ett `JFormattedTextField`-objekt har två separata egenskaper: ett *värde* (som programmet sätter eller hämtar i fältet) och en *textrepresentation* (som är vad som syns i fältet). Användaren arbetar genomgående med det som syns, alltså fältets innehåll, medan programmet genomgående använder sig av värdet. Omvandlingen mellan värde och textrepresentation utförs av en *formaterare*, ett stödobjekt av någon underklass till den obundna nästlade klassen `AbstractFormatter`.

Klassen `JFormattedTextField` hör till namnrymden `javax.swing` och tillkom i version 1.4 av standardbiblioteket.

Komponenter i Swing

Klassen JFormattedTextField – centrala konstruktorer

Konstruktor	*Beskrivning*
`JFormattedTextField()`	initiera till att välja formaterare från värdets typ
`JFormattedTextField(fmt)`	initiera till att använda formatet `fmt`
`JFormattedTextField(fmtr)`	initiera till att använda formateraren `fmtr`

Exempel:

```
JFormattedTextField postCode = new JFormattedTextField();
postCode.setValue(new Integer(11771));

Format numericFormat = NumberFormat.getInstance();
postCode = new JFormattedTextField(numericFormat);

MaskFormatter pcFormatter = new MaskFormatter("### ##");
postCode = new JFormattedTextField(pcFormatter);
```

Det finns tre vanliga konstruktorer för initiering av objekt av klassen `JFormattedTextField`:

- Konstruktorn utan argument initierar ett `JFormattedTextField`-objekt till att välja formaterare baserat på typen på värdet i fältet:

  ```
  JFormattedTextField postCode = new JFormattedTextField();
  ```

- Konstruktorn med argument av klassen `Format` från namnrymden `java.text` initierar ett `JFormattedTextField`-objekt till att använda en formaterare för det format som är argument:

  ```
  postCode = new JFormattedTextField(NumberFormat.getInstance());
  ```

- Konstruktorn med argument av underklasser till den nästlade klassen `AbstractFormatter` initierar ett `JFormattedTextField`-objekt till att använda en explicit angiven formaterare. Detta är t ex vanligt när s k *maskformaterare* ska användas, som i detalj styr hur textfältet får fyllas i:

  ```
  postCode = new JFormattedTextField(new MaskFormatter("### ##"));
  ```

  Klassen `MaskFormatter` finns i namnrymden `javax.swing.text` och hanterar inmatning där varje tecken kontrolleras mot ett i förväg bestämt mönster, där t ex ett brädgårdstecken betyder en siffra. Se klassens dokumentation för ytterligare detaljer!

Komponenter i Swing

Klassen JFormattedTextField – centrala tjänster

Tjänst	*Beskrivning*
`getValue()`	returnera fältets värde
`setValue(obj)`	ändra värdet till `obj`
`setColumns(size)`	sätt bredden på fältet till ca `size` tecken
`setFocusLostBehavior(mode)`	åtgärd när användaren lämnar fältet; se nedan
`addActionListener(obj)`	registrera lyssnare till `ActionEvent`-händelser
`addPropertyChangeListener( "value", obj)`	registrera lyssnare till ändringar i fältets värde

Klasskonstant	*Innebörd*
`REVERT`	återställ alltid text till gällande värde
`COMMIT`	uppdatera värde från text om det är giltigt
`COMMIT_OR_REVERT`	uppdatera värde om giltigt; återställ annars text
`PERSIST`	uppdatera inte värde; behåll text (dvs gör inget)

Det finns en stor uppsättning tjänster för formaterade textfält. De vanligaste är:

- Med tjänsten `getValue()` kan vi hämta fältets värde. Med tjänsten `setValue()` kan fältets värde ändras. Värdet måste repesenteras av ett objekt – inte ett värde av en primitiv datatyp – t ex ett objekt av en wrapperklass eller klassen `java.util.Date`.
- Med tjänsten `setColumns()` kan vi bestämma bredden på fältet, uttryckt i genomsnittsbreda tecken. Denna tjänst fanns redan i klassen `JTextField`, men användes sällan där eftersom bredden kunde sättas med argument i konstruktorn.
- Med tjänsten `setFocusLostBehavior()` kan vi bestämma hur programmet ska bete sig när fältet mister tangentbordsfokus, dvs när användaren lämnar fältet. Det finns fyra tillåtna värden, samtliga klasskonstanter i klassen `JFormattedTextField`:

  `JFormattedTextField.REVERT` innebär att textrepresentationen alltid återställs till det gällande värdet.

  `JFormattedTextField.COMMIT` innebär att värdet uppdateras om texten i fältet representerar ett korrekt värde; om inte, ändras inte värdet och textrepresentationen återställs inte.

  `JFormattedTextField.COMMIT_OR_REVERT` innebär att värdet uppdateras om texten i fältet representerar ett korrekt värde; om inte, återställs textrepresentationen till det senaste korrekta värdet. Detta är det förvalda läget.

  `JFormattedTextField.PERSIST` innebär att textrepresentationen behålls, men värdet uppdateras inte – dvs inget händer när användaren lämnar fältet.

- Med tjänsten `addActionListener()` registreras argumentet som lyssnare till `ActionEvent`-händelser från det formaterade textfältet. Därmed kan vi knyta händelser till att användaren trycker på **Enter**-tangenten under inmatning i fältet. Med tjänsten `removeActionListener()` avregistreras lyssnare.
- Med tjänsten `addPropertyChangeListener()` och första argumentet satt till textsträngen `"value"` registreras andra argumentet som lyssnare till händelsen att fältet ändrar värde (eller, mer exakt, att komponentegenskapen med namnet `value` ändrar tillstånd – vi bygger här på att Swing-komponenter är JavaBeans-komponenter). Detta kan användas för att uppdatera andra fält när det inmatade värdet i ett fält förändras. Med tjänsten `removePropertyChangeListener()` avregistreras lyssnare.

Ett numeriskt inmatningsfält – definition

```
import java.awt.*;
import java.awt.event.*;
import java.text.NumberFormat;
import javax.swing.*;

public class MovieForm extends JFrame implements ActionListener {

    private JTextField          myNameField      = new JTextField(40);
    private JTextField          myDirectorField  = new JTextField(40);
    private JTextField          myCountryField   = new JTextField(40);
    private JFormattedTextField myYearField;

    public MovieForm() {
        ...
        formPanel.add(entryHeader);

        NumberFormat yearFormat = NumberFormat.getInstance();
        yearFormat.setGroupingUsed(false);
        myYearField = new JFormattedTextField(yearFormat);
        myYearField.setColumns(4);

        addRow(formPanel, "Film:",               myNameField,     true);
        addRow(formPanel, "Regiss\u00f6r:",      myDirectorField, true);
        addRow(formPanel, "Ursprungsland:",      myCountryField,  true);
        addRow(formPanel, "Produktions\u00e5r:", myYearField,     false);
        ...
    }
    ...
}
```

Vi kommer nu att ändra inmatningsfältet som `myYearField` refererar till i vårt formulär så att det endast tar emot numerisk inmatning genom att utnyttja ett objekt av klassen `JFormattedTextField`. Objekt av denna klass kan användas för att kontrollera att inmatad text endast accepteras om den går att tolka som ett giltigt värde av den typ som presenteras i fältet.

Vi hade kunnat välja att initiera objektet av klassen `JFormattedTextField` med konstruktorn utan argument, och lita till att formatet styrs av att vi sätter ett heltalsobjekt, t ex av klassen `Integer`, som fältets värde. Det enda som inte blir bra med denna lösning är att standardformateringen för heltal i många språkområden separerar presentationen av värdet i grupper om tre siffror (i fallet svenska med blanktecken som separator). I de flesta språk utom spanska är detta ett felaktigt sätt att skriva årtal. Vi väljer därför i stället den konstruktor som kräver ett objekt av klassen `Format` från namnrymden `java.text`.

Vi inleder med att skapa formatobjektet. Formatobjekt för numeriska värden skapas enklast med klass-tjänsten `getInstance()` i klassen `NumberFormat` från namnrymden `java.text`. När vi väl har ett standardinitierat formatobjekt kan vi lätt slå av siffergrupperingen:

```
NumberFormat yearFormat = NumberFormat.getInstance();
yearFormat.setGroupingUsed(false);
```

Vi kan nu skapa inmatningsfältet av klassen `JFormattedTextField`. Vi måste också bestämma den önskade bredden på inmatningsfältet, eftersom detta inte kan göras i konstruktorn (som vi brukar göra för vanliga `JTextField`-objekt):

```
myYearField = new JFormattedTextField(yearFormat);
myYearField.setColumns(4);
```

Ett numeriskt inmatningsfält – hantering

```
public class MovieForm extends JFrame implements ActionListener {

    ...

    private void clearForm() {
        myNameField.setText("");
        myDirectorField.setText("");
        myCountryField.setText("");
        myYearField.setValue(null);
    }

    private void fetchForm() {
        myNameField.setText(theMovie.getName());
        myDirectorField.setText(theMovie.getDirector());
        myCountryField.setText(theMovie.getCountry());
        myYearField.setValue(new Long(theMovie.getProductionYear()));
    }

    private void storeForm() {
        theMovie.setName(myNameField.getText());
        theMovie.setDirector(myDirectorField.getText());
        theMovie.setCountry(myCountryField.getText());
        Long year = (Long) myYearField.getValue();
        if (year == null)
            theMovie.setProductionYear(0);
        else
            theMovie.setProductionYear(year.intValue());
    }
}
```

De återstående förändringarna kommer i de metoder som hanterar innehållet i det numeriska inmatningsfältet. För objekt av klassen `JTextField` är det normala att vi hämtar resp sätter värdet med hjälp av tjänsterna `getText()` resp `setText()`. För objekt av klassen `JFormattedTextField` bör vi undvika att själva arbeta med textrepresentationen i fältet – det är formateringsobjektets uppgift. I stället blir det värdet i fältet vi hämtar med tjänsten `getValue()` resp sätter med tjänsten `setValue()`.

I tjänsten `fetchForm()` hämtar vi produktionsåret från `Movie`-objektet på vanligt sätt, men därefter kapslar vi in det i ett wrapperobjekt av klassen `Long` och sätter det som nytt värde i inmatningsfältet med tjänsten `setValue()`. Observera att vi måste hantera värdet som ett objekt – inte som ett värde av en primitiv datatyp. Vi väljer objekt av wrapperklassen `Long` eftersom detta är den klass av objekt som `NumberFormat`-formateraren normalt returnerar för heltalsvärden.

I tjänsten `clearForm()` ska vi radera samtliga inmatningsfält. Vi vill alltså i alla fält visa att värdet är tomt eller saknas. För ett numeriskt inmatningsfält är det närmaste vi kan komma att ange att ett värde saknas att sätta det till `null`, vilket också är det initiala tillståndet innan användaren matar in något i fältet. Detta kommer att presenteras som en tom textsträng.

I tjänsten `storeForm()` ska vi hämta det heltal som matats in i fältet med tjänsten `getValue()` och lagra detta i `Movie`-objektet. Standardinställningen för vad som händer när användaren lämnar fältet, klasskonstanten `JFormattedTextField.COMMIT_OR_REVERT`, innebär att inmatning i fältet bara accepteras om det är ett giltigt värde – annars återgår fältet till sitt tidigare innehåll. Det betyder att om något har matats in i fältet och accepterats, så vet vi att det är ett korrekt numeriskt värde inkapslat i ett wrapperobjekt, som vi kan få tag i med tjänsten `getValue()`. Vi måste emellertid tänka på som en möjlighet att formuläret bekräftas av användaren utan att inmatning någonsin sker i det numeriska fältet. I det läget kommer `getValue()` att returnera `null`, något vi måste vara beredda på att hantera.