

Läsanvisningar

Börja med att läsa kapitlen i kursboken, därefter resurserna på Internet och till sist själva lektionsmaterialet.

Kursboken (5-8:e upplagan): Kapitel 6 och 8

Internet: The Java Tutorial, Trail: Creating a GUI With JFC/Swing: - Getting Started with Swing
(<http://docs.oracle.com/javase/tutorial/uiswing/start/index.html>)

Internet: The Java Tutorial, Trail: Creating a GUI With JFC/Swing: - Using Swing Components
(<http://docs.oracle.com/javase/tutorial/uiswing/components/index.html>)

JApplet ingår ej. Du behöver inte lusläsa alla How to... utan skumma igenom dessa. Försök dock få en uppfattning om vad de olika komponenterna gör och om du någon gång behöver använda någon av dessa kan du återkomma hit och läsa om dem.

Internet: The Java Tutorial, Trail: Creating a GUI With JFC/Swing: - Laying Out Components Within a Container
(<http://docs.oracle.com/javase/tutorial/uiswing/layout/index.html>)

I avsnittet How to Use ... är det i första hand FlowLayout, GridLayout och BorderLayout du bör titta på. Avsnittet Creating a Custom Layout Manager ingår ej i lektionen.

Grafiska användargränssnitt, del 1

Swing och AWT (Abstract Window Toolkit)

I den första versionen av Java (JDK 1.0) användes paketet java.awt för att skapa grafiska applikationer. Detta paket innehåller klasser med vilka man kan skapa fönster, knappar, textfält, menyer med mera. Det finns även klasser för att kunna rita s.k. frihandsgrafik (linjer, cirklar m.m.) i en applikation.

Paketet AWT är plattformsoberoende så att ett program skrivet med detta paket kan exekvera på samtliga Javaplattformar och i de användargränssnitt som dessa plattformar har. Det vill säga ett fönster ser ut som ett Windows-fönster om applikationen exekveras på en Windows-maskin och ser ut som ett Solaris-fönster om den körs på Solaris-maskin. Detta kan ses som en fördel då användaren av applikationen känner igen sig med de komponenter som de normalt brukar använda.

Paketet AWT har dock ett par nackdelar. Alla komponenter (knappar med mera) i AWT är så kallade tungviktskomponenter vilket innebär att de ritas i ett eget fyrkantigt och ogenomskinligt område som hanteras av operativsystemet själv. Alla komponenter i AWT förlitar sig på att det finns en motsvarande native komponent, en så kallad peer, skriven för den aktuella plattformen koden körs på. Det innebär att enbart komponenter som finns för alla plattformar kan användas. Komponenter som enbart finns för en eller några plattformar kan inte tas med eftersom koden då inte längre är plattformsoberoende. Som exempelvis kan trädlistkontrollen som finns i Windows (utforskaren) nämnas. Detta var ett problem med AWT eftersom program såg lite primitiva ut och saknat avancerade komponenter.

På grund av de nackdelar AWT drogs med var det vanligt att många utvecklare skrev sina egna

GUI-komponenter. Därför påbörjades 1997 ett projekt som kallades Swing där syftet var att ta fram ett paket med komponenter helt skrivna i Java. Resultatet av detta projekt blev paketet javax.swing och inkluderades som standard från och med version 1.2 av Java.

Swing bygger vidare på AWT och ersätter nästan alla AWTs komponenter och dessutom utökades antalet komponenter i swing med betydligt fler och mer avancerade komponenter. Både AWT och Swing kan användas och innehåller även en del komponenter med samma namn. Till exempel finns det knappar i båda AWT och Swing. För att skilja dessa komponenter åt börjar alla komponenter i Swing på bokstaven J. En knapp i Swing heter JButton, medan en knapp i AWT heter Button.

Det är helt möjligt att i en applikation samtidigt använda en JButton tillsammans med en Button, men det är något som inte rekommenderas och bör undvikas. Finns det inte verkligen speciella skäl ska Swing användas framför AWT när nya applikationer skapas.

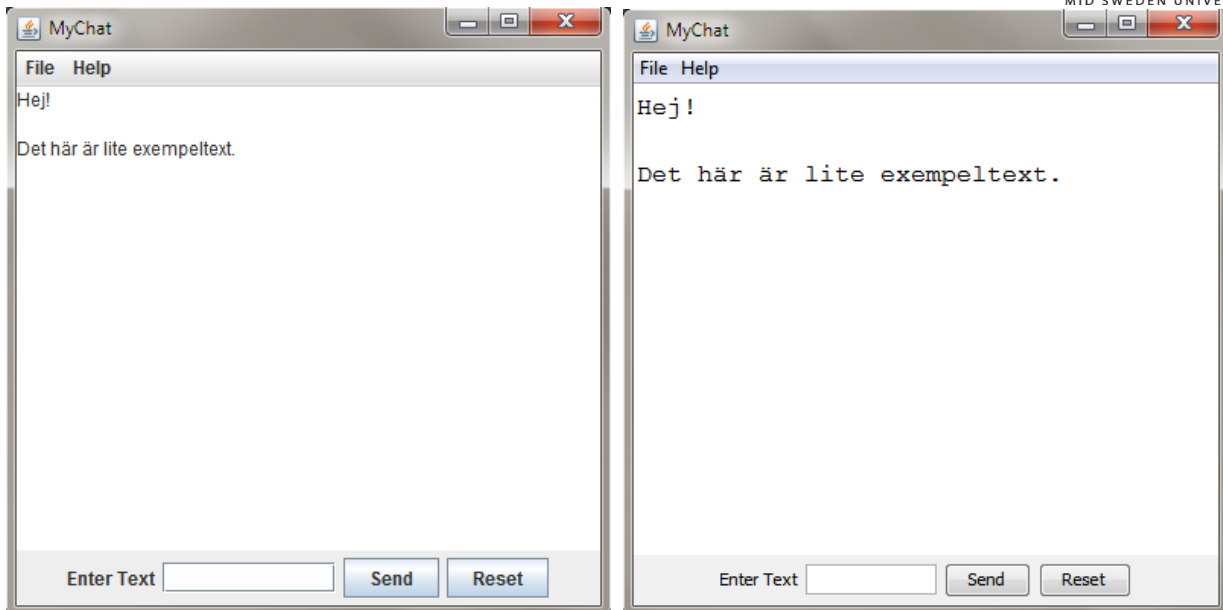
Den stora skillnaden är att i stort sett alla komponenter i Swing är lättviktskomponenter. De förlitar sig inte på någon motsvarande native komponent utan alla komponenter är helt skrivna i Java. De behöver inte längre vara fyrkantiga och inte heller ogenomskinliga.

Koden som styr komponentens utseende i Swing ligger i separata klasser (så kallade UI-delegate) och det gör det relativt enkelt att ändra utseendet på komponenterna. Man säger att man ändrar komponenternas look-and-feel (till och med under körning av applikationen). Som default används en look-and-feel som ser likadan ut på alla plattformar, nämligen Java look-and-feel (även kallad Motif). Här ser till exempel en knapp likadan ut oavsett om applikationen körs i Windows eller i Linux.

Det går att byta look-and-feel och även skapa en helt ny egen look-and-feel om man så vill. För att byta look-and-feel, till exempelvis systemets look-and-feel, kan man i applikationens main-metod skriva följande:

```
public class GUIExample {
    public static void main(String args[]) {
        try {
            // Set system L&F
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        }
        catch (Exception e) {
            // En handfull undantag ska fångas i stället för Exception
        }
        //Creating the JFrame
    }
}
```

Utseendet på komponenterna ändras nu så att de efterliknar, så långt det går, komponenterna på den plattform applikationen körs på. Vänstra bilden nedan visar en applikation vars look-and-feel är standard och den högra bilden visar samma applikation men med systemets look-and-feel (körs på Windows).



Komponenter - JComponent

Komponenter är ett väldigt centralt begrepp när vi skapar applikationer med ett grafiskt användargränssnitt. Komponenter är det vi använder för att bygga upp användargränssnittet och exempel på olika komponenter är: fönster, knappar och textfält med flera. I stort sett allting som syns på skärmen i ett grafiskt användargränssnitt är komponenter av olika slag. När vi använder Swing för att skapa ett användargränssnitt är det klassen `JComponent` som utgör grunden för alla komponenter. Denna klass innehåller metoder som är gemensamma för alla komponenter och några vanliga och användbara metoder är:

```
setBackground(Color c)
setForeground(Color c)
```

Dessa metoder tar båda ett `Color`-objekt och ändrar för- och bakgrundsfärgen på komponenten. Förgrundsfärgen är vanligtvis färgen på komponentens text. Bakgrundsfärgen ändras endast om komponenten är ogenomskinlig (opaque). För att sätta bakgrundsfärgen till svart kan vi skriva: `setBackground(new Color(0,0,0));`

```
setFont(Font f)
```

Med metoden `setFont` kan vi ändra utseendet på texten i komponenten. Vi kan till exempel ändra fonten till Courier genom att skriva `setFont(new Font("Courier", Font.PLAIN, 16));` `Font.PLAIN` anger att det inte ska vara fet eller kursiv stil och 16 anger storleken.

```
setToolTipText(String s)
```

Med metoden `setToolTipText()` kan vi ange en text som ska visas om vi för muspekaren ovanför komponenten och håller den stilla där ett tag. Det kan vara användbart för att beskriva för användaren vad komponenten gör.

```
setEnabled(boolean b)
```

Med metoden `setEnabled` kan vi bestämma om komponenten ska vara tillgänglig eller inte. På till exempel en knapp kan metoden användas för att avgöra om knappen ska kunna tryckas på eller inte.

```
setVisible(boolean b)
```

Med denna metod kan vi ange om en komponent ska visas eller inte.

```
setPreferredSize(Dimension d)  
setMaximumSize(Dimension d)  
setMinimumSize(Dimension d)
```

Med dessa metoder kan vi ge ett förslag om hur stor vi vill att komponenten ska vara. Notera att det endast är ett förslag och hur stor komponenten sen faktiskt blir beror på bland annat vilken layouthanterare som placerar ut komponenten (mer om detta senare). `PreferredSize` är den önskvärda storleken vi vill ha och `Minimum-` respektive `MaximumSize` anger den absolut minsta och största storleken. Storleken anges i antal pixlar. För att ange en komponents minsta storlek till 100 pixlar bred och 50 pixlar hög skriver vi `setMinimumSize(new Dimension(100, 50))`;

```
paintComponent(Graphics g)  
Graphics getGraphics()
```

Om vi har behov att rita "frihands"-grafik i en komponent kan metoderna ovan utnyttjas. Med hjälp av `Graphics`-objektet kan vi rita linjer, cirklar och andra figurer direkt i komponenten. Vi kan t.ex. använda en knapp och sen rita ett rött kryss ovanpå knappen. Du som kommer från Java I har provat på att rita med `Graphics` i samband med att du provade på Applets.

Det finns betydligt fler metoder vi kan anropa på en komponent och eftersom denna klass är superklass till alla andra komponenter rekommenderas du att titta igenom dess metoder i till exempel API.

Notera att vi aldrig direkt använder klassen `JComponent` och skapar objekt av den klassen, utan det är subklasser till `JComponent` vi använder när vi skapar vårt grafiska användargränssnitt.

Behållare - Container

Innan en komponent kan visas på skärmen måste den läggas till i en så kallad behållare (container). En container kan vara ett fönster eller en annan yta i vilken vi kan placera våra komponenter för att sen visa dem. En applikation består oftast av flera komponenter och för att på ett smidigt sätt kunna hantera alla komponenter används olika containrar för att gruppera ihop olika komponenter. En container är i grund och botten en behållare för komponenter och utgörs av klassen `Container`.

De fyra mest grundläggande typer av Containers är `JFrame`, `JPanel`, `JDialog` och `JApplet`.

- **JFrame**
Detta är ett så kallat applikationsfönster och är den container vi skapar för att överhuvudtaget kunna visa våra komponenter på skärmen. En `JFrame` är ett fönster som har en ram runt fönstret, minimera-, maximera- och stäng-knapp, samt eventuellt en meny.
- **JDialog**
Detta är en speciell typ av fönster som används för att interagera med användaren. I Java I har vi redan provat på att använda dialogrutor för in- och utmatning av data från användaren och programmen.
- **JApplet**
Detta är en container som får sin yta tilldelad av webbläsaren. Det är denna klass vi använder när vi vill placera komponenter på en webbsida. Mer om detta i lektion 7.

- **JPanel**
Detta kan ses som en tom yta vi kan använda för att ”rita” på. Med rita menas att vi kan placera komponenter i en JPanel som vi i sin tur kan placera i till exempel ett applikationsfönster. När man skapar avancerade användargränssnitt brukar det vara nödvändigt att först placera komponenter i en panel innan de läggs till i fönstret. Med rita menas även frihandsgrafik. Det vill säga om vi behöver rita linjer, cirklar och andra figurer brukar det vara vanligt att använda en JPanel.
- **JComponent**
Även JComponent är en container vilket innebär att vi till exempel kan placera en knapp i en knapp (varför man nu skulle vilja göra det?).

För att lägga till en komponent i en container anropar vi metoden `add`. Denna metod tar som argument vilken komponent som helst (även en annan container kan vi lägga in i en container).

```
add(Component comp)
```

Komponenterna som läggs till en container placeras i en lista i samma ordning som de läggs till. Denna lista är också utgångspunkt för en så kallad tabuleringsordning, det vill säga när användaren använder Tab-tangenten för att förflytta sig mellan olika komponenter i ett fönster. Det finns metoder vi kan anropa för att ändra komponenternas index i listan samt vilken tabuleringsordning som ska användas.

Några andra användbara metoder i klassen `Container` är:

```
int getWidth()  
int getHeight()
```

Dessa metoder returnerar ett heltal som anger hur bred och hög containern är (i antal pixlar). Det kan vara användbart bland annat om vi ska rita frihandsgrafik så vi vet att vi håller oss inom den synliga delen.

```
Component getComponent(int n)  
Component getComponentAt(int x, int y)
```

Metoderna ovan kan användas för att få en referens till en komponent i containern. Antingen genom att specificera komponentens index i listan eller genom att ge x- och y-koordinater.

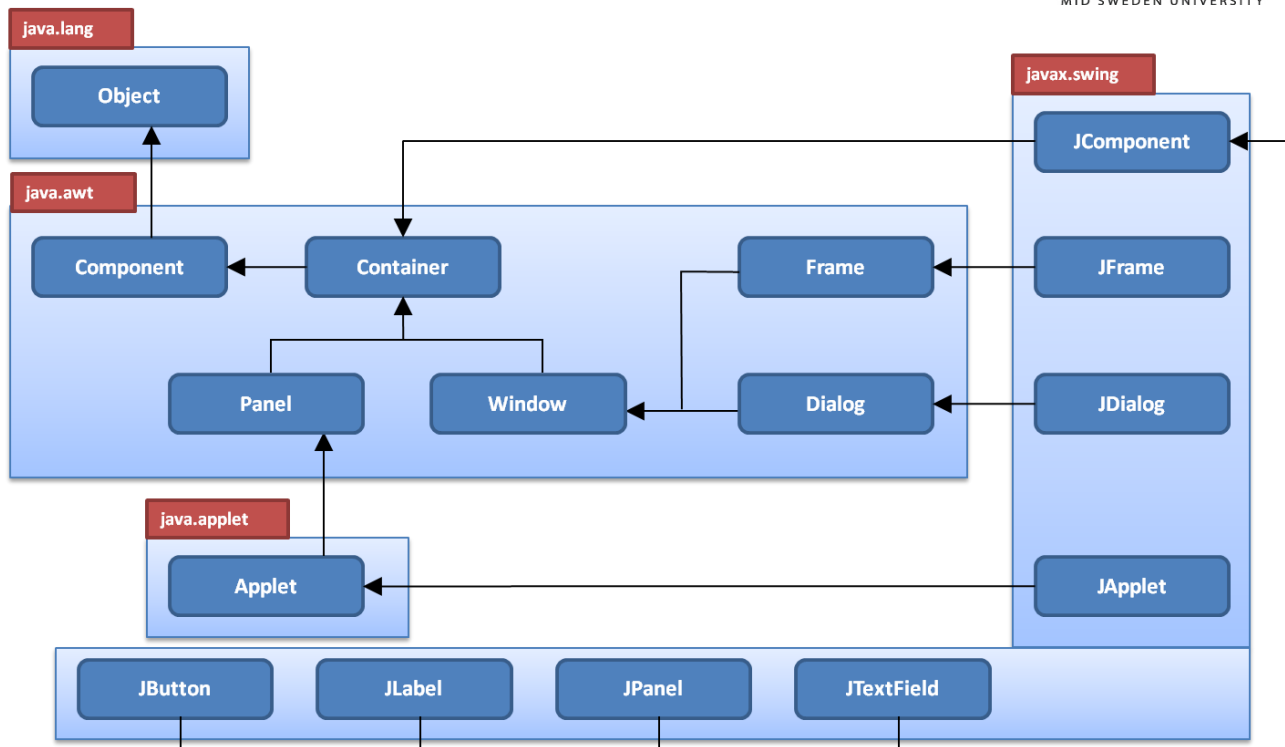
```
remove(int index)  
remove(Component comp)  
removeAll()
```

Metoderna ovan används för att ta bort en komponent, eller alla, från containern. Behöver normalt inte göras i ett program, men det kan vara bra att känna till att möjligheten finns.

```
setLayout(LayoutManager mgr)
```

Med metoden ovan anger vi vilken layouthanterare som ska användas. En layouthanterare styr hur komponenter i behållaren ska placeras ut i förhållande till varandra. Mer om detta lite senare.

Många metoder är gemensamma för `JComponent` och `Container` och anledningen till detta är att `JComponent` ärver sina egenskaper från `Container`. Som nämnts tidigare så är en `JComponent` en `Container`. Detta framgår tydligt i bilden nedan.

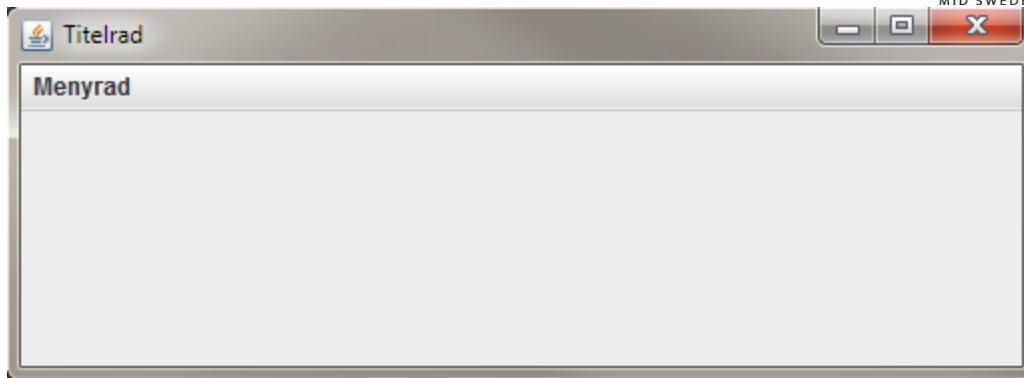


Bilden ovan visar alltså hur klasserna i paketen AWT och Swing hör ihop med varandra och i vilka paket det ligger. Som vanligt ärver alla objekt i Java på ett eller annat sätt från klassen Object. Vi går inte in på några detaljer, men notera bland annat att klassen Component är superklassen till alla typer av komponenter som vi använder i en grafisk applikation. Notera även att Container i sin tur är en Component. Det innebär att en container i sin tur kan vara en komponent i en annan container.

Applikationsfönster - JFrame

En JFrame är ett fönster som i andra system brukar kallas ett applikationsfönster eller ramfönster. Det är ett komplett fönster som vi kan använda för att placera de komponenter vi vill ska synas i användargränssnittet. Ett fönster innehåller väldigt kortfattat följande:

1. En titelrad där man oftast skriver ut namnet på applikationen. Innehåller även en programikon. Denna kan användaren ta tag med musen och flytta runt fönstret på skärmen.
2. Knappar för att minimera- och maximera storleken på fönstret.
3. En stäng-knapp som används för att t.ex. avsluta applikationen. Observera att det som default inte händer något om vi trycker på knappen. Vi måste själv definiera vad som ska hända.
4. En gräns/ram runt fönstret som kan användas för att ändra storleken på fönstret.
5. En yta att placera komponenter på.
6. Samt eventuell en menyrad.



Det första vi måste göra när vi skapar en applikation som ska använda ett grafiskt användargränssnitt är att skapa en JFrame. För att göra detta kan vi använda någon av dessa konstruktorer:

```
JFrame() // Constructs a new frame that is initially invisible  
JFrame(String title) // Creates a new, invisible frame with the specified  
title
```

Några användbara metoder vi kan anropa på en JFrame är:

```
setSize(int width, int height)
```

Ett anrop till metoden ovan anger vilken storlek fönstret ska ha. För att ange att fönstret ska vara 500 pixlar brett och 300 pixlar högt skriver vi `myJFrame.setSize(500, 300);`

```
setBounds(int x, int y, int width, int height)
```

Förutom att sätta storleken på fönstret kan man bestämma var på skärmen fönstret ska placeras. För att ange att fönstret ska vara 500 pixlar brett och 300 pixlar högt och placerat på position 50, 100 (50 pixlar från högra kanten och 100 pixlar från övre kanten på skärmen) skriver vi `myJFrame.setBounds(500, 300, 50, 100);`

```
setTitle(String title)
```

Använd metoden ovan för att ange fönstrets titel. Kan vara användbart om du använt konstruktorn utan argument eller om du under applikationens körning behöver byta titel.

```
setResizable(boolean resizable)
```

Anropa metoden ovan med `false` om du inte vill att det ska gå att ändra storleken på fönstret (det vill säga om det inte ska gå att dra i ramen runt fönstret). Default är `true`, det vill säga att det går att ändra storleken på fönstret.

```
setVisible(boolean visible)
```

Metoden ovan är väldigt viktig att anropa när vi lagt till alla komponenter i fönstret. Metoden anger nämligen om fönstret ska vara synligt eller inte och som default är alla fönster ej synliga (`false`).

```
setIconImage(Image image)  
setIconImages(List<? extends Image> icons)
```

Med dessa metoder kan vi ange vilken bild/ikon som ska synas till vänster i titelraden på fönstret.

På många plattformar används olika storlekar på ikonerna i olika sammanhang. I till exempel Windows är ikonerna 16x16 pixlar i titelraden, men när användaren trycker alt + tabb för att bläddra mellan öppna program, visas en ikon med storleken 32x32 pixlar. Med metoden `setIconImages` kan du ange flera ikoner med olika storlekar och den storlek som är mest lämpad för tillfället kommer att användas.

```
setDefaultCloseOperation(int operation)
```

Här kan vi på ett snabbt och enkelt sätt ange vad som ska ske när användaren trycker på fönstrets stäng-knapp i titelraden. Det finns fyra fördefinierade värden som kan användas:

- `WindowConstants.DO_NOTHING_ON_CLOSE` (heltalsvärde 0)
Gör inget, stäng inte ned. Överlåt till andra metoder att stänga ned istället.
- `WindowConstants.HIDE_ON_CLOSE` (heltalsvärde 1)
Göm fönstret, men släng inte bort det. Det går att visa fönstret igen om man så önskar.
- `WindowConstants.DISPOSE_ON_CLOSE` (heltalsvärde 2)
Göm fönstret och frigör resurser använda av fönstret.
- `WindowConstants.EXIT_ON_CLOSE` (heltalsvärde 3)
Det mest använda alternativet för att stänga och avsluta en applikation. Det som händer är att `System.exit(0)` anropas. Om man vill spara en öppen fil eller något sådant, kanske detta inte är rätt alternativ att välja.

```
pack()
```

I stället för att själv bestämma storleken på fönstret med `setSize` eller `setBounds` kan du anropa denna metod. Metoden beräknar själv lämplig storlek på fönstret utifrån den önskade storleken (preferred size) på alla komponenter som har placerats i fönstret.

```
setLocationRelativeTo(Component c)
```

Det här är en användbar metod för att centrera ett fönster på skärmen. Det egentliga syftet med metoden är att placera ett fönster relativt en annan komponent. Väldigt ofta används dock metoden bara för att centrera fönstret på skärmen. Det görs genom att anropa metoden med `null` som argument.

För att demonstrera de flesta metoder ovan tittar vi nu på ett litet exempel. Det är ett fönster utan några komponenter i, men som har en titel, ikoner och några andra egenskaper satta.

```
import javax.swing.*;
import java.awt.*;
import java.util.*;

public class HelloWorldJFrame extends JFrame {
    public HelloWorldJFrame(String title) {
        // Sätt titel på fönstret
        super(title);

        // Vad ska ske när vi stänger fönstret?
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

        // Sätt fönstrets storlek
        setSize(300, 150);

        // Storleken ska inte gå att ändra
```



```
setResizable(false);

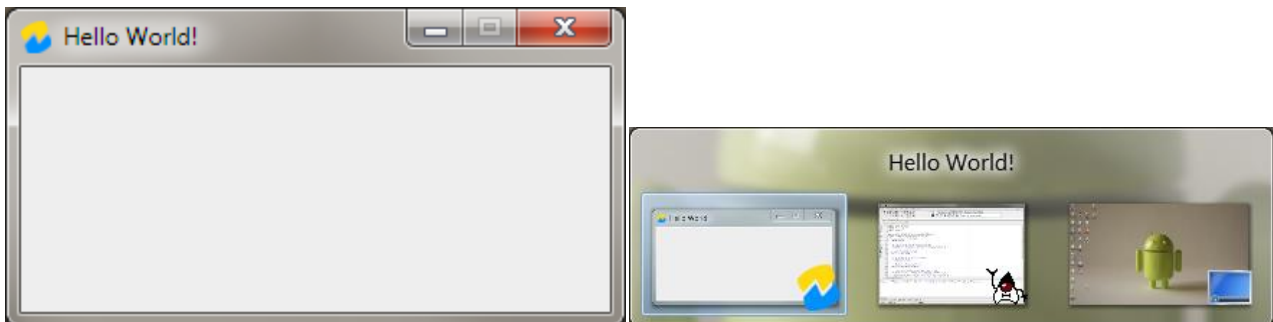
// Centrera fönstret på skärmen
setLocationRelativeTo(null);

// Ange vilka ikoner som fönstret ska använda
ArrayList<Image> iconImages = new ArrayList<Image>();
iconImages.add(new ImageIcon("miun16x16.png").getImage());
iconImages.add(new ImageIcon("miun32x32.png").getImage());
iconImages.add(new ImageIcon("miun64x64.png").getImage());
setIconImages(iconImages);

// Gör fönstret synligt
setVisible(true);
}

public static void main(String args[]) {
    // Skapa en ny instans av vårt fönster
    new HelloWorldJFrame("Hello World!");
}
}
```

När vi skapar en JFrame är det vanligt att skriva en klass som ärver just JFrame. I dess konstruktor sätter vi egenskaperna fönstret ska ha och sist i konstruktorn gör vi fönstret synligt. Notera koden för att sätta vilka ikoner som ska användas i fönstret. Jag går inte in på någon djupare förklaring på vad som sker utan använd koden i dina klasser om du behöver använda ikoner. Observera att bildfilerna i mitt exempel ligger i samma katalog som källkodsfilen.



Som du ser i bilden ovan används ikonerna dels i titelraden och dels i Windows task switcher (alt + tabb) med olika storlekar på ikonerna.

JLabel

En JLabel består endast av en fast text som inte kan manipuleras på något sätt av användaren. Det går till exempel inte att markera texten med hjälp av musen eller att skriva in ny text med tangentbordet. En JLabel används mest för att ge information eller ledtexter till användaren för att denna lättare ska förstå vad som ska göras, eller som förklaringar till vad andra komponenter gör.

För att skapa en JLabel kan man använda ett par olika konstruktörer och den absolut vanligaste är den som tar en String som argument. Denna sträng innehåller den text som ska synas. Det är möjligt att via metoder sätta och hämta texten på en JLabel. Detta görs genom att anropa metoderna `setText(String)` för att sätta ny text och `getText()` för att hämta den aktuella texten i en JLabel.

I Swing är det möjligt att ge många komponenter en bild istället för, eller tillsammans med, en text. Detta görs genom att skapa något som kallas för en `ImageIcon`. För att skapa en ny `ImageIcon` anger man i konstruktorn vilken bild det är som ska användas (som en string och bilden måste då ligga i

samma katalog som källkodsfilen). Sen kan man använda denna ImageIcon i konstruktorn när man skapar sin JLabel. Har man både bild och text samtidigt kan det vara bra att ange var bilden ska placeras i förhållande till eventuell text. Detta görs genom att anropa metoden setHorizontalTextPosition() och skicka med SwingConstants.LEFT (eller .CENTER, .RIGHT).

Det är även lätt att ändra textens färg på en JLabel. Detta görs genom att anropa metoden setForeground(Color) som tar ett Color-objekt som argument. En annan bra sak med JLabel (och många andra komponenter) är att vi kan formatera texten med enklare html-taggar. För fler metoder som kan anropas på en JLabel bör du titta i API:n).

Ofta används JLabel som en ledtext till en textruta (JTextField). I ett forumlära där användaren till exempel ska skriva in sin kontaktinformation kan ett antal JLabel användas som etiketter för de olika textfälten. Det kan vara en JLabel i vilken det står Förnamn: och en JLabel där det står Efternamn: etc. För att i kod knyta till vilken komponent en JLabel är etikett för används metoden setLabelFor(Component c). Om vi även i samband med detta anropar metoden setDisplayedMnemonic(char aChar) kommer textfältet JLabel är etikett för att få focus om användaren trycker alt + aChar.

Några exempel på användning av JLabel:

<pre>JLabel jLabel = new JLabel("En textsträng."); String text = jLabel.getText();</pre>	
<pre>JLabel jLabel = new JLabel(new ImageIcon("figur.png")); jLabel.setHorizontalTextPosition(JLabel.RIGHT); jLabel.setText("Text och bild."); jLabel.setForeground(Color.red);</pre>	
<pre>JLabel jLabel = new JLabel("<html><h1>Rubrik</h1></html>"); jLabel.setHorizontalTextPosition(JLabel.LEFT); jLabel.setIcon(new ImageIcon("figur.png")); jLabel.setIconTextGap(50); // default is 4</pre>	
<pre>JLabel jLabel = new JLabel("Förnamn:"); JTextField txt = new JTextField(10); jLabel.setLabelFor(txt); jLabel.setDisplayedMnemonic('F');</pre>	

JButton

En annan komponent som är väldigt vanlig i ett användargränssnitt är en knapp. I Swing heter denna komponent JButton och det är en komponent med vilken en användare kan använda musen för att klicka på för att utföra något. När knappen trycks ner genereras ett så kallat ActionEvent som måste tas omhand om vi vill att något ska hända vid knapptryckningen. Mer om detta i kommande lektion.

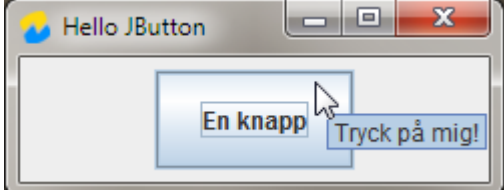
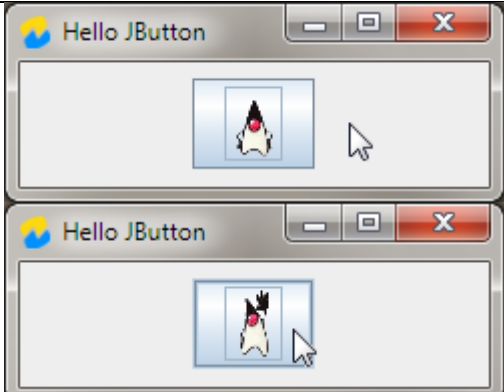
För att skapa en JButton är det enklast och mest använda sättet att i konstruktorn ange den text som

ska stå på knappen. Med metoden `setToolTipText(String)` kan man ange vilken "hjälpstext" som ska synas om man håller muspekaren stilla över knappen en kortstund. Kan vara bra att använda för att ge användaren information om vad som händer när man trycker på knappen. Denna metod kan för övrigt användas på alla komponenter i Swing.

Man kan även skapa en knapp med en bild istället för (eller tillsammans med en) text. Detta görs precis som med en `JLabel` med en `ImageIcon`. Med metoden `setRolloverIcon(ImageIcon)` kan man även ange den bild som ska synas när man för muspekaren över knappen, man får en så kallad mouse-over effekt. Det finns andra varianter av "set icon" för att byta ikon för knappens alla olika tillstånd (`selected`, `enabled`, `disabled`, `pressed`).

Det kan ibland vara nödvändigt att förhindra att en knapp ska gå att tryckas på i en applikation. För detta finns metoden `setEnabled(boolean)` som "gråar ut" knappen om man skickar `false` som argument (det går inte att trycka på den). För att återställa knappen skickar man `true` som argument.

Några exempel på användning av `JButton`:

<pre>JButton jButton = new JButton("En knapp"); jButton.setPreferredSize(new Dimension(100, 50)); jButton.setToolTipText("Tryck på mig!");</pre>	
<pre>JButton jButton = new JButton(new ImageIcon("figur.png")); jButton.setRolloverIcon(new ImageIcon("figur2.png")); // when using rollover icons, set the pressed // icon to the same icon rollover is using jButton.setPressedIcon(jButton.getRolloverIcon());</pre>	
<pre>JButton jButton = new JButton(); jButton.setText("En knapp med bild"); jButton.setHorizontalTextPosition(JButton.LEFT); ; jButton.setIcon(new ImageIcon("figur.png")); jButton.setMnemonic(KeyEvent.VK_E); // Aktivera med Alt+E</pre>	
<pre>JButton jButton = new JButton("En knapp"); jButton.setEnabled(false);</pre>	
<pre>JButton jButton = new JButton("<html><i>En knapp</i></html>"); jButton.setForeground(Color.blue); jButton.setBorderPainted(false); jButton.setFocusable(false);</pre>	

JTextField

Vill man på ett enkelt sätt låta användaren mata in text till applikationen kan man använda sig av en `JTextField`. Detta är en komponent som består av en enda rad där text kan skrivas in.

Trycker användaren på enter när textmarkören befinner sig i textfältet genereras ett `ActionEvent` (precis som om man klickar på en knapp). Detta kan vara väldigt smidigt att använda i applikationer (se nästa lektion).

En `JTextField` har givetvis metoderna `setText()` och `getText()` som vi vid det här laget vet vad de utför. Men i och med att man i ett textfält kan markera en viss del av texten med muspekaren finns även metoden `getSelectedText()` som returnerar en sträng innehållandes den text som är markerad i textfältet.

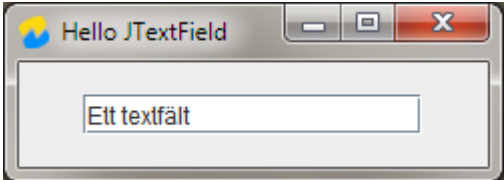
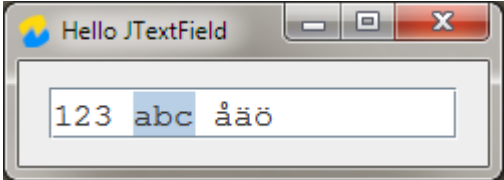
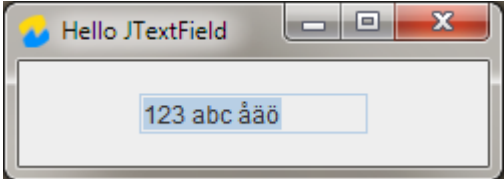
Om vi vill kan vi ändra den font som texten i textfältet ska ha (fungerar även på alla komponenter i Swing). För att göra detta anropar vi metoden `setFont(Font)` som tar ett `Font`-objekt som argument. För att skapa en ny `Font` måste man ange vilken font, stil och storlek det ska vara.

```
Font f = new Font("Courier", Font.PLAIN, 16);
```

Raden ovan skapar ett nytt `Font`-objekt med typsnittet Courier. Stilen är helt vanligt, det vill säga den är varken fet eller kursiv. Storleken sätts till 16 punkter. Med `Font.BOLD` och `Font.ITALIC` fås fet eller kursiv stil.

Med metoden `setEditable(boolean)` kan vi ange om det ska gå att ändra texten i textfältet eller inte (jämför med `setEnabled()` för `JButton`).

Några exempel på användning av `JTextField`:

<pre>JTextField txt = new JTextField("Ett textfält"); txt.setColumns(15); String text = txt.getText();</pre>	
<pre>JTextField txt = new JTextField(20); txt.setText("123 abc åäö"); txt.setFont(new Font("Courier", Font.PLAIN, 16)); txt.setSelectionStart(4); txt.setSelectionEnd(7); String text = txt.getSelectedText();</pre>	
<pre>JTextField txt = new JTextField("123 abc åäö", 10); txt.setEditable(false); txt.selectAll();</pre>	

JTextArea

En `JTextField` kan som sagt endast visa en rad med text. Om du har behov av att kunna visa flera rader med text måste du använda en `JTextArea`. Bortsett från hur många rader med text som kan

visas, liknar JTextArea väldigt mycket JTextField. Många av metoderna som kan anropas på en JTextField kan även användas på en JTextArea. Något vi dock måste tänka på när vi skapar en JTextArea är att vi måste lägga till scroll-lister själv om vi vill kunna scrolla upp/ned eller höger/vänster i textrutan. Detta läggs inte till automatiskt.

För att skapa en JTextArea med scroll-lister börjar vi med att skapa själva textrutan. Detta görs vanligtvis genom att anropa konstruktorn som inte tar några argument. Scroll-listerna skapas sen av klassen JScrollPane och till denna konstruktor skickar vi den textruta som ska ha scroll-lister. Ex:

```
JTextArea txt = new JTextArea();  
JScrollPane scroll = new JScrollPane(txt);
```

JScrollPane kan ses som en behållare för en JTextArea så när vi sätter storleken och placeringen gör vi detta på JScrollPane och inte på textrutan. Samma sak när vi lägger till textrutan i till exempel en JFrame. Det är JScrollPane vi lägger till och inte JTextArea. En JScrollPane kan för övrigt användas för alla komponenter.

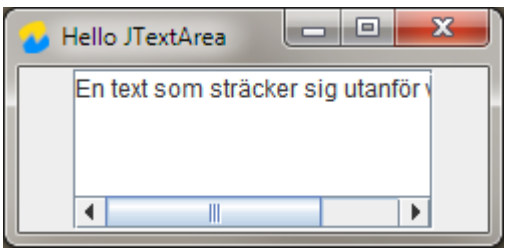
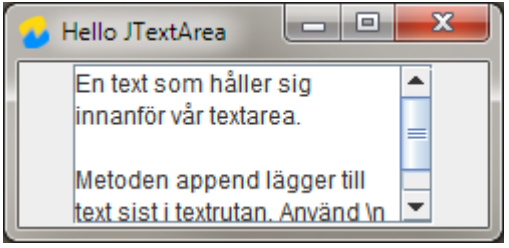
Default skrivs all text i en textruta på samma rad, den bryts inte när texten kommer utanför textrutans gränser. För att få till radbrytning anropar vi metoden `setLineWrap(boolean)` där vi skickar med `true` om vi vill att texten ska brytas eller `false` om vi inte vill att den ska brytas.

När texten bryts kan det ske mitt i ett ord, vilket inte är speciellt snyggt. För att ange att radbrytning endast får ske mellan orden i texten anropar vi metoden `setWrapStyleWord(boolean)` med `true`.

Med metoderna `setText(String)` och `getText()` sätter vi och returnerar vi texten i textrutan. Vill vi lägga till text sist i textrutan och behålla det som redan står där får vi anropa metoden `append(String)`.

Vi kan även ersätta en viss text som redan står i textrutan med en annan text genom att anropa metoden `replaceRange(String, int, int)` där vi skickar med den nya strängen samt start- och slutpositionen på de bokstäver som ska ersättas.

Några exempel på användning av JTextField:

<pre>JTextArea txt = new JTextArea(); JScrollPane scroll = new JScrollPane(txt); scroll.setPreferredSize(new Dimension(180,80)); txt.setText("En text som sträcker sig utanför vår textarea.");</pre>	
<pre>JTextArea txt = new JTextArea(); JScrollPane scroll = new JScrollPane(txt); scroll.setPreferredSize(new Dimension(180,80)); txt.setLineWrap(true); txt.setWrapStyleWord(true); txt.setText("En text som håller sig utanför vår textarea."); txt.append("\n\nMetoden append lägger till text sist i textrutan."); txt.append(" Använd \\n för att infoga radbrytningar."); txt.replaceRange("håller sig innanför", 12, 30);</pre>	

De komponenter vi gått igenom ovan är de mest vanliga komponenterna. Som du vet finns det betydligt många fler komponenter som kan användas. Hur alla dessa komponenter används får du själv ta reda på genom att läsa till exempel kurslitteraturen. Har du frågor om någon komponent tveka inte att höra av dig, genom till exempel ett inlägg i forumet.

LayoutManager

För att bestämma komponenters storlek och position i en container, som till exempel JFrame, används en LayoutManager. Det är ett speciellt objekt som dynamiskt placerar ut komponenterna i en container. Varje LayoutManager har en speciell placeringsstrategi och tar bland annat i beräkning containerns och komponenternas (önskade) storlek för att bestämma hur komponenterna ska placeras ut. För att ange vilken LayoutManager en container ska använda anropar vi:

```
setLayout (LayoutManager manager)
```

För vissa containers kan vi även ange LayoutManager direkt i konstruktorn när vi skapar containern.

Det finns ett stort antal olika LayoutManager, men många av dem används endast i speciella komponenter. När det gäller LayoutManager för att placera komponenter i en container finns det en handfull användbara sådana.

- BorderLayout
Placerar komponenter i fem olika områden. Endast en komponent i varje område.
- BoxLayout
Placerar komponenter antingen i en rad eller i en kolumn.
- CardLayout
Innehåller olika "kort" som i sin tur kan innehålla olika komponenter. Endast ett kort är synligt åt gången. Man kan välja vilket kort som ska vara synligt genom att till exempel bläddra framåt eller bakåt.
- FlowLayout
Placerar ut komponenter en och en efter varandra i en rad. Börjar på en ny rad om utrymmet inte räcker till.
- GridBagLayout
Placerar ut komponenter i ett rutnät. Varje rad och kolumn kan ha olika storlekar och en komponent kan uppta flera rader och/eller kolumner.
- GridLayout
Placerar ut komponenterna i ett rutnät där alla komponenter blir lika stora.
- GroupLayout
Framtagen för att användas tillsammans med ett grafiskt utvecklingsverktyg där GUI byggs genom dra-och-släpp av komponenter. Svår att använda om vi kodar "för hand".
- SpringLayout
Samma syfte som GroupLayout, men lite lättare att använda vid kodning för hand. Smidig att använda för att skapa formulär.

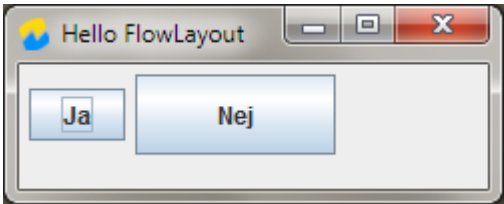
Av dessa olika LayoutManager är FlowLayout, BorderLayout och GridLayout de som är enklast att använda. Genom att kombinera dessa går det att bygga avancerade GUI med många komponenter.

FlowLayout

Den absolut enklaste av dessa att använda är nog FlowLayout som placerar ut komponenterna i den

ordning de läggs till i containern (med metoden add). Komponenterna placeras ut från vänster till höger och uppifrån och ner (kan även placeras höger till vänster om `ComponentOrientation.RIGHT_TO_LEFT` används i containern). När vi skapar en ny `FlowLayout` är det möjligt att i konstruktorn ange hur komponenterna ska justeras i förhållande till containern. Möjliga värden är `CENTER`, `LEADING`, `LEFT`, `RIGHT` och `TRAILING`. Det är även möjligt att ange hur många pixlars mellanrum det ska vara mellan varje komponent och rad.

Några exempel på användning av `FlowLayout`:

<pre>JFrame frame = new JFrame("Hello FlowLayout"); frame.setLayout(new FlowLayout()); frame.setSize(250, 100); frame.add(new JButton("Ja")); frame.add(new JButton("Nej")); frame.add(new JButton("Kanske")); frame.setVisible(true);</pre>	
<pre>JFrame frame = new JFrame("Hello FlowLayout"); frame.setLayout(new FlowLayout()); frame.getContentPane().setComponentOrientation(ComponentOrientation.RIGHT_TO_LEFT); frame.setSize(250, 100); frame.add(new JButton("Ja")); frame.add(new JButton("Nej")); frame.add(new JButton("Kanske")); frame.setVisible(true);</pre>	
<pre>JFrame frame = new JFrame("Hello FlowLayout"); frame.setLayout(new FlowLayout(FlowLayout.LEFT)); frame.setSize(250, 100); JButton c1 = new JButton("Ja"); JButton c2 = new JButton("Nej"); c2.setPreferredSize(new Dimension(100, 40)); frame.add(c1); frame.add(c2); frame.setVisible(true);</pre>	
<pre>JFrame frame = new JFrame("Hello FlowLayout"); frame.setLayout(new FlowLayout(FlowLayout.LEFT, 15, 10)); // 15 px mellan varje komponent och 10 px mellan varje rad frame.setSize(250, 100); JButton c1 = new JButton("Ja"); JButton c2 = new JButton("Nej"); c2.setPreferredSize(new Dimension(100, 40)); frame.add(c1); frame.add(c2); frame.setVisible(true);</pre>	

BorderLayout

Denna `LayoutManager` placerar som sagt ut komponenter i fem namngivna områden: `PAGE_START`, `PAGE_END`, `LINE_START`, `LINE_END` och `CENTER`. Komponenternas storlek ändras så att de hela tiden upptar största möjliga utrymme i varje område. Viktigt att tänka på är att

varje område i en BorderLayout endast rymmer en komponent. Om du behöver lägga till flera komponenter i ett område (t.ex. "PAGE_START") måste du använda dig av en container av något slag i vilken du lägger komponenterna. Därefter lägger du till din container i PAGE_START.

Före version 1.4 av Java användes andra namn på de olika områdena (WEST, EAST, NORTH, SOUTH och CENTER), men nu är rekommendationen att använda de tidigare nämnda namnen för att låta gränssnittet bättre anpassas till språk som använder annan orientering än vänster-till-höger.

Några exempel på användning av BorderLayout:

```
JFrame frame = new JFrame("Hello
BorderLayout");
frame.setLayout(new BorderLayout());
frame.setSize(250, 100);
JLabel c1 = new JLabel("PAGE_START",
JLabel.CENTER);
c1.setOpaque(true);
c1.setBackground(Color.YELLOW);

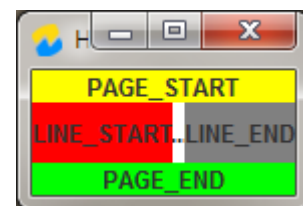
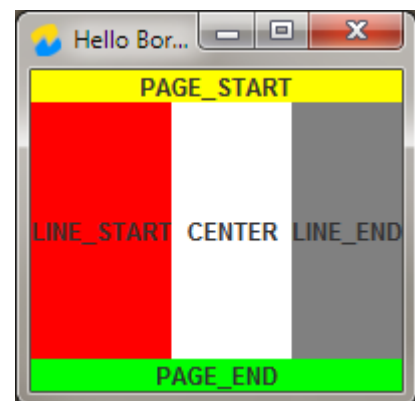
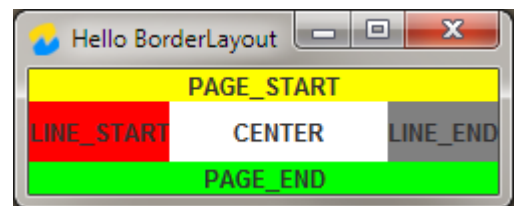
JLabel c2 = new JLabel("PAGE_END",
JLabel.CENTER);
c2.setOpaque(true);
c2.setBackground(Color.GREEN);

JLabel c3 = new JLabel("CENTER",
JLabel.CENTER);
c3.setOpaque(true);
c3.setBackground(Color.WHITE);

JLabel c4 = new JLabel("LINE_START",
JLabel.CENTER);
c4.setOpaque(true);
c4.setBackground(Color.RED);

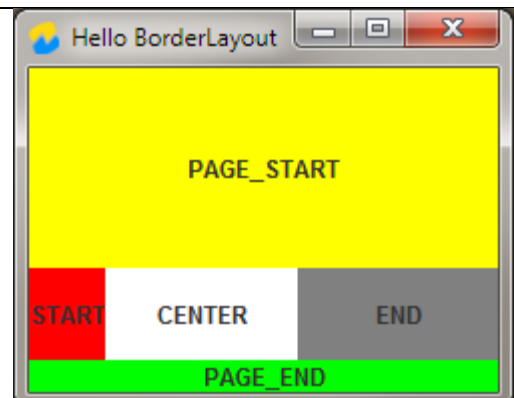
JLabel c5 = new JLabel("LINE_END",
JLabel.CENTER);
c5.setOpaque(true);
c5.setBackground(Color.GRAY);

// Lägg till komponenten i fönstret
frame.add(c1, BorderLayout.PAGE_START);
frame.add(c2, BorderLayout.PAGE_END);
frame.add(c3, BorderLayout.CENTER);
frame.add(c4, BorderLayout.LINE_START);
frame.add(c5, BorderLayout.LINE_END);
frame.setVisible(true);
```



```
JFrame frame = new JFrame("Hello
BorderLayout");
frame.setLayout(new BorderLayout());
frame.setSize(250, 200);
JLabel c1 = new JLabel("PAGE_START",
JLabel.CENTER);
c1.setOpaque(true);
c1.setBackground(Color.YELLOW);
c1.setPreferredSize(new Dimension(0, 100));

JLabel c2 = new JLabel("PAGE_END",
JLabel.CENTER);
c2.setOpaque(true);
c2.setBackground(Color.GREEN);
```

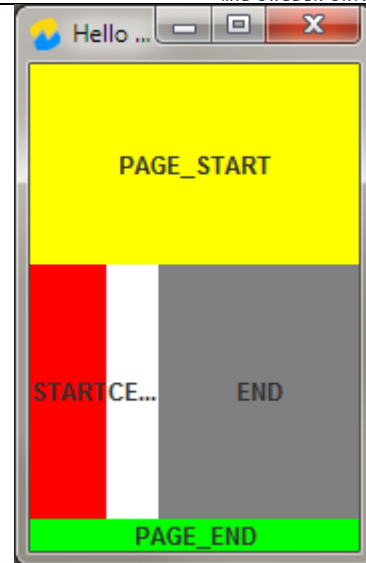



```
JLabel c3 = new JLabel("CENTER",
JLabel.CENTER);
c3.setOpaque(true);
c3.setBackground(Color.WHITE);

JLabel c4 = new JLabel("START", JLabel.CENTER);
c4.setOpaque(true);
c4.setBackground(Color.RED);

JLabel c5 = new JLabel("END", JLabel.CENTER);
c5.setOpaque(true);
c5.setBackground(Color.GRAY);
c5.setPreferredSize(new Dimension(100, 0));

// Lägg till komponenten i fönstret
frame.add(c1, BorderLayout.PAGE_START);
frame.add(c2, BorderLayout.PAGE_END);
frame.add(c3, BorderLayout.CENTER);
frame.add(c4, BorderLayout.LINE_START);
frame.add(c5, BorderLayout.LINE_END);
frame.setVisible(true);
```



GridLayout

När det finns behov av att placera komponenter i ett rutnät med lika stora celler är GridLayout perfekt att använda. När vi skapar en GridLayout kan vi i konstruktorn ange hur många rader och kolumner med komponenter det ska finnas. I varje cell får det plats endast en komponent. Det finns även möjlighet att i konstruktorn ange hur många pixlars mellanrum det ska vara mellan varje komponent.

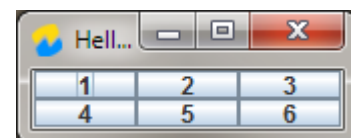
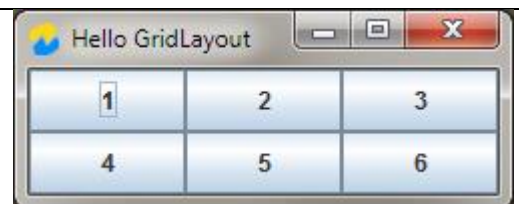
Komponenterna placeras ut antingen vänster-till-höger eller höger-till-vänster beroende på vilken ComponentOrientation containern har. Om vi har angett ett värde större än 0 för både antal kolumner och rader så kommer värdet för antal kolumner att ignoreras om vi lägger till fler komponenter än vad som ursprungligen fick plats. Det vill säga om vi skapat en GridLayout med 3 rader och 3 kolumner och lägger till 12 komponenter så blir resultatet 4 kolumner och 3 rader..

Några exempel på användning av GridLayout:

```
JFrame frame = new JFrame("Hello GridLayout");
frame.setLayout(new GridLayout(2, 3));
frame.setSize(250, 100);
```

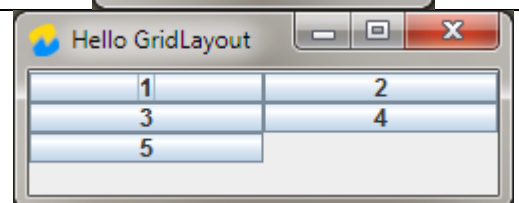
```
frame.add(new JButton("1"));
frame.add(new JButton("2"));
frame.add(new JButton("3"));
frame.add(new JButton("4"));
frame.add(new JButton("5"));
frame.add(new JButton("6"));
```

```
frame.setVisible(true);
```



```
JFrame frame = new JFrame("Hello GridLayout");
frame.setLayout(new GridLayout(4, 1));
frame.setSize(250, 100);
```

```
JButton c1 = new JButton("1");
c1.setPreferredSize(new Dimension(999, 999));
```



```
frame.add(c1);  
frame.add(new JButton("2"));  
frame.add(new JButton("3"));  
frame.add(new JButton("4"));  
frame.add(new JButton("5"));  
  
frame.setVisible(true);
```

SpringLayout

Som nämnts tidigare är främsta syftet med SpringLayout att den ska användas i en utvecklingsmiljö som erbjuder skapande av GUI med hjälp av dra-och-släpp av komponenter. Det vill säga användaren använder musen för att placera ut komponenterna som ska ingå i det grafiska användargränssnittet. Tittar man på koden som automatiskt produceras av en sådan miljö blir det snabbt väldigt många rader med kod och oftast ganska svårbegriplig kod.

Det är möjligt att koda för hand även med SpringLayout och vi kan i stort sett härma alla typer av LayoutManager med en SpringLayout. Dock krävs ganska mycket kod för att få till det. Jag kommer inte gå in på någon detaljerad förklaring av SpringLayout utan nöjer mig med att visa korta exempel på hur det fungerar.

Kort beskrivit fungerar en SpringLayout som så att komponenter placeras ut i förhållande till varandras kantar, samt kanterna på den container komponenterna finns i. Det går till exempel att placera en komponent låt säga 5 pixlar till höger och 5 pixlar nedanför containerns övre vänstra hörn. Därefter kan vi placera en annan komponent 10 pixlar till höger om första komponentens vänstra kant. Detta görs genom att anropa följande metod i SpringLayout:

```
putConstraint(String e1, Component c1, int pad, String e2, Component c2)
```

Här anger e1 vilken kant på komponenten c1 som ska placeras pad pixlar från kant e2 på komponent c2. Vilken kant som avses anger man med någon av följande värden:

- SpringLayout.NORTH
- SpringLayout.SOUTH
- SpringLayout.EAST
- SpringLayout.WEST
- SpringLayout.VERTICAL_CENTER
- SpringLayout.HORIZONTAL_CENTER
- SpringLayout.BASELINE

Följande exempel visar hur en SpringLayout kan användas för att placera ut en knapp så att knappens fyra kantar hela tiden placeras 10 pixlar från fönstrets kanter.

```
import javax.swing.*;  
import java.awt.*;  
import java.awt.event.*;  
import java.util.*;  
  
public class HelloSpringLayout extends JFrame {  
  
    public HelloSpringLayout(String title) {  
        // Sätt titel på fönstret  
        super(title);  
    }  
}
```

```
// Vad ska ske när vi stänger fönstret?
setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

// Centrera fönstret på skärmen
setLocationRelativeTo(null);

// Ange vilka ikoner som fönstret ska använda
ArrayList<Image> iconImages = new ArrayList<Image>();
iconImages.add(new ImageIcon("miun16x16.png").getImage());
iconImages.add(new ImageIcon("miun32x32.png").getImage());
iconImages.add(new ImageIcon("miun64x64.png").getImage());
setIconImages(iconImages);

// Initiera alla komponenter
initComponents();

// Sätt storleken på fönstret
setSize(250, 100);

// Gör fönstret synligt
setVisible(true);
}

private void initComponents() {
    // Ange vilken layout som ska användas
    SpringLayout layout = new SpringLayout();
    setLayout(layout);

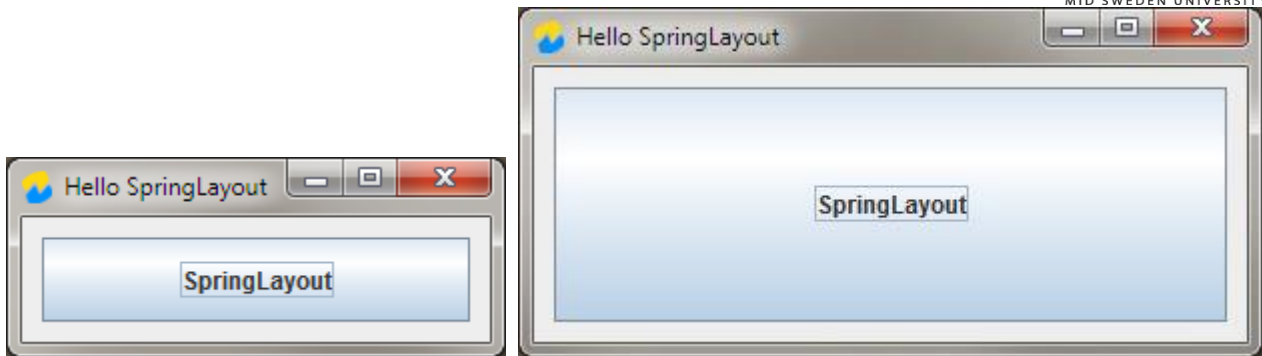
    JButton c1 = new JButton("SpringLayout");

    // Placera vardera hörn på knappen 10 pixlar från aktuell container
    layout.putConstraint(SpringLayout.WEST, c1, 10, SpringLayout.WEST,
        getContentPane());
    layout.putConstraint(SpringLayout.NORTH, c1, 10, SpringLayout.NORTH,
        getContentPane());
    layout.putConstraint(SpringLayout.EAST, c1, -10, SpringLayout.EAST,
        getContentPane());
    layout.putConstraint(SpringLayout.SOUTH, c1, -10, SpringLayout.SOUTH,
        getContentPane());

    // Lägg till komponenten i fönstret
    add(c1);
}

public static void main(String args[]) {
    // Skapa en ny instans av vårt fönster
    new HelloSpringLayout("Hello SpringLayout");
}
}
```

Första raden med `putConstraint` anger att den västra kanten på knappen ska placeras 10 pixlar från västra kanten på containern. Nästa rad anger att knappens norra kant ska placeras 10 pixlar från containerns norra kant. Därefter placeras östra kanten på knappen -10 pixlar från östra kanten på containerna och samma sak med knappens södra kant som placeras -10 pixlar från containerns södra kant. Notera att -10 används för att kanten ska hamna innanför fönstret.



Anledningen till att jag nämner SpringLayout är kanske främst för att det är relativt snabbt och smidigt gjort att skapa en layout för formulärdata. Som exempel kan vi utgå från inlämningsuppgift 1 och de uppgifter som ska anges för en kund i banken. Vill vi skapa ett grafisk användargränssnitt där användaren kan mata in data för en ny kund behöver vi en textruta (JTextField) för varje kunduppgift samt en beskrivande etikett (JLabel).

I exemplen på Oracles sidor finns en klass med namnet SpringUtilities. Denna klass innehåller två metoder, makeGrid och makeCompactGrid, som kan användas för att skapa en SpringLayout med alla nödvändiga constraints (putConstraint) för att placera ut komponenter i ett rutnät. Vi kan då placera alla våra etiketter i en kolumn och alla textfält i en annan. Tyvärr ingår inte SpringUtilities i Javas API utan du måste ladda ner klassen själv och inkludera den i dina projekt om du vill använda klassen.

<http://docs.oracle.com/javase/tutorial/uiswing/examples/layout/SpringGridProject/src/layout/SpringUtilities.java>

Nedanstående exempel visar hur vi kan skapa ett formulär för att mata in kunduppgifter till SmåStålar AB.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;

public class CustomerForm extends JFrame {
    private String[] labels = {"Förnamn", "Efternamn", "Personnummer",
    "Adress", "Postnummer", "Postort", "E-post"};
    private int[] mnemonicIndex = {0, 0, 0, 0, 4, 4, 2};
    private JLabel[] jLabels = new JLabel[labels.length];
    private JTextField[] jTextFields = new JTextField[labels.length];

    public CustomerForm(String title) {
        // Sätt titel på fönstret
        super(title);

        // Vad ska ske när vi stänger fönstret?
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

        // Centrera fönstret på skärmen
        setLocationRelativeTo(null);

        // Ange vilka ikoner som fönstret ska använda
        ArrayList<Image> iconImages = new ArrayList<Image>();
        iconImages.add(new ImageIcon("miun16x16.png").getImage());
        iconImages.add(new ImageIcon("miun32x32.png").getImage());
        iconImages.add(new ImageIcon("miun64x64.png").getImage());
        setIconImages(iconImages);
    }
}
```

```
// Initiera alla komponenter
initComponents();

// Sätt storleken på fönstret
setSize(250, 200);

// Gör fönstret synligt
setVisible(true);
}

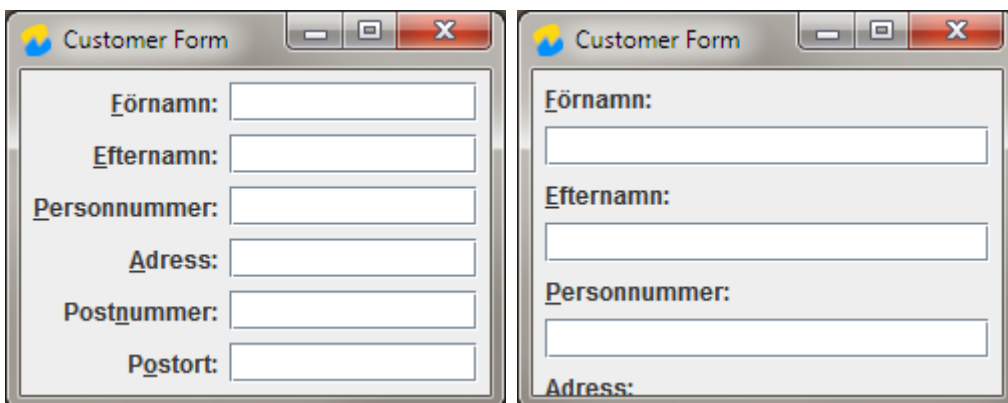
private void initComponents() {
    // Ange vilken layout som ska användas
    GroupLayout layout = new GroupLayout();
    setLayout(layout);

    // Skapa alla komponenter
    for (int i = 0; i < labels.length; i++) {
        // JLabel
        jLabels[i] = new JLabel(labels[i] + ":", JLabel.TRAILING);
        add(jLabels[i]);

        // JTextField
        jTextFields[i] = new JTextField(10);
        jLabels[i].setLabelFor(jTextFields[i]);
        jLabels[i].setDisplayedMnemonic(labels[i].charAt(mnemonicIndex[i]));
        add(jTextFields[i]);
    }

    SpringUtilities.makeCompactGrid(getContentPane(),
        labels.length, 2, // rows, cols
        6, 6, // initX, initY
        6, 6); // xPad, yPad
}

public static void main(String args[]) {
    // Skapa en ny instans av vårt fönster
    new CustomerForm("Customer Form");
}
}
```



Vill vi i stället att etiketterna ska placeras ovanför textrutorna behöver vi bara ändra på två ställen i koden. Vi använder JLabel.LEADING istället för JLabel.TRAILING samt sätter antal rader till $\text{labels.length} * 2$ och antal kolumner till 1 (när vi anropar makeCompactGrid). Resultatet blir då enligt bilden ovan till höger.

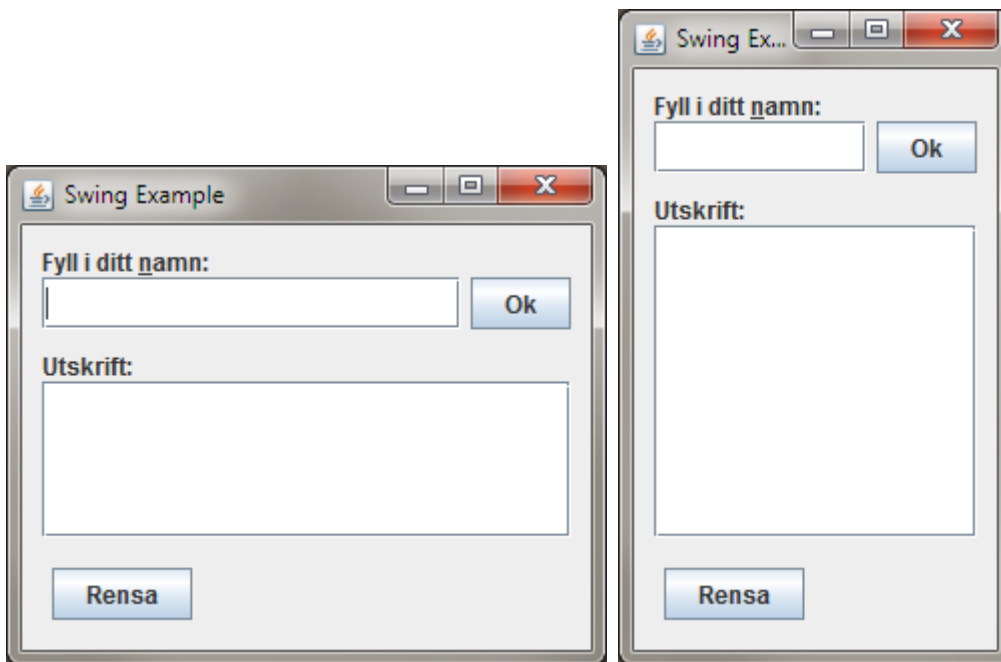
Absolut positionering

Det finns också möjlighet till absolut positionering av en komponent i en container. För att göra detta sätter vi layouten i containern till null, det vill säga till ingenting alls. Nu måste vi för varje komponent själv ange storlek (setSize) och position (setLocation). Detta tillvägagångssätt är inget som rekommenderas då det är svårt att anpassa sitt GUI så det ser bra ut på alla plattformar (olika komponenter kan vara olika stor beroende på vilken look-and-feel som används).

JPanel

Den sista komponenten vi ska titta på i denna lektion är JPanel. Vi har valt att beskriva denna sist eftersom vi först ville gå igenom vad LayoutManager är. Veldig ofta används JPanel som en container åt andra komponenter. I ett grafiskt användargränssnitt kan många olika JPanel användas för att uppnå det utseende man vill ha. Varje JPanel kan ha en egen LayoutManager för att placera ut sina komponenter. Förutom att innehålla komponenter används JPanel även för att skapa egna komponenter och för att rita frihandsgrafik. Detta är något vi tittar närmare på i lektion 5 och lektion 6.

Låt säga vi vill skapa ett användargränssnitt enligt bilderna nedan där texttrutan och textfältets storlek ändras utifrån storleken på fönstret:



Vi kan då kombinera ett flertal olika JPanel med olika LayoutManagers enligt följande exempel:

```
import javax.swing.*;
import java.awt.*;
import java.util.*;

public class SwingExample extends JFrame
{
    // Instansvariabler (normalt alla komponenter)
    private JLabel nameLabel;
    private JLabel outputLabel;
    private JButton okButton;
```

```
private JButton clearButton;
private JTextField nameTextField;
private JTextArea outputTextArea;

public SwingExample(String title) {
    // Sätt titel på fönstret
    super(title);

    // Vad ska ske när vi stänger fönstret?
    setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

    // Centrera fönstret på skärmen
    setLocationRelativeTo(null);

    // Ange vilken layout som ska användas i fönstret
    setLayout(new BorderLayout());

    // Ange vilka ikoner som fönstret ska använda
    ArrayList<Image> iconImages = new ArrayList<Image>();
    iconImages.add(new ImageIcon("miun16x16.png").getImage());
    iconImages.add(new ImageIcon("miun32x32.png").getImage());
    iconImages.add(new ImageIcon("miun64x64.png").getImage());
    setIconImages(iconImages);

    // Initiera alla komponenter
    initComponents();

    // Sätt storleken på fönstret
    setSize(300, 250);

    // Gör fönstret synligt
    setVisible(true);
}

private void initComponents() {
    // Skapa och sätt LayoutManager för alla JPanel
    JPanel namePanel = new JPanel(new BorderLayout(5, 0));
    JPanel outputPanel = new JPanel(new BorderLayout());
    JPanel clearPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
    JPanel formPanel = new JPanel(new BorderLayout(10, 10));

    // Skapa komponenter för name panel och lägg till dessa
    okButton = new JButton("Ok");
    nameTextField = new JTextField();
    nameLabel = new JLabel("Fyll i ditt namn:");
    nameLabel.setDisplayedMnemonic('n');
    nameLabel.setLabelFor(nameTextField);

    namePanel.add(nameLabel, BorderLayout.PAGE_START);
    namePanel.add(nameTextField, BorderLayout.CENTER);
    namePanel.add(okButton, BorderLayout.LINE_END);

    // Skapa komponenter för output panel och lägg till dessa
    outputLabel = new JLabel("Utskrift:");

    outputTextArea = new JTextArea();
    outputTextArea.setEditable(false);
    outputTextArea.setLineWrap(true);
    outputTextArea.setWrapStyleWord(true);
    JScrollPane scroll = new JScrollPane(outputTextArea);

    outputPanel.add(outputLabel, BorderLayout.PAGE_START);
```

```
outputPanel.add(scroll, BorderLayout.CENTER);

// Skapa komponenter för clear panel och lägg till dessa
clearButton = new JButton("Rensa");
clearPanel.add(clearButton);

// Lägg till panelerna i form panel
formPanel.add(namePanel, BorderLayout.PAGE_START);
formPanel.add(outputPanel, BorderLayout.CENTER);
formPanel.add(clearPanel, BorderLayout.PAGE_END);

// Sätt en 10 pixlar bred osynlig kant runt form panel
formPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

// Lägg till form panel i fönstret
add(formPanel, BorderLayout.CENTER);
}

public static void main(String args[]) {
    // Skapa en ny instans av vårt fönster
    new SwingExample("Swing Example");
}
}
```

Hur skulle ovanstående användargränssnitt implementeras med en `SpringLayout` i stället? Prova gärna själv att ändra ovanstående exempel så att en enda `SpringLayout` används. Hör av dig till kursansvariga om du är intresserad av ett lösningsförslag.

Ett generellt råd är att planera hur det grafiska användargränssnittet ska se ut innan du börjar skriva kod. Skissa gärna på ett papper och fundera över vilka ”delfönster” som finns. Varje delfönster låter du bli en egen `JPanel`. Kanske dina delfönster kan delas upp i ännu fler delfönster? Vilken typ av `LayoutManager` bör du använda i varje delfönster?