Introduction to graphics programming in Java

Mads Rosendahl

February 13, 2009

Introduction. Writing graphics applications in Java using Swing can be quite a daunting experience which requires understanding of some large libraries, and fairly advanced aspects of Java. In these notes we will show that by using a small subset of the Swing package we can write a wide range of graphics programs. To make this possible we have constructed three small classes that simplifies three of the more complex aspects of graphics programming: 2D-graphics, layout of components, and event-handling.

Prerequisites. These notes are written for an introductory programming course. Most of the examples just uses a single main method and can be understood early in such a course. Some of the later examples contains other static methods and in some final examples we define some classes.

Overview. The notes are organized in three parts - corresponding to the three classes mentioned below: We examine how to draw various geometric shapes on a canvas. We examine a range of standard Swing components and show how to place them in a window. Finally we show how to react to events from components in a window.

Material. The notes uses three classes written specifically for these notes. They are:

- JCanvas. A JComponent that behaves as a Graphics2D object. You can perform the usual drawing operations directly on a JCanvas.
- JBox. A container for other Swing components. A JBox works almost like a Box object in Swing, but offers easier control over the layout of components in the box. This means that a JBox can be used as an alternative to a number of different layout managers.
- JEventQueue. An event-handler for the usual events in Swing programs. The events are placed in a queue and the program can then extract them. Using a JEventQueue is an alternative to writing programs in an event-driven style.

They can all be used independently of each other, so that one may each case may use a standard Swing approach instead. These classes are all available from http://akira.ruc.dk/~madsr/swing/.

These notes are still work in progress. If you have suggestions, comments, corrections, please email them to me: madsr@ruc.dk. The programs have been tested on Java 1.5 with Windows XP. They do not work with earlier versions of Java but should work on other platforms running Java 1.5.
Contents

1 Frames and windows
  1.1 JFrame .................................................. 4
  1.2 Import libraries ...................................... 5

2 JCanvas ....................................................... 5
  2.1 JCanvas, general controls ......................... 7
  2.2 Specify how outlines are drawn .................. 8
  2.3 Specify how shapes are filled .................... 9
  2.4 Text in a canvas ..................................... 10
  2.5 Images .................................................. 10

3 Components in Swing ...................................... 12
  3.1 Introducing JBox ....................................... 13
  3.2 Setting size .......................................... 14
  3.3 Color and font ....................................... 17
  3.4 Labels .................................................. 17
  3.5 Buttons ............................................... 18
  3.6 CheckBox, Radiobutton and Togglebutton ........ 20
  3.7 Textfield and textarea ............................. 21
  3.8 Slider, Spinner and ProgressBar .................. 23
  3.9 ComboBox and List .................................. 26
  3.10 SplitPane and Scrollpane ......................... 27
  3.11 Borders ............................................... 28
  3.12 Other controls for Swing components .......... 30
  3.13 Adding and removing content of a box ........ 31

4 Types of events ............................................ 32
  4.1 Introducing JEventQueue ............................. 33
  4.2 Events from components with state ............... 37
  4.3 Text components ..................................... 38
  4.4 Events from a canvas ................................ 40
  4.5 Timers .................................................. 41
  4.6 Menus .................................................. 43
  4.7 Events from the frame ............................... 45

5 Animation ................................................... 46
  5.1 Slideshow ............................................. 47
  5.2 Bouncing with gravity ............................... 50
  5.3 Rolling background .................................. 51

6 Sound .......................................................... 52

7 More on drawing on a canvas .............................. 52
1 Frames and windows

Let us look at a simple Java program that creates a window. In Java terms a window is called a frame and it consists of:

- A top part with a title, a little icon and buttons to minimize, maximize and close the window.
- An optional menu area
- A content area where we can place buttons, text, drawings etc.

Let us just create such a frame and examine it a bit.

```java
import javax.swing.JFrame;

public class EmptyFrame{
    public static void main(String args[]){
        JFrame frame=new JFrame("EmptyFrame");
        frame.setSize(600,600);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

This will result in the following window:

The program uses the JFrame class which is part of the Java Swing package. To be able to use this class we need to import it into the program. That is done with the import statement on the first line. In the main method we construct the frame and the title is passed on as a parameter to the constructor. The title does not have to be the same as the name of the class. We then specify the size of the whole frame - in this case 600 pixels wide and 600 pixels high. We want
the program to stop when the user closes the window by clicking on the cross in the top right hand corner. Finally the frame is made visible by displaying it on the screen.

Notice that the program does not terminate just because we reach the end of the main method. In a sense the window is a separate program that runs in parallel with your own program. You tell it what to display but the window itself is able to be redrawn when it has been minimized.

This empty frame is only the first step on the way. There is a lot more to graphics programming. We will look at the following aspects.

- Putting standard components in a window: buttons, text areas, scroll panels, menus etc.
- Constructing your own graphics: draw shapes, images, text
- Handling events from components.

### 1.1 JFrame

The previous example uses the JFrame class. In these notes we will give an overview of the methods in classes in small tables. For the JFrame class the central methods are listed here.

| JFrame | import javax.swing.*;
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>new JFrame(String title)</td>
<td>Constructs a new, initially invisible JFrame object with the specified title.</td>
</tr>
<tr>
<td>void add(JComponent comp)</td>
<td>Specify the content of a frame. In section 3 we show how to add several items to a frame by placing them in boxes.</td>
</tr>
<tr>
<td>void setDefaultCloseOperation(int operation)</td>
<td>Sets the operation that will happen by default when the user initiates a “close” on this frame. Possible values are WindowConstants.DO NOTHING ON CLOSE, WindowConstants.HIDE ON CLOSE, WindowConstants.DISPOSE ON CLOSE and JFrame.EXIT ON CLOSE.</td>
</tr>
<tr>
<td>void setDefaultCloseOperation(int operation)</td>
<td>Sets the operation that will happen by default when the user initiates a ”close” on this frame. Possible values are WindowConstants.DO NOTHING ON CLOSE, WindowConstants.HIDE ON CLOSE, WindowConstants.DISPOSE ON CLOSE and JFrame.EXIT ON CLOSE.</td>
</tr>
<tr>
<td>void setIconImage(BufferedImage image)</td>
<td>Sets the image to be displayed in the minimized icon for this frame. Images are created as BufferedImage. See loadimage in section 2.5</td>
</tr>
<tr>
<td>void setJMenuBar(JMenuBar menubar)</td>
<td>Sets the menubar for this frame. Menus are discussed in section 4.6.</td>
</tr>
<tr>
<td>void setLocation(Point p)</td>
<td>Places the window at this location on the screen.</td>
</tr>
<tr>
<td>void setResizable(boolean resizable)</td>
<td>Sets whether this frame is resizable by the user.</td>
</tr>
<tr>
<td>void setSize(int width, int height)</td>
<td>Resizes this component so that it has width width and height height. Controlling the size of frames and content is further discussed in section 3.2.</td>
</tr>
<tr>
<td>void setTitle(String title)</td>
<td>Sets the title for this frame to the specified string.</td>
</tr>
<tr>
<td>void setVisible(boolean b)</td>
<td>Shows or hides this frame depending on the value of parameter b.</td>
</tr>
</tbody>
</table>
1.2 Import libraries

When we program for swing we need to include a number of classes from several different packages in the standard libraries. The examples in these notes will include import statements for some of the following packages.

<table>
<thead>
<tr>
<th>Package</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.swing.*</td>
<td>Standard components in swing: JFrame, JLabel, SwingConstants, WindowConstants, ImageIcon, BorderFactory</td>
</tr>
<tr>
<td>java.awt.*</td>
<td>Color, Font, some shapes, paints and strokes</td>
</tr>
<tr>
<td>java.awt.geom.*</td>
<td>Special shapes: Ellipse2D:Double, GeneralPath,..</td>
</tr>
<tr>
<td>java.awt.image.*</td>
<td>BufferedImage</td>
</tr>
<tr>
<td>java.util.*</td>
<td>EventObject</td>
</tr>
</tbody>
</table>

Inheritance plays a major role in the design of the graphics packages in Java. We will avoid a description of inheritance hierarchies whenever they are not important for the use of the classes. We will regularly describe parameters to methods as less specific than defined and return types as more specific. This can be done without losing precision or correctness.

2 JCanvas

A JCanvas object is an area in a window in which you can draw various shapes, images and text. It is not part of the standard swing distribution, but you can obtain the code from http://akira.ruc.dk/~madsr/swing/JCanvas.java. A JCanvas object contains all the methods from the Graphics2D object, normally used in swing graphics. The advantage is that you can draw in windows without designing new classes and you do not have to be familiar with inheritance in object oriented programming.

Components in a frame. Our first example just contained an empty frame. We will now display a canvas with some drawings in the frame. We will do this by adding a component to the frame. You should only add one component, but this may be a container that has several other components in it. We will discuss how to do that in section 3. For now you should add the component before you make the window visible.

Example. Let us start with a small example that shows most of the simple commands to draw on a canvas. You can draw the outline of various shapes or fill these shapes with a color. You can control the thickness of the pen and the color used when drawing.

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

public class DrawCanvas{
    public static void main(String args[]){
        JFrame frame=new JFrame();
        frame.setSize(600,600);
        frame.setTitle("DrawCanvas");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```
JCanvas canvas = new JCanvas();
frame.add(canvas);
frame.setVisible(true);
canvas.setStroke(new BasicStroke(5));
canvas.drawRect(10,20,100,200);
canvas.drawOval(120,20,100,200);
canvas.drawRoundRect(230,20,100,200,20,20);
canvas.drawArc(340,20,100,200,45,270);
canvas.drawLine(450,20,500,220);
canvas.drawDashedLine(10,480,20,530,220);
canvas.setPaint(Color.gray);
canvas.fillRect(10,320,100,200);
canvas.fillOval(120,320,100,200);
canvas.fillRoundRect(230,320,100,200,20,20);
canvas.fillArc(340,320,100,200,45,270);
}
}

This will result in the following window:

In the example above we create the canvas, we make it visible and then we draw a number of shapes and lines on it. If you look carefully at the screen when the program runs you may be able to see that it is not all displayed at the same time. We could also have made the drawings before it was visible. the drawings would then be displayed at once.

Compilation. To run this program you must make sure that you have downloaded the JCanvas.java file. If you compile your files directly from a command prompt then you should just place the JCanvas.java in the same directory as your own java file. The compiler will then find the JCanvas.java file when it need it. If you use an integrated development editor, like JEditor, then you will create a project and you should then add the JCanvas.java to the project.
Coordinate system. The coordinate system on the screen is a bit unusual in that the y-axis points down. The origin is the top left hand corner and the positive x and y values are to the right and below. In this example we \((x, y) = (335, 370)\)

In the rest of this section we will describe the operations we can perform on a canvas.

2.1 JCanvas, general controls

<table>
<thead>
<tr>
<th>JCanvas</th>
<th>Initial controls</th>
</tr>
</thead>
<tbody>
<tr>
<td>void setBackground(Color c)</td>
<td>Set the background color used by clear. You can also start by using fillRect to fill a suitably big area with a color.</td>
</tr>
<tr>
<td>void clear()</td>
<td>Clear the canvas to the background color</td>
</tr>
<tr>
<td>void startBuffer()</td>
<td>Start double buffering: nothing is shown until you call endBuffer();</td>
</tr>
<tr>
<td>void endBuffer()</td>
<td>End double buffering.</td>
</tr>
<tr>
<td>void print()</td>
<td>Print the content of the canvas. One unit corresponds to 1/72 of an inch.</td>
</tr>
<tr>
<td>void print(double scale)</td>
<td>Print the content of the canvas. Scale the content relative to the default of one unit corresponding to 1/72 of an inch.</td>
</tr>
<tr>
<td>void writeToImage(String s, int w, int h)</td>
<td>Write the content of the canvas to an image file with name s. Draw everything from point (0,0) to (w,h). The filename should end with &quot;.jpg&quot; or &quot;.png&quot;.</td>
</tr>
<tr>
<td>void sleep(int ms)</td>
<td>Sleep for ms 1/1000 of a second. This makes it easier to make small animations.</td>
</tr>
<tr>
<td>int getHeight()</td>
<td>return the height of the canvas. See section 3.2 for more about controlling the size of frames and their content.</td>
</tr>
<tr>
<td>int getWidth()</td>
<td>return the width of the canvas</td>
</tr>
</tbody>
</table>
void setClip(int x, int y, int w, int h) specify a clipping rectangle. Further drawing operation will only have effect inside this rectangle.

**Double buffering** Normally drawing operations are done one at a time. If you want to make animations it is important to create a whole scene and display it as one single operation. Otherwise the screen will flicker and show partial scenes. To avoid this you call startBuffer, make your drawing in the background and then call endBuffer when it is finished. This is called double-buffering.

<table>
<thead>
<tr>
<th>JCanvas</th>
<th>Simple outlines</th>
</tr>
</thead>
<tbody>
<tr>
<td>void drawLine(int x1, int y1, int x2, int y2)</td>
<td>Draw a line from (x1,y1) to (x2,y2)</td>
</tr>
<tr>
<td>void drawDashedLine(int d, int x1, int y1, int x2, int y2)</td>
<td>Draw a dashed line from (x1,y1) to (x2,y2) with dash length d</td>
</tr>
<tr>
<td>void drawOval(int x, int y, int w, int h)</td>
<td>Draw an oval with top-left (x,y), width w and height h</td>
</tr>
<tr>
<td>void drawRect(int x, int y, int w, int h)</td>
<td>Draw a rectangle with top-left (x,y), width w and height h</td>
</tr>
<tr>
<td>void drawRoundRect(int x, int y, int w, int h, int aw, int ah)</td>
<td>Draw a rectangle with rounded corners with top-left (x,y), width w and height h. Corners are arcs with width aw and height ah.</td>
</tr>
<tr>
<td>void drawArc(int x, int y, int w, int h, int start, int angle)</td>
<td>Draw parts of an oval with top-left (x,y), width w and height h. Start at angle start and span angle. Angles are measured in degrees.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JCanvas</th>
<th>Fill simple shapes</th>
</tr>
</thead>
<tbody>
<tr>
<td>void fillOval(int x, int y, int w, int h)</td>
<td>Fill an oval with top-left (x,y), width w and height h</td>
</tr>
<tr>
<td>void fillRect(int x, int y, int w, int h)</td>
<td>Fill a rectangle with top-left (x,y), width w and height h</td>
</tr>
<tr>
<td>void fillRoundRect(int x, int y, int w, int h, int aw, int ah)</td>
<td>Fill a rectangle with rounded corners with top-left (x,y), width w and height h. Corners are arcs with width aw and height ah.</td>
</tr>
<tr>
<td>void fillArc(int x, int y, int w, int h, int start, int angle)</td>
<td>Fill parts of an oval with top-left (x,y), width w and height h. Start at angle start and span angle. Angles are measured in degrees.</td>
</tr>
</tbody>
</table>

### 2.2 Specify how outlines are drawn

This section may seem like making to much out of a small thing - and it probably does just that. When you draw a line you can specify how it is drawn. This will normally just be to say how wide a line should be. You can, however, also specify how the end of lines should look and how to draw corners when you connect lines.

<table>
<thead>
<tr>
<th>JCanvas</th>
<th>Change Stroke</th>
</tr>
</thead>
<tbody>
<tr>
<td>void setStroke(Stroke s)</td>
<td>Set the stroke used in outlines</td>
</tr>
<tr>
<td>Stroke getStroke()</td>
<td>Get the current stroke.</td>
</tr>
</tbody>
</table>
Stroke

```
new BasicStroke(float width)
```

Constructs a solid BasicStroke with the specified line width and with default values for the cap and join styles.

```
new BasicStroke(float width, int cap, int join)
```

Constructs a solid BasicStroke with the specified attributes.
- **CAP_BUTT**  Ends unclosed subpaths and dash segments with no added decoration.
- **CAP_ROUND** Ends unclosed subpaths and dash segments with a round decoration that has a radius equal to half of the width of the pen.
- **CAP_SQUARE** Ends unclosed subpaths and dash segments with a square projection that extends beyond the end of the segment to a distance equal to half of the line width.
- **JOIN_BEVEL** Joins path segments by connecting the outer corners of their wide outlines with a straight segment.
- **JOIN_MITER** Joins path segments by extending their outside edges until they meet.
- **JOIN_ROUND** Joins path segments by rounding off the corner at a radius of half the line width.

```
new BasicStroke(float width, int cap, int join, float miterlimit, float[] dash, float dash_phase)
```

More general stroke to construct various dashed patterns. In most cases you can use drawDashedLine instead.

2.3 Specify how shapes are filled

You will normally fill an area with a color. Paint does come in more than single colors - you can paint with a texture or with a double-color that changes gradually from one point on the screen to another.

<table>
<thead>
<tr>
<th>JCanvas</th>
<th>Paint</th>
</tr>
</thead>
<tbody>
<tr>
<td>void setPaint(Paint p)</td>
<td></td>
</tr>
<tr>
<td>Paint getPaint()</td>
<td></td>
</tr>
</tbody>
</table>

**JCanvas**

- **void setPaint(Paint p)**  Set the paint used in fill operations
- **Paint getPaint()**  Get the current paint.

**Paint**

```
new Color(int r, int g, int b)
```

Creates an opaque sRGB color with the specified red, green, and blue values in the range (0 - 255).
To get a brown color you can write
```
Color brown = new Color(255,128,0);
```

- **Color.black**  The color black.
- **Color.blue**  The color blue.
- **Color.cyan**  The color cyan.
- **Color.darkGray**  The color dark gray.
- **Color.gray**  The color gray.
- **Color.green**  The color green.
- **Color.lightGray**  The color light gray.
The color magenta.
The color orange.
The color pink.
The color red.
The color white.
The color yellow.

new GradientPaint(float x1, float y1, Color color1, float x2, float y2, Color color2)
Constructs a simple acyclic GradientPaint object. It gradually changes from color color1
to color color2 from point (x1,y1) to point (x2,y2).

new TexturePaint(BufferedImage txtr, Rectangle2D anchor)
Constructs a TexturePaint object. It uses an image (see section 2.5) to fill a shape by repeatedly
displaying it side by side.

2.4 Text in a canvas

JCanvas

void setFont(Font f)
Set the font

Font getFont()
Get the current font

void drawString(String str, int x, int y)
Draw the string str staring at point (x,y)

void drawOutline(String str, int x, int y)
Draw the outline of string str staring at point (x,y). Each character is seen as lines and drawn,
rather than shapes and filled.

Font

new Font(String name, int style, int size)
Creates a new Font from the specified name,
style and point size.
name - the font name.
style - the style constant for the Font The
style argument is an integer bitmask that
may be Font.PLAIN, Font.BOLD, Font.ITALIC or
Font.BOLD|Font.ITALIC.
size - the point size of the Font

The font name can be one of: "Dialog", "DialogInput", "Monospaced", "Serif", or "SansSerif".
You can also use the various other fonts that are installed on your computer.

2.5 Images

We may draw various images on a canvas. Images are typically stored in jpg, gif or png format.
We have a number of operations to read, write, display and manipulate images.

JCanvas

import java.awt.image.*;

void.drawImage(BufferedImage im, int x, int y)
Draw the image with top left at point (x,y)

void drawScaledImage(BufferedImage im, int x, int y, int w, int h)
Draw the image with top left at point (x,y),
scaled to have width w and height h.
void drawScaledImage(BufferedImage im, int x, int y, float scale)

Draw the image with top left at point \((x, y)\), scaled with factor \(float\).

static BufferedImage loadImage(String s)

Reads an image from file \(s\).

static BufferedImage scaleImage(BufferedImage im, int w, int h)

Create a scaled instance of the image with width \(w\) and height \(h\).

static BufferedImage tileImage(BufferedImage im, int w, int h)

Create a new image of width \(w\) and height \(h\), tiled with the given image.

static BufferedImage cropImage(BufferedImage im, int x, int y, int w, int h)

Create a new image which is the selection from point \((x, y)\) with width \(w\) and height \(h\).

static BufferedImage rotateImage(BufferedImage im)

rotate an image 90 degrees clockwise.

static BufferedImage storeImage(BufferedImage im, String s)

write an image to file \(s\). The filename should end with "jpg" or "png".

When we load an image we may ask about the size of the image. We may want to rescale the image so it fills the available space.

In some of the later section we will need another kind of image called an Icon. There are no good reasons for these two classes - they behave in the same way but icons are used as images on labels and buttons. When we need to load an Icon, use the IconImage class.

new IconImage(String fileName)

create an IconImage object

Let us look at how one may use the image operations to achieve various effects.

CheckImage

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

public class CheckImage{
    public static void main(String args[]){
        JFrame frame=new JFrame("CheckImage");
        frame.setSize(800,600);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JCanvas canvas=new JCanvas();
        frame.add(canvas);
        frame.setVisible(true);

        BufferedImage img1=canvas.loadImage("res/Fox.jpg");
        BufferedImage img2=canvas.scaleImage(img1,400,300);
        BufferedImage img3=canvas.cropImage(img2,130,60,110,115);
        canvas.storeImage(img3,"fox1.jpg");
        BufferedImage img4=canvas.scaleImage(img3,2);
        BufferedImage img5=canvas.rotateImage(img3);
        BufferedImage img6=canvas.loadImage("fox1.jpg");
        BufferedImage img7=canvas.rotateImage(canvas.rotateImage(img6));
        BufferedImage img8=canvas.tileImage(img3,530,200);
        BufferedImage img9=canvas.rotateImage(img8);
    }
}
```
More about graphics. We will describe several other operations you can perform on a canvas in section 7. We will now look at other components you can place in a window.

3 Components in Swing

This section of the notes on graphics programming in Java covers the standard components in a window. We will look at which components one can put into a window and how we place them in the window. We will use the class JBox, which is almost identical to the Box class in the standard Swing package, but it is extended with a few extra features so that it can be used as a general layout structure for a large number of applications. In this part of the notes we will show which components we can place in a window and how we control where they are placed. In the next section (4) we will show how we can be informed about mouse-clicks and other types of input to a program.
**Component structure.** We can distinguish between the containers that contain other components and base components that display or receive information.

<table>
<thead>
<tr>
<th>JComponent</th>
<th>Basic Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>new JLabel(...)</td>
<td>a label, see section 3.4</td>
</tr>
<tr>
<td>new JButton(...)</td>
<td>a button, see section 3.5</td>
</tr>
<tr>
<td>new JRadioButton(...)</td>
<td>a radiobutton, see section 3.5</td>
</tr>
<tr>
<td>new JCheckBox(...)</td>
<td>a checkbox, see section 3.5</td>
</tr>
<tr>
<td>new JToggleButton(...)</td>
<td>a togglebutton, see section 3.5</td>
</tr>
<tr>
<td>new JCanvas()</td>
<td>a canvas</td>
</tr>
<tr>
<td>new JTextField(...)</td>
<td>a textfield, see section 3.7</td>
</tr>
<tr>
<td>new JTextArea(...)</td>
<td>a textarea, see section 3.7</td>
</tr>
<tr>
<td>new JFormattedTextField(...)</td>
<td>a formatted textfield, see section 3.7</td>
</tr>
<tr>
<td>new JSlider(...)</td>
<td>a slider, see section 3.8</td>
</tr>
<tr>
<td>new Progressbar(...)</td>
<td>a progress bar, see section 3.8</td>
</tr>
<tr>
<td>new JComboBox(...)</td>
<td>a combo box, see section 3.9</td>
</tr>
<tr>
<td>new JList(...)</td>
<td>a list, see section 3.9</td>
</tr>
<tr>
<td>new JSeparator(...)</td>
<td>a separator</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JComponent</th>
<th>Basic Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>JBox.hbox(...)</td>
<td>a horizontal box</td>
</tr>
<tr>
<td>JBox.vbox(...)</td>
<td>a vertical box</td>
</tr>
<tr>
<td>new JScrollPane(...)</td>
<td>a scroll pane</td>
</tr>
<tr>
<td>new JSplitPanePane(...)</td>
<td>a split pane</td>
</tr>
<tr>
<td>new JTappedPanePane(...)</td>
<td>a tabbed pane</td>
</tr>
</tbody>
</table>

### 3.1 Introducing JBox

We will structure components in a window by placing them in boxes, in very much the same way as LaTeX and Tex places text on a page. We will use the class JBox, which (like JCanvas) is not part of the standard Swing distribution but can be obtained from [http://akira.ruc.dk/~madsr/swing/JBox.java](http://akira.ruc.dk/~madsr/swing/JBox.java). The class can be used instead of the class Box in the Swing package, but it is extended with some features for alignment of the content and it inherits from JComponent so that the usual size setting operations can be used and the boxes can be decorated with borders.

<table>
<thead>
<tr>
<th>JBox</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>static JBox hbox(int align, JComponent c1,...)</td>
<td>Create a horizontal box and add the components c1,... to the box. The components are aligned in the box according to the value align. Possible values are: JBox.BOTTOM, JBox.CENTER and JBox.TOP</td>
</tr>
<tr>
<td>static JBox hbox(JComponent c1,...)</td>
<td>Create a horizontal box and add the components c1,... to the box. Components are aligned along the bottom of the box.</td>
</tr>
</tbody>
</table>
**3.2 Setting size**

In this section we will discuss how you set the size of components to achieve a specific layout. The first thing you need to do when you design a window frame is to figure out which components you want and where they should be placed in relation to each other. Assume some initial/minimum size of the whole frame but allow the window to be displayed. You may use information about the size of the whole screen to set the size of your frame.

**Set the size.** The remaining part of constructing the frame is to place components in boxes a set the size of some of the components and boxes. The easiest way to setting the size of a component is to use the `setSize` method in `JBox`.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>static JBox vbox(int align, JComponent c1,...)</code></td>
<td>Create a vertical box and add the components c1,... to the box. JBox.LEFT, JBox.CENTER and JBox.RIGHT.</td>
</tr>
<tr>
<td><code>static JBox vbox(JComponent c1,...)</code></td>
<td>Create a vertical box and add the components c1,... to the box. Components are aligned along the left edge of the box.</td>
</tr>
<tr>
<td><code>static JComponent hglue()</code></td>
<td>Create a horizontal glue element.</td>
</tr>
<tr>
<td><code>static JComponent hspace(int n)</code></td>
<td>Create a horizontal strut with the given length.</td>
</tr>
<tr>
<td><code>static JComponent vglue()</code></td>
<td>Create a vertical glue element.</td>
</tr>
<tr>
<td><code>static JComponent vspace(int n)</code></td>
<td>Create a vertical strut with the given height.</td>
</tr>
</tbody>
</table>

If you want finer control over the size of components then you may use the following methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>JBox</code></td>
<td></td>
</tr>
<tr>
<td><code>static JComponent setSize(JComponent c,int w,int h)</code></td>
<td>Set the size of the component c to width w and height h.</td>
</tr>
<tr>
<td><code>static JComponent setSize(JComponent c,int w,int h,int dw,int dh)</code></td>
<td>Set the size of the component c to width w and height h, but allow it to stretch or shrink with width dw and height dh.</td>
</tr>
<tr>
<td><code>static int getScreenHeight()</code></td>
<td>return the height of the screen. This is how high the frame may be and still be visible.</td>
</tr>
<tr>
<td><code>static int getScreenWidth()</code></td>
<td>return the width of the screen. This is how wide the frame may be and still be visible.</td>
</tr>
</tbody>
</table>

If you want finer control over the size of components then you may use the following methods

<table>
<thead>
<tr>
<th>Component Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int getHeight()</code></td>
<td>Returns the current height of this component.</td>
</tr>
<tr>
<td><code>int getWidth()</code></td>
<td>Returns the current width of this component.</td>
</tr>
<tr>
<td><code>void setMinimumSize(Dimension minimumSize)</code></td>
<td>Sets the minimum size of this component to a constant value.</td>
</tr>
<tr>
<td><code>void setPreferredSize(Dimension preferredSize)</code></td>
<td>Sets the preferred size of this component.</td>
</tr>
<tr>
<td><code>void setMaximumSize(Dimension maximumSize)</code></td>
<td>Sets the maximum size of this component to a constant value.</td>
</tr>
</tbody>
</table>

**The size of the outer frame.** You can use the methods `getScreenHeight` and `getScreenWidth` to find the maximum size a frame can have and still be visible. You should be aware that there may be som decorations on your screen (System tray) so that the whole screen cannot be used.
Inside the frame you should then place some container - typically a box. You will notice that this box is a bit smaller than the frame since the frame contains a header, borders and possibly a menubar. You should not try to set the size of this container but just let it fill the whole frame. Inside this frame you can then place an number of components. It is a good idea to fix the size of most of the components but let one area be flexible so that the whole frame is filled.

**Example.** In the first example we will achieve a certain design without fixing the size of components. We want to place three labels in a frame: one in the top left hand corner, one centered, and one in the bottom right hand corner.

We will later present a large number of different components one may place in these boxes. In the first few examples we will use labels which are small components with a text string. The simplest version is:

<table>
<thead>
<tr>
<th>JLabel</th>
<th>Create a label with text s</th>
</tr>
</thead>
<tbody>
<tr>
<td>new JLabel(String s)</td>
<td></td>
</tr>
</tbody>
</table>

```java
import javax.swing.*;

public class CheckLabel0{
    public static void main(String args[]){
        JFrame frame=new JFrame("CheckLabel0");
        frame.setSize(300,300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JLabel l1=new JLabel("Label 1");
        JLabel l2=new JLabel("Label 2");
        JLabel l3=new JLabel("Label 3");
        JBox body=
        JBox.vbox(
            JBox.hbox(l1,JBox.hglue()),
            JBox.vglue(),
            JBox.hbox(JBox.hglue(),l2,JBox.hglue()),
            JBox.vglue(),
            JBox.hbox(JBox.hglue(),l3)
        );
        frame.add(body);
        frame.setVisible(true);
    }
}
```

![Image of the frame with three labels](image-url)
Example  By default a label is as small as the text in it. In the next example we will set the size of some of the labels in a frame. We will also use the alignment to control where components are placed in a box. By default components in a horizontal box (an hbox) is placed along the bottom, and the components in a vertical box (a vbox) is placed along the left edge. We will use the setBackground method to give components different color so that we can see the actual size of the components.

```
public class CheckSize1{
    public static void main(String args[]){
        JFrame frame=new JFrame("CheckSize");
        frame.setSize(500,500);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JLabel l1 = new JLabel("Label l1");
        JLabel l2 = new JLabel("Label l2");
        JLabel l3 = new JLabel("Label l3");
        JLabel l4 = new JLabel("Label l4");
        JLabel l5 = new JLabel("Label l5");
        JLabel l6 = new JLabel("Label l6");
        JLabel l7 = new JLabel("Label l7");
        JLabel l8 = new JLabel("Label l8");
        JLabel l9 = new JLabel("Label l9");
        l1.setBackground(Color.lightGray);l1.setOpaque(true);
        l2.setBackground(Color.white); l2.setOpaque(true);
        l3.setBackground(Color.yellow); l3.setOpaque(true);
        l4.setBackground(Color.orange); l4.setOpaque(true);
        l5.setBackground(Color.gray); l5.setOpaque(true);
        l6.setBackground(Color.lightGray);l6.setOpaque(true);
        l7.setBackground(Color.yellow); l7.setOpaque(true);
        l8.setBackground(Color.white); l8.setOpaque(true);
        l9.setBackground(Color.orange); l9.setOpaque(true);
        JBox.setSize(l1,300,150); JBox.setSize(l3,100,100);
        JBox.setSize(l5,300,150); JBox.setSize(l6,100,100);
        JBox.setSize(l7,100,100); JBox.setSize(l9,300,150);
        JBox body= JBox.vbox(JBox.hbox(JBox.CENTER,l1,l2,l3),
                            JBox.hbox(JBox.TOP ,l4,l5,l6),
                            JBox.hbox(JBox.BOTTOM,l7,l8,l9));
        frame.add(body);
        frame.setVisible(true);
    }
}
```

The frame contains a vertical box with three horizontal boxes. The top row is aligned along the center, the middle row along the top and the bottom row along the bottom.

Size behavior may vary.  When you place several components in a frame you will notice that some like to expand while others shrink. By default each component has a minimum, preferred and maximum size. For some they are as small as possible, while others want to grow. To achieve a reasonable appearance you will have to fix sizes of you components and decide which should be allowed to grow in size.

If you have several components of the same type - e.g. several buttons or several text field it is a good idea to give them the same size. It may be an idea to write a small method that constructs buttons and at the same time sets the size of the button. You may also let it set the font, font size, color, borders etc.
3.3 Color and font

<table>
<thead>
<tr>
<th>JComponent</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>void <code>setFont(Font font)</code></td>
<td>Set the font used to display text in the component.</td>
</tr>
<tr>
<td>void <code>setForeground(Color fg)</code></td>
<td>Sets the foreground color of this component.</td>
</tr>
<tr>
<td>void <code>setOpaque(boolean isOpaque)</code></td>
<td>If true the component paints every pixel within its bounds.</td>
</tr>
<tr>
<td>void <code>setBackground(Color bg)</code></td>
<td>Sets the background color of this component.</td>
</tr>
</tbody>
</table>

The `setFont` may be called for a variety of different components - even components with no text. In that case it will just have no effect. Similarly you may call the `setBackground` method, but it will have no effect if the component is not opaque. Some components are by default not opaque which means that only parts of the component is drawn, while the rest is invisible and just shows the background.

3.4 Labels

Labels are small components with some text and/or a small image

<table>
<thead>
<tr>
<th>JLabel</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>new <code>JLabel(IconImage image)</code></td>
<td>Creates a JLabel instance with the specified image.</td>
</tr>
<tr>
<td>new <code>JLabel(String text)</code></td>
<td>Creates a JLabel instance with the specified text.</td>
</tr>
<tr>
<td>new <code>JLabel(String text, IconImage icon, int horizontalAlignment)</code></td>
<td>Creates a JLabel instance with the specified text, image, and horizontal alignment. Alignment should have one of the values in <code>n SwingConstants</code>: RIGHT, LEFT, CENTER</td>
</tr>
</tbody>
</table>
void setHorizontalAlignment(int alignment)
Sets the alignment of the label’s contents along the X axis.
Alignment should have one of the values in SwingConstants: RIGHT, LEFT, CENTER

void setHorizontalTextPosition(int textPosition)
Sets the horizontal position of the label’s text, relative to its image.
Position has one of the values in SwingConstants: RIGHT, LEFT, CENTER

void setText(String text)
Defines the single line of text this component will display.

void setVerticalAlignment(int alignment)
Sets the alignment of the label’s contents along the Y axis.
Alignment has one of the values in SwingConstants: TOP, BOTTOM, CENTER

void setVerticalTextPosition(int textPosition)
Sets the vertical position of the label’s text, relative to its image.
TextPosition has one of the values in SwingConstants: TOP, BOTTOM, CENTER

Both labels and buttons are small components with an icon and/or a text. The methods controlling them are essentially the same. We will show an example of these controls in the next section.

### 3.5 Buttons

A button is an area you can press with the mouse. There may be some text and/or an icon display on the button.

This example shows some buttons with icons and a button with text. There is also a checkbox and a radiobutton (see below).

![Button Example](image)

<table>
<thead>
<tr>
<th>JButton</th>
</tr>
</thead>
<tbody>
<tr>
<td>new JButton(String text)</td>
</tr>
<tr>
<td>new JButton(String text, IconImage icon)</td>
</tr>
<tr>
<td>void setEnabled(boolean b)</td>
</tr>
</tbody>
</table>
void setHorizontalAlignment(int alignment)
Sets the horizontal alignment of the icon and text. Alignment is one of the values from SwingConstants: RIGHT, LEFT, CENTER

void setHorizontalTextPosition(int textPosition)
Sets the horizontal position of the text relative to the icon. TextPosition is one of the values from SwingConstants: RIGHT, LEFT, CENTER

void setIconTextGap(int iconTextGap)
If both the icon and text properties are set, this property defines the space between them. The default value of this property is 4 pixels.

void setText(String text)
Sets the button’s text.

void setVerticalAlignment(int alignment)
Sets the vertical alignment of the icon and text. Alignment is one of the values from SwingConstants: TOP, BOTTOM, CENTER

void setVerticalTextPosition(int textPosition)
Sets the vertical position of the text relative to the icon. TextPosition is one of the values from SwingConstants: TOP, BOTTOM, CENTER

The next example shows how one may place text and icons on a button. We have nine buttons, some with both a text string and an icon.

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

public class CheckButton{
    public static void main(String args[]){
        JFrame frame=new JFrame("CheckButton");
        frame.setSize(500,400);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ImageIcon icon= new ImageIcon("res/cross.gif");
        JButton l1=new JButton("Button 1");
        JButton l2=new JButton(icon);
        JButton l3=new JButton("Button 3",icon);
        JButton l4=new JButton("Button 4",icon);
        l4.setVerticalAlignment(SwingConstants.TOP);
        JButton.setSize(l4,150,150);
        JButton l5=new JButton("Button 5",icon);
        l5.setVerticalAlignment(SwingConstants.CENTER);
        JButton.setSize(l5,150,150);
        JButton l6=new JButton("Button 6",icon);
        l6.setVerticalAlignment(SwingConstants.BOTTOM);
        JButton.setSize(l6,150,150);
        JButton l7=new JButton("Button 7",icon);
        l7.setHorizontalTextPosition(SwingConstants.RIGHT);
        l7.setVerticalTextPosition(SwingConstants.TOP);
        l7.setIconTextGap(50);
        JButton l8=new JButton("Button 7",icon);
        l8.setHorizontalTextPosition(SwingConstants.CENTER);
        l8.setVerticalTextPosition(SwingConstants.BOTTOM);
        l8.setIconTextGap(50);
        JButton l9=new JButton("Button 7",icon);
        l9.setHorizontalTextPosition(SwingConstants.LEFT);
        l9.setIconTextGap(50);
        l9.setVerticalTextPosition(SwingConstants.BOTTOM);
        JButton body=
            JBox.vbox(
                JBox.hbox(l1,l2,l3),
                JBox.hbox(l4,l5,l6),
                JBox.hbox(l7,l8,l9));
```
3.6 CheckBox, Radiobutton and Togglebutton

Checkboxes, radiobuttons and togglebuttons are very similar to buttons. When you press them (select them) they stay selected until you press them again.

<table>
<thead>
<tr>
<th>JCheckBox</th>
</tr>
</thead>
<tbody>
<tr>
<td>new JCheckBox(String text)</td>
</tr>
<tr>
<td>boolean isSelected()</td>
</tr>
<tr>
<td>void setSelected(boolean b)</td>
</tr>
<tr>
<td>Plus methods from JButton</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JRadioButton</th>
</tr>
</thead>
<tbody>
<tr>
<td>new JRadioButton(String text)</td>
</tr>
<tr>
<td>boolean isSelected()</td>
</tr>
<tr>
<td>void setSelected(boolean b)</td>
</tr>
<tr>
<td>Plus methods from JButton</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JToggleButton</th>
</tr>
</thead>
<tbody>
<tr>
<td>new JToggleButton(String text)</td>
</tr>
<tr>
<td>new JToggleButton(String text, IconImage icon)</td>
</tr>
<tr>
<td>boolean isSelected()</td>
</tr>
<tr>
<td>void setSelected(boolean b)</td>
</tr>
<tr>
<td>Plus methods from JButton</td>
</tr>
</tbody>
</table>
A togglebutton looks like an ordinary button. You select it with a single press and a second press will deselect it.

Sometimes you may have several radiobuttons or checkboxes where at most one can be set at a time. You can obtain this effect if you create a ButtonGroup object and register some checkboxes and radiobuttons in this object. The buttongroup object is just a controller that deselects the other buttons when one is selected. You should also add the checkboxes and radiobuttons to the window.

<table>
<thead>
<tr>
<th>ButtonGroup()</th>
<th>Creates a new ButtonGroup.</th>
</tr>
</thead>
<tbody>
<tr>
<td>new ButtonGroup()</td>
<td>Adds the button to the group.</td>
</tr>
</tbody>
</table>

The arguments to add on a ButtonGroup can be any of the selectable buttons. This is JCheckBox, JRadioButton, JToggleButton, JCheckBoxMenuItem and JRadioButtonMenuItem. The last to will be discussed in section 4.6 and we will also discuss events from buttons in section 4.2.

### 3.7 Textfield and textarea

A textfield is a single line area where the user can type in text.

<table>
<thead>
<tr>
<th>JTextField</th>
<th>Constructs a new TextField.</th>
</tr>
</thead>
<tbody>
<tr>
<td>new JTextField()</td>
<td>Constructs a new TextField initialized with the specified text.</td>
</tr>
<tr>
<td>String getText()</td>
<td>Returns the text contained in this TextComponent.</td>
</tr>
<tr>
<td>boolean isEditable()</td>
<td>Returns the boolean indicating whether this TextComponent is editable or not.</td>
</tr>
<tr>
<td>void setEditable(boolean b)</td>
<td>Sets the specified boolean to indicate whether or not this TextComponent should be editable.</td>
</tr>
<tr>
<td>void setHorizontalAlignment(int alignment)</td>
<td>Sets the horizontal alignment of the text. Alignment is one of the values from SwingConstants: RIGHT, LEFT, CENTER</td>
</tr>
<tr>
<td>void setText(String t)</td>
<td>Sets the text of this TextComponent to the specified text.</td>
</tr>
</tbody>
</table>

A textarea is editable area of text. It may consist of several lines.
### JTextArea

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>new JTextArea(String text)</code></td>
<td>Constructs a new TextArea with the specified text displayed.</td>
</tr>
<tr>
<td><code>void append(String str)</code></td>
<td>Appends the given text to the end of the document.</td>
</tr>
<tr>
<td><code>int getLineCount()</code></td>
<td>Determines the number of lines contained in the area.</td>
</tr>
<tr>
<td><code>String getText()</code></td>
<td>Returns the text contained in this TextComponent.</td>
</tr>
<tr>
<td><code>boolean isEditable()</code></td>
<td>Returns the boolean indicating whether this TextComponent is editable or not.</td>
</tr>
<tr>
<td><code>void setEditable(boolean b)</code></td>
<td>Specifies whether or not this TextComponent should be editable.</td>
</tr>
<tr>
<td><code>void setHorizontalAlignment(int alignment)</code></td>
<td>Sets the horizontal alignment of the text. Alignment is one of the values from SwingConstants: RIGHT, LEFT, CENTER</td>
</tr>
<tr>
<td><code>void setLineWrap(boolean wrap)</code></td>
<td>Sets the line-wrapping policy of the text area. If set to true the lines will be wrapped if they are too long to fit within the allocated width.</td>
</tr>
<tr>
<td><code>void setText(String t)</code></td>
<td>Sets the text of this TextComponent to the specified text.</td>
</tr>
<tr>
<td><code>void setWrapStyleWord(boolean word)</code></td>
<td>Sets the style of wrapping used if the text area is wrapping lines. If set to true the lines will be wrapped at word boundaries (whitespace) if they are too long to fit within the allocated width.</td>
</tr>
</tbody>
</table>

A formatted text field is almost like a textfield but internally the values are stored as objects from other classes than String. You can use it when you want a date or a number and by default the textfield will not accept which is not correct. If you try to write some illegal text then it will just be ignored.

### JFormattedTextField

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>new JFormattedTextField(Object val)</code></td>
<td>Constructs a new Formatted TextField. The value can be an object from the classes Integer, Float, Double, or Date</td>
</tr>
<tr>
<td><code>Object getValue()</code></td>
<td>Returns the object contained in this TextComponent.</td>
</tr>
<tr>
<td><code>void setValue(Object value)</code></td>
<td>Set the value stored in this component.</td>
</tr>
</tbody>
</table>

**Example**  This example shows some textfields and also shows that you can control the alignment of the text in a field

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

public class CheckTextField1{
    public static void main(String args[]){
        JFrame frame=new JFrame("CheckTextField");
```
3.8 Slider, Spinner and ProgressBar

A Slider can be used to enter numeric values in bounded area. The user can drag a knob to specify a value.

| JSlider |
|-----------------|---------------------------------|
| new JSlider()   | Creates a horizontal slider with the range 0 to 100 and an initial value of 50. |
| new JSlider(int orientation, int min, int max, int value) | Creates a slider with the specified orientation and the specified minimum, maximum, and initial values. Orientation should have the value SwingConstants.VERTICAL or SwingConstants.HORIZONTAL |
### JSpinner

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void setExtent(int extent)</td>
<td>Sets the size of the range &quot;covered&quot; by the knob.</td>
</tr>
<tr>
<td>void setInverted(boolean b)</td>
<td>Specify true to reverse the value-range shown for the slider and false to put the value range in the normal order.</td>
</tr>
<tr>
<td>void setMajorTickSpacing(int n)</td>
<td>This method sets the major tick spacing.</td>
</tr>
<tr>
<td>void setMinorTickSpacing(int n)</td>
<td>This method sets the minor tick spacing.</td>
</tr>
<tr>
<td>void setPaintLabels(boolean b)</td>
<td>Determines whether labels are painted on the slider.</td>
</tr>
<tr>
<td>void setPaintTicks(boolean b)</td>
<td>Determines whether tick marks are painted on the slider.</td>
</tr>
<tr>
<td>void setPaintTrack(boolean b)</td>
<td>Determines whether the track is painted on the slider.</td>
</tr>
<tr>
<td>void setSnapToTicks(boolean b)</td>
<td>Specifying true makes the knob (and the data value it represents) resolve to the closest tick mark next to where the user positioned the knob.</td>
</tr>
<tr>
<td>void setValue(int n)</td>
<td>Sets the slider's current value.</td>
</tr>
<tr>
<td>int getValue()</td>
<td>Gets the slider's current value.</td>
</tr>
</tbody>
</table>

A spinner is a small field where the user can select a value by stepping through a range of values. A spinner will just display one value at a time. Values can be numbers, dates or be selected from a list of strings.

### SpinnerModel

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>new JSpinner()</td>
<td>Constructs a spinner with an Integer SpinnerNumberModel with initial value 0 and no minimum or maximum limits.</td>
</tr>
<tr>
<td>new JSpinner(SpinnerModel model)</td>
<td>Constructs a complete spinner with pair of next/previous buttons and an editor for the SpinnerModel.</td>
</tr>
<tr>
<td>Object getValue()</td>
<td>Returns the current value of the model, typically this value is displayed by the editor.</td>
</tr>
<tr>
<td>void setValue(Object value)</td>
<td>Changes current value of the model, typically this value is displayed by the editor.</td>
</tr>
</tbody>
</table>

new SpinnerDateModel() | Constructs a SpinnerDateModel whose initial value is the current date, calendarField is equal to Calendar.DAY_OF_MONTH, and for which there are no start/end limits. |
new SpinnerListModel(Object[] values) | Constructs a SpinnerModel whose sequence of values is defined by the specified array. |
new SpinnerNumberModel(double value, double minimum, double maximum, double stepSize) | Constructs a SpinnerNumberModel with the specified value, minimum/maximum bounds, and stepSize. |
new SpinnerNumberModel(int value, int minimum, int maximum, int stepSize) | Constructs a SpinnerNumberModel with the specified value, minimum/maximum bounds, and stepSize. |
The progressbar A progressbar may be used when you will tell the user how long it takes before an initialization is finished. You need to update the value yourself and this would normally require the use of a timer and events. We will here just show an example where nothing really happens while we wait.

<table>
<thead>
<tr>
<th>JProgressBar</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>new JProgressBar()</strong></td>
<td>create a progress bar</td>
</tr>
<tr>
<td><strong>void setValue(int n)</strong></td>
<td>set the value in the progress bar. The value should be between 0 and 100.</td>
</tr>
<tr>
<td><strong>void setStringPainted(boolean b)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>void setBorderPainted(boolean b)</strong></td>
<td></td>
</tr>
</tbody>
</table>
3.9 ComboBox and List

A ComboBox allows the use to select a value from a small drop-down menu. With a List all options are displayed and you may select several of the possible values.

<table>
<thead>
<tr>
<th>JComboBox</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>new JComboBox(Object[] items)</td>
<td>Creates a JComboBox that contains the elements in the specified array.</td>
</tr>
<tr>
<td>void addItem(Object anObject)</td>
<td>Adds an item to the item list.</td>
</tr>
<tr>
<td>int getItemCount()</td>
<td>Returns the number of items in the list.</td>
</tr>
<tr>
<td>Object getItemAt(int index)</td>
<td>Returns the list item at the specified index.</td>
</tr>
<tr>
<td>Object getSelectedItem()</td>
<td>Returns the current selected item.</td>
</tr>
<tr>
<td>void setEditable(boolean aFlag)</td>
<td>Specify whether the JComboBox field is editable.</td>
</tr>
<tr>
<td>void setEnabled(boolean b)</td>
<td>Enables the comboBox so that items can be selected.</td>
</tr>
<tr>
<td>void setMaximumRowCount(int count)</td>
<td>Sets the maximum number of rows the JComboBox displays.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JList</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>new JList(Object[] listData)</td>
<td>Constructs a JList that displays the elements in the specified array.</td>
</tr>
<tr>
<td>Object getSelectedValue()</td>
<td>Returns the first selected value.</td>
</tr>
<tr>
<td>boolean getSelectedIndex(int index)</td>
<td>Returns true if index is selected.</td>
</tr>
</tbody>
</table>
3.10 SplitPane and JScrollPane

A SplitPane allows you to display two components with a draggable border between them. A scrollpane you to put a big component into a small container so that only a part of it is displayed.

<table>
<thead>
<tr>
<th>JSplitPane</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>new JSplitPane(int newOrientation, JComponent newLeftComponent, JComponent newRightComponent)</td>
<td>Creates a new JSplitPane with the specified orientation and with the specified components that do not do continuous redrawing. The option newOrientation is one of the constants in JSplitPane: HORIZONTAL_SPLIT or VERTICAL_SPLIT.</td>
</tr>
</tbody>
</table>
int getDividerLocation()  
Returns the last value passed to setDividerLocation.

void setDividerLocation(int location)  
Sets the location of the divider. The location is measured in pixels from the top or left.

void setDividerSize(int newSize)  
Sets the size of the divider.

void setResizeWeight(double value)  
Specifies how to distribute extra space when the size of the split pane changes.

JScrollPane

new JScrollPane(JComponent view)  
Creates a JScrollPane that displays the contents of the specified component, where both horizontal and vertical scrollbars appear whenever the component’s contents are larger than the view.

void setHorizontalScrollBarPolicy(int policy)  
Determines when the horizontal scrollbar appears in the scrollpane. The options are one of the constants in ScrollPaneConstants: HORIZONTAL_SCROLLBAR_AS_NEEDED, HORIZONTAL_SCROLLBAR_NEVER, HORIZONTAL_SCROLLBAR_ALWAYS

void setVerticalScrollBarPolicy(int policy)  
Determines when the vertical scrollbar appears in the scrollpane. The options are one of the constants in ScrollPaneConstants: VERTICAL_SCROLLBAR_AS_NEEDED, VERTICAL_SCROLLBAR_NEVER, VERTICAL_SCROLLBAR_ALWAYS

3.11 Borders

You can decorate your components and containers with various types of borders. In principle you should be able to put borders around any component in swing. If the effect does not look right, you may have to put the component into a box and put a border around that box.

JComponent

void setBorder(Border border)  
Sets the border of this component.

BorderFactory

static Border createCompoundBorder(Border outsideBorder, Border insideBorder)  
Creates a compound border specifying the border objects to use for the outside and inside edges.

static Border createEmptyBorder(int top, int left, int bottom, int right)  
Creates an empty border that takes up space but which does no drawing, specifying the width of the top, left, bottom, and right sides.

static Border createEtchedBorder()  
Creates a border with an “etched” look using the component’s current background color for highlighting and shading.

static Border createLineBorder(Color color, int thickness)  
Creates a line border with the specified color and width.

static Border createMatteBorder(int top, int left, int bottom, int right, Color color)  
Creates a border where the width can be different on the four sides.
static Border
createLoweredBevelBorder()

Creates a border with a lowered beveled edge, using brighter shades of the component’s current background color for highlighting, and darker shading for shadows.

static Border
createRaisedBevelBorder()

Creates a border with a raised beveled edge, using brighter shades of the component’s current background color for highlighting, and darker shading for shadows.

static Border
createTitledBorder(String title)

Creates a new title border specifying the text of the title, using the default border (etched), using the default text position (sitting on the top line) and default justification (leading) and using the default font and text color determined by the current look and feel.

static Border
createTitledBorder(Border border, String title, int titleJustification, int titlePosition, Font titleFont, Color titleColor)

Adds a title to an existing border, specifying the text of the title along with its positioning, font, and color. titleJustification is one of the values LEFT, RIGHT, or CENTER from javax.swing.Border.TitledBorder. titleJustification is one of the values ABOVE_TOP, TOP, BELOW_TOP, ABOVE_BOTTOM, BOTTOM, BELOW_BOTTOM, from the same class.

CheckBorders1

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

public class CheckBorders1{
    public static void main(String args[]){
        JFrame frame=new JFrame("CheckBorders1");
        frame.setSize(600,600);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JTextArea t1=new JTextArea("Text 1");
        JTextArea t2=new JTextArea("Text 2");
        JTextArea t3=new JTextArea("Text 3");
        JTextArea t4=new JTextArea("Text 4");
        JTextArea t5=new JTextArea("Text 5");
        JTextArea t6=new JTextArea("Text 6");
        JTextArea t7=new JTextArea("Text 7");
        JTextArea t8=new JTextArea("Text 8");
        JTextArea t9=new JTextArea("Text 9");
        JTextArea t10=new JTextArea("Text 10");

        JBox body=JBox.hbox(JBox.hspace(50),
                         JBox.vbox(JBox.vspace(50),t1,JBox.vspace(50),t2,JBox.vspace(50),
                                   t3,JBox.vspace(50),t4,JBox.vspace(50),t5,JBox.vspace(50),
                                   JBox.hspace(50),
                                   JBox.vbox(JBox.vspace(50),t6,JBox.vspace(50),t7,JBox.vspace(50),
                                             t8,JBox.vspace(50),t9,JBox.vspace(50),t10,JBox.vspace(50),
                                             JBox.hspace(50));

        t1.setBorder(BorderFactory.createEmptyBorder(10,5,10,5));
        t2.setBorder(BorderFactory.createEtchedBorder());
        t3.setBorder(BorderFactory.createLineBorder(Color.lightGray,5));
        t4.setBorder(BorderFactory.createMatteBorder(10,5,10,5,Color.lightGray));
        t5.setBorder(BorderFactory.createLoweredBevelBorder());
        t6.setBorder(BorderFactory.createRaisedBevelBorder());
        t7.setBorder(BorderFactory.createTitledBorder("title 7"));
    }
}
```
3.12 Other controls for Swing components

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>JComponent requestFocusInWindow()</code></td>
<td>Requests that this Component gets the input focus. You may want a specific text field or the canvas be the component that by default receive keyboard input.</td>
</tr>
<tr>
<td><code>void setToolTipText(String text)</code></td>
<td>Registers the text to display in a tool tip. Tool Tips are small text boxes that pops up when the mouse passes over a component.</td>
</tr>
<tr>
<td><code>void setCursor(Cursor c)</code></td>
<td>specify a cursor for a component. Possible values of cursors are listed below.</td>
</tr>
</tbody>
</table>
When you specify a cursor for a component then the possible standard values are:

- `Cursor.getPredefinedCursor(Cursor.MOVE_CURSOR)`
- `Cursor.getPredefinedCursor(Cursor.TEXT_CURSOR)`
- `Cursor.getPredefinedCursor(Cursor.CROSSHAIR_CURSOR)`
- `Cursor.getPredefinedCursor(Cursor.DEFAULT_CURSOR)`
- `Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR)`

### 3.13 Adding and removing content of a box

Occasionally you may want to change the content in a window. This can be done in three easy steps. Make sure that the content you want to change is in a box somewhere in the frame. Now you should:

- remove the content of the box by calling `removeAll`.
- ask the frame to layout its content again by calling `validate`.
- repaint the frame by calling `repaint`.

<table>
<thead>
<tr>
<th>JBox</th>
<th>JFrame</th>
</tr>
</thead>
<tbody>
<tr>
<td>void add(JComponent c)</td>
<td>void validate()</td>
</tr>
<tr>
<td>void removeAll()</td>
<td>void repaint()</td>
</tr>
<tr>
<td>remove all components in a box</td>
<td>Check the layout of components in the window</td>
</tr>
</tbody>
</table>

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

public class CheckAddRemove{
    public static void main(String args[]){
        JFrame frame=new JFrame("CheckAddRemove");
        frame.setSize(600,600);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JButton button1=new JButton("Start");
        button1.setFont(new Font("Arial",Font.BOLD,40));
        JEventQueue events=new JEventQueue();
        events.listenTo(button1,"Button1");
        JBox body = JBox.vbox(button1);
        frame.add(JBox.vbox(JBox.hglue(),body,JBox.hglue()),JBox.vglue());
        frame.setVisible(true);
        events.waitEvent();//wait for button press
        body.removeAll();
        JRadioButton button2=new JRadioButton("Button2");
        events.listenTo(button2,"Button2");
        body.add(button2);
        frame.validate();
        frame.repaint();
    }
}
```
4 Types of events

This part of the notes on graphics programming in Java covers event handling. When the user interacts with a program through a graphical user interface the underlying system will generate so-called events, and the program must then “listen” to these events. We will show how to make graphics programs react to events from the user interface. We will not describe the usual event-driven paradigm, normally used in Swing programs. Instead we use a simple event queue where the graphics system pushes events and the program fetches events.

The central feature of a graphics program is that the user may interact with the program in a number of different ways. One may click with mouse on a canvas, push some buttons and write text in fields. These interactions are called events and the challenge is now how to receive these events in an orderly way in the program.

The standard way to receive events in swing is to write some methods that should be called when events occur. We then inform the components which method it should call, and we then add suitable code to process the event in the method. The complication, however, is that these methods are placed in anonymous inner classes and that is not a favourite subject in an introductory programming course.

In these notes we will use an alternative event handling strategy. It is based on a central event loop where the programmer fetches events from an event queue. With this approach it is possible to write event based programs without a lot of experience in object oriented programming.

In these notes we will cover the main types of events that may occur in a graphics based program.

- Events from pressing a button. This is probably the simplest type of event. When a button is pressed it generates an event and the program should perform a suitable action.

- Events from state based components. By this we mean events from components that have a value and where the user can change this value. This includes check boxes, radio buttons, spinners, sliders, combo boxes and lists. When the value is changed the system generates an event (or several events).

- Events from text component.

- Events from a canvas. In a canvas you may use the mouse to click and use the keyboard to control what the program does.

- Timers. The program may specify that certain things should happen at regular intervals. This can be achieved by starting a timer that generates special events at regular intervals.
• Menus. Menus in the top of windows are really just buttons you can press. The need special treatment because you can set up mnemonics (e.g. Alt-f for the file menu) and accelerator keys (e.g. Control-s to save).

• Window events. The user may resize the window or close the window.

4.1 Introducing JEventQueue

The JEventQueue is not part of the standard Swing library but can be obtained from http://akira.ruc.dk/~madsr/swing/JEventQueue.java. A JEventQueue object can be set to listen to events from swing components. The typical program will then use a central event loop where you process one event at a time. The components you listen to are given names to identify them and when you receive an event you can then ask which component generated the event.

<table>
<thead>
<tr>
<th>JEventQueue</th>
</tr>
</thead>
<tbody>
<tr>
<td>new JEventQueue()</td>
</tr>
<tr>
<td>void listenTo(JComponent c, String name)</td>
</tr>
<tr>
<td>EventObject waitEvent()</td>
</tr>
<tr>
<td>String getName(EventObject e)</td>
</tr>
<tr>
<td>boolean hasEvent()</td>
</tr>
</tbody>
</table>

The class EventObject is part of the java.util package and thus not specific to swing programs. Almost all events that occur in swing are generated as EventObjects but normally as belonging to more specific sub-classes. As a programmer you will not have to know about the inheritance hierarchy. You need to know which component generated the event and what type of event it was.

The typical program will use the structure outline below. You create the eventqueue and set it to listen to a number of components. At the same time you register a name you want to use to identify the component. In the event loop you wait for an event, fetch the name of the component that generated it, and then you perform the appropriate action.

```java
JEventQueue events = new JEventQueue();
events.listenTo(button1, "button1");
events.listenTo(button2, "button2");
...
while(true){
    EventObject event = events.waitEvent();
    String name = events.getName(event);
    if(name.equals("button1")){
        ...
    }else if(name.equals("button2")){
        ...
    }else ...
}
```
Some components may generate several different types of events and you may want to refine the program to have different actions for different types of events from a component. We will show some examples of this later.

Example: A temperature converter

Let us examine a slightly larger example, which also illustrates how you design a user interface for an application. The goal is to construct a small program that makes it possible to convert Celsius to Fahrenheit and vice versa.

The first step is to make a mock-up of the application. This may just be a small drawing of what the window should look like.

```
37
← Celsius
98.6
← Fahrenheit
```

The window should have a field for Celsius and a field for Fahrenheit. If you write a temperature in the Celsius box and press the Fahrenheit button then the program should convert the temperature to Fahrenheit and write it in the Fahrenheit box. You may also write a number in the Fahrenheit field and get it converted to a Celsius degree.

The next step is to write a small program with these components in a window.

```java
import javax.swing.*;

public class CheckListener1{
   public static void main(String args[]){
      JFrame frame=new JFrame("CheckListener1");
      frame.setSize(600,400);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      JButton button1=new JButton("<<< Celsius");
      JButton button2=new JButton("<<< Fahrenheit");
      JTextField field1=new JTextField(""褰");
      JTextField field2=new JTextField(""褰");

      JBox body=
         JBox.vbox(
         JBox.hbox(field1,button1),
         JBox.hbox(field2,button2)
         );
      frame.add(body);
      frame.setVisible(true);
   }
}
```
The result is not very satisfactory. The problem is that the default sizes of the buttons and textfields do not fit our requirements. By default a button is as small as the text in it and a textfield may grow as large as the space allows. The result is not very attractive looking. If you have several components of the same type then it is often a good idea that they have the same size. It can be an idea to write methods that construct the components and at the same time control their appearance. In this way it is easier to adjust it later if the design was not quite right.

In our little program we will fix the size of the buttons and textfield and put some glue around it so that the fields and buttons a centered in the window. We will also make the font size a bit bigger and place the text on the buttons to the left.

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

public class CheckListener2{
    static JButton myButton(String s){
        JButton c=new JButton(s);
        c.setSize(c,300,40);
        c.setFont(new Font("Arial",Font.BOLD,24));
        c.setHorizontalAlignment(SwingConstants.LEFT);
        return c;
    }
    static JTextField myTextField(String s){
        JTextField c=new JTextField(s);
        c.setSize(c,200,40);
        c.setFont(new Font("Arial",Font.BOLD,24));
        return c;
    }
    public static void main(String args[]){
        JFrame frame=new JFrame("CheckListener2");
        frame.setSize(600,400);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JButton button1=myButton("<<< Celsius");
        JButton button2=myButton("<<< Fahrenheit");
        JTextField field1=myTextField(""");
        JTextField field2=myTextField("");
        VBox body=
            VBox.vbox(
                VGlue(),
                HBox(JBox.hglue(),field1,button1,JBox.hglue()),
                HBox(JBox.hglue(),field2,button2,JBox.hglue()),
                VGlue()
            );
        frame.add(body);
        frame.setVisible(true);
    }
}
```
That was better. Now we have the user interface, but it does not do anything. You can write text in the fields and press the buttons but it will not make any conversions. The conversions are placed in the event loop. It uses the `getText` and `setText` methods on the textfield to fetch values and set the converted values. The code for the graphical user interface is not changed. We have only added the event loop at the end of the program.

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

public class CheckListener3{
    static JButton myButton(String s){
        JButton c=new JButton(s);
        c.setSize(c,300,40);
        c.setFont(new Font("Arial",Font.BOLD,24));
        c.setHorizontalAlignment(SwingConstants.LEFT);
        return c;
    }
    static JTextField myTextField(String s){
        JTextField c=new JTextField(s);
        c.setSize(c,200,40);
        c.setFont(new Font("Arial",Font.BOLD,24));
        return c;
    }
    static String format(double d){
        int x=(int) (d*10);
        return ""+(x/10)+"."+(x%10);
    }
    public static void main(String args[]){
        JFrame frame=new JFrame("CheckListener3");
        frame.setSize(600,600);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JButton button1=myButton("<<< Celsius");
        JButton button2=myButton("<<< Fahrenheit");
        JTextField field1=myTextField("");
        JTextField field2=myTextField("");
        JBox body=JBox.vbox(JBox.vglue(),JBox.hbox(JBox.hglue(),field1,button1,JBox.hglue()),JBox.hbox(JBox.hglue(),field2,button2,JBox.hglue()),JBox.vglue());
    }
}
```
The program is almost finished. In a full version we should also check for errors in the input.

4.2 Events from components with state

The next example shows how to listen to and obtain the state from some other Swing components. In this example we construct a number of different components and place them in a long vbox. The eventqueue listens to the components and when events occur we print out the state of the component.

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

public class CheckState{
    public static void main(String args[]){
        JFrame frame=new JFrame("CheckState");
        frame.setSize(600,400);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JRadioButton button1=new JRadioButton("Yes");
        JRadioButton button2=new JRadioButton("No");
        JCheckBox button3=new JCheckBox("CheckBox");
        JToggleButton button4=new JToggleButton("ToggleButton");
        JSpinner spinner=new JSpinner();
        JSlider slider=new JSlider();
        ButtonGroup group=new ButtonGroup();
        group.add(button1); group.add(button2); //only one can be set at a time.
        String[] data1={"combo1","combo2","combo3","combo4"};
        String[] data2={"list1","list2","list3","list4"};
        JComboBox combo=new JComboBox(data1);
        JList list=new JList(data2);
        JBox body=
            JBox.vbox(
                button1,button2,button3,button4,
                spinner,slider, combo, list
            );
        frame.add(body);
        frame.setVisible(true);
    }
}
```
JEventQueue events=new JEventQueue();
events.listenTo(button1,"button1");
events.listenTo(button2,"button2");
events.listenTo(button3,"button3");
events.listenTo(button4,"button4");
events.listenTo(spinner,"spinner");
events.listenTo(slider,"slider");
events.listenTo(combo,"combo");
events.listenTo(list,"list");

while(true){
    EventObject event=events.waitEvent();
    String name= events.getName(event);
    if(name.equals("button1"))
        System.out.println("Button1: "+button1.isSelected());
    else if(name.equals("button2"))
        System.out.println("Button2: "+button2.isSelected());
    else if(name.equals("button3"))
        System.out.println("Button3: "+button3.isSelected());
    else if(name.equals("button4"))
        System.out.println("Button4: "+button4.isSelected());
    else if(name.equals("spinner"))
        System.out.println("Spinner: "+spinner.getValue());
    else if(name.equals("slider"))
        System.out.println("Slider: "+slider.getValue());
    else if(name.equals("combo"))
        System.out.println("Combo: "+combo.getSelectedItem());
    else if(name.equals("list"))
        System.out.println("List: "+list.getSelectedValue());
    else
        System.out.println("Error: unknown event :"+name);
}

This program will generate the following user interface. It would probably need some work to give it an acceptable design. You may notice that by default the spinner and combobox may grow in size and the slider can grow in length.

4.3 Text components

The problem with listening to events from text components is that they generate a large number of possible events. If you type a character in a text field you get an event when the key is pressed, an event when the key is being typed, an event when the text is changed and finally an event when the key is being released. In a textfield, pressing the return key will generate an action
event. A newline character cannot be typed in a text field and it will not change the text, but it will still generate an event.

You will, in many situation, not listen to events from text fields. It may be sufficient to let the user type in a text in the field and then press a button when the text has been written. This is exactly what we did with the temperature converter. If you do listen to events from a text component you may just ignore all events except for keys being typed and action keys being pressed. Action keys are all the keys on a keyboard that do not correspond to a character. This includes the arrows for cursor movements, Home, Page Up, function keys etc.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>isKeyEvent(EventObject e)</code></td>
<td>Returns true if the event is a KeyEvent</td>
</tr>
<tr>
<td><code>isKeyTyped(EventObject e)</code></td>
<td>Returns true if the event is a character being typed.</td>
</tr>
<tr>
<td><code>isKeyPressed(EventObject e)</code></td>
<td>Returns true if the event is a key being pressed.</td>
</tr>
<tr>
<td><code>isKeyReleased(EventObject e)</code></td>
<td>Returns true if the event is a key being released.</td>
</tr>
<tr>
<td><code>isActionPerformed(EventObject e)</code></td>
<td>Returns true if it is an ActionPerformed event.</td>
</tr>
<tr>
<td><code>isActionKey(EventObject e)</code></td>
<td>Returns true if the event is pressing an action key. Action keys are keys like Home, End, Page Up, etc.</td>
</tr>
<tr>
<td><code>isActionEvent(EventObject e)</code></td>
<td>Returns true if the event is an ActionEvent</td>
</tr>
<tr>
<td><code>isDocumentEvent(EventObject e)</code></td>
<td>Returns true if the event is a DocumentEvent</td>
</tr>
</tbody>
</table>

In the example below we listen to events from a text field and a text area. Keys being typed and action keys being pressed is then recorded in a separate text area.

```java
public class CheckTextComp{
    public static void main(String args[]){
        JFrame frame=new JFrame("CheckTextComp");
        frame.setSize(600,600);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JTextField c1=new JTextField();
        JTextArea c2=new JTextArea();
        JTextArea c3=new JTextArea();
        JBox body=JBox.vbox(
            JBox.setSize(c1,600,30,600,0),JBox.setSize(c2,600,200,600,0),
            new JLabel("KeyEvents:"),new JScrollPane(c3));
        body.setFont(new Font("Arial",Font.PLAIN,18));
        c3.setFont(new Font("Arial",Font.PLAIN,18));
        frame.add(body);
    }
}
```
22    frame.setVisible(true);
23    JEventQueue events=new JEventQueue();
24    events.listenTo(c1,"C1");
25    events.listenTo(c2,"C2");
26    for(;;){
27        EventObject event=events.waitEvent();
28        String name=events.getName(event);
29        if(events.isKeyTyped(event))
30            c3.setText(c3.getText() + name +": Typed "
31            + events.getKeyChar(event));
32        if(events.isActionKey(event))
33            c3.setText(c3.getText() + name +": Action 
34            + events.getKeyText(event));
35    }
36    }
37
4.4 Events from a canvas

On a canvas you will typically listen to keyboard events and events from the mouse. The
keyboard events behave as with text components. You will probably ignore most of the keyboard
events and only take actions when characters are being typed and action keys are being pressed.

As with keyboard events there is a number of methods to make it easier to access information
about mouse events.

<table>
<thead>
<tr>
<th>JEventQueue</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static boolean isMouseEvent(EventObject e)</td>
<td>Returns true if it is a mouse event</td>
</tr>
<tr>
<td>static boolean isMousePressed(EventObject e)</td>
<td>Returns true if it is an event from pressing a mouse button</td>
</tr>
<tr>
<td>static boolean isMouseClicked(EventObject e)</td>
<td>Returns true if it is an event from clicking a mouse button</td>
</tr>
</tbody>
</table>
static boolean isMouseReleased(EventObject e) Returns true if it is an event from releasing a mouse button
static int getX(EventObject e) Return the x coordinate of the mouse location
static int getY(EventObject e) Return the y coordinate of the mouse location
static int getMouseButton(EventObject e) Return information about which mouse button was pressed
static int getMouseClickCount(EventObject e) Return the number of clicks on the mouse button.

The following example shows how we may listen to mouse events from a canvas. The program will draw a circle whenever you click the mouse on the canvas. If you drag the mouse, i.e. press the mouse button down, move the mouse and the release it, then the program will draw a line from where you pressed the mouse button to where you released it.

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

public class CheckMouse{
    public static void main(String args[]){
        JFrame frame=new JFrame("CheckMouse");
        frame.setSize(600,600);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JCanvas canvas=new JCanvas();
        frame.add(canvas);
        frame.setVisible(true);
        JEventQueue events=new JEventQueue();
        events.listenTo(canvas,"canvas");
        int x0=0,y0=0;
        while(true){
            EventObject event=events.waitEvent();
            if(events.isMouseEvent(event)){
                int x=events.getX(event);
                int y=events.getY(event);
                if(events.isMousePressed(event)){x0=x;y0=y;}
                if(events.isMouseClicked(event))
                    canvas.drawOval(x0-5,y0-5,10,10);
                if(events.isMouseReleased(event))
                    canvas.drawLine(x0,y0,x,y);
            }
        }
    }
}
```

4.5 Timers

Timers use the internal clock to fire events at regular intervals. Timers are not specific to graphical applications and the example below will start a couple of timers and just register their events.

You may use the following operations to start and stop timers.
**JEventQueue**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void startTimer(int interval, String name)</code></td>
<td>Start the timer. The timer will fire with interval milliseconds intervals. Register the name with the timer in the eventqueue.</td>
</tr>
<tr>
<td><code>void stopTimer(String name)</code></td>
<td>Stop the timer and remove any events in the queue from the timer.</td>
</tr>
<tr>
<td><code>void sleep(int ms)</code></td>
<td>Sleep for ms milliseconds</td>
</tr>
</tbody>
</table>

This example starts two timers which fire events with 1 second and 2.5 second intervals. The second timer is stopped when it has fired two events and a new timer is started. The third timer fire events with 1.5 second intervals.

```java
import java.util.*;

public class CheckTimers{

    public static void main(String args[]){

        JEventQueue events=new JEventQueue();

        events.startTimer(1000,"timer1");

        events.startTimer(2500,"timer2");

        int t=0;

        while(t<10){
            EventObject event=events.waitEvent();

            String name=events.getName(event);

            System.out.println(t+":"+name);

            if(name.equals("timer1"))t++;

            if(name.equals("timer2")&&t>3){
                events.stopTimer("timer2");
                events.startTimer(1500,"timer3");
            }
        }

        System.exit(0);
    }
}
```

The output from the program is as follows

```
0:timer1
1:timer1
2:timer2
2:timer1
3:timer1
4:timer2
4:timer1
5:timer1
6:timer3
6:timer1
7:timer3
7:timer1
8:timer1
9:timer3
9:timer1
```

The example shows that timer2 has stopped at time 4 (when t > 3) and that timer3 started at that time.
4.6 Menus

Your window may include a menu bar near the top of the window. A menu bar contains a number of menus, and each menu can contain menu items and other menus. The `JEventQueue` class contains some utility methods to make the construction of a menu bar easier. The methods just construct the appropriate object, asks the event queue to listen to events from it and return the object. With these methods the menu bar can be constructed as one big expression.

When we create the menus we may register some short hand code to access the menus. We may specify mnemonic characters with menus and menu items. In the example below the file menu has 'F' as mnemonic character. If you press 'Alt' and 'F' down simultaneously the file menu is opened. You can then select one of the menu items in several different ways. You can click on it with the mouse, you can use the mnemonic character (e.g. 'N' for the 'New' item), or you can move down the list with the down arrow key and select it with 'enter'.

Menu items can also be accessed with special key strokes. We may specify that the 'exit' menu item in the 'file' menu can be accessed using the 'Control-q' key stroke.

<table>
<thead>
<tr>
<th><strong>JEventQueue</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>JMenuBar jmenubar(JMenu... j)</code></td>
<td>Construct a menubar with the menu list as content.</td>
</tr>
<tr>
<td><code>JMenuBar jmenubar(Font f,JMenu... j)</code></td>
<td>Construct a menubar with the menu list as content. Use the font f for all menus and menu items.</td>
</tr>
<tr>
<td><code>JMenu jmenu(String s,JComponent... j)</code></td>
<td>Construct a menu with the component list as content.</td>
</tr>
<tr>
<td><code>JMenu jmenu(String s,char c, JComponent... j)</code></td>
<td>Construct a menu with the component list as content. Use the character c as mnemonic.</td>
</tr>
<tr>
<td><code>JMenuItem jmenuItem(String s)</code></td>
<td>Construct a menu item and set the event queue to listen to it.</td>
</tr>
<tr>
<td><code>JMenuItem jmenuItem(String s,char c)</code></td>
<td>Construct a menu item and set the event queue to listen to it. Use the character c as mnemonic.</td>
</tr>
<tr>
<td><code>JMenuItem jmenuItem(String s,char c, KeyStroke k)</code></td>
<td>Construct a menu item and set the event queue to listen to it. Use the character c as mnemonic and the keystroke k as accelerator key.</td>
</tr>
<tr>
<td><code>JMenuItem jcheckboxmenuitem(String s)</code></td>
<td>Construct a checkbox menu item and set the event queue to listen to it.</td>
</tr>
<tr>
<td><code>JMenuItem jradiobuttonmenuitem(String s)</code></td>
<td>Construct a radiobutton menu item and set the event queue to listen to it.</td>
</tr>
<tr>
<td><code>static KeyStroke control(char c)</code></td>
<td>Construct the control-c keystroke. The keystroke may be used as accelerator key for a menu item</td>
</tr>
<tr>
<td><code>static KeyStroke controlShift(char c)</code></td>
<td>Construct the control-shift-c keystroke. The keystroke may be used as accelerator key for a menu item</td>
</tr>
<tr>
<td><code>new JSeparator()</code></td>
<td>Construct a horizontal line that may be inserted in a menu</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>JFrame</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void setJMenuBar(JMenuBar menubar)</code></td>
<td>Add the menubar to the frame.</td>
</tr>
</tbody>
</table>
import java.awt.*;
import javax.swing.*;
import java.util.*;

public class CheckMenus{
    public static void main(String args[]){
        JFrame frame=new JFrame("CheckMenus");
        frame.setSize(600,250);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JTextArea area=new JTextArea();
        frame.add(area);
        JEventQueue events=new JEventQueue();
        frame.setJMenuBar(
            events.jmenubar(new Font("Arial",Font.PLAIN,18),
                events.jmenu("File",'F',
                    events.jmenuitem("New","n",events.control('N')),
                    events.jmenuitem("Open","o",events.control('O')),
                    events.jmenuitem("Save","s",events.control('S')),
                    events.jmenuitem("Save as","a",events.control('A')),
                    events.jmenuitem("Exit","x",events.control('Q'))),
                events.jmenu("Edit","E",
                    events.jmenuitem("edit 1"),
                    events.jmenuitem("edit 2"),
                    events.jmenuitem("edit 3"),
                    events.jradiobuttonmenuitem("edit 4"),
                    new JSeparator(),
                    events.jmenuitem("edit 5")),
                events.jmenu("View",'V',
                    events.jmenu("View more","M",
                        events.jmenuitem("help 1"),
                        events.jmenuitem("help 2"),
                        events.jmenuitem("help 3"))
            ),
        frame.setVisible(true);
        for(;;){
            EventObject event=events.waitEvent();
            String name=events.getName(event);
            area.append("Event: "+name+"\n");
            if(name.equals("Exit"))System.exit(0);
        }
    }
}
4.7 Events from the frame

You may also listen to events from the frame itself. You will then receive events when the window loses focus, is iconified, moved etc. Probably the most useful events in this category are events closing the window and resizing the window.

<table>
<thead>
<tr>
<th>JEventQueue</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>static boolean isWindowClosing(EventObject e)</td>
<td>Return true if it is a windows closing event</td>
</tr>
<tr>
<td>static boolean isWindowResizing(EventObject e)</td>
<td>Return true if the window has been resized</td>
</tr>
</tbody>
</table>

This example shows how you can bring up a small dialog to confirm whether a window should be closed.

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

public class CheckClose{
    public static void main(String args[]){
        JFrame frame=new JFrame("CheckClose");
        frame.setSize(600,600);
        frame.setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
        frame.setVisible(true);

        JEventQueue events=new JEventQueue();
        events.listenTo(frame,"Frame");
        while(true){
            EventObject event = events.waitEvent();
            String name=events.getName(event);
            if(events.isWindowClosing(event)){
                int i=JOptionPane.showConfirmDialog(frame,"Ready to Close?");
                if(i==0)System.exit(0);
            }
            if(events.isWindowResizing(event))
                System.out.println("resizing! "+frame.getSize());
        }
    }
}
```
If you try to close the window it will bring up a dialog asking you whether you will close the window. If you select 'No' then the close operation is ignored and the program continues.

More

There is still one area missing in these notes: Frame based animation. The last part of the notes will cover the more advanced areas of graphics. We will look at how to make moving background, sprites that move around on the screen, etc.

5 Animation

This section is a short introduction to frame based animation. The aim is make moving pictures and we do it by repainting the whole frame repeatedly. It is an alternative to sprite based animation where one uses a fixed background and let smaller pictures (sprites) move over the background.

The first and most important aspect of animation is the use of double-buffering. We paint the next frame in a new buffer and only when the buffer is complete will we display it. If we did not buffer the drawings we will see a very unattractive flickering of the screen.

The second most important aspect is not to over strain the system. It is important that you insert small periods of ”sleep” in your program so that the underlying graphics systems can repaint windows and react to other events.

The typical repainting loop may look like the following snippet.

```
<table>
<thead>
<tr>
<th>Frame loop</th>
</tr>
</thead>
<tbody>
<tr>
<td>[while(true){</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
\[} |
```

We will show how to use this style in a number of small examples.
5.1 Slideshow

The first example is a small slide show with three pictures which are changed every 2 seconds. The idea here is to load the image, find the height and width of the image and of the canvas, then compute how much the image can be rescaled and still fit.

The image will be centered on the canvas. If the window is resized then the next image will be displayed to fit the window.

There are some obvious extensions to this little program. We should have controls so that the slide show can be stopped and to go back and forth in the list of images. The image should be redrawn immediately if the window is resized and not just when the next image is going to be displayed.

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

public class SlideShow{
    public static void main(String args[]){
        JFrame frame=new JFrame("SlideShow");
        frame.setSize(600,600);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JCanvas canvas=new JCanvas();
        frame.add(canvas);
        frame.setVisible(true);

        String[] files={"res/canal.jpg","res/heron.jpg","res/fox.jpg"};
        int i=0;
        while(true){
            BufferedImage im=canvas.loadImage(files[i]);
            int h=canvas.getHeight(); //window height
            int w=canvas.getWidth(); //window width
            double h1=im.getHeight(); //image height
            double w1=im.getWidth(); //image width
            double f=Math.min(h/h1,w/w1); // Scale factor
            if(f<0.2)f=0.2;
            int h2=(int) (h1*f), w2=(int) (w1*f);
            canvas.startBuffer();
            canvas.clear();
            canvas.drawScaledImage(im,(w-w2)/2,(h-h2)/2,w2,h2);
            canvas.endBuffer();
            i=(i+1)%files.length;
            canvas.sleep(2000);
        }
    }
}
```

Bouncing a ball on walls

The next example will show how a little bit of physics can be used in animation. We will look at a ball rolling around in a room, seen from above. We will store the speed of the ball as a speed in the x- and y-axis. When the ball hits a vertical wall (wall in the y-direction) then the speed in the y-direction is unchanged, but the speed in the x-direction changes sign. It is the same principle as with light being reflected in a mirror.
This little program can be generalized in a number of ways. Let us just show what happens if we allow several balls in the same room and introduce a bit of randomness when they hit the wall. When you have several actors in a window it may be a good idea to separate it into a number of objects. Information about where the object is and how it moves is stored with the object and not in the central animation loop. We will introduce a Ball class that stores the position and speed of the ball. When we create the ball we give it a color and a maximum speed. Whenever the ball bounces off a wall its speed is changed with a small random number, but never more than the maximum speed. In each iteration we move the ball with its speed. We check the size of the canvas every time since it may be resized and we change the speed when it bounces off the wall.
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;

public class BounceBall3{
    static int vmax=10;
    public static void main(String args[]){
        JFrame frame=new JFrame("BounceBall3");
        frame.setSize(600,600);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JCanvas canvas=new JCanvas();
        frame.add(canvas);
        frame.setVisible(true);
        Ball list[] = new Ball[6];
        list[0]=new Ball(Color.red,50,15);
        list[1]=new Ball(Color.blue,200,5);
        list[2]=new Ball(Color.green,100,10);
        list[3]=new Ball(Color.yellow,50,15);
        list[4]=new Ball(Color.lightGray,25,20);
        list[5]=new Ball(Color.cyan,75,10);
        while(true){
            int mx=canvas.getWidth();
            int my=canvas.getHeight();
            for(Ball b:list)b.move(mx,my);
            canvas.startBuffer();
            canvas.clear();
            for(Ball b:list)b.draw(canvas);
            canvas.setPaint(Color.black);
            canvas.endBuffer();
            canvas.sleep(10);
        }
    }
}

class Ball{
    int x=0,y=0,d=50,vmax=10,vx=vmax,vy=vmax;
    int mx=500,my=500;
    Color color;
    Ball(Color c,int d,int vmax){color=c;this.d=d;this.vmax=vmax;}
    int random(int v){return (int) (Math.random()*v);} 
    int adj(int v){ return Math.max(Math.min(v+random(3)-1,vmax),-vmax); }
    void move(int mx,int my){
        this.mx=mx;this.my=my;
        if(x<0&&vx<0){x=0;vx=adj(-vx);}
        if(y<0&&vy<0){y=0;vy=adj(-vy);}
        if(x+d>mx&&vx>0){x=mx-d;vx=adj(-vx);}
        if(y+d>my&&vy>0){y=my-d;vy=adj(-vy);}
        if(vx==0)vx=1;if(vy==0)vy=1;
    }
    int getX(){return x;}
    int getY(){return y;}
    void draw(JCanvas canvas){
        canvas.setPaint(color);
        canvas.fillOval(x,y,d,d);
    }
}
5.2 Bouncing with gravity

The next example shows how to make a ball bounce under the effect of gravity. The height of the ball as a function of time is given by

\[ h(t) = 4 \cdot h_1 \cdot \left( \frac{t}{t_1} - \frac{t^2}{t_1^2} \right) \]

where \( h_1 \) is the maximum height the ball reaches and \( t_1 \) is the time it takes for the ball to hit the ground again. This is just the solution to the equation

\[ h(t) = a \cdot t^2 + b \cdot t + c \]

where \( h(0) = 0 \), \( h(t_1/2) = h_1 \) and \( h(t_1) = 0 \).

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

public class BounceBall{
    static double sqr(double d){return d*d;}
    public static void main(String args[]){
        JFrame frame=new JFrame("BounceBall");
        frame.setSize(600,600);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JCanvas canvas=new JCanvas();
        frame.add(canvas);
        frame.setVisible(true);

        int d=100; // ball diameter
        int h0=500; // ground depth from top of canvas
        int h1=450-d; // height above ground (bottom of ball)
        int h2=35; // height below: deformation at bounce
        int x=200; // horizontal position of ball
        double t=0; // time
        double t1=300; // time above ground
        double t2=340; // full circle
        while(true){
            t++;
            if(t>t2)t=0;
            int h=0; //height
            if(t<t1)
                h=(int) (4*h1*(t/t1-sqr(t)/sqr(t1)));

            // Draw the ball
            // Draw the ground
        }
    }
}
```
else
    h=(int) (-4*h2*((t-t1)/(t2-t1)-sqr(t-t1)/sqr(t2-t1)));
canvas.startBuffer();
canvas.clear();
canvas.setPaint(Color.gray);
canvas.fillRect(0,h0,x*2+d,20);
int d1= Math.max(0,-h); // deformation
canvas.setPaint(Color.red);
canvas.fillOval(x,h0-h-d+d1,d-d1);
canvas.endBuffer();
canvas.sleep(10);
//if(t==50)canvas.writeToImage("bounceballA.jpg",600,600);
//if(t==310)canvas.writeToImage("bounceballB.jpg",600,600);
}
}
}

5.3 Rolling background

The last example shows how to create a moving background on a screen. Backgrounds and some objects in games are often painted with a texture. A texture as an image that can be tiled, i.e. painted repeatedly next to each other to give the effect of a large surface.

The trick with a moving background is to create a larger background than actually seen. We create an image which is exactly the height of the tile higher than the window. We then just draw a suitable section of this image in the window.

---

RollBackground

```
import java.awt.Rectangle;
import java.awt.image.BufferedImage;
import javax.swing.JFrame;

public class RollBackground{
    public static void main(String args[]){
        JFrame frame=new JFrame("Roll Background");
        frame.setSize(600,600);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JCanvas canvas=new JCanvas();
        frame.add(canvas);
        frame.setVisible(true);

        BufferedImage image=canvas.loadImage("res/oceano.gif");
        int iHeight=image.getHeight();
        int height=canvas.getHeight()-20;
        int width=canvas.getWidth()-20;
        BufferedImage tiles=canvas.tileImage(image,width,height+iHeight);
        int j=0;
        while(true){
            canvas.startBuffer();
            canvas.clear();
            canvas.setClip(10, 10, width, height);
            canvas.drawImage(tiles,10,10-j);
            j++;if(j==iHeight)j=0;
            canvas.endBuffer();
            canvas.sleep(10);
        }
    }
}
```
6 Sound

An animation works better with sound. The JCanvas contains a few methods to provide sound ways to play sound in animation. Sound can either be used as general background musik/sound or special sound clips that are used for special events (explosions, movements etc).

<table>
<thead>
<tr>
<th>JCanvas</th>
<th>Draw or fill general shapes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clip loadClip(String filename)</td>
<td>load a .wav file into a java program as a sound clip</td>
</tr>
<tr>
<td>int clipLength(Clip clip)</td>
<td>return the length of the sound clip in milliseconds</td>
</tr>
<tr>
<td>void playClip(Clip clip)</td>
<td>play the sound clip. The method returns so you may have to use the sleep method to wait while it plays.</td>
</tr>
<tr>
<td>void stopClip(Clip clip)</td>
<td>stop playing the sound clip</td>
</tr>
<tr>
<td>void loopClip(Clip clip, int count)</td>
<td>play the sound clip count times.</td>
</tr>
</tbody>
</table>

Here is a small example that plays a little music piece, plays half the piece and then plays it twice.

```java
import javax.sound.sampled.Clip;

public class PlaySound{
    public static void main(String args[]){
        Clip musica = JCanvas.loadClip("res/musica.wav");
        int milisec= JCanvas.clipLength(musica);
        JCanvas.playClip(musica);
        JCanvas.sleep(milisec);
        System.out.println("done");
        //play the first half of the piece
        JCanvas.playClip(musica);
        JCanvas.sleep(milisec/2);
        JCanvas.stopClip(musica);
        System.out.println("done");
        // pause for a second then play it twice and pause again
        JCanvas.sleep(1000);
        JCanvas.loopClip(musica,2);
        JCanvas.sleep(milisec*2);
        System.out.println("done");
        JCanvas.sleep(1000);
    }
}
```

7 More on drawing on a canvas

In section 2 we introduced a larger number of drawing operations. We will now describe some more advanced operations on a canvas. We will describe shapes as an alternative to the drawing and filling operations described in section 2.
Shapes. A shape is an object that describes, well..., a shape. You can then draw the outline of a shape or you can fill the shape with a color. You can make shapes that describe a variety of curves and even add several lines and curves together to form complex shapes.

<table>
<thead>
<tr>
<th>JCanvas</th>
<th>Draw or fill general shapes</th>
</tr>
</thead>
<tbody>
<tr>
<td>void draw(Shape s)</td>
<td>draw the outline of shape s</td>
</tr>
<tr>
<td>void draw(Shape s, int x, int y)</td>
<td>Draw a shape, moved x to the right and y down</td>
</tr>
<tr>
<td>void fill(Shape s)</td>
<td>Fill a shape</td>
</tr>
<tr>
<td>void fill(Shape s, int x, int y)</td>
<td>Fill a shape, moved x to the right and y down</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Shape</th>
<th>Shape constructors</th>
</tr>
</thead>
<tbody>
<tr>
<td>new Line2D.Double(double x1, double y1, double x2, double y2)</td>
<td>A line from (x1,y1) to (x2,y2)</td>
</tr>
<tr>
<td>new Rectangle2D.Double(double x, double y, double w, double h)</td>
<td>A rectangle with top-left at (x,y), width w and height h</td>
</tr>
<tr>
<td>new RoundRectangle2D.Double(double x, double y, double w, double h, double aw, double ah)</td>
<td>A rectangle with top-left at (x,y), width w and height h. Corners are arcs with width aw and height ah.</td>
</tr>
<tr>
<td>new Ellipse2D.Double(double x, double y, double w, double h)</td>
<td>An ellipsis rectangle with top-left at (x,y), width w and height h.</td>
</tr>
<tr>
<td>new CubicCurve2D.Double(double x1, double y1, double cx1, double cy1, double cx2, double cy2, double x2, double y2)</td>
<td>A cubic curve from (x1,y1) to (x2,y2) with control points (cx1, cy1) and (cx2, cy2).</td>
</tr>
<tr>
<td>new QuadCurve2D.Double(double x1, double y1, double cx, double cy, double x2, double y2)</td>
<td>A quadratic curve from (x1,y1) to (x2,y2) with control point (cx, cy).</td>
</tr>
<tr>
<td>new GeneralPath()</td>
<td>A general path - see below</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GeneralPath</th>
<th>General path</th>
</tr>
</thead>
<tbody>
<tr>
<td>void moveTo(double x, double y)</td>
<td>The next part of the path will continue from (x,y).</td>
</tr>
<tr>
<td>void lineTo(double x, double y)</td>
<td>Draw a line from the current point to (x,y).</td>
</tr>
<tr>
<td>void curveTo(float x1, float y1, float x2, float y2, float x3, float y3)</td>
<td>Draw a curve from the current point to (x3,y3) using (x1,y1) and (x2,y2) as control points.</td>
</tr>
<tr>
<td>void quadTo(float x1, float y1, float x2, float y2)</td>
<td>Draw a quadratic curve from the current point to (x2,y2) using (x1,y1) as control point.</td>
</tr>
<tr>
<td>void append(Shape s, boolean connect)</td>
<td>Appends the shape to the path, possibly connecting it to the existing path.</td>
</tr>
<tr>
<td>void closePath()</td>
<td>Adds a straight line from the current point to the coordinates of the last moveTo</td>
</tr>
</tbody>
</table>

Transformations. You can transform the coordinate system you when drawing. The four basic transformations are rotation, scaling, translation and shearing. Transformations are especially important when you want text to be drawn horizontally or tilted an angle.

A transformation is represented internally as an AffineTransform object, but you do not have to worry about that. Instead you can use operation to rotate, scale etc.
### Transformations

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AffineTransform.getTransform()</td>
<td>Return the current transformation</td>
</tr>
<tr>
<td>void setTransform(AffineTransform t)</td>
<td>restore a saved transformation. Use it to restore the original transformation after some transformations.</td>
</tr>
<tr>
<td>void rotate(double theta)</td>
<td>rotate the coordinate system theta radians around (0,0)</td>
</tr>
<tr>
<td>void rotate(double theta, double x, double y)</td>
<td>rotate the coordinate system theta radians around the point (x,y)</td>
</tr>
<tr>
<td>void scale(double sx, double sy)</td>
<td>scale the coordinate system a factor sx in x-direction and sy in the y-direction</td>
</tr>
<tr>
<td>void shear(double shx, double shy)</td>
<td>shift point shx<em>y in x-direction and shy</em>x in y-direction</td>
</tr>
<tr>
<td>void translate(double tx, double ty)</td>
<td>translate the coordinate system with (tx,ty)</td>
</tr>
</tbody>
</table>

#### Transparency

A Composition is a description of how you paint something on top of other things. Normally you paint over something and cannot be seen anymore. You may, however, paint with a partially transparent color so some of the underlying color is still visible. There are quite a few effects you can obtain in this way, but the easiest is a level of transparency.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Composite.getComposite()</td>
<td>Return the current composition</td>
</tr>
<tr>
<td>void setComposite(Composite comp)</td>
<td>set the composition</td>
</tr>
</tbody>
</table>

A Composition is constructed as

\[
\text{AlphaComposite.getInstance}(\text{AlphaComposite.SRC\_OVER},0.1f)
\]

where the second argument is a number between 0 and 1. The lower the more transparent.

#### Final example

Let us put some of these methods into a more advanced example. We will show how you can use transformation to tilt text and gradient painting change the color.

```java
import javax.swing.*;
import java.awt.*;
import java.awt.font.*;
import java.awt.geom.*;

public class CheckFontCanvas{
    public static void main(String args[]){
        JFrame frame=new JFrame("CheckFontCanvas");
        frame.setSize(800,800);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JCanvas canvas=new JCanvas();
        frame.add(canvas);
        frame.setVisible(true);
        int x,y,y1,sz=100,wd=400;
        Font ft1=new Font("Arial",Font.BOLD,sz);
        String s="Swing";
        FontMetrics fm=canvas.getFontMetrics(ft1);
        int w1=fm.stringWidth(s);
        int dc=fm.getMaxDescent();
```
x=100; y=25; y1=y+sz-dc;
canvas.setFont(ft1);
canvas.setStroke(new BasicStroke(3));
canvas.setPaint(Color.yellow);
canvas.drawString(s, x+3, y1);
canvas.setPaint(Color.red);
canvas.drawString(s, x+3, y1);

x=100; y=150; y1=y+sz-dc;
canvas.setPaint(Color.darkGray);
canvas.fillRect(x-5, y-5, wd+5, sz+15);
canvas.setPaint(Color.lightGray);
canvas.drawString(s, x+4, y1+4);
canvas.setPaint(Color.white);
canvas.drawString(s, x, y1);

x=100; y=275; y1=y+sz-dc;
canvas.setPaint(Color.lightGray);
canvas.fillRect(x-5, y-5, wd+5, sz+15);
canvas.setPaint(Color.white);
canvas.drawString(s, x+4, y1+4);
canvas.setPaint(Color.black);
canvas.drawString(s, x, y1);

x=100; y=400; y1=y+sz-dc;
Color top_color = new Color(200, 200, 0);
Color side_color = new Color(100, 100, 0);
for (int i = 0; i < 22; i++) {
    canvas.setPaint(top_color);
    canvas.drawString(s, x+i, y1+i-1);
    canvas.setPaint(side_color);
    canvas.drawString(s, x+i-1, y1+i);
}
canvas.setPaint(Color.yellow);
canvas.drawString(s, x+20, y1+20);

x=100; y=525; y1=y+sz-dc;
AffineTransform origTransform = canvas.getTransform();
canvas.setPaint(Color.lightGray);
canvas.translate(x+20, y1);
canvas.shear(3, 0);
canvas.scale(1, 0.5);
canvas.drawString(s, 0, 0);
canvas.setTransform(origTransform);
canvas.setPaint(Color.red);
canvas.drawString(s, x+20, y1);

x=100; y=650; y1=y+sz-dc;
canvas.setPaint(new GradientPaint(x, y, Color.yellow, x+400, y1+dc, Color.red));
canvas.drawString(s, x, y1);
canvas.setPaint(Color.orange);
canvas.drawRect(x-5, y-5, w1+10, sz+10);
canvas.drawString(s, x+20, y1+20);
}
References

[1] An Introduction to Graphical User Interfaces with Java Swing
    Paul Fischer
    Addison Wesley, 2005

    Sun Microsystems, 2001

    Andrew Davison
    O’Reilly, 2005

[4] Object-Oriented Programming featuring Graphical Applications in Java
    Michael J Laszlo
    Addison Wesley, 2002

    Kathy Walrath, Mary Campione, Alison Huml, Sharon Zakhour
    Addison Wesley, 2004
Index

Border, 29
BorderFactory
  createCompoundBorder, 29
  createEmptyBorder, 29
  createEtchedBorder, 29
  createLineBorder, 29
  createLoweredBevelBorder, 30
  createMatteBorder, 29
  createRaisedBevelBorder, 30
  createTitledBorder, 30
BufferedImage
  getHeight, 11
  getWidth, 11
ButtonGroup()
  add, 21
  ButtonGroup, 21
Clip, 54
EventObject, 34
Font
  Font, 10
GeneralPath
  append, 56
  closePath, 56
  curveTo, 56
  lineTo, 56
  moveTo, 56
  quadTo, 56
IconImage
  IconImage, 11
JBox
  add, 32
  getScreenHeight, 14
  getScreenWidth, 14
  hbox, 14
  hglue, 14
  hspace, 14
  removeAll, 32
  setSize, 14
  vbox, 14
  vglue, 14
  vspace, 14
JButton
  JButton, 19
  setEnabled, 19
  setHorizontalAlignment, 19
  setHorizontalTextPosition, 19
  setIconTextGap, 19
  setText, 19
  setVerticalAlignment, 19
  setVerticalTextPosition, 19
JCanvas
  clear, 7
  clipLength, 54
  cropImage, 11
  draw, 55
  drawArc, 8
  drawDashedLine, 8
  drawImage, 11
  drawLine, 8
  drawOutline, 10
  drawOval, 8
  drawRect, 8
  drawRoundRect, 8
  drawScaledImage, 11
  drawString, 10
  endBuffer, 7
  fill, 55
  fillArc, 8
  fillOval, 8
  fillRect, 8
  fillRoundRect, 8
  getComposite, 56
  getFont, 10
  getClip, 8
  getHeight, 8
  getPaint, 9
  getStroke, 9
  getTransform, 56
  getWidth, 8
  loadClip, 54
  loadImage, 11
  loopClip, 54
  playClip, 54
  print, 7
  rotate, 56
  rotateImage, 11
  scale, 56
  scaleImage, 11
  setBackground, 7
  setClip, 8
  setComposite, 56
  setFont, 10
  setPaint, 9
  setStroke, 9
  setTransform, 56
shear, 56
sleep, 7
startBuffer, 7
stopClip, 54
storeImage, 11
tileImage, 11
translate, 56
writeToImage, 7

JCheckBox
isSelected, 21
JCheckBox, 21
setSelected, 21

JComboBox
addItem, 27
getItemAt, 27
getItemCount, 27
getSelectedItem, 27
JComboBox, 27
setEditable, 27
setEnabled, 27
jMaximumRowCount, 27

JComponent
gHeight, 15
gWidth, 15
JBox.hbox, 13
JBox.vbox, 13
JButton, 13
JCanvas, 13
JCheckBox, 13
JComboBox, 13
JFormattedTextField, 13
JLabel, 13
JList, 13
JRadioButton, 13
JSeparator, 13
JSlider, 13
JSplitPane, 13
JTappedPane, 13
JTextArea, 13
JTextField, 13
JToggleButton, 13
ProgressBar, 13
requestFocusInWindow, 31
setBackground, 17
setBorder, 29
setCursor, 31
setFont, 17
setForeground, 17
setMinimumSize, 15
setOpaque, 17
setPreferredSize, 15
setToolTipText, 31

JEventQueue
ccontrol, 46
ccontrolShift, 46
ggetKeyChar, 41
ggetKeyText, 41
ggetMouseButton, 43
ggetMouseClickCount, 43
ggetMouseX, 43
ggetMouseY, 43
ggetName, 34
hasEvent, 34
isActionEvent, 41
isActionKey, 41
isActionPerformed, 41
isDocumentEvent, 41
isKeyEvent, 41
isKeyPressed, 41
isKeyReleased, 41
isKeyUp, 41
isMouseClicked, 43
isMouseReleased, 43
isWindowClosing, 47
isWindowResizing, 47
jcheckbokmenuitem, 45
JEventQueue, 34
jmenu, 45
jmenubar, 45
jmenitem, 45
jradiobuttonmenuitem, 45
JSeparator, 46
listenTo, 34
sleep, 44
startTimer, 44
stopTimer, 44
waitEvent, 34

JFormattedTextField
getValue, 23
JFormattedTextField, 23
setValue, 23

JFrame
add, 4
JFrame, 4
repaint, 32
setDefaultCloseOperation, 4
setIconImage, 4

58
setJMenuBar, 4
setLocation, 4
setResizable, 4
setSize, 4
setTitle, 4
setVisible, 4
validate, 32
JLabel
   JLabel, 15, 18
   setHorizontalAlignment, 18
   setHorizontalTextPosition, 18
   setText, 18
   setVerticalAlignment, 18
   setVerticalTextPosition, 18
JList
   getSelectedIndex, 27
   getSelectedValue, 27
JJList, 27
JMenu, 45
JMenuBar, 45
JMenuItem, 45
JProgressBar
   JProgressBar, 26
   setBorderPainted, 26
   setStringPainted, 26
   setValue, 26
JRadioButton
   isSelected, 21
   JRadioButton, 21
   setSelected, 21
JScrollPane
   JScrollPane, 29
   setHorizontalScrollBarPolicy, 29
   setVerticalScrollBarPolicy, 29
JSlider
   getValue, 25
   JSlider, 24
   setExtent, 24
   setInverted, 24
   setMajorTickSpacing, 24
   setMinorTickSpacing, 24
   setPaintLabels, 24
   setPaintTicks, 24
   setPaintTrack, 24
   setSnapToTicks, 25
   setValue, 25
JSplitPane
   JSplitPane, 28
   getDividerLocation, 28
   JSplitPane, 28
   setDividerLocation, 28
   setDividerSize, 28
   setResizeWeight, 28
JTextArea
   append, 22
   getLineCount, 22
   getText, 22
   isEditable, 22
   JTextArea, 22
   setEditable, 22
   setHorizontalAlignment, 22
   setLineWrap, 22
   setText, 22
   setWrapStyleWord, 23
JTextField
   getText, 22
   isEditable, 22
   JTextField, 22
   setEditable, 22
   setHorizontalAlignment, 22
   setText, 22
JToggleButton
   isSelected, 21
   JToggleButton, 21
   setSelected, 21
Paint
   Color, 10
   GradientPaint, 10
   TexturePaint, 10
Shape
   CubicCurve2D.Double, 55
   Ellipse2D.Double, 55
   GeneralPath, 55
   Line2D.Double, 55
   QuadCurve2D.Double, 55
   Rectangle2D.Double, 55
   RoundRectangle2D.Double, 55
SpinnerModel
   SpinnerDateModel, 25
   SpinnerListModel, 25
   SpinnerNumberModel, 25
Stroke
   BasicStroke, 9