

Introduction to graphics programming in Java

Mads Rosendahl

October 10, 2006

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 simplify 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 use a single `main` method and can be understood early in such a course. Some of the later examples contain 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 use 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 in each case 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	3
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	39
4.4	Events from a canvas	40
4.5	Timers	42
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	More on drawing on a canvas	52

1 Frames and windows

Let us look at a simple java program the 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.

EmptyFrame

```
1  import javax.swing.JFrame;
2
3  public class EmptyFrame{
4      public static void main(String args[]){
5          JFrame frame=new JFrame("EmptyFrame");
6          frame.setSize(600,600);
7          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
8          frame.setVisible(true);
9      }
10 }
```

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.

<code>JFrame</code>	<code>import javax.swing.*;</code>
<code>new <u>JFrame</u>(String title)</code>	Constructs a new, initially invisible <code>JFrame</code> object with the specified title.
<code>void <u>add</u>(JComponent comp)</code>	Specify the content of a frame. In section 3 we show how to add several items to a frame by placing them in boxes.
<code>void <u>setDefaultCloseOperation</u>(int operation)</code>	Sets the operation that will happen by default when the user initiates a "close" on this frame. Possible values are <code>WindowConstants.DO_NOTHING_ON_CLOSE</code> , <code>WindowConstants.HIDE_ON_CLOSE</code> , <code>WindowConstants.DISPOSE_ON_CLOSE</code> and <code>JFrame.EXIT_ON_CLOSE</code> .
<code>void <u>setIconImage</u>(BufferedImage image)</code>	Sets the image to be displayed in the minimized icon for this frame. Images are created as <code>BufferedImage</code> . See <code>loadImage</code> in section 2.5
<code>void <u>setJMenuBar</u>(JMenuBar menubar)</code>	Sets the menubar for this frame. Menus are discussed in section 4.6.
<code>void <u>setLocation</u>(Point p)</code>	Places the window at this location on the screen. To create a point call <code>new Point(int x,int y)</code>
<code>void <u>setResizable</u>(boolean resizable)</code>	Sets whether this frame is resizable by the user.
<code>void <u>setSize</u>(int width, int height)</code>	Resizes this component so that it has width <code>width</code> and height <code>height</code> . Controlling the size of frames and content is further discussed in section 3.2.
<code>void <u>setTitle</u>(String title)</code>	Sets the title for this frame to the specified string.
<code>void <u>setVisible</u>(boolean b)</code>	Shows or hides this frame depending on the value of parameter <code>b</code> .

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.

Package	Classes
java.swing.*	Standard components in swing: JFrame, JLabel,... SwingConstants, WindowConstants, ImageIcon, BorderFactory
java.awt.*	Color, Font, some shapes, paints and strokes
java.awt.geom.*	Special shapes: Ellipse2D:Double, GeneralPath,...
java.awt.image.*	BufferedImage
java.util.*	EventObject

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.

DrawCanvas

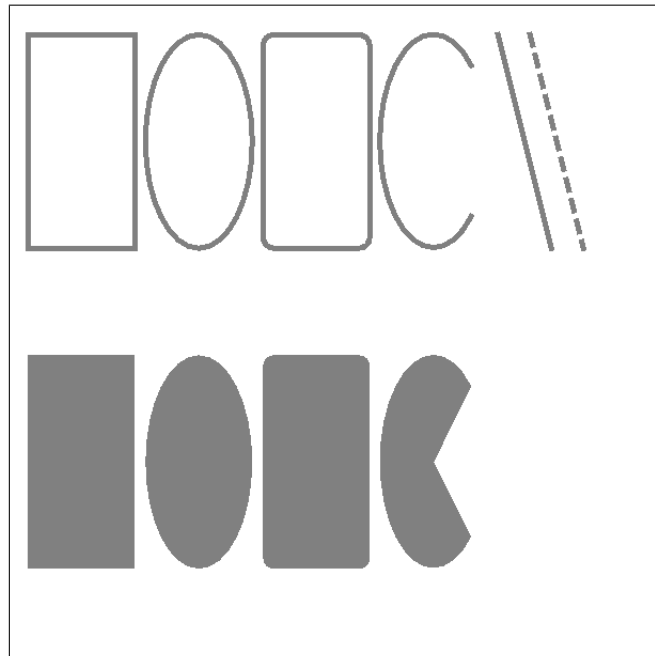
```
1  import java.awt.*;
2  import javax.swing.*;
3
4  public class DrawCanvas{
5      public static void main(String args[]){
6          JFrame frame=new JFrame();
7          frame.setSize(600,600);
8          frame.setTitle("DrawCanvas");
9          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10         JCanvas canvas = new JCanvas();
```

```

11     frame.add(canvas);
12     frame.setVisible(true);
13
14     canvas.setStroke(new BasicStroke(5));
15     canvas.drawRect(10,20,100,200);
16     canvas.drawOval(120,20,100,200);
17     canvas.drawRoundRect(230,20,100,200,20,20);
18     canvas.drawArc(340,20,100,200,45,270);
19     canvas.drawLine(450,20,500,220);
20     canvas.drawDashedLine(10,480,20,530,220);
21
22     canvas.setPaint(Color.gray);
23     canvas.fillRect(10,320,100,200);
24     canvas.fillOval(120,320,100,200);
25     canvas.fillRoundRect(230,320,100,200,20,20);
26     canvas.fillArc(340,320,100,200,45,270);
27 }
28 }

```

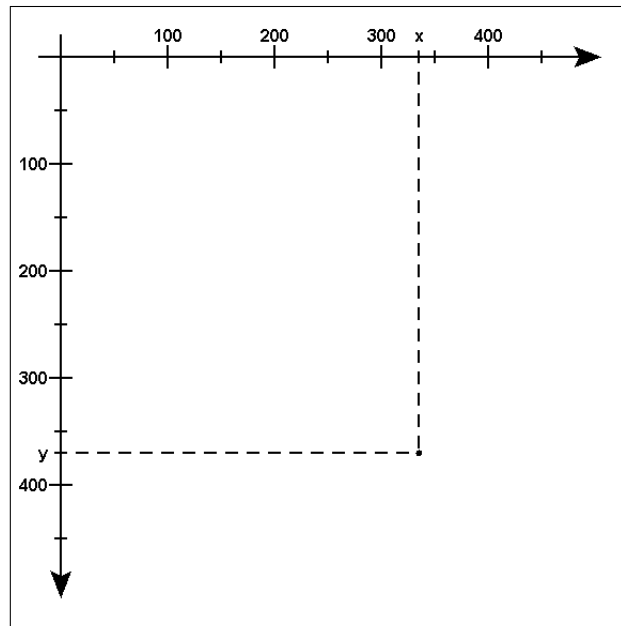
This will result in the following window:



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

JCanvas	Initial controls
<code>void <u>setBackground</u>(Color c)</code>	Set the background color used by <code>clear</code> . You can also start by using <code>fillRect</code> to fill a suitably big area with a color.
<code>void <u>clear</u>()</code>	Clear the canvas to the background color
<code>void <u>startBuffer</u>()</code>	Start double buffering: nothing is shown until you call <code>endbuffer()</code> ;
<code>void <u>endBuffer</u>()</code>	End double buffering.
<code>void <u>print</u>()</code>	Print the content of the canvas. One unit corresponds to 1/72 of an inch.
<code>void <u>print</u>(double scale)</code>	Print the content of the canvas. Scale the content relative to the default of one unit corresponding to 1/72 of an inch.
<code>void <u>writeToImage</u>(String s,int w,int h)</code>	Write the content of the canvas to an image file with name <code>s</code> . Draw everything from point (0,0) to (w,h). The filename should end with ".jpg" or ".png".
<code>void <u>sleep</u>(int ms)</code>	Sleep for <code>ms</code> 1/1000 of a second. This makes it easier to make small animations.

<code>int getHeight()</code>	return the height of the canvas. See section 3.2 for more about controlling the size of frames and their content.
<code>int getWidth()</code>	return the width of the canvas
<code>void setClip(int x,int y,int w,int h)</code>	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.

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

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

2.2 Specify how outlines are drawn

This section may seem like making too 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.

JCanvas	Change Stroke
<code>void <u>setStroke</u>(Stroke s)</code>	Set the stroke used in outlines
<code>Stroke <u>getStroke</u>()</code>	Get the current stroke.
Stroke	
<code>new <u>BasicStroke</u>(float width)</code>	Constructs a solid BasicStroke with the specified line width and with default values for the cap and join styles.
<code>new <u>BasicStroke</u>(float width, int cap, int join)</code>	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.
<code>new <u>BasicStroke</u>(float width, int cap, int join, float miterlimit, float[] dash, float dash_phase)</code>	More general stroke to construct various dashed patterns. In most cases you can use <code>drawDashedLine</code> 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.

JCanvas	Change Paint
<code>void <u>setPaint</u>(Paint p)</code>	Set the paint used in fill operations
<code>Paint <u>getPaint</u>()</code>	Get the current paint.
Paint	
<code>new <u>Color</u>(int r, int g, int b)</code>	Creates an opaque sRGB color with the specified red, green, and blue values in the range (0 - 255).
<code>Color.black</code>	The color black.
<code>Color.blue</code>	The color blue.
<code>Color.cyan</code>	The color cyan.
<code>Color.darkGray</code>	The color dark gray.
<code>Color.gray</code>	The color gray.
<code>Color.green</code>	The color green.
<code>Color.lightGray</code>	The color light gray.

<code>Color.magenta</code>	The color magenta.
<code>Color.orange</code>	The color orange.
<code>Color.pink</code>	The color pink.
<code>Color.red</code>	The color red.
<code>Color.white</code>	The color white.
<code>Color.yellow</code>	The color yellow.
<code>new GradientPaint(float x1, float y1, Color color1, float x2, float y2, Color color2)</code>	Constructs a simple acyclic GradientPaint object. It gradually changes from color <code>color1</code> to color <code>color2</code> from point <code>(x1,y1)</code> to point <code>(x2,y2)</code> .
<code>new TexturePaint(BufferedImage txtr, Rectangle2D anchor)</code>	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	Draw text
<code>void setFont(Font f)</code>	Set the font
<code>Font getFont()</code>	Get the current font
<code>void drawString(String str, int x, int y)</code>	Draw the string <code>str</code> staring at point <code>(x,y)</code>
<code>void drawOutline(String str, int x, int y)</code>	Draw the outline of string <code>str</code> staring at point <code>(x,y)</code> . Each character is seen as lines and drawn, rather than shapes and filled.
Font	
<code>new Font(String name, int style, int size)</code>	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 <code>Font.PLAIN</code> , <code>Font.BOLD</code> , <code>Font.ITALIC</code> or <code>Font.BOLD Font.ITALIC</code> . 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	<code>import java.awt.image.*;</code>
<code>void drawImage(BufferedImage im,int x,int y)</code>	Draw the image with top left at point <code>(x,y)</code>
<code>void drawScaledImage(BufferedImage im,int x,int y,int w,int h)</code>	Draw the image with top left at point <code>(x,y)</code> , scaled to have width <code>w</code> and height <code>h</code> .

<code>void drawScaledImage(BufferedImage im,int x,int y,float scale)</code>	Draw the image with top left at point (x,y), scaled with factor float.
<code>static BufferedImage loadImage(String s)</code>	Reads an image from file s.
<code>static BufferedImage scaleImage(BufferedImage im, int w, int h)</code>	Create a scaled instance of the image with width w and height h
<code>static BufferedImage tileImage(BufferedImage im, int w, int h)</code>	Create a new image of width w and height h, tiled with the given image.
<code>static BufferedImage cropImage(BufferedImage im, int x, int y, int w, int h)</code>	Create a new image which is the selection from point (x,y) with width w and height h.
<code>static BufferedImage rotateImage(BufferedImage im)</code>	rotate an image 90 degrees clockwise.
<code>static BufferedImage storeImage(BufferedImage im, String s)</code>	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.

BufferedImage	<code>import java.awt.image.*;</code>
<code>int getHeight()</code>	return the height of an image
<code>int getWidth()</code>	return the width of an image

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.

IconImage	
<code>new ImageIcon(String fileName)</code>	create an ImageIcon object

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

CheckImage

```

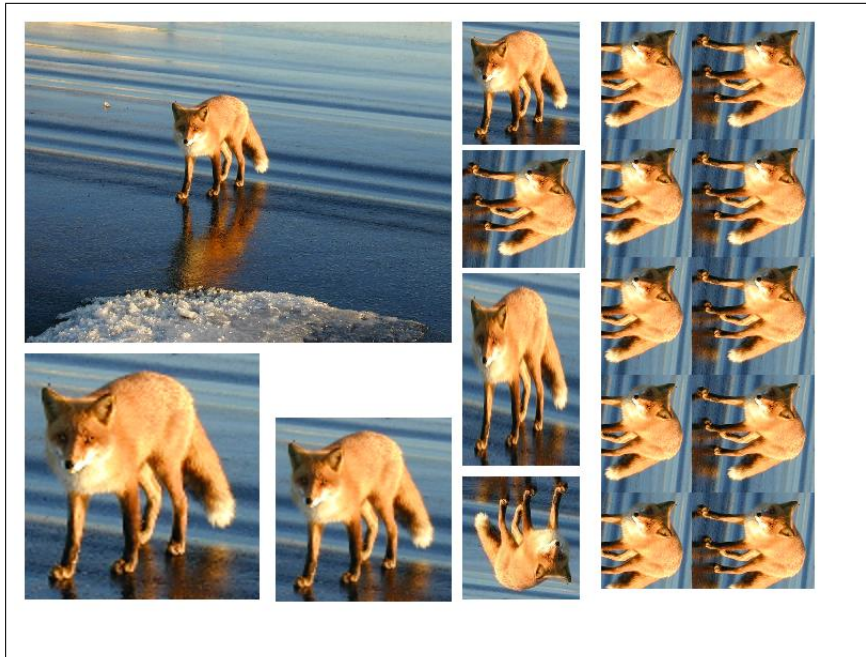
1  import java.awt.*;
2  import java.awt.image.*;
3  import javax.swing.*;
4  import java.util.*;
5
6  public class CheckImage{
7      public static void main(String args[]){
8          JFrame frame=new JFrame("CheckImage");
9          frame.setSize(800,600);
10         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11         JCanvas canvas=new JCanvas();
12         frame.add(canvas);
13         frame.setVisible(true);
14
15         BufferedImage img1=canvas.loadImage("res/Fox.jpg");
16         BufferedImage img2=canvas.scaleImage(img1,400,300);
17         BufferedImage img3=canvas.cropImage(img2,130,60,110,115);
18         canvas.storeImage(img3,"fox1.jpg");
19         BufferedImage img4=canvas.scaleImage(img3,2);
20         BufferedImage img5=canvas.rotateImage(img3);
21         BufferedImage img6=canvas.loadImage("fox1.jpg");
22         BufferedImage img7=canvas.rotateImage(canvas.rotateImage(img6));
23         BufferedImage img8=canvas.tileImage(img3,530,200);
24         BufferedImage img9=canvas.rotateImage(img8);

```

```

25
26     canvas.drawImage(img2,10,10);
27     canvas.drawImage(img4,10,320);
28     canvas.drawScaledImage(img3,245,380,1.5f);
29     canvas.drawImage(img3,420,10);
30     canvas.drawImage(img5,420,130);
31     canvas.drawScaledImage(img3,420,245,110,180);
32     canvas.drawImage(img7,420,435);
33     canvas.drawImage(img9,550,10);
34     canvas.writeToImage("checkimage.jpg",800,600);
35 }
36 }

```



More about graphics. We will describe several other operations you can perform on a canvas in section 6. 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.

JComponent	Basic Components
new <u>JLabel</u> (...)	a label, see section 3.4
new <u>JButton</u> (...)	a button, see section 3.5
new <u>JRadioButton</u> (...)	a radiobutton, see section 3.5
new <u>JCheckBox</u> (...)	a checkbox, see section 3.5
new <u>JToggleButton</u> (...)	a togglebutton, see section 3.5
new <u>JCanvas</u> ()	a canvas
new <u>JTextField</u> (...)	a textfield, see section 3.7
new <u>JTextArea</u> (...)	a textarea, see section 3.7
new <u>JFormattedTextField</u> (...)	a formatted textfield, see section 3.7
new <u>JSpinner</u> (...)	a spinner, see section 3.8
new <u>JSlider</u> (...)	a slider, see section 3.8
new <u>ProgressBar</u> (...)	a progress bar, see section 3.8
new <u>JComboBox</u> (...)	a combo box, see section 3.9
new <u>JList</u> (...)	a list, see section 3.9
new <u>JSeparator</u> (...)	a separator
JComponent	Basic Components
<u>JBox.hbox</u> (...)	a horizontal box
<u>JBox.vbox</u> (...)	a vertical box
new <u>JScrollPane</u> (...)	a scroll pane
new <u>JSplitPane</u> (...)	a split pane
new <u>JTabbedPane</u> (...)	a tabbed pane

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>. 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.

JBox	
static <code>JBox hbox(int align, JComponent c1,...)</code>	Create a horizontal box and add the components <code>c1,...</code> to the box. The components are aligned in the box according to the value <code>align</code> . Possible values are: <code>JBox.BOTTOM</code> , <code>JBox.CENTER</code> and <code>JBox.TOP</code>
static <code>JBox hbox(JComponent c1,...)</code>	Create a horizontal box and add the components <code>c1,...</code> to the box. Components are aligned along the bottom of the box.

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

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`.

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

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

JComponent	
<code>int getHeight()</code>	Returns the current height of this component.
<code>int getWidth()</code>	Returns the current width of this component.
<code>void setMinimumSize(Dimension minimumSize)</code>	Sets the minimum size of this component to a constant value.
<code>void setPreferredSize(Dimension preferredSize)</code>	Sets the preferred size of this component.
<code>void setMaximumSize(Dimension maximumSize)</code>	Sets the maximum size of this component to a constant value.

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 some 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:

JLabel	
new <u>JLabel</u> (String s)	Create a label with text s

CheckLabel0

```

1  import javax.swing.*;
2
3  public class CheckLabel0{
4      public static void main(String args[]){
5          JFrame frame=new JFrame("CheckLabel0");
6          frame.setSize(300,300);
7          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
8          JLabel l1=new JLabel("Label 1");
9          JLabel l2=new JLabel("Label 2");
10         JLabel l3=new JLabel("Label 3");
11         JBox body=
12             JBox.vbox(
13                 JBox.hbox(l1,JBox.hglue()),
14                 JBox.vglue(),
15                 JBox.hbox(JBox.hglue(),l2,JBox.hglue()),
16                 JBox.vglue(),
17                 JBox.hbox(JBox.hglue(),l3)
18             );
19         frame.add(body);
20         frame.setVisible(true);
21     }
22 }
```



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.

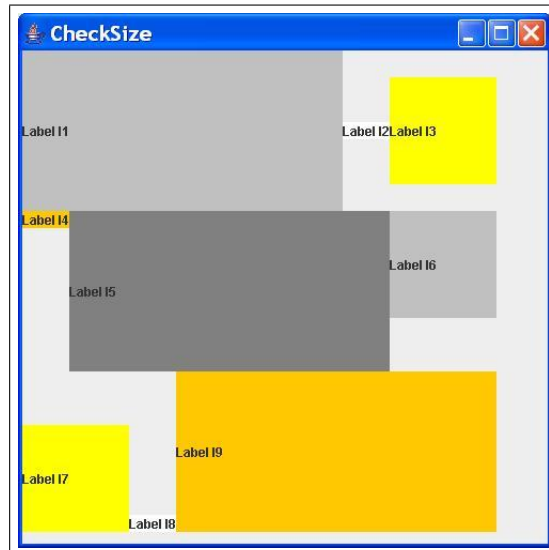
CheckSize1

```
1  import java.awt.*;
2  import javax.swing.*;
3
4  public class CheckSize1{
5      public static void main(String args[]){
6          JFrame frame=new JFrame("CheckSize");
7          frame.setSize(500,500);
8          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
9          JLabel l1 = new JLabel("Label 11");
10         JLabel l2 = new JLabel("Label 12");
11         JLabel l3 = new JLabel("Label 13");
12         JLabel l4 = new JLabel("Label 14");
13         JLabel l5 = new JLabel("Label 15");
14         JLabel l6 = new JLabel("Label 16");
15         JLabel l7 = new JLabel("Label 17");
16         JLabel l8 = new JLabel("Label 18");
17         JLabel l9 = new JLabel("Label 19");
18         l1.setBackground(Color.lightGray);l1.setOpaque(true);
19         l2.setBackground(Color.white);    l2.setOpaque(true);
20         l3.setBackground(Color.yellow);    l3.setOpaque(true);
21         l4.setBackground(Color.orange);    l4.setOpaque(true);
22         l5.setBackground(Color.gray);      l5.setOpaque(true);
23         l6.setBackground(Color.lightGray);l6.setOpaque(true);
24         l7.setBackground(Color.yellow);    l7.setOpaque(true);
25         l8.setBackground(Color.white);     l8.setOpaque(true);
26         l9.setBackground(Color.orange);    l9.setOpaque(true);
27         JBox.setSize(l1,300,150);          JBox.setSize(l3,100,100);
28         JBox.setSize(l5,300,150);          JBox.setSize(l6,100,100);
29         JBox.setSize(l7,100,100);          JBox.setSize(l9,300,150);
30         JBox body= JBox.vbox(JBox.hbox(JBox.CENTER,l1,l2,l3),
31                               JBox.hbox(JBox.TOP    ,l4,l5,l6),
32                               JBox.hbox(JBox.BOTTOM,l7,l8,l9));
33         frame.add(body);
34         frame.setVisible(true);
35     }
36 }
```

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 your 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

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

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

JLabel	
new <u>JLabel</u> (IconImage image)	Creates a JLabel instance with the specified image.
new <u>JLabel</u> (String text)	Creates a JLabel instance with the specified text.
new <u>JLabel</u> (String text, IconImage icon, int horizontalAlignment)	Creates a JLabel instance with the specified text, image, and horizontal alignment. Alignment should have one of the values in n <code>SwingConstants</code> : RIGHT, LEFT, CENTER

<code>void <u>setHorizontalAlignment</u>(int alignment)</code>	Sets the alignment of the label's contents along the X axis. Alignment should have one of the values in <code>SwingConstants</code> : <code>RIGHT</code> , <code>LEFT</code> , <code>CENTER</code>
<code>void <u>setHorizontalTextPosition</u>(int textPosition)</code>	Sets the horizontal position of the label's text, relative to its image. Position has one of the values in <code>SwingConstants</code> : <code>RIGHT</code> , <code>LEFT</code> , <code>CENTER</code>
<code>void <u>setText</u>(String text)</code>	Defines the single line of text this component will display.
<code>void <u>setVerticalAlignment</u>(int alignment)</code>	Sets the alignment of the label's contents along the Y axis. Alignment has one of the values in <code>SwingConstants</code> : <code>TOP</code> , <code>BOTTOM</code> , <code>CENTER</code>
<code>void <u>setVerticalTextPosition</u>(int textPosition)</code>	Sets the vertical position of the label's text, relative to its image. TextPosition has one of the values in <code>SwingConstants</code> : <code>TOP</code> , <code>BOTTOM</code> , <code>CENTER</code>

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).



<code>JButton</code>	
<code>new <u>JButton</u>(String text)</code>	Creates a button with text.
<code>new <u>JButton</u>(String text, ImageIcon icon)</code>	Creates a button with initial text and an icon.

<code>void <u>setEnabled</u>(boolean b)</code>	Enables (or disables) the button. If disabled, then the button cannot be pressed.
<code>void <u>setHorizontalAlignment</u>(int alignment)</code>	Sets the horizontal alignment of the icon and text. Alignment is one of the values from <code>SwingConstants</code> : <code>RIGHT</code> , <code>LEFT</code> , <code>CENTER</code>
<code>void <u>setHorizontalTextPosition</u>(int textPosition)</code>	Sets the horizontal position of the text relative to the icon. <code>TextPosition</code> is one of the values from <code>SwingConstants</code> : <code>RIGHT</code> , <code>LEFT</code> , <code>CENTER</code>
<code>void <u>setIconTextGap</u>(int iconTextGap)</code>	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.
<code>void <u>setText</u>(String text)</code>	Sets the button's text.
<code>void <u>setVerticalAlignment</u>(int alignment)</code>	Sets the vertical alignment of the icon and text. Alignment is one of the values from <code>SwingConstants</code> : <code>TOP</code> , <code>BOTTOM</code> , <code>CENTER</code>
<code>void <u>setVerticalTextPosition</u>(int textPosition)</code>	Sets the vertical position of the text relative to the icon. <code>TextPosition</code> is one of the values from <code>SwingConstants</code> : <code>TOP</code> , <code>BOTTOM</code> , <code>CENTER</code>

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.

CheckBox

```

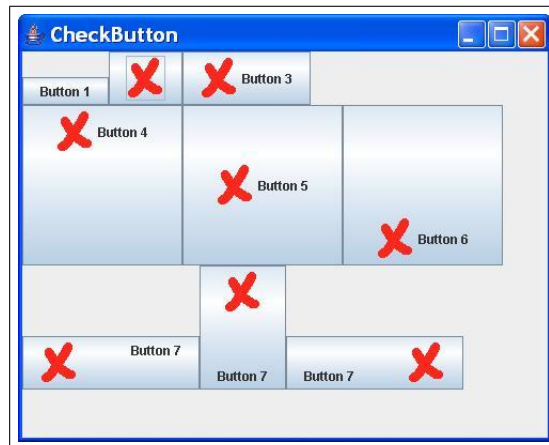
1  import java.awt.*;
2  import javax.swing.*;
3
4  public class CheckBox{
5      public static void main(String args[]){
6          JFrame frame=new JFrame("CheckBox");
7          frame.setSize(500,400);
8          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
9          ImageIcon icon= new ImageIcon("res/cross.gif");
10         JButton l1=new JButton("Button 1");
11         JButton l2=new JButton(icon);
12         JButton l3=new JButton("Button 3",icon);
13
14         JButton l4=new JButton("Button 4",icon);
15         l4.setVerticalAlignment(SwingConstants.TOP);
16         JBox.setSize(l4,150,150);
17         JButton l5=new JButton("Button 5",icon);
18         l5.setVerticalAlignment(SwingConstants.CENTER);
19         JBox.setSize(l5,150,150);
20         JButton l6=new JButton("Button 6",icon);
21         l6.setVerticalAlignment(SwingConstants.BOTTOM);
22         JBox.setSize(l6,150,150);
23
24         JButton l7=new JButton("Button 7",icon);
25         l7.setHorizontalTextPosition(SwingConstants.RIGHT);
26         l7.setVerticalTextPosition(SwingConstants.TOP);
27         l7.setIconTextGap(50);
28         JButton l8=new JButton("Button 7",icon);
29         l8.setHorizontalTextPosition(SwingConstants.CENTER);
30         l8.setVerticalTextPosition(SwingConstants.BOTTOM);
31         l8.setIconTextGap(50);
32         JButton l9=new JButton("Button 7",icon);
33         l9.setHorizontalTextPosition(SwingConstants.LEFT);
34         l9.setIconTextGap(50);
35         l9.setVerticalTextPosition(SwingConstants.BOTTOM);
36
37         JBox body=
38             JBox.vbox(

```

```

39         JBox.hbox(11,12,13),
40         JBox.hbox(14,15,16),
41         JBox.hbox(17,18,19)
42     );
43     frame.add(body);
44     frame.setVisible(true);
45 }
46 }

```



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.

JCheckBox	
<code>new JCheckBox(String text)</code>	Creates an initially unselected check box with text.
<code>boolean isSelected()</code>	Returns the state of the button.
<code>void setSelected(boolean b)</code>	Sets the state of the button.
Plus methods from JButton	
JRadioButton	
<code>new JRadioButton(String text)</code>	Creates an unselected radio button with the specified text.
<code>boolean isSelected()</code>	Returns the state of the button.
<code>void setSelected(boolean b)</code>	Sets the state of the button.
Plus methods from JButton	
JToggleButton	
<code>new JToggleButton(String text)</code>	Creates an togglebutton button with the specified text.
<code>new JToggleButton(String text, ImageIcon icon)</code>	Creates a togglebutton that has the specified text and image.
<code>boolean isSelected()</code>	Returns the state of the button.
<code>void setSelected(boolean b)</code>	Sets the state of the button.
Plus methods from JButton	

A togglebutton looks like an ordinary button. You select it with a single press and a second press will deselect it.

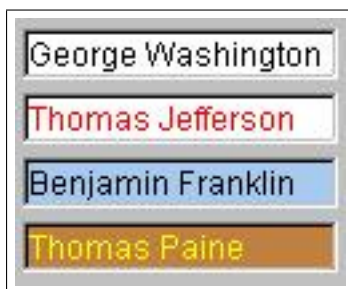
Sometimes you may have several radiobuttons or checkboxes where at most one can be set at a time. You can obtain this effect if you create a ButtonGroup object and register some checkboxes and radiobuttons in this object. The buttongroup object is just a controller that deselects the other buttons when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

ButtonGroup()	
<code>new ButtonGroup()</code>	Creates a new ButtonGroup.
<code>void add(AbstractButton b)</code>	Adds the button to the group.

The arguments to add on a ButtonGroup can be any of the selectable buttons. This is JCheckBox, JRadioButton, JToggleButton, JCheckBoxMenuItem and JRadioButtonMenuItem. The last two 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.



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

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

JTextArea	
<code>new JTextArea(String text)</code>	Constructs a new TextArea with the specified text displayed.

<code>void <u>append</u>(String str)</code>	Appends the given text to the end of the document.
<code>int <u>getLineCount</u>()</code>	Determines the number of lines contained in the area.
<code>String <u>getText</u>()</code>	Returns the text contained in this TextComponent.
<code>boolean <u>isEditable</u>()</code>	Returns the boolean indicating whether this TextComponent is editable or not.
<code>void <u>setEditable</u>(boolean b)</code>	Specifies whether or not this TextComponent should be editable.
<code>void <u>setHorizontalAlignment</u>(int alignment)</code>	Sets the horizontal alignment of the text. Alignment is one of the values from <code>SwingConstants</code> : <code>RIGHT</code> , <code>LEFT</code> , <code>CENTER</code>
<code>void <u>setLineWrap</u>(boolean wrap)</code>	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.
<code>void <u>setText</u>(String t)</code>	Sets the text of this TextComponent to the specified text.
<code>void <u>setWrapStyleWord</u>(boolean word)</code>	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.

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

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

CheckTextField1

```

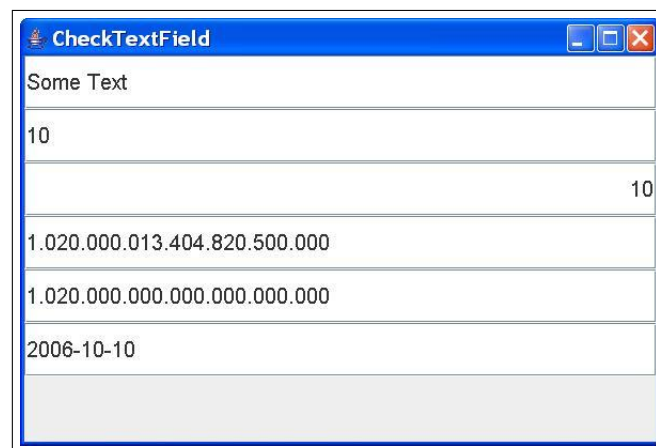
1  import java.awt.*;
2  import javax.swing.*;
3  import java.util.*;
4
5  public class CheckTextField1{
6      public static void main(String args[]){
7          JFrame frame=new JFrame("CheckTextField");
8          frame.setSize(600,400);
9          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10
11         JTextField c1=new JTextField("Some Text");
12         JFormattedTextField c10=new JFormattedTextField(new Integer(10));

```

```

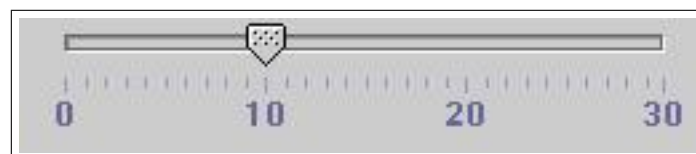
13      JFormattedTextField c11=new JFormattedTextField(new Integer(10));
14      JFormattedTextField c12=new JFormattedTextField(new Float(10.2E20));
15      JFormattedTextField c13=new JFormattedTextField(new Double(10.2E20));
16      JFormattedTextField c14=new JFormattedTextField(new Date());
17
18      JBox body=JBox.vbox(c1,c10,c11,c12,c13,c14,JBox.vglue());
19      body.setFont(new Font("Arial",Font.PLAIN,20));
20      JBox.setSize(c1,600,50,10000,0);
21      JBox.setSize(c10,600,50,10000,0);
22      JBox.setSize(c11,600,50,10000,0);
23      c11.setHorizontalAlignment(SwingConstants.RIGHT);
24      JBox.setSize(c12,600,50,10000,0);
25      JBox.setSize(c13,600,50,10000,0);
26      JBox.setSize(c14,600,50,10000,0);
27      frame.add(body);
28      frame.setVisible(true);
29  }
30  }

```



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.



JSlder	
<code>new JSlder()</code>	Creates a horizontal slider with the range 0 to 100 and an initial value of 50.
<code>new JSlder(int orientation, int min, int max, int value)</code>	Creates a slider with the specified orientation and the specified minimum, maximum, and initial values. Orientation should have the value <code>SwingConstants.VERTICAL</code> or <code>SwingConstants.HORIZONTAL</code>

<code>void <u>setExtent</u>(int extent)</code>	Sets the size of the range "covered" by the knob.
<code>void <u>setInverted</u>(boolean b)</code>	Specify true to reverse the value-range shown for the slider and false to put the value range in the normal order.
<code>void <u>setMajorTickSpacing</u>(int n)</code>	This method sets the major tick spacing.
<code>void <u>setMinorTickSpacing</u>(int n)</code>	This method sets the minor tick spacing.
<code>void <u>setPaintLabels</u>(boolean b)</code>	Determines whether labels are painted on the slider.
<code>void <u>setPaintTicks</u>(boolean b)</code>	Determines whether tick marks are painted on the slider.
<code>void <u>setPaintTrack</u>(boolean b)</code>	Determines whether the track is painted on the slider.
<code>void <u>setSnapToTicks</u>(boolean b)</code>	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.
<code>void <u>setValue</u>(int n)</code>	Sets the sliders current value.
<code>int <u>getValue</u>()</code>	Get the sliders current value.

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.

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

SpinnerModel	
<code>new <u>SpinnerDateModel</u>()</code>	Constructs a <code>SpinnerDateModel</code> whose initial value is the current date, <code>calendarField</code> is equal to <code>Calendar.DAY_OF_MONTH</code> , and for which there are no start/end limits.
<code>new <u>SpinnerListModel</u>(Object[] values)</code>	Constructs a <code>SpinnerModel</code> whose sequence of values is defined by the specified array.
<code>new <u>SpinnerNumberModel</u>(double value, double minimum, double maximum, double stepSize)</code>	Constructs a <code>SpinnerNumberModel</code> with the specified value, minimum/maximum bounds, and <code>stepSize</code> .
<code>new <u>SpinnerNumberModel</u>(int value, int minimum, int maximum, int stepSize)</code>	Constructs a <code>SpinnerNumberModel</code> with the specified value, minimum/maximum bounds, and <code>stepSize</code> .

CheckSpinner

```
1  import javax.swing.*;
2
3  public class CheckSpinner{
4      public static void main(String args[]){
5          JFrame frame=new JFrame("CheckSpinner");
6          frame.setSize(400,250);
7          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
8
9          JSpinner      c1=new JSpinner();
10         JSpinner      c2=new JSpinner(new SpinnerDateModel());
11         JSpinner      c3=new JSpinner(new SpinnerNumberModel(3.1,1.1,9.9,0.1));
12         JSpinner      c4=new JSpinner(new SpinnerNumberModel(6,0,30,3));
13         String weekdays[]={"Monday","Tuesday","Wednesday","Thursday","Friday",
14                             "Saturday","Sunday"};
15         JSpinner      c5=new JSpinner(new SpinnerListModel(weekdays));
16
17         frame.add(JBox.vbox(c1,c2,c3,c4,c5,JBox.vglue()));
18         frame.setVisible(true);
19     }
20 }
```



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.

JProgressBar	
new JProgressBar()	create a progress bar
void setValue(int n)	set the value in the progress bar. The value should be between 0 and 100.
void setStringPainted(boolean b)	
void setBorderPainted(boolean b)	

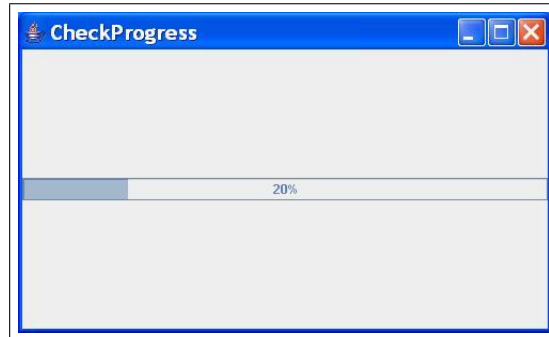
CheckProgress

```
1  import javax.swing.*;
2
3  public class CheckProgress{
4      public static void main(String args[]){
5          JFrame frame=new JFrame("CheckProgress");
6          frame.setSize(500,300);
7          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
8          JProgressBar bar=new JProgressBar();
9          frame.add(JBox.vbox(JBox.vglue(),bar,JBox.vglue()));
10         frame.setVisible(true);
11         bar.setStringPainted(true);
12         bar.setBorderPainted(true);
13         for(int i=0;i<=100;i++){
```

```

14         bar.setValue(i);
15         JCanvas.sleep(200);
16     }
17 }
18 }

```



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.

JComboBox	
<code>new JComboBox(Object[] items)</code>	Creates a JComboBox that contains the elements in the specified array.
<code>void addItem(Object anObject)</code>	Adds an item to the item list.
<code>int getItemCount()</code>	Returns the number of items in the list.
<code>Object getItemAt(int index)</code>	Returns the list item at the specified index.
<code>Object getSelectedItem()</code>	Returns the current selected item.
<code>void setEditable(boolean aFlag)</code>	Specify whether the JComboBox field is editable.
<code>void setEnabled(boolean b)</code>	Enables the combo box so that items can be selected.
<code>void setMaximumRowCount(int count)</code>	Sets the maximum number of rows the JComboBox displays.
JList	
<code>new JList(Object[] listData)</code>	Constructs a JList that displays the elements in the specified array.
<code>Object getSelectedValue()</code>	Returns the first selected value.
<code>boolean getSelectedIndex(int index)</code>	Returns true if index is selected.

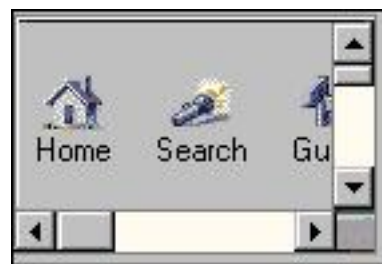
CheckComboList

```
1  import javax.swing.*;
2
3  public class CheckComboList{
4      public static void main(String args[]){
5          JFrame frame=new JFrame("CheckComboList");
6          frame.setSize(400,200);
7          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
8
9          String[] data1={"choice 1","choice 2","choice 3","choice 4"};
10         String[] data2={"select 1","select 2","select 3","select 4"};
11         JComboBox combo=new JComboBox(data1);
12         JList list=new JList(data2);
13         JBox body=JBox.vbox(combo,JBox.vspace(20),list,JBox.vglue());
14         frame.add(body);
15         frame.setVisible(true);
16     }
17 }
```



3.10 SplitPane and Scrollpane

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.



JSplitPane	
<code>new JSplitPane(int newOrientation, JComponent newLeftComponent, JComponent newRightComponent)</code>	Creates a new JSplitPane with the specified orientation and with the specified components that do not do continuous redrawing. The option <code>newOrientation</code> is one of the constants in JSplitPane: <code>HORIZONTAL_SPLIT</code> or <code>VERTICAL_SPLIT</code> .

<code>int <u>getDividerLocation</u>()</code>	Returns the last value passed to <code>setDividerLocation</code> .
<code>void <u>setDividerLocation</u>(int location)</code>	Sets the location of the divider. The location is measured in pixels from the top or left.
<code>void <u>setDividerSize</u>(int newSize)</code>	Sets the size of the divider.
<code>void <u>setResizeWeight</u>(double value)</code>	Specifies how to distribute extra space when the size of the split pane changes.
JScrollPane	
<code>new <u>JScrollPane</u>(JComponent view)</code>	Creates a <code>JScrollPane</code> 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.
<code>void <u>setHorizontalScrollBarPolicy</u>(int policy)</code>	Determines when the horizontal scrollbar appears in the scrollpane. The options are one of the constants in <code>ScrollPaneConstants</code> : <code>HORIZONTAL_SCROLLBAR_AS_NEEDED</code> , <code>HORIZONTAL_SCROLLBAR_NEVER</code> , <code>HORIZONTAL_SCROLLBAR_ALWAYS</code>
<code>void <u>setVerticalScrollBarPolicy</u>(int policy)</code>	Determines when the vertical scrollbar appears in the scrollpane. The options are one of the constants in <code>ScrollPaneConstants</code> : <code>VERTICAL_SCROLLBAR_AS_NEEDED</code> , <code>VERTICAL_SCROLLBAR_NEVER</code> , <code>VERTICAL_SCROLLBAR_ALWAYS</code>

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	
<code>void <u>setBorder</u>(Border border)</code>	Sets the border of this component.
BorderFactory	
<code>static Border <u>createCompoundBorder</u>(Border outsideBorder, Border insideBorder)</code>	Creates a compound border specifying the border objects to use for the outside and inside edges.
<code>static Border <u>createEmptyBorder</u>(int top, int left, int bottom, int right)</code>	Creates an empty border that takes up space but which does no drawing, specifying the width of the top, left, bottom, and right sides.
<code>static Border <u>createEtchedBorder</u>()</code>	Creates a border with an "etched" look using the component's current background color for highlighting and shading.
<code>static Border <u>createLineBorder</u>(Color color, int thickness)</code>	Creates a line border with the specified color and width.
<code>static Border <u>createMatteBorder</u>(int top, int left, int bottom, int right, Color color)</code>	Creates a border where the width can be different on the four sides.

static Border <u>createLoweredBevelBorder()</u>	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 <u>createRaisedBevelBorder()</u>	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 <u>createTitledBorder(String title)</u>	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 <u>createTitledBorder</u> (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

```

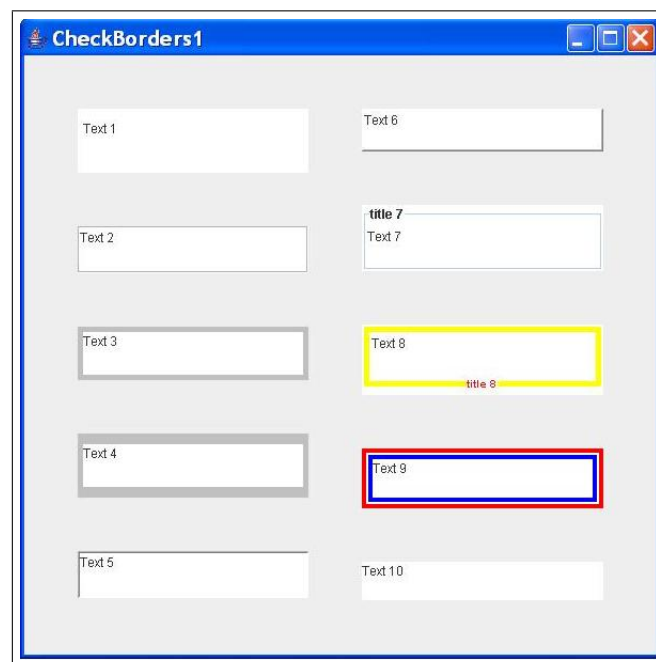
1  import java.awt.*;
2  import javax.swing.*;
3  import javax.swing.border.*;
4
5  public class CheckBorders1{
6      public static void main(String args[]){
7          JFrame frame=new JFrame("CheckBorders1");
8          frame.setSize(600,600);
9          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10
11          JTextArea t1=new JTextArea("Text 1");
12          JTextArea t2=new JTextArea("Text 2");
13          JTextArea t3=new JTextArea("Text 3");
14          JTextArea t4=new JTextArea("Text 4");
15          JTextArea t5=new JTextArea("Text 5");
16          JTextArea t6=new JTextArea("Text 6");
17          JTextArea t7=new JTextArea("Text 7");
18          JTextArea t8=new JTextArea("Text 8");
19          JTextArea t9=new JTextArea("Text 9");
20          JTextArea t10=new JTextArea("Text 10");
21
22          JBox body=JBox.hbox(JBox.hspace(50),
23              JBox.vbox(JBox.vspace(50),t1,JBox.vspace(50),t2,JBox.vspace(50),
24                  t3,JBox.vspace(50),t4,JBox.vspace(50),t5,JBox.vspace(50)),
25              JBox.hspace(50),
26              JBox.vbox(JBox.vspace(50),t6,JBox.vspace(50),t7,JBox.vspace(50),
27                  t8,JBox.vspace(50),t9,JBox.vspace(50),t10,JBox.vspace(50)),
28              JBox.hspace(50));
29
30          t1.setBorder(BorderFactory.createEmptyBorder(10,5,10,5));
31          t2.setBorder(BorderFactory.createEtchedBorder());
32          t3.setBorder(BorderFactory.createLineBorder(Color.lightGray,5));
33          t4.setBorder(BorderFactory.createMatteBorder(10,5,10,5,Color.lightGray));
34          t5.setBorder(BorderFactory.createLoweredBevelBorder());
35          t6.setBorder(BorderFactory.createRaisedBevelBorder());

```

```

36     t7.setBorder(BorderFactory.createTitledBorder("title 7"));
37     t8.setBorder(BorderFactory.createTitledBorder(
38         BorderFactory.createLineBorder(Color.yellow,5),
39         "title 8",TitledBorder.CENTER,TitledBorder.BOTTOM,
40         new Font("Arial",Font.BOLD,10),Color.red));
41     t9.setBorder(BorderFactory.createCompoundBorder(
42         BorderFactory.createLineBorder(Color.red,4),
43         BorderFactory.createCompoundBorder(
44             BorderFactory.createLineBorder(Color.white,2),
45             BorderFactory.createLineBorder(Color.blue,4))));
46
47     frame.add(body);
48     frame.setVisible(true);
49 }
50 }

```



3.12 Other controls for Swing components

JComponent	
boolean <u>requestFocusInWindow</u> ()	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.
void <u>setToolTipText</u> (String text)	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.
void <u>setCursor</u> (Cursor c)	specify a cursor for a component. Possible values of cursors are listed below.

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`.

JBox	
<code>void add(JComponent c)</code>	
<code>void removeAll()</code>	remove all components in a box
JFrame	
<code>void validate()</code>	Check the layout of components in the window
<code>void repaint()</code>	Repaint the window

CheckAddRemove

```

1  import javax.swing.*;
2  import java.awt.Font;
3
4  public class CheckAddRemove{
5      public static void main(String args[]){
6          JFrame frame=new JFrame("CheckAddRemove");
7          frame.setSize(600,600);
8          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
9          JButton button1=new JButton("Start");
10         button1.setFont(new Font("Arial",Font.BOLD,40));
11         JEventQueue events=new JEventQueue();
12         events.listenTo(button1,"Button1");
13         JBox body = JBox.vbox(button1);
14         frame.add(JBox.vbox(JBox.vglue(),
15             JBox.hbox(JBox.hglue(),body,JBox.hglue()),
16             JBox.vglue()));
17         frame.setVisible(true);
18
19         events.waitEvent();//wait for button press
20         body.removeAll();
21         JRadioButton button2=new JRadioButton("Button2");
22         events.listenTo(button2,"Button2");
23         body.add(button2);
24         frame.validate();
25         frame.repaint();
26     }

```

```
27         events.waitEvent();//wait for button press
28         body.removeAll();
29         JCheckBox button3=new JCheckBox("Button3");
30         events.listenTo(button3,"Button3");
31         body.add(button3);
32         frame.validate();
33         frame.repaint();
34
35         events.waitEvent();//wait for button press
36         System.exit(0);
37     }
38 }
```

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, radiobuttons, 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. They 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.

JEventQueue	
<code>new JEventQueue()</code>	Construct an event queue
<code>void <u>listenTo</u>(JComponent c,String name)</code>	Let the <code>JEventQueue</code> listen to event from component c. Register the name with the component.
<code>EventObject <u>waitEvent</u>()</code>	wait for an event to happen. Remove the event from the queue and return it.
<code>String <u>getName</u>(EventObject e)</code>	Return the name of the component that generated the event
<code>boolean <u>hasEvent</u>()</code>	Returns true if there are events in the queue.

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 now 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.

```

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 ...
}

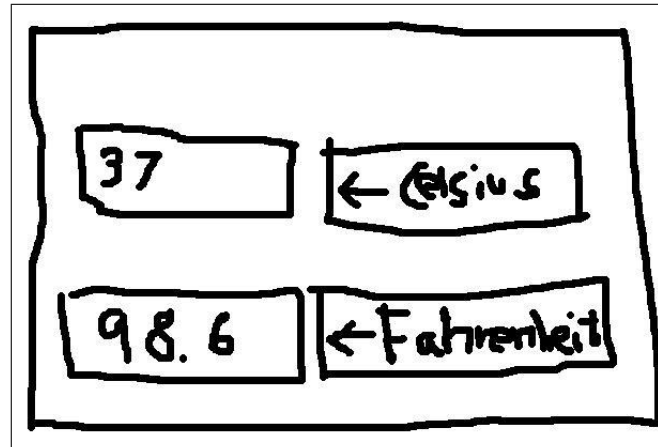
```

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.



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.

CheckListener1

```
1  import javax.swing.*;
2
3  public class CheckListener1{
4      public static void main(String args[]){
5          JFrame frame=new JFrame("CheckListener1");
6          frame.setSize(600,400);
7          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
8
9          JButton button1=new JButton("<<< Celsius");
10         JButton button2=new JButton("<<< Fahrenheit");
11         JTextField field1=new JTextField("");
12         JTextField field2=new JTextField("");
13
14         JBox body=
15             JBox.vbox(
16                 JBox.hbox(field1,button1),
17                 JBox.hbox(field2,button2)
18             );
19         frame.add(body);
20         frame.setVisible(true);
21     }
22 }
```



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 text field and put some glue around it so that the fields and buttons are centered in the window. We will also make the font size a bit bigger and place the text on the buttons to the left.

CheckListener2

```

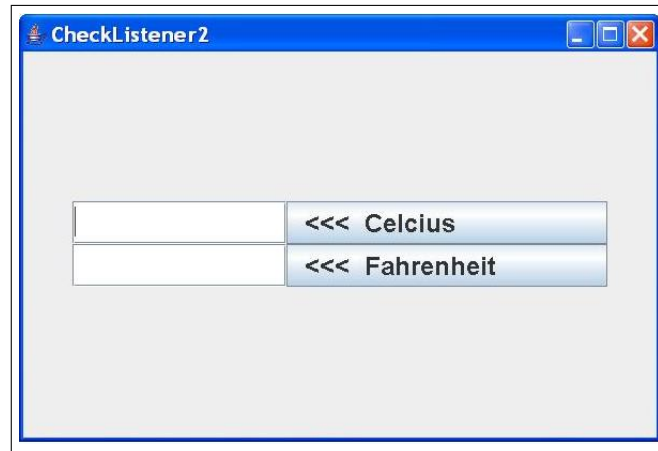
1  import java.awt.*;
2  import javax.swing.*;
3
4  public class CheckListener2{
5      static JButton myButton(String s){
6          JButton c=new JButton(s);
7          JBox.setSize(c,300,40);
8          c.setFont(new Font("Arial",Font.BOLD,24));
9          c.setHorizontalAlignment(SwingConstants.LEFT);
10         return c;
11     }
12     static JTextField myTextField(String s){
13         JTextField c=new JTextField(s);
14         JBox.setSize(c,200,40);
15         c.setFont(new Font("Arial",Font.BOLD,24));
16         return c;
17     }
18     public static void main(String args[]){
19         JFrame frame=new JFrame("CheckListener2");
20         frame.setSize(600,400);
21         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
22
23         JButton button1=myButton("<<<  Celsius");
24         JButton button2=myButton("<<<  Fahrenheit");
25         JTextField field1=myTextField("");
26         JTextField field2=myTextField("");
27         JBox body=
28             JBox.vbox(
29                 JBox.vglue(),
30                 JBox.hbox(JBox.hglue(),field1,button1,JBox.hglue()),
31                 JBox.hbox(JBox.hglue(),field2,button2,JBox.hglue()),
32                 JBox.vglue()
33             );
34         frame.add(body);

```

```

35     frame.setVisible(true);
36 }
37 }

```



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.

CheckListener3

```

1  import java.awt.*;
2  import javax.swing.*;
3  import java.util.EventObject;
4
5  public class CheckListener3{
6      static JButton myButton(String s){
7          JButton c=new JButton(s);
8          JBox.setSize(c,300,40);
9          c.setFont(new Font("Arial",Font.BOLD,24));
10         c.setHorizontalAlignment(SwingConstants.LEFT);
11         return c;
12     }
13     static JTextField myTextField(String s){
14         JTextField c=new JTextField(s);
15         JBox.setSize(c,200,40);
16         c.setFont(new Font("Arial",Font.BOLD,24));
17         return c;
18     }
19     static String format(double d){
20         int x=(int) (d*10);
21         return ""+(x/10)+"."+ (x%10);
22     }
23     public static void main(String args[]){
24         JFrame frame=new JFrame("CheckListener3");
25         frame.setSize(600,600);
26         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
27
28         JButton button1=myButton("<<< Celsius");
29         JButton button2=myButton("<<< Fahrenheit");
30         JTextField field1=myTextField("");
31         JTextField field2=myTextField("");
32         JBox body=
33             JBox.vbox(
34                 JBox.vglue(),
35                 JBox.hbox(JBox.hglue(),field1,button1,JBox.hglue()),
36                 JBox.hbox(JBox.hglue(),field2,button2,JBox.hglue()),

```

```

37         JBox.vglue()
38     );
39     frame.add(body);
40     frame.setVisible(true);
41
42     JEventQueue events=new JEventQueue();
43     events.listenTo(button1,"celcius");
44     events.listenTo(button2,"fahrenheit");
45     while(true){
46         EventObject event=events.waitEvent();
47         String name=events.getName(event);
48         if(name.equals("celcius")){
49             double f=Double.parseDouble(field2.getText());
50             field1.setText(format((f-32)/1.8));
51         }else
52         if(name.equals("fahrenheit")){
53             double c=Double.parseDouble(field1.getText());
54             field2.setText(format(c*1.8+32));
55         }
56     }
57 }
58 }

```

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.

CheckState

```

1  import javax.swing.*;
2  import java.util.*;
3
4  public class CheckState{
5      public static void main(String args[]){
6          JFrame frame=new JFrame("CheckState");
7          frame.setSize(600,400);
8          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
9
10         JRadioButton button1=new JRadioButton("Yes");
11         JRadioButton button2=new JRadioButton("No");
12         JCheckBox button3=new JCheckBox("CheckBox");
13         JToggleButton button4=new JToggleButton("ToggleButton");
14
15         JSpinner spinner=new JSpinner();
16         JSlider slider=new JSlider();
17
18         ButtonGroup group=new ButtonGroup();
19         group.add(button1); group.add(button2); //only one can be set at a time.
20
21         String[] data1={"combo1","combo2","combo3","combo3"};
22         String[] data2={"list1","list2","list3","list4"};
23
24         JComboBox combo=new JComboBox(data1);
25         JList list=new JList(data2);
26
27         JBox body=
28             JBox.vbox(
29                 button1,button2,button3,button4,

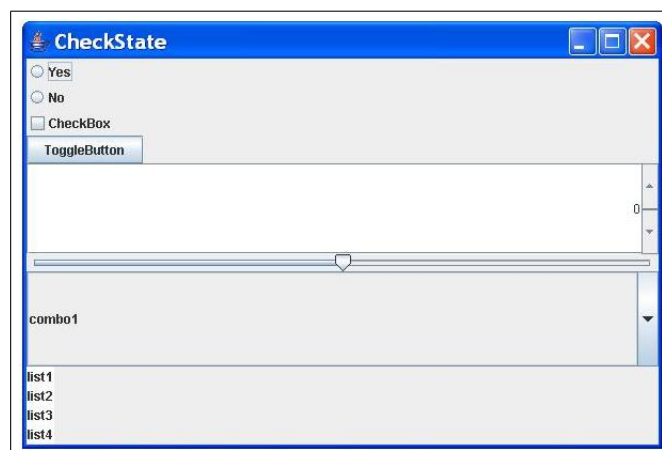
```

```

30         spinner,slider, combo,list
31     );
32     frame.add(body);
33     frame.setVisible(true);
34
35     JEventQueue events=new JEventQueue();
36     events.listenTo(button1,"button1");
37     events.listenTo(button2,"button2");
38     events.listenTo(button3,"button3");
39     events.listenTo(button4,"button4");
40     events.listenTo(spinner,"spinner");
41     events.listenTo(slider,"slider");
42     events.listenTo(combo,"combo");
43     events.listenTo(list,"list");
44
45     while(true){
46         EventObject event=events.waitEvent();
47         String name= events.getName(event);
48         if(name.equals("button1"))
49             System.out.println("Button1: "+button1.isSelected());
50         else if(name.equals("button2"))
51             System.out.println("Button2: "+button2.isSelected());
52         else if(name.equals("button3"))
53             System.out.println("Button3: "+button3.isSelected());
54         else if(name.equals("button4"))
55             System.out.println("Button4: "+button4.isSelected());
56         else if(name.equals("spinner"))
57             System.out.println("Spinner: "+spinner.getValue());
58         else if(name.equals("slider"))
59             System.out.println("Slider: "+slider.getValue());
60         else if(name.equals("combo"))
61             System.out.println("Combo: "+combo.getSelectedItem());
62         else if(name.equals("list"))
63             System.out.println("List: "+list.getSelectedValue());
64         else
65             System.out.println("Error: unknown event :"+name);
66     }
67 }
68 }

```

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.

JEventQueue	
static boolean <u>isKeyEvent</u> (EventObject e)	Returns true if the event is a KeyEvent
static boolean <u>isKeyTyped</u> (EventObject e)	Returns true if the event is a character being typed.
static boolean <u>isKeyPressed</u> (EventObject e)	Returns true if the event is a key being pressed.
static boolean <u>isKeyReleased</u> (EventObject e)	Returns true if the event is a key being released.
static boolean <u>isActionPerformed</u> (EventObject e)	Returns true if it is an ActionPerformed event.
static boolean <u>isActionKey</u> (EventObject e)	Returns true if the event is pressing an action key. Action keys are keys like Home, End, Page Up, etc.
static boolean <u>isActionEvent</u> (EventObject e)	Returns true if the event is an ActionEvent
static boolean <u>isDocumentEvent</u> (EventObject e)	Returns true if the event is a DocumentEvent
static char <u>getKeyChar</u> (EventObject e)	If the event is typing a character, return this character
static String <u>getKeyText</u> (EventObject e)	If the event is pressing an action key, return a textual representation of this

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.

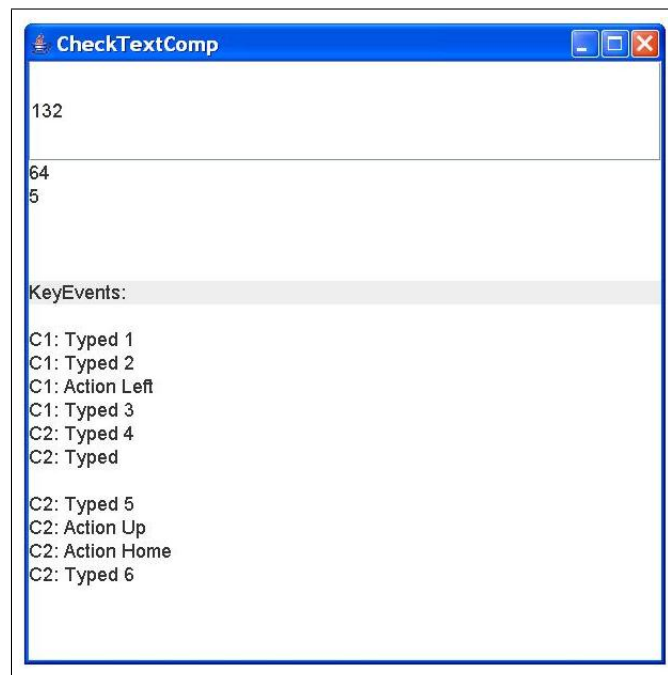
CheckTextComp

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4  import java.util.*;
5
6  public class CheckTextComp{
7      public static void main(String args[]){
8          JFrame frame=new JFrame("CheckTextComp");
9          frame.setSize(600,600);
10         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11     }
```

```

12      JTextField      c1=new JTextField();
13      JTextArea       c2=new JTextArea();
14      JTextArea       c3=new JTextArea();
15
16      JBox body=JBox.vbox(
17          JBox.setSize(c1,600,30,600,0),JBox.setSize(c2,600,200,600,0),
18          new JLabel("KeyEvents:"),new JScrollPane(c3));
19      body.setFont(new Font("Arial",Font.PLAIN,18));
20      c3.setFont(new Font("Arial",Font.PLAIN,18));
21      frame.add(body);
22      frame.setVisible(true);
23
24      JEventQueue events=new JEventQueue();
25      events.listenTo(c1,"C1");
26      events.listenTo(c2,"C2");
27      for(;;){
28          EventObject event=events.waitEvent();
29          String name=events.getName(event);
30          if(events.isKeyTyped(event))
31              c3.setText(c3.getText()+"\n"+name+": Typed "
32                  +events.getKeyChar(event));
33          if(events.isActionKey(event))
34              c3.setText(c3.getText()+"\n"+name+": Action "
35                  +events.getKeyText(event));
36      }
37 }
38 }

```



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.

JEventQueue	
static boolean <u>isMouseEvent</u> (EventObject e)	Returns true if it is a mouse event
static boolean <u>isMousePressed</u> (EventObject e)	Returns true if it is an event from pressing a mouse button
static boolean <u>isMouseClicked</u> (EventObject e)	Returns true if it is an event from clicking a mouse button
static boolean <u>isMouseReleased</u> (EventObject e)	Returns true if it is an event from releasing a mouse button
static int <u>getMouseX</u> (EventObject e)	Return the x coordinate of the mouse location
static int <u>getMouseY</u> (EventObject e)	Return the y coordinate of the mouse location
static int <u>getMouseButton</u> (EventObject e)	Return information about which mouse button was pressed
static int <u>getMouseClickCount</u> (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.

CheckMouse

```

1  import javax.swing.*;
2  import java.util.*;
3
4  public class CheckMouse{
5      public static void main(String args[]){
6          JFrame frame=new JFrame("CheckMouse");
7          frame.setSize(600,600);
8          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
9
10         JCanvas canvas=new JCanvas();
11         frame.add(canvas);
12         frame.setVisible(true);
13
14         JEventQueue events=new JEventQueue();
15         events.listenTo(canvas,"canvas");
16         int x0=0,y0=0;
17         while(true){
18             EventObject event=events.waitEvent();
19             if(events.isMouseEvent(event)){
20                 int x=events.getMouseX(event);
21                 int y=events.getMouseY(event);
22                 if(events.isMousePressed(event)){x0=x;y0=y;}
23                 if(events.isMouseClicked(event))
24                     canvas.drawOval(x0-5,y0-5,10,10);
25                 if(events.isMouseReleased(event))
26                     canvas.drawLine(x0,y0,x,y);
27             }
28         }
29     }
30 }
```

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	
void <u>startTimer</u> (int interval, String name)	Start the timer. The timer will fire with <u>interval</u> milliseconds intervals. Register the <u>name</u> with the timer in the eventqueue.
void <u>stopTimer</u> (String name)	Stop the timer and remove any events in the queue from the timer.
void <u>sleep</u> (int ms)	Sleep for ms milliseconds

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.

CheckTimers

```
1  import java.util.*;
2
3  public class CheckTimers{
4      public static void main(String args[]){
5
6          JEventQueue events=new JEventQueue();
7          events.startTimer(1000,"timer1");
8          events.startTimer(2500,"timer2");
9          int t=0;
10         while(t<10){
11             EventObject event=events.waitEvent();
12             String name=events.getName(event);
13             System.out.println(t+": "+name);
14             if(name.equals("timer1"))t++;
15             if(name.equals("timer2")&& t>3){
16                 events.stopTimer("timer2");
17                 events.startTimer(1500,"timer3");
18             }
19         }
20         System.exit(0);
21     }
22 }
```

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 is stop at time and that timer3 is 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 eventqueue 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'.

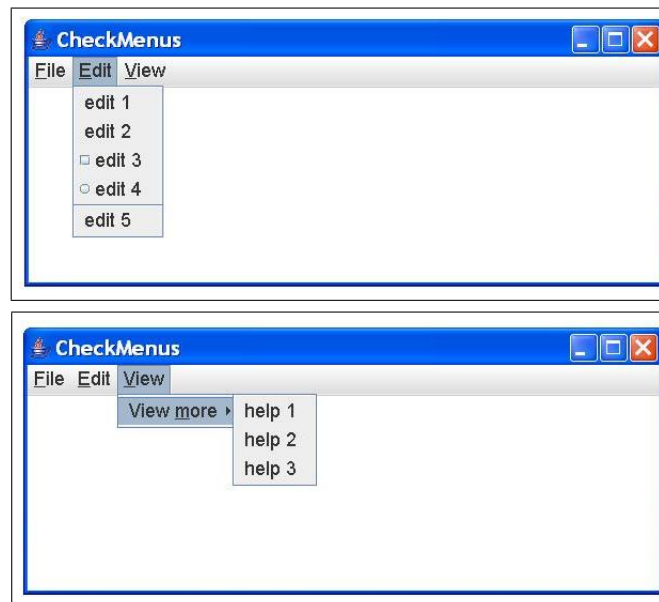
Menu items can also be accessed with special key strokes. We may specify that the 'exit' menu item in the 'file' menu can be accessed using the 'Control-q' key stroke.

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

CheckMenus

```
1  import java.awt.*;
2  import javax.swing.*;
3  import java.util.*;
4
5  public class CheckMenus{
6      public static void main(String args[]){
7          JFrame frame=new JFrame("CheckMenus");
8          frame.setSize(600,250);
9          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10
11          JTextArea area=new JTextArea();
12          frame.add(area);
13          JEventQueue events=new JEventQueue();
14          frame.setJMenuBar(
15              events.jmenubar(new Font("Arial",Font.PLAIN,18),
16                  events.jmenu("File",'F',
17                      events.jmenuitem("New",'n',events.control('N')),
18                      events.jmenuitem("Open",'o',events.control('O')),
19                      events.jmenuitem("Save",'s',events.control('S')),
20                      events.jmenuitem("Save as",'a',events.control('A')),
21                      events.jmenuitem("Exit",'x',events.control('Q'))
22                  ),
23                  events.jmenu("Edit",'E',
24                      events.jmenuitem("edit 1"),
25                      events.jmenuitem("edit 2"),
26                      events.jcheckboxxmenuitem("edit 3"),
27                      events.jradiobuttonmenuitem("edit 4"),
28                      new JSeparator(),
29                      events.jmenuitem("edit 5")
30                  ),
31                  events.jmenu("View",'V',
32                      events.jmenu("View more",'M',
33                          events.jmenuitem("help 1"),
34                          events.jmenuitem("help 2"),
35                          events.jmenuitem("help 3")
36                      )
37                  )
38              );
39          frame.setVisible(true);
40          for(;;){
41              EventObject event=events.waitEvent();
42              String name=events.getName(event);
43              area.append("Event: "+name+"\n");
44              if(name.equals("Exit"))System.exit(0);
45          }
46      }
47  }
48 }
```





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.

JEventQueue	
static boolean <code>isWindowClosing(EventObject e)</code>	Return true if it is a windows closing event
static boolean <code>isWindowResizing(EventObject e)</code>	Return true if the window has been resized

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

CheckClose

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4  import java.util.*;
5
6  public class CheckClose{
7      public static void main(String args[]){
8          JFrame frame=new JFrame("CheckClose");
9          frame.setSize(600,600);
10         frame.setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
11         frame.setVisible(true);
12
13         JEventQueue events=new JEventQueue();
14         events.listenTo(frame,"Frame");
15         while(true){
16             EventObject event = events.waitEvent();
17             String name=events.getName(event);
18             if(events.isWindowClosing(event)){
19                 int i=JOptionPane.showConfirmDialog(frame,"Ready to Close?");
20                 if(i==0)System.exit(0);
21             }
22             if(events.isWindowResizing(event))
23                 System.out.println("resizing! "+frame.getSize());

```

```
24     }  
25   }  
26 }
```

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 import 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.

Frame loop

```
while(true){  
    canvas.startBuffer();  
    canvas.clear();  
    // draw something on the canvas  
  
    canvas.endBuffer();  
    canvas.sleep(20);  
}
```

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.

SlideShow

```
1  import java.awt.*;
2  import java.awt.image.*;
3  import javax.swing.*;
4
5  public class SlideShow{
6      public static void main(String args[]){
7          JFrame frame=new JFrame("SlideShow");
8          frame.setSize(600,600);
9          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10         JCanvas canvas=new JCanvas();
11         frame.add(canvas);
12         frame.setVisible(true);
13
14         String[] files={"res/canal.jpg","res/heron.jpg","res/fox.jpg"};
15         int i=0;
16         while(true){
17             BufferedImage im=canvas.loadImage(files[i]);
18             int h=canvas.getHeight(); //window height
19             int w=canvas.getWidth(); // window width
20             double h1=im.getHeight(); //image height
21             double w1=im.getWidth(); //image width
22             double f=Math.min(h/h1,w/w1); // Scale factor
23             if(f<0.2)f=0.2;
24             int h2=(int) (h1*f), w2=(int) (w1*f);
25             canvas.startBuffer();
26             canvas.clear();
27             canvas.drawScaledImage(im,(w-w2)/2,(h-h2)/2,w2,h2);
28             canvas.endBuffer();
29             i=(i+1)%files.length;
30             canvas.sleep(2000);
31         }
32     }
33 }
```

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.

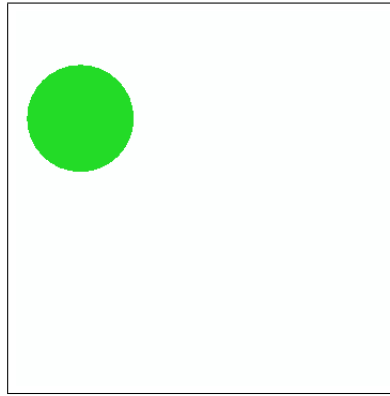


Image of a green ball in a room with four walls.

BounceBall1

```
1  import java.awt.*;
2  import java.awt.geom.*;
3  import javax.swing.*;
4
5  public class BounceBall1{
6      public static void main(String args[]){
7          JFrame frame=new JFrame("BounceBall1");
8          frame.setSize(600,600);
9          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10         JCanvas canvas=new JCanvas();
11         frame.add(canvas);
12         frame.setVisible(true);
13
14         int vx=8,vy=10,x=0,y=50,d=50;
15         while(true){
16             int mx=canvas.getWidth();
17             int my=canvas.getHeight();
18             canvas.startBuffer();
19             canvas.clear();
20             canvas.setPaint(Color.green);
21             x+=vx; y+=vy;
22             if(x<0){x=0;vx=-vx;}
23             if(y<0){y=0;vy=-vy;}
24             if(x+d>mx){x=mx-d;vx=-vx;}
25             if(y+d>my){y=my-d;vy=-vy;}
26             canvas.fillOval(x,y,d,d);
27             canvas.endBuffer();
28             canvas.sleep(20);
29         }
30     }
31 }
```

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.

BounceBall3

```
1  import java.awt.*;
2  import java.awt.geom.*;
3  import javax.swing.*;
4
5  public class BounceBall3{
6      static int vmax=10;
7      public static void main(String args[]){
8          JFrame frame=new JFrame("BounceBall3");
9          frame.setSize(600,600);
10         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11         JCanvas canvas=new JCanvas();
12         frame.add(canvas);
13         frame.setVisible(true);
14
15         Ball list[]=new Ball[6];
16         list[0]=new Ball(Color.red,50,15);
17         list[1]=new Ball(Color.blue,200,5);
18         list[2]=new Ball(Color.green,100,10);
19         list[3]=new Ball(Color.yellow,50,15);
20         list[4]=new Ball(Color.lightGray,25,20);
21         list[5]=new Ball(Color.cyan,75,10);
22
23         while(true){
24             int mx=canvas.getWidth();
25             int my=canvas.getHeight();
26             for(Ball b:list)b.move(mx,my);
27             canvas.startBuffer();
28             canvas.clear();
29             for(Ball b:list)b.draw(canvas);
30             canvas.setPaint(Color.black);
31             canvas.endBuffer();
32             canvas.sleep(10);
33         }
34     }
35 }
36
37 class Ball{
38     int x=0,y=0,d=50,vmax=10,vx=vmax,vy=vmax;
39     int mx=500,my=500;
40     Color color;
41     Ball(Color c,int d,int vmax){color=c;this.d=d;this.vmax=vmax;}
42     int random(int v){return (int) (Math.random()*v);}
43     int adj(int v){ return Math.max(Math.min(v+random(3)-1,vmax),-vmax); }
44     void move(int mx,int my){
45         this.mx=mx;this.my=my;
46         x+=vx; y+=vy;
47         if(x<0&&vx<0){x=0;vx=adj(-vx);}
48         if(y<0&&vy<0){y=0;vy=adj(-vy);}
49         if(x+d>mx&&vx>0){x=mx-d;vx=adj(-vx);}
50         if(y+d>my&&vy>0){y=my-d;vy=adj(-vy);}
51         if(vx==0)vx=1;if(vy==0)vy=1;
52     }
53     int getX(){return x;}
54     int getY(){return y;}
55     void draw(JCanvas canvas){
56         canvas.setPaint(color);
57         canvas.fillOval(x,y,d,d);
58     }
59 }
60 }
```

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

$$h(t) = 4 * h_1 * (t/t_1 - t^2/t_1^2)$$

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

$$h(t) = a * t^2 + b * t + c$$

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



BounceBall

```
1  import java.awt.*;
2  import java.awt.geom.*;
3  import javax.swing.*;
4
5  public class BounceBall{
6      static double sqr(double d){return d*d;}
7      public static void main(String args[]){
8          JFrame frame=new JFrame("BounceBall");
9          frame.setSize(600,600);
10         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11         JCanvas canvas=new JCanvas();
12         frame.add(canvas);
13         frame.setVisible(true);
14
15         int    d=100;    // ball diameter
16         int    h0=500;   // ground depth from top of canvas
17         int    h1=450-d; // height above ground (bottom of ball)
18         int    h2=35;    // height below: deformation at bounce
19         int    x=200;    // horizontal position of ball
20         double t=0;      // time
21         double t1=300;   // time above ground
22         double t2=340;   // full circle
23         while(true){
24             t++;if(t>t2)t=0;
25             int h=0; //height
```

```

26         if(t<t1)
27             h=(int) (4*h1*(t/t1-sqr(t)/sqr(t1)));
28         else
29             h=(int) (-4*h2*((t-t1)/(t2-t1)-sqr(t-t1)/sqr(t2-t1)));
30         canvas.startBuffer();
31         canvas.clear();
32         canvas.setPaint(Color.gray);
33         canvas.fillRect(0,h0,x*2+d,20);
34         int d1= (int) Math.max(0,-h); //deformation
35         canvas.setPaint(Color.red);
36         canvas.fillOval(x,h0-h-d,d+d1,d-d1);
37         canvas.endBuffer();
38         canvas.sleep(10);
39         if(t==50) canvas.writeToImage("bounceballA.jpg",600,600);
40         if(t==310) canvas.writeToImage("bounceballB.jpg",600,600);
41     }
42 }
43 }

```

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 is 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

```

1  import java.awt.Rectangle;
2  import java.awt.image.BufferedImage;
3  import javax.swing.JFrame;
4
5  public class RollBackground{
6      public static void main(String args[]){
7          JFrame frame=new JFrame("Roll Background");
8          frame.setSize(600,600);
9          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10         JCanvas canvas=new JCanvas();
11         frame.add(canvas);
12         frame.setVisible(true);
13         //
14         BufferedImage image=canvas.loadImage("res/oceano.gif");
15         int iHeight=image.getHeight();
16         int height=canvas.getHeight()-20;
17         int width=canvas.getWidth()-20;
18         BufferedImage tiles=canvas.tileImage(image,width,height+iHeight);
19         int j=0;
20         while(true){
21             canvas.startBuffer();
22             canvas.clear();
23             canvas.setClip(10, 10, width, height);
24             canvas.drawImage(tiles,10,10-j);
25             j++;;if(j>=iHeight)j=0;
26             canvas.endBuffer();
27             canvas.sleep(10);
28         }
29     }
30 }

```

6 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.

JCanvas	Draw or fill general shapes
<code>void <u>draw</u>(Shape s)</code>	draw the outline of shape <code>s</code>
<code>void <u>draw</u>(Shape s,int x,int y)</code>	Draw a shape, moved <code>x</code> to the right and <code>y</code> down
<code>void <u>fill</u>(Shape s)</code>	Fill a shape
<code>void <u>fill</u>(Shape s,int x,int y)</code>	Fill a shape, moved <code>x</code> to the right and <code>y</code> down
Shape	Shape constructors
<code>new <u>Line2D.Double</u>(double x1, double y1, double x2, double y2)</code>	A line from (x1,y1) to (x2,y2)
<code>new <u>Rectangle2D.Double</u>(double x, double y, double w, double h)</code>	A rectangle with top-left at (x,y), width <code>w</code> and height <code>h</code>
<code>new <u>RoundRectangle2D.Double</u>(double x, double y, double w, double h, double aw, double ah)</code>	A rectangle with top-left at (x,y), width <code>w</code> and height <code>h</code> . Corners are arcs with width <code>aw</code> and height <code>ah</code> .
<code>new <u>Ellipse2D.Double</u>(double x, double y, double w, double h)</code>	An ellipsis rectangle with top-left at (x,y), width <code>w</code> and height <code>h</code> .
<code>new <u>CubicCurve2D.Double</u>(double x1, double y1, double cx1, double cy1, double cx2, double cy2, double x2, double y2)</code>	A cubic curve from (x1,y1) to (x2,y2) with control points (cx1,cy1) and (cx2,cy2).
<code>new <u>QuadCurve2D.Double</u>(double x1, double y1, double cx, double cy, double x2, double y2)</code>	A quadratic curve from (x1,y1) to (x2,y2) with control point (cx,cy).
<code>new <u>GeneralPath</u>()</code>	A general path - see below
GeneralPath	General path
<code>void <u>moveTo</u>(double x,double y)</code>	The next part of the path will continue from (x,y).
<code>void <u>lineTo</u>(double x,double y)</code>	Draw a line from the current point to (x,y).
<code>void <u>curveTo</u>(float x1, float y1, float x2, float y2, float x3, float y3)</code>	Draw a curve from the current point to (x3,y3) using (x1,y1) and (x2,y2) as control points.
<code>void <u>quadTo</u>(float x1, float y1, float x2, float y2)</code>	Draw a quadratic curve from the current point to (x2,y2) using (x1,y1) as control point.
<code>void <u>append</u>(Shape s, boolean connect)</code>	Appends the shape to the path, possibly connecting it to the existing path.
<code>void <u>closePath</u>()</code>	Adds a straight line from the current point to the coordinates of the last <code>moveTo</code>

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.

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

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.

JCanvas	Transparency
<code>Composite getComposite()</code>	Return the current composition
<code>void setComposite(Composite comp)</code>	set the composition

A `Composition` is constructed as

```
AlphaComposite.getInstance(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.

CheckFontCanvas

```

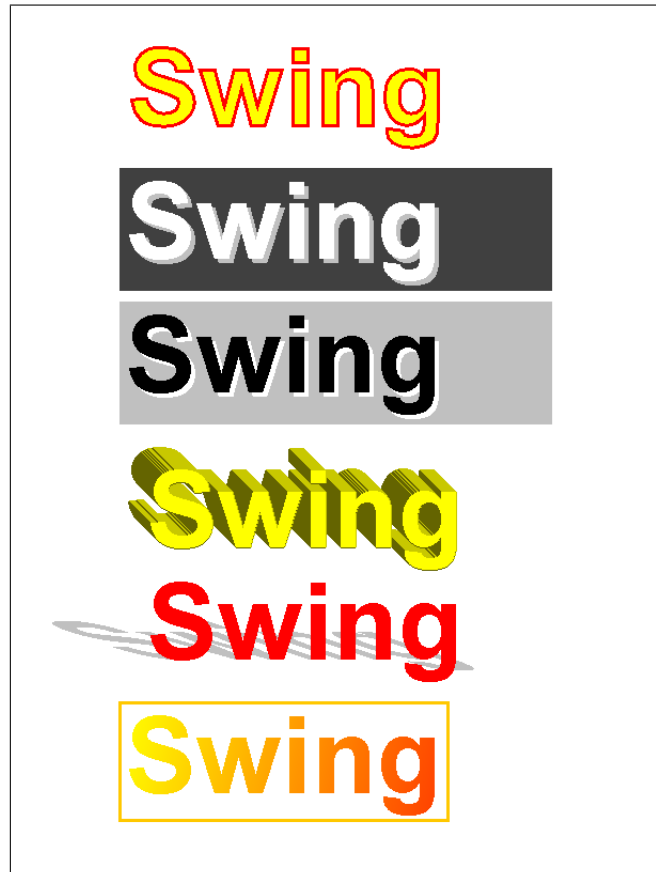
1  import javax.swing.*;
2  import java.awt.*;
3  import java.awt.font.*;
4  import java.awt.geom.*;
5
6  public class CheckFontCanvas{
7      public static void main(String args[]){
8          JFrame frame=new JFrame("CheckFontCanvas");
9          frame.setSize(800,800);
10         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11         JCanvas canvas=new JCanvas();

```

```

12     frame.add(canvas);
13     frame.setVisible(true);
14     int x,y,y1,sz=100,wd=400;
15
16     Font ft1=new Font("Arial",Font.BOLD,sz);
17     String s="Swing";
18     FontMetrics fm=canvas.getFontMetrics(ft1);
19     int w1=fm.stringWidth(s);
20     int dc=fm.getMaxDescent();
21
22     x=100;y=25;y1=y+sz-dc;
23     canvas.setFont(ft1);
24     canvas.setStroke(new BasicStroke(3));
25     canvas.setPaint(Color.yellow);
26     canvas.drawString(s,x+3,y1);
27     canvas.setPaint(Color.red);
28     canvas.drawOutline(s,x+3,y1);
29
30     x=100;y=150;y1=y+sz-dc;
31     canvas.setPaint(Color.darkGray);
32     canvas.fillRect(x-5,y-5,wd+5,sz+15);
33     canvas.setPaint(Color.lightGray);
34     canvas.drawString(s,x+4,y1+4);
35     canvas.setPaint(Color.white);
36     canvas.drawString(s,x,y1);
37
38     x=100;y=275;y1=y+sz-dc;
39     canvas.setPaint(Color.lightGray);
40     canvas.fillRect(x-5,y-5,wd+5,sz+15);
41     canvas.setPaint(Color.white);
42     canvas.drawString(s,x+4,y1+4);
43     canvas.setPaint(Color.black);
44     canvas.drawString(s,x,y1);
45
46     x=100;y=400;y1=y+sz-dc;
47     Color top_color = new Color(200, 200, 0);
48     Color side_color = new Color(100, 100, 0);
49     for (int i = 0; i < 22; i++) {
50         canvas.setPaint(top_color);
51         canvas.drawString(s, x+i, y1+i-1);
52         canvas.setPaint(side_color);
53         canvas.drawString(s, x+i-1,y1+i);
54     }
55     canvas.setPaint(Color.yellow);
56     canvas.drawString(s, x+20, y1+20);
57
58     x=100;y=525;y1=y+sz-dc;
59     AffineTransform origTransform = canvas.getTransform();
60     canvas.setPaint(Color.lightGray);
61     canvas.translate(x+20,y1);
62     canvas.shear(3, 0);
63     canvas.scale(1, 0.5);
64     canvas.drawString(s, 0, 0);
65     canvas.setTransform(origTransform);
66     canvas.setPaint(Color.red);
67     canvas.drawString(s,x+20,y1);
68
69     x=100;y=650;y1=y+sz-dc;
70     canvas.setPaint(new GradientPaint(x,y,Color.yellow,x+400,y1+dc,Color.red));
71     canvas.drawString(s,x,y1);
72     canvas.setPaint(Color.orange);
73     canvas.drawRect(x-5,y-5,w1+10,sz+10);
74 }
75 }

```



References

- [1] An Introduction to Graphical User Interfaces with Java Swing
Paul Fischer
Addison Wesley, 2005
- [2] Programmer's Guide to the Java 3D API
Sun Microsystems, 2001
- [3] Killer Game Programming in Java
Andrew Davison
O'Reilly, 2005
- [4] Object-Oriented Programming featuring Graphical Applications in Java
Michael J Laszlo
Addison Wesley, 2002
- [5] JFC Swing Tutorial, The: A Guide to Constructing GUIs
Kathy Walrath, Mary Campione, Alison Huml, Sharon Zakhour
Addison Wesley, 2004

Index

- Border, 28
- BorderFactory
 - createCompoundBorder, 28
 - createEmptyBorder, 28
 - createEtchedBorder, 28
 - createLineBorder, 28
 - createLoweredBevelBorder, 29
 - createMatteBorder, 28
 - createRaisedBevelBorder, 29
 - createTitledBorder, 29
- BufferedImage
 - getHeight, 11
 - getWidth, 11
- ButtonGroup()
 - add, 21
 - ButtonGroup, 21
- EventObject, 33
- Font
 - Font, 10
- GeneralPath
 - append, 52
 - closePath, 52
 - curveTo, 52
 - lineTo, 52
 - moveTo, 52
 - quadTo, 52
- IconImage
 - IconImage, 11
- JBox
 - add, 31
 - getScreenHeight, 14
 - getScreenWidth, 14
 - hbox, 13
 - hglue, 14
 - hspace, 14
 - removeAll, 31
 - setSize, 14
 - vbox, 14
 - vglue, 14
 - vspace, 14
- JButton
 - JButton, 18
 - setEnabled, 19
 - setHorizontalAlignment, 19
 - setHorizontalTextPosition, 19
 - setIconTextGap, 19
 - setText, 19
 - setVerticalAlignment, 19
 - setVerticalTextPosition, 19
- JCanvas
 - clear, 7
 - cropImage, 11
 - draw, 52
 - drawArc, 8
 - drawDashedLine, 8
 - drawImage, 10
 - drawLine, 8
 - drawOutline, 10
 - drawOval, 8
 - drawRect, 8
 - drawRoundRect, 8
 - drawScaledImage, 10, 11
 - drawString, 10
 - endBuffer, 7
 - fill, 52
 - fillArc, 8
 - fillOval, 8
 - fillRect, 8
 - fillRoundRect, 8
 - getComposite, 53
 - getFont, 10
 - getHeight, 8
 - getPaint, 9
 - getStroke, 9
 - getTransform, 53
 - getWidth, 8
 - loadImage, 11
 - print, 7
 - rotate, 53
 - rotateImage, 11
 - scale, 53
 - scaleImage, 11
 - setBackground, 7
 - setClip, 8
 - setComposite, 53
 - setFont, 10
 - setPaint, 9
 - setStroke, 9
 - setTransform, 53
 - shear, 53
 - sleep, 7
 - startBuffer, 7

- storeImage, 11
- tileImage, 11
- translate, 53
- writeToImage, 7
- JCheckBox
 - isSelected, 20
 - JCheckBox, 20
 - setSelected, 20
- JComboBox
 - addItem, 26
 - getItemAt, 26
 - getItemCount, 26
 - getSelectedItem, 26
 - JComboBox, 26
 - setEditable, 26
 - setEnabled, 26
 - setMaximumRowCount, 26
- JComponent
 - getHeight, 14
 - getWidth, 14
 - JBox.hbox, 13
 - JBox.vbox, 13
 - JButton, 13
 - JCanvas, 13
 - JCheckBox, 13
 - JComboBox, 13
 - JFormattedTextField, 13
 - JLabel, 13
 - JList, 13
 - JRadioButton, 13
 - JScrollPane, 13
 - JSeparator, 13
 - JSlider, 13
 - JSpinner, 13
 - JSplitPane, 13
 - JTappedPane, 13
 - JTextArea, 13
 - JTextField, 13
 - JToggleButton, 13
 - ProgressBar, 13
 - requestFocusInWindow, 30
 - setBackground, 17
 - setBorder, 28
 - setCursor, 30
 - setFont, 17
 - setForeground, 17
 - setMaximumSize, 14
 - setMinimumSize, 14
 - setOpaque, 17
 - setPreferredSize, 14
 - setToolTipText, 30
- JEventQueue
 - control, 43
 - controlShift, 43
 - getKeyChar, 39
 - getKeyText, 39
 - getMouseButton, 41
 - getMouseClickCount, 41
 - getMouseX, 41
 - getMouseY, 41
 - getName, 33
 - hasEvent, 33
 - isActionEvent, 39
 - isActionKey, 39
 - isActionPerformed, 39
 - isDocumentEvent, 39
 - isKeyEvent, 39
 - isKeyPressed, 39
 - isKeyReleased, 39
 - isKeyTyped, 39
 - isMouseClicked, 41
 - isMouseEvent, 41
 - isMousePressed, 41
 - isMouseReleased, 41
 - isWindowClosing, 45
 - isWindowResizing, 45
 - jcheckboxboxmenuItem, 43
 - JEventQueue, 33
 - jmenu, 43
 - jmenubar, 43
 - jmenuItem, 43
 - jradiobuttonmenuItem, 43
 - JSeparator, 43
 - listenTo, 33
 - sleep, 42
 - startTimer, 42
 - stopTimer, 42
 - waitEvent, 33
- JFormattedTextField
 - getValue, 22
 - JFormattedTextField, 22
 - setValue, 22
- JFrame
 - add, 4
 - JFrame, 4
 - repaint, 31
 - setDefaultCloseOperation, 4
 - setIconImage, 4
 - setJMenuBar, 4
 - setLocation, 4

- setResizable, 4
 - setSize, 4
 - setTitle, 4
 - setVisible, 4
 - validate, 31
- JLabel
 - JLabel, 15, 17
 - setHorizontalAlignment, 18
 - setHorizontalTextPosition, 18
 - setText, 18
 - setVerticalAlignment, 18
 - setVerticalTextPosition, 18
- JList
 - getSelectedIndex, 26
 - getSelectedValue, 26
 - JList, 26
- JMenu, 43
- JMenuBar, 43
- JMenuItem, 43
- JProgressBar
 - JProgressBar, 25
 - setBorderPainted, 25
 - setStringPainted, 25
 - setValue, 25
- JRadioButton
 - isSelected, 20
 - JRadioButton, 20
 - setSelected, 20
- JScrollPane
 - JScrollPane, 28
 - setHorizontalScrollBarPolicy, 28
 - setVerticalScrollBarPolicy, 28
- JSlider
 - getValue, 24
 - JSlider, 23
 - setExtent, 24
 - setInverted, 24
 - setMajorTickSpacing, 24
 - setMinorTickSpacing, 24
 - setPaintLabels, 24
 - setPaintTicks, 24
 - setPaintTrack, 24
 - setSnapToTicks, 24
 - setValue, 24
- JSpinner
 - getValue, 24
 - JSpinner, 24
 - setValue, 24
- JSplitPane
 - getDividerLocation, 28
 - JSplitPane, 27
 - setDividerLocation, 28
 - setDividerSize, 28
 - setResizeWeight, 28
- JTextArea
 - append, 22
 - getLineCount, 22
 - getText, 22
 - isEditable, 22
 - JTextArea, 21
 - setEditable, 22
 - setHorizontalAlignment, 22
 - setLineWrap, 22
 - setText, 22
 - setWrapStyleWord, 22
- JTextField
 - getText, 21
 - isEditable, 21
 - JTextField, 21
 - setEditable, 21
 - setHorizontalAlignment, 21
 - setText, 21
- JToggleButton
 - isSelected, 20
 - JToggleButton, 20
 - setSelected, 20
- Paint
 - Color, 9
 - GradientPaint, 10
 - TexturePaint, 10
- Shape
 - CubicCurve2D.Double, 52
 - Ellipse2D.Double, 52
 - GeneralPath, 52
 - Line2D.Double, 52
 - QuadCurve2D.Double, 52
 - Rectangle2D.Double, 52
 - RoundRectangle2D.Double, 52
- SpinnerModel
 - SpinnerDateModel, 24
 - SpinnerListModel, 24
 - SpinnerNumberModel, 24
- Stroke
 - BasicStroke, 9