Scrolling

Scrolling needed when:
- Content is too big to fit in a given area

JScrollArea
- Defines the given area
- Like all containers, clips out descendant components outside its bounds
- Displays scroll bars (if necessary)
- Moves the location of its child component

Like a viewport through which you can see a sub-rectangle of a component

Usage:
- instead of
  - add(component)
- use:
  - add(new JScrollArea(component))

Graphics

Each component in a hierarchy is drawn with a paint method
- `paint`
  - `paintComponent`
    - Paints the background if the component is opaque
    - You often override this method
  - `paintBorder`
  - `paintChildren`

By overriding `paintComponent`, you can do custom painting.

When is `paintComponent` called?
- System-triggered
  - becomes visible
  - component is resized
  - something obscured it and no longer does
- App-triggered (for example, checkbox was checked)
  - Your code calls one of:
    - `repaint()` // invalidates entire component
    - `repaint(x, y, width, height)` // invalidates rectangular area of component

JScrollArea demo

```java
public class ScrollTest extends JFrame {
    public ScrollTest() {
        super("ScrollTest");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(300, 100);
        setLayout(new BorderLayout());

        JPanel textPanel = new JPanel();
        add(textPanel, BorderLayout.CENTER);
        JTextArea memoDisplay = new JTextArea("initial text", 20, 30);
        textPanel.add(new JScrollPane(memoDisplay));
        pack();
    }
}
```
Repaint manager

Repaint manager accumulates repaint requests
- Only repaints components that intersect area that needs repainting
- Clips painting to bounds of component and all its ancestors

Graphics

paintComponent is passed a Graphics parameter
- Graphics object is an object that can be drawn (painted) to

Has lots of state information:
- color: set to the component’s foreground property
- font: set to the component’s font property
- translation: set so that (0, 0) is at the upper-left corner of the component
- clip rectangle: set to the area of the component that needs repainting

Has lots of methods:
- drawLine
- drawRect/fillRect (and Oval and RoundRect, and Arc)
- drawString
- ...
- accessors for state info
- mutators for state info

Drawing is done on a coordinate system

Top-left of a component is (0, 0)
- coordinates increase down and to the right
- Each coordinate is one pixel (picture element)

Coordinates are infinitely thin and lie between pixels

Drawing methods

Drawing:
- Drawn with a one-pixel pen that hangs down and to the right
- drawLine(x1, y1, x2, y2)
  - Draws line from point (x1, y1) to (x2, y2)
- drawRect(x, y, width, height)
  - Draws rectangle with top-left at (x, y)
- drawRoundRect(x, y, width, height, radius)
  - Draws rectangle with rounded corners of given radius
- drawOval(x, y, width, height)
  - Draws oval inscribed in given rectangle
- drawArc(x, y, width, height, startAngle, arcAngle)
  - Draws an arc inscribed in given rectangle with given angles

Filling:
- No pen, instead colors inside the lines
- fillRect, fillRoundRect, fillOval, fillArc
Graphics Example

```java
GraphicsTest() {
    super("GraphicsTest");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    JPanel p = new JPanel() {
        public void paintComponent(Graphics g) {
            super.paintComponent(g); // to erase, if opaque
            g.drawOval(0, 0, 100, 100);
            g.fillRect(30, 30, 2, 2);
            g.fillRect(70, 30, 2, 2);
            g.drawArc(20, 20, 60, 60, -30, -120);
        }
    };
    p.setSize(100, 100);
    p.setPreferredSize(new Dimension(100, 100));
    p.setForeground(Color.RED);
    p.setBackground(Color.YELLOW);
    add(p);
    pack();  // resize window to fit contents
}
```

Color

Usually represented in RGB
- One component each for red, green, and blue
- From none of a component to all of a component
  - Represented as one of:
    - float between 0 and 1
    - integer between 0 and 255

Examples
- (1.0, 0.0, 0.0) is pure red
- (1.0, 1.0, 1.0) is pure white
- (0.0, 0.0, 0.0) is pure black
- (x, x, x) is a gray
- (0.5, 0.0, 0.0) is a dark red
- (1.0, 1.0, 0.0) is pure yellow

Color class has many predefined colors:
- BLACK, WHITE, RED, BLUE

Timer

Swing provides a timer
- java.swing.Timer (there's a different Timer in java.util.Timer)
It is initialized with a number of milliseconds between firings, and an ActionListener:
- timer = new Timer(500, anActionListener)
Start the timer with:
- timer.start();
Stop the timer with:
- timer.stop();

Timer sample

```java
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class TimerTest extends JFrame {
    private int size;
    private int lightnessPercentage;
    private Timer timer;

    public static void main(String[] args) {
        TimerTest window = new TimerTest();
        window.setVisible(true);
    }

    TimerTest() {
        super("TimerTest");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        timer = new Timer(1000, new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                TimerTest.this.setTitle(System.currentTimeMillis() + "");
                if (lightnessPercentage == 100)
                  timer.stop();
            }
        });
        timer.start();
    }
}
```
MVC

Traditional Model/View/Controller
- Model holds the data and computations. No user interface.
  - It notifies listeners when it changes
- View displays the data
  - It listens to the model, updating itself as-needed
- Controller
  - Handles user interaction, updating the model

Swing:
- View and controller are bound together
- If you don't provide a model, one is created for you
  - By default, each checkbox is linked to its own model
  - Model actually holds listeners and notifies listeners on change

Biggest advantage
- Multiple views of the same data

How the model notifies the views
Change occurs to model
Model notifies views
- Either notification includes what has changed, or:
  - View must query back to model to find what has changed

Example model

```
ModelTest() {
    super("ModelTest");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(300, 100);
    setLayout(new FlowLayout());

    JCheckBox checkbox = new JCheckBox("Check me");
    add(checkbox);
    JCheckBox checkbox2 = new JCheckBox("Check me 2");
    add(checkbox2);
    checkbox.setModel(checkbox2.getModel());
}
```