Lecture 27 – Swing Control Programming

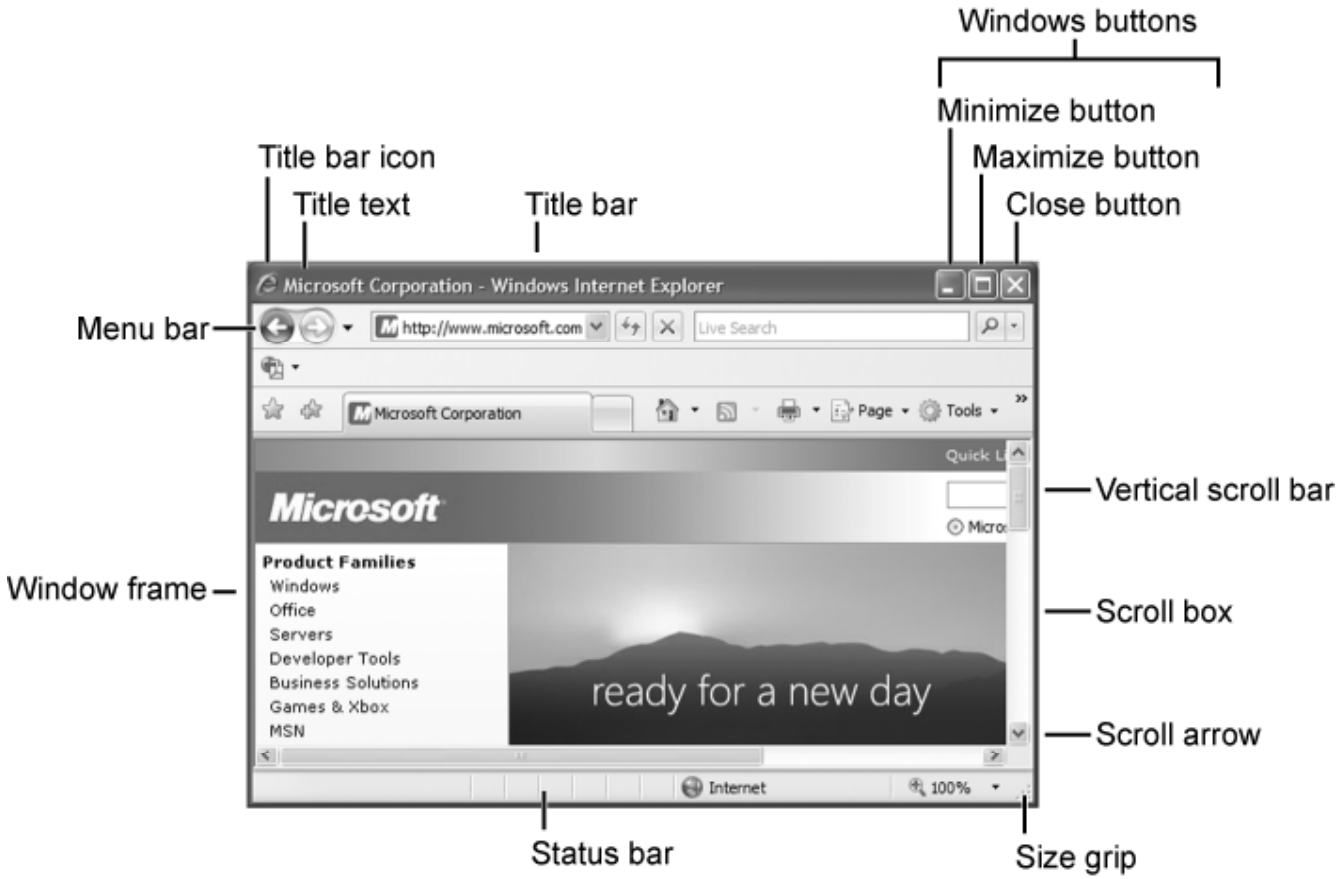
Lecturer: Gerald Weber

Overview

- Range of controls
- ‘hello worlds’ revisited
- Model v. view
- Tree control
- Customising controls

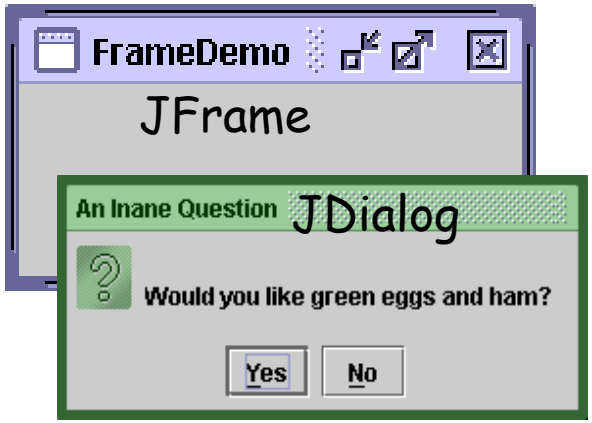
Components of a Window

- Windows XP Window Components

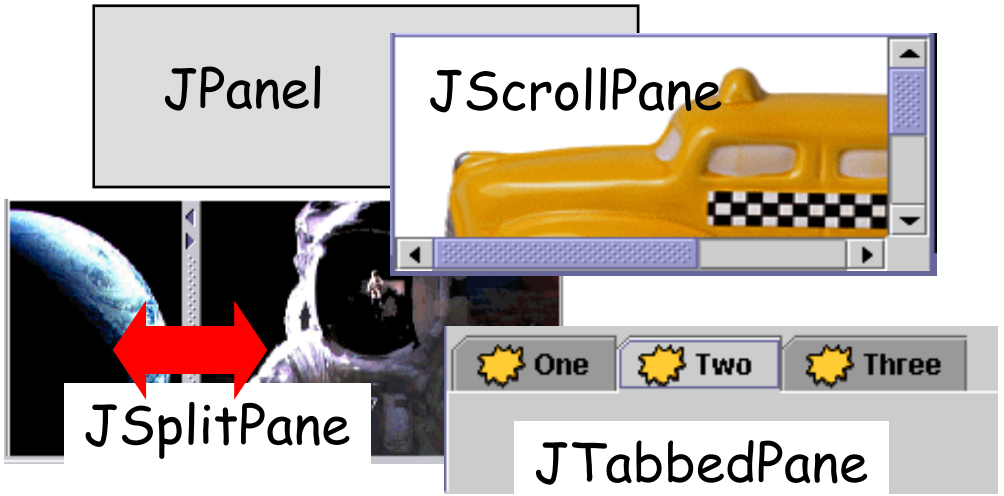


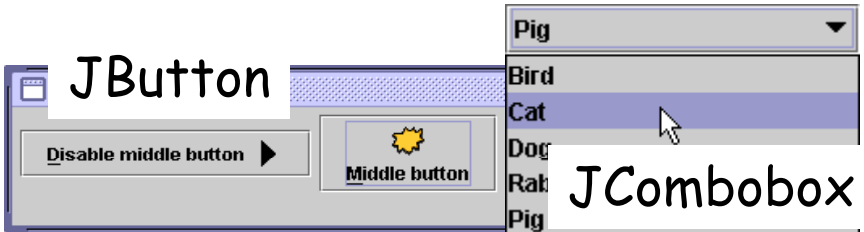
Swing Widgets

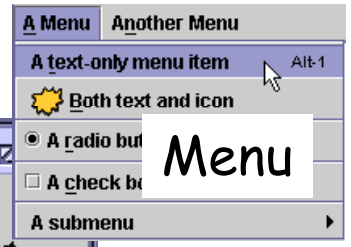
Top-Level Containers

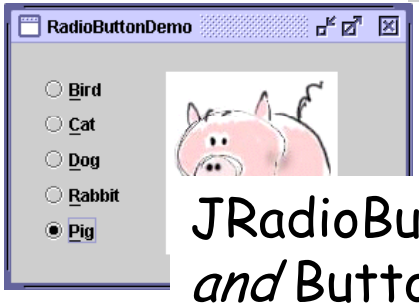


General-Purpose Containers

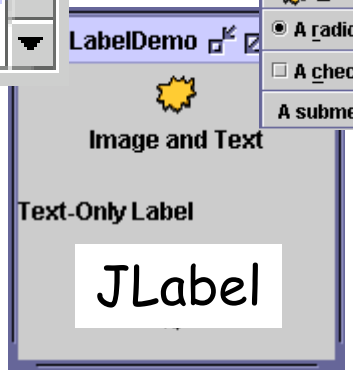




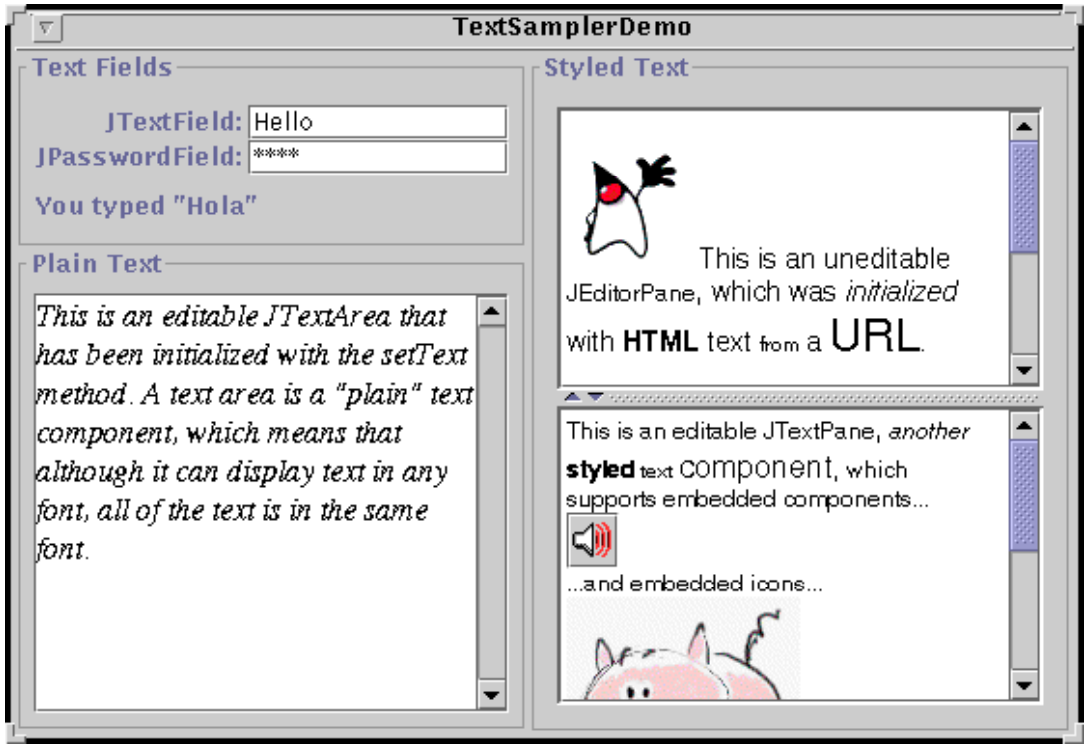


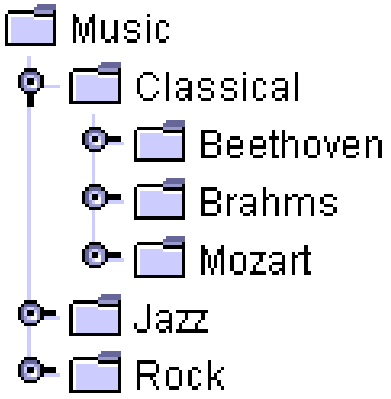


JRadioButton
and ButtonGroup



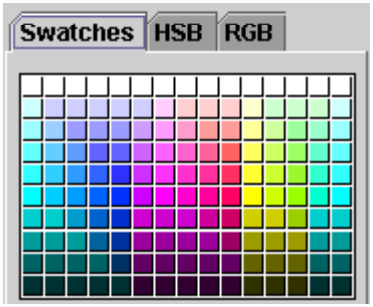
More Swing Widgets





First Name	Last Name	Favorite Food
Jeff	Dinkins	
Ewan	Dinkins	
Amy	Fowler	
Hania	Gajewska	
David	Geary	

JTable



JColorChooser

Another ‘hello world’

```
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JButton;
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JMenuBar;
import javax.swing.JMenu;
import javax.swing.JMenuItem;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.BorderLayout;

public class HelloWorldSwing extends JFrame implements ActionListener {
  JLabel label;

  public static void main(String[] args) {
    HelloWorldSwing frame=new HelloWorldSwing(args.length>0 ? args[0] : "Hello World");
  }

  public HelloWorldSwing(String myhello) {
....
```

Can also use “*” but nice to know what you’ve got

Extend the root class to *be* a JFrame, and have it provide our button event handler

Put the main action in the constructor

```
....
public HelloWorldSwing(String myhello) {
    label = new JLabel(myhello);
    getContentPane().add(label,BorderLayout.CENTER);
    label.setHorizontalAlignment(JLabel.CENTER);

    JButton b0 = new JButton("Nothing");
    b0.setActionCommand("click");
    b0.addActionListener(this);
    JButton b1 = new JButton("Close");
    b1.setActionCommand("closewindow");
    b1.addActionListener(this);
    getRootPane().setDefaultButton(b1);
    JPanel bp = new JPanel();
    getContentPane().add(bp,BorderLayout.PAGE_END);
    bp.add(b0);
    bp.add(b1);
....
```

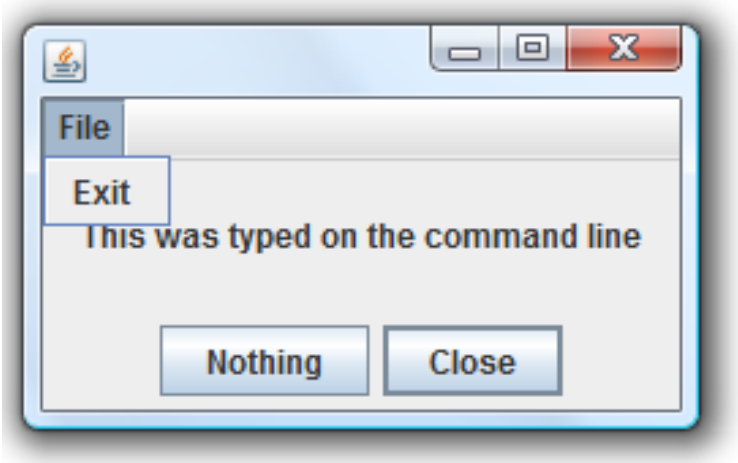
BorderLayout is default layout manager – more on these next lecture

Different user-defined command names for the different button; both go to the actionListener on our main object ('this')

Make a JPanel to contain the buttons; put it at the 'end' of the main content pane

```
    ....
    bp.add(b1);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(250,150);
    setLocationRelativeTo(null);

    final JMenu menu = new JMenu("File");
    final JMenuItem item1 = new JMenuItem("Exit");
    item1.addActionListener(this);
    menu.add(item1);
    JMenuBar menubar = new JMenuBar();
    menubar.add(menu);
    setJMenuBar(menubar);

    setVisible(true);
  }

  public void actionPerformed(ActionEvent e) {
    if (e.getActionCommand() == "closewindow") {
       setVisible(false);
       dispose(); }
    else
     label.setText("Click!");
       };
}
```

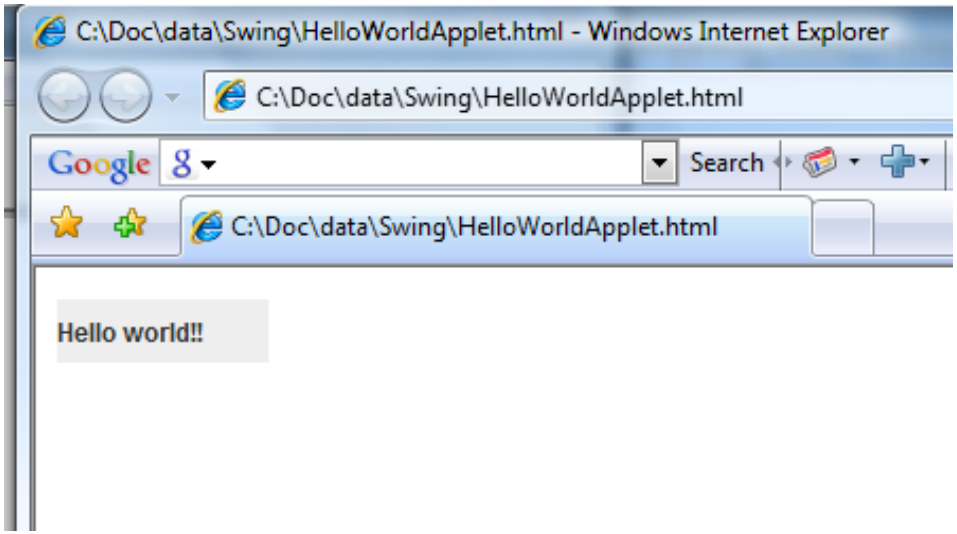
Centres frame relative to whole screen



Provide the method for button presses, sense the action command and run the appropriate code

Change label's data

A ‘hello world’ applet



The Java:

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

public class HelloWorldApplet extends JApplet {

   public void init() {
      getContentPane().add(new JLabel("Hello world!!"));
   }

}
```

Start the action in the applet’s init method

The HTML:

```
<applet code="HelloWorldApplet" width="100" height="30"></applet>
```

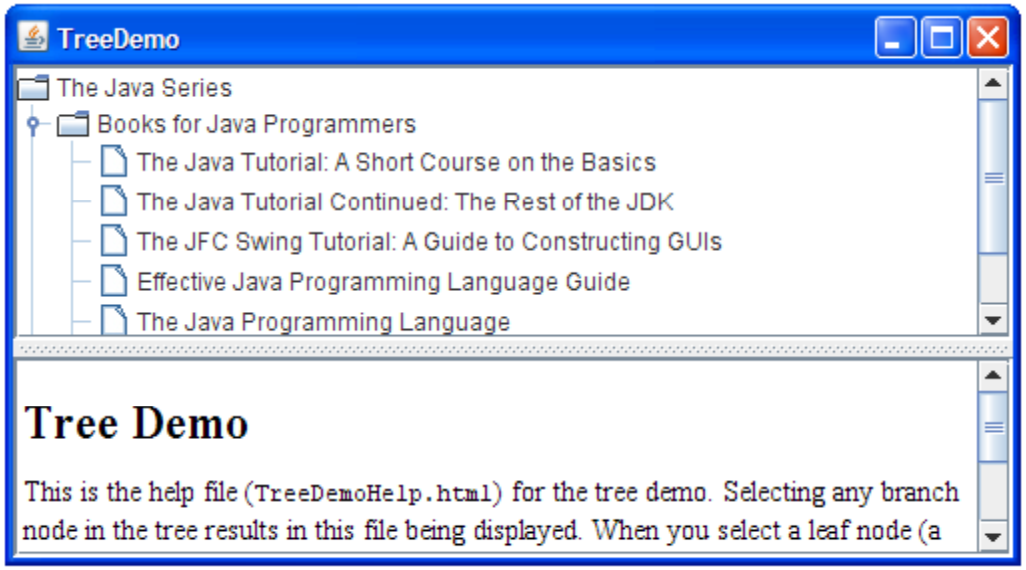
Separation of Model and View

- Use different classes for Model and View:
  - Model: the data that is presented by a widget
  - View: the actual presentation on the screen
- The data of a GUI component may be represented using several model objects, e.g. for
  - Displayed data (e.g. list items in **`JList`**: **`ListModel`**)
  - Widget state (e.g. selections in **`JList`**: **`ListSelectionModel`**)
- Advantages
  - Data independent of views, e.g. can be displayed in several views
  - Model concept is integrated with event concept: changes of the model trigger well-defined events

List Model Example

```
…
listModel = new DefaultListModel();
listModel.addElement("Alan Sommerer");
…
list = new JList(listModel);
…
public void actionPerformed(
  ActionEvent e) {
    int index =
      list.getSelectedIndex();
    listModel.remove(index);
}
…
```

Tree example




```
private JTree tree;
...
public TreeDemo() {
    ...
    DefaultMutableTreeNode
        top =
         new DefaultMutableTreeNode("The Java Series");
    createNodes(top);
    tree = new JTree(top);
    ...
    JScrollPane treeView = new JScrollPane(tree);
...
private void createNodes(DefaultMutableTreeNode top) {
    DefaultMutableTreeNode category = null;
    DefaultMutableTreeNode book = null;

    category = new DefaultMutableTreeNode("Books for Java
Programmers");
    top.add(category);

    book = new DefaultMutableTreeNode(new BookInfo
        ("The Java Tutorial: A Short Course on the Basics",
        "tutorial.html"));
    category.add(book);
```

Give the root node to a JTree to display, and give the JTree to a scroll pane to manage

About the tree controls

- The data (‘model) is a bunch of tree nodes (DefaultMutableTreeNode)
  - Any tree node has a .add that lets you give it children
  - You can construct the node with either
    - A string – that’ll be the label of that node
    - Any object, as long as the object implements toString
    - Any object if you then override the convertValueToText of the JTree that’ll be viewing it
- Construct a JTree with the root DefaultMutableTreeNode as its parameter to render the user control

Exampe from http://java.sun.com/docs/books/tutorial/uiswing/components/tree.html

Back to the tree example

- We create our custom BookInfo object (which implements toString for the sake of the JTree)
  - It also defines a URL, which we'll use when it's selected by the user

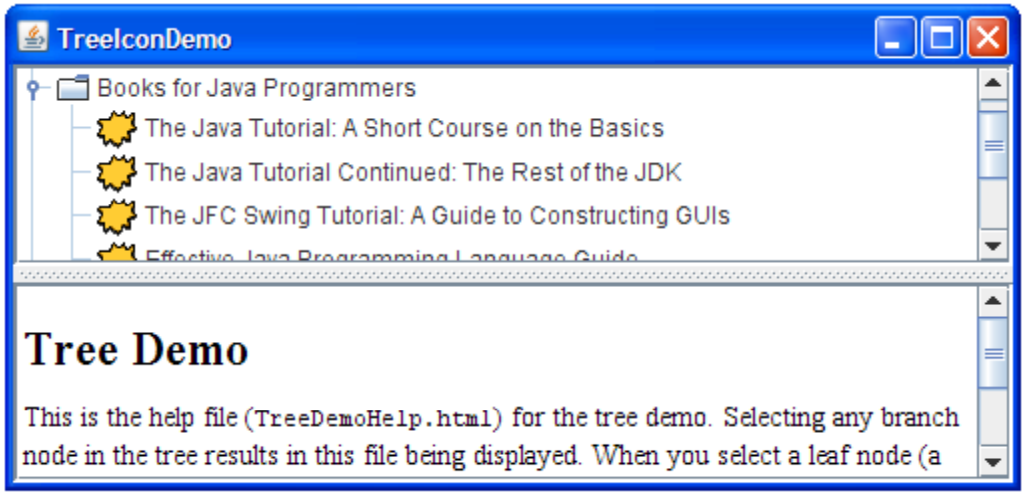
```
private class BookInfo {
        public String bookName;
        public URL bookURL;

        public BookInfo(String book, String filename) {
            bookName = book;
            bookURL = getClass().getResource(filename);
            if (bookURL == null) {
                System.err.println("Couldn't find file: "+ filename);
            }
        }

        public String toString() {
            return bookName;
        }
    }
```

Responding to action on the JTree

```
 //Where the tree is initialized:
    tree.getSelectionModel().setSelectionMode
            (TreeSelectionModel.SINGLE_TREE_SELECTION);

    //Listen for when the selection changes.
    tree.addTreeSelectionListener(this);
...
public void valueChanged(TreeSelectionEvent e) {
//Returns the last path element of the selection.
//This method is useful only when the selection model allows a single
selection.
    DefaultMutableTreeNode node = (DefaultMutableTreeNode)
                       tree.getLastSelectedPathComponent();

    if (node == null)
    //Nothing is selected.
    return;

    Object nodeInfo = node.getUserObject();
    if (node.isLeaf()) {
        BookInfo book = (BookInfo)nodeInfo;
        displayURL(book.bookURL);
    } else {
        displayURL(helpURL);
    }
}
```

Our leaf nodes are books (with URLs), so go ahead and display the selected URL content (in a separate pane)

Customising tree display



- You can create a cell renderer and change the icons it uses for LeafIcon, OpenIcon and ClosedIcon

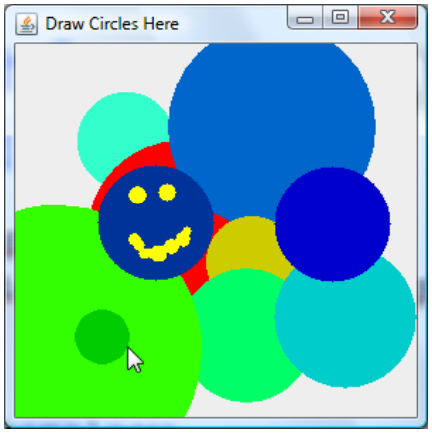
```
ImageIcon leafIcon =
createImageIcon("images/middle.gif");
if (leafIcon != null) {
    DefaultTreeCellRenderer renderer =
      new DefaultTreeCellRenderer();
    renderer.setLeafIcon(leafIcon);
    tree.setCellRenderer(renderer);
}
```

Drawing Your Own Components

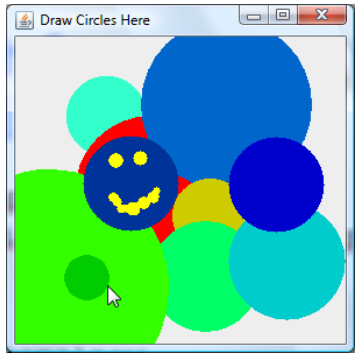
- **`paint()`**: called by the system whenever drawing is necessary
  - Calls **`paintComponent()`**, **`paintBorder()`**, **`paintChildren()`**
  - Override **`paintComponent()`** for custom look
- **`repaint()`**: call it if you need to trigger redrawing of components
  - You can give "dirty region" as argument
  - Asynchronously calls **`paint()`**
- **Transparency**: component does not draw all pixels in its bounds
  - Underlying components may need to be redrawn (slower)
  - Use **`setOpacity()`**: if true then you must draw **all** the component's pixels in **`paintComponent()`** (or screen garbage)
- **Double Buffering**: component (including children) is first drawn on off-screen bitmap, then off-screen bitmap is copied to screen
  - Reduces flickering
  - In Swing used by default

CirclePaint Part 1
Custom Component

```
public class Canvas extends JComponent {
   class Circle { float x, y, r; Color col; }
   java.util.Vector<Circle> circles
      = new java.util.Vector<Circle>();
   Circle current;

   public Canvas() {
      setOpaque(true);
      setBackground(Color.white);
      // add mouse listeners, see next slides…
   }

   public void paintComponent(Graphics g)  {
      // see next slides…
   }
}
```



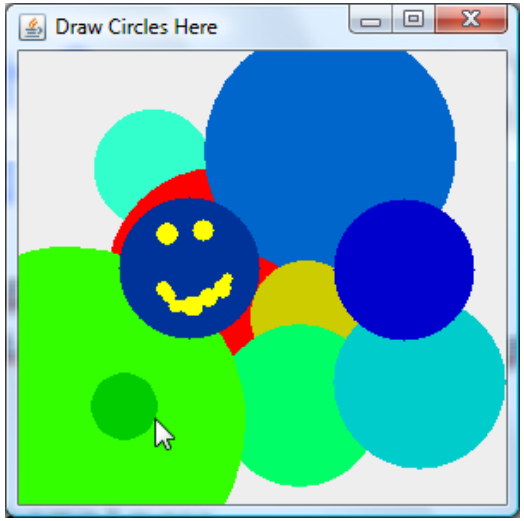
CirclePaint Part 2
Event Listeners




```
addMouseListener(new MouseAdapter() {
   public void mousePressed(MouseEvent e) {
       current = new Circle();
       current.x = e.getX();
       current.y = e.getY();
       current.col = CirclePaint.colorChooser.getColor();
   }
   public void mouseReleased(MouseEvent e) {
       if(current!=null) circles.add(current); }
   public void mouseExited(MouseEvent e) { current = null; }
});
addMouseMotionListener(new MouseMotionAdapter() {
   public void mouseDragged(MouseEvent e) {
       if(current==null) return;
       current.r =  (float)Math.sqrt(
            (e.getX() - current.x) * (e.getX() - current.x)
          + (e.getY() - current.y) * (e.getY() - current.y));
       repaint();
   }
});
```

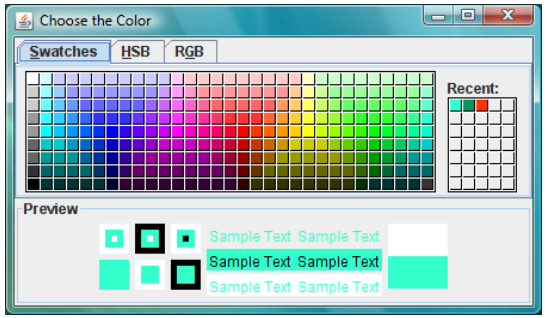
CirclePaint Part 3
Paint Method

```
public void paintComponent(Graphics g) {
   g.clearRect(0, 0, this.getWidth(), this.getHeight());
   for(Circle c : circles) {
        g.setColor(c.col);
        ((Graphics2D)g).fill((Shape)new
        java.awt.geom.Ellipse2D.Float(
          c.x-c.r, c.y-c.r, 2*c.r, 2*c.r));
   }
   if(current!=null) {
        g.setColor(current.col);
        ((Graphics2D)g).fill((Shape)new
        java.awt.geom.Ellipse2D.Float(
          current.x-current.r, current.y-current.r,
          2*current.r, 2*current.r));
   }
}
```



CirclePaint Part 4

Main Class




```
public class CirclePaint {
   public static JColorChooser colorChooser
        = new JColorChooser();

   public static void main(String[] args) {
        JFrame frame1 = new JFrame("Choose the Color");
        frame1.setSize(450, 260);
        frame1.setDefaultCloseOperation(
               JFrame.EXIT_ON_CLOSE);
        frame1.getContentPane().add(colorChooser);
        frame1.setVisible(true);

        JFrame frame2 = new JFrame("Draw Circles Here");
        frame2.setSize(300, 300);
        frame2.getContentPane().add(new Canvas());
        frame2.setVisible(true);
   }
}
```

Summary

- Swing provides a host of familiar components (aka ‘widgets’)
- You can override their default behaviours to achieve customisation
- Writing action listeners (interfaces and adapters) provides interesting dynamic capabilities