Java Foundation Classes

"It's spelled JFC,

but it's pronounced Swing!"

Java Foundation Classes

- *"In retrospect, AWT 1.0 was a good initial step…"* but it has several weaknesses
  - difficult to implement properly
    - coping with the vagaries of different platforms
  - restrictive rendering & event handling
  - difficult to extend
- "if at first you don't succeed…"
  - Netscape & IBM adapting technologies for Java
    - Netscape: IFC
    - IBM: Taligent work, experience, etc.

Java Foundation Classes

- Five APIs
  - builds on top of (part of) existing AWT
  - Java 2D
    - low-level graphics facility
    - licensed from Taligent
  - swing
    - 2nd-generation component set and GUI architecture
    - 100% Java
  - accessibility
    - screen reader, screen magnification, mouseless operation
  - drag & drop
    - incorporated from JavaBeans specification

Java Foundation Classes

- Java 2D
  - three new packages
    - java.awt.color
    - java.awt.font
    - java.awt.geom
  - some changes to existing packages
    - java.awt.image
    - java.awt
  - aim is to make *"new functionality available to your existing applications without rewriting old code"*

Java Foundation Classes

- java.awt
  - BasicStroke class provides linewidths
  - AlphaComposite class deals with transparency
  - Rectangle now extends java.awt.geom.Rectangle2D (+ other similar changes)
  - enhanced Font class lets you get at installed fonts
  - new Graphics2D class provides coordinate transformations, antialiasing, etc.
- java.awt.color
  - support for CMYK, Standard RGB, CIEXYZ
  - color profiles allow mapping between device-dependent and device-independent colors

Java Foundation Classes

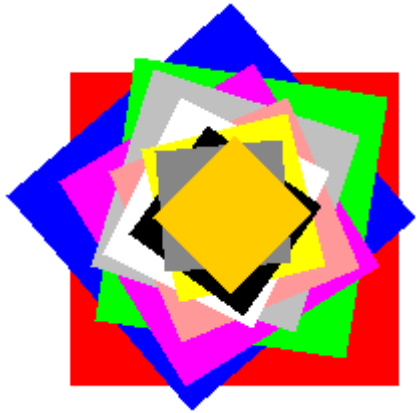

```
import java.awt.*;
import java.awt.geom.Ellipse2D;
import java.awt.font.TextLayout;
import javax.swing.*;

public class Oval extends JApplet
  {
  private final Object AntiAlias = RenderingHints.VALUE_ANTIALIAS_ON,
                       Rendering = RenderingHints.VALUE_RENDER_SPEED;

  public void paint(Graphics g)
    {
    Graphics2D g2 = (Graphics2D) g;
    BasicStroke bs = new BasicStroke (8.0F, BasicStroke.CAP_BUTT,
                                      BasicStroke.JOIN_MITER, 1.0F,
                                      new float [] {10.0F}, 0.0F);

    g2.setStroke (bs);
    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, AntiAlias);
    g2.setRenderingHint(RenderingHints.KEY_RENDERING, Rendering);
    Ellipse2D.Float e = new Ellipse2D.Float (10F, 10F, 330F, 100F);
    g2.setColor (Color.pink);
    g2.fill (e);
    g2.setColor (Color.red);
    g2.draw (e);
    g2.setColor (Color.black);
    Font f = new Font ("Helvetica", Font.BOLD, 36);
    TextLayout tl = new TextLayout ("Hello World", f,
                                    g2.getFontRenderContext ());
    tl.draw (g2, 40F, 75F);
    }
  }
```

Java Foundation Classes

- java.awt.geom
  - support for basic shapes
  - AffineTransform class
    - translation, rotation, shear, scale
  - Double, Float, Integer versions of Point
  - CubicCurve2D & QuadCurve2D class heirarchies
  - Area and GeneralPath classes
    - Area permits alteration of an existing geometric shape
- java.awt.image
  - color conversions, color-table modifications
  - new BufferedImage class exposes its pixmap
- java.awt.font
  - glyphs and text with multiple fonts & attributes
  - can be transformed like any shape

Java Foundation Classes

```
public void paint (Graphics g) {
  Dimension theSize = getSize ();
  Graphics2D g2 = (Graphics2D) g;
  g2.setRenderingHints (Graphics2D.ANTIALIASING, Graphics2D.ANTIALIAS_ON);

  GeneralPath p = new GeneralPath (1);
  p.moveTo (theSize.width / 6, theSize.height / 6);
  p.lineTo (theSize.width * 5 / 6, theSize.height / 6);
  p.lineTo (theSize.width * 5/ 6, theSize.height * 5 / 6);
  p.lineTo (theSize.width / 6, theSize.height * 5 / 6);
  p.closePath ();

  g2.setColor (Color.blue);
  g2.draw (p);

  AffineTransform at = new AffineTransform ();
  at.scale (.5, .5);
  at.translate (theSize.width / 2, theSize.height / 2);
  g2.setTransform (at);
  g2.setColor (Color.red);
  g2.fill (p);

  Color [] colorArray = { Color.blue, Color.green, Color.magenta, Color.lightGray,
                          Color.pink, Color.white, Color.yellow, Color.black,
                          Color.gray, Color.orange };


  for (int n = 0;  n < colorArray.length; n ++) {
    at.scale (.9, .9);
    at.rotate (15, theSize.width / 2, theSize.height / 2);
    g2.setTransform (at);
    g2.setColor (colorArray [n]);
    g2.fill (p);
    }
  }
```



Java Foundation Classes

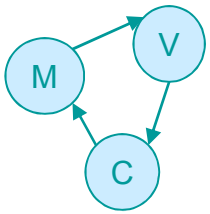
- swing
  - lightweight
    - not associated with a peer in the underlying OS
    - easy to create/port
  - Model-View-Controller architecture
    - separable model
    - pluggable look & feel
      - expect atrocities!
  - various other aspects
    - packages
    - component gallery
    - layout control
    - events and interfaces

Java Foundation Classes

- swing design goals
  - overall:
    - *"To build a set of extensible GUI components to enable developers to more rapidly develop powerful Java frontends for commercial applications."*
  - in addition, swing should:
    - be 100% pure Java
    - be capable of supporting multiple *look-and-feels*
      - System, Metal, Motif exist (Mac version in ß)
    - enable the power of model-driven programming
    - adhere to JavaBeans design principles
    - provide compatibility with the existing AWT

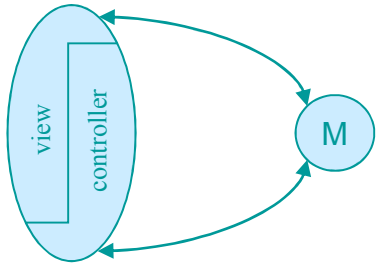
Java Foundation Classes

- Model-View-Controller architecture
  - originated with SmallTalk
  - application is broken into three basic portions
    - model
      - the application's data
    - view
      - visual representation of the data
    - controller
      - takes user input on the view and manipulates the model data
  - each portion is independent of the others



Java Foundation Classes

- swing modifies ‘pure’ MVC
  - view and controller often intimately associated so strict separation difficult to achieve or valueless in practice (difficult to have a controller that doesn’t know specifics of it’s view. Also doesn’t make sense to combine scrollable view and button controller, for example)
  - swing combines view & controller into a single *“UI Delegate”*
    - 2 purposes:
      - make development easier
      - facilitate pluggable look-and-feel

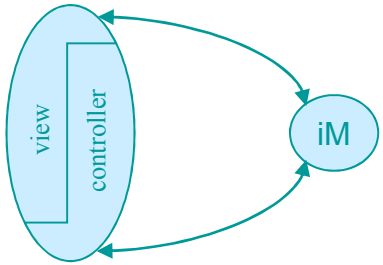


Java Foundation Classes

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

public class DelegateExample extends JFrame
  {
  public DelegateExample (Delegate d)
    {
    super ("DelegateExample");
    getContentPane ().add (d);
    }

  public static void main (String [] args)
    {
    try
      {
      UIManager.setLookAndFeel (UIManager.getSystemLookAndFeelClassName ());

      DelegateExample d = new DelegateExample (new Delegate (new IntModel (0)));
      d.addWindowListener
        (
        new WindowAdapter ()
          {
          public void windowClosing (WindowEvent e) { System.exit (0); }
          }
        );
      d.setSize (new Dimension (250, 60));
      d.show ();
      }
    catch (Exception e)
      { /* SQUELCH! */ }
    }
  }
```

Java Foundation Classes

```
class Delegate extends JPanel
  {
  public Delegate (final IntModel iM)
    {
    final JTextField view = new JTextField (iM.getValue () + "", 10);
    add (view);
    JButton controller = new JButton ("Increment");
    controller.addActionListener
      (
      new ActionListener ()
        {
        public void actionPerformed (ActionEvent e)
          {
          int val = iM.getValue () + 1;
          iM.setValue (val);
          view.setText (val + "");
          }
        }
      );
    add (controller);
    }
  }

class IntModel
  {
  private int model;
  public IntModel (int model) { setValue (model); }
  public int getValue () { return (model); }
  public void setValue (int model) { this.model = model; }
  }
```



Java Foundation Classes

- conventional to consider two major categories of model
  - GUI-state
    - e.g. state of a button or current selection in a list
    - separate model not strictly required but
      - helps in situations such as shared whiteboard
      - also if one control should update another
  - application-data
    - data required by application to “get the job done”
      - data underlying a spreadsheet cell
      - final value of a control, etc.
- also have “crazy mixed up” models
  - e.g. BoundedRangeModel
    - GUI-state when backing a slider control
    - can usefully be application-data as well

Java Foundation Classes

- component-to-model mapping

Component	Model Interface	Model Type
JButton	ButtonModel	GUI
JToggleButton	ButtonModel	GUI/data
JCheckBox	ButtonModel	GUI/data
JRadioButton	ButtonModel	GUI/data
JMenu	ButtonModel	GUI
JMenuItem	ButtonModel	GUI
JCheckBoxMenuItem	ButtonModel	GUI/data
JRadioButtonMenuItem	ButtonModel	GUI/data
JComboBox	ComboBoxModel	data
JProgressBar	BoundedRangeModel	GUI/data
JScrollBar	BoundedRangeModel	GUI/data
JSlider	BoundedRangeModel	GUI/data
JTabbedPane	SingleSelectionModel	GUI
JList	ListModel	data
JList	ListSelectionModel	GUI
JTable	TableModel	data
JTable	TableColumnModel	GUI
JTree	TreeModel	data
JTree	TreeSelectionModel	GUI
JEditorPane	Document	data
JTextPane	Document	data
JTextArea	Document	data
JTextField	Document	data
JPasswordField	Document	data

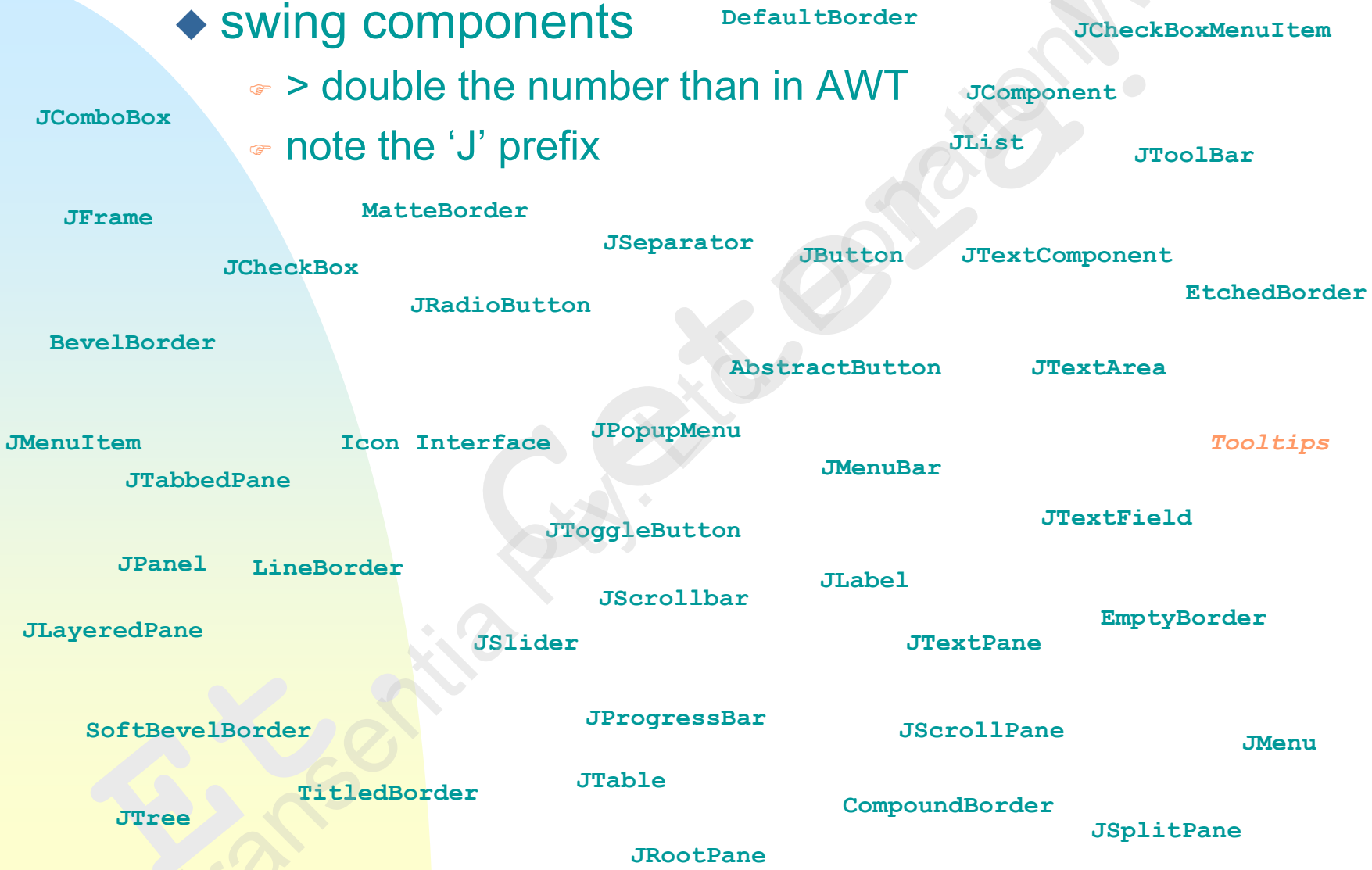
Java Foundation Classes

- swing packages
  - currently 15, more expected

javax.accessibility
javax.swing
javax.swing.border
javax.swing.colorchooser
javax.swing.event
javax.swing.filechooser
javax.swing.plaf
javax.swing.plaf.basic
javax.swing.plaf.metal
javax.swing.plaf.multi
javax.swing.table
javax.swing.text
javax.swing.text.html
javax.swing.tree
javax.swing.undo

Java Foundation Classes

- swing components
  - > double the number than in AWT
  - note the ‘J’ prefix



Java Foundation Classes

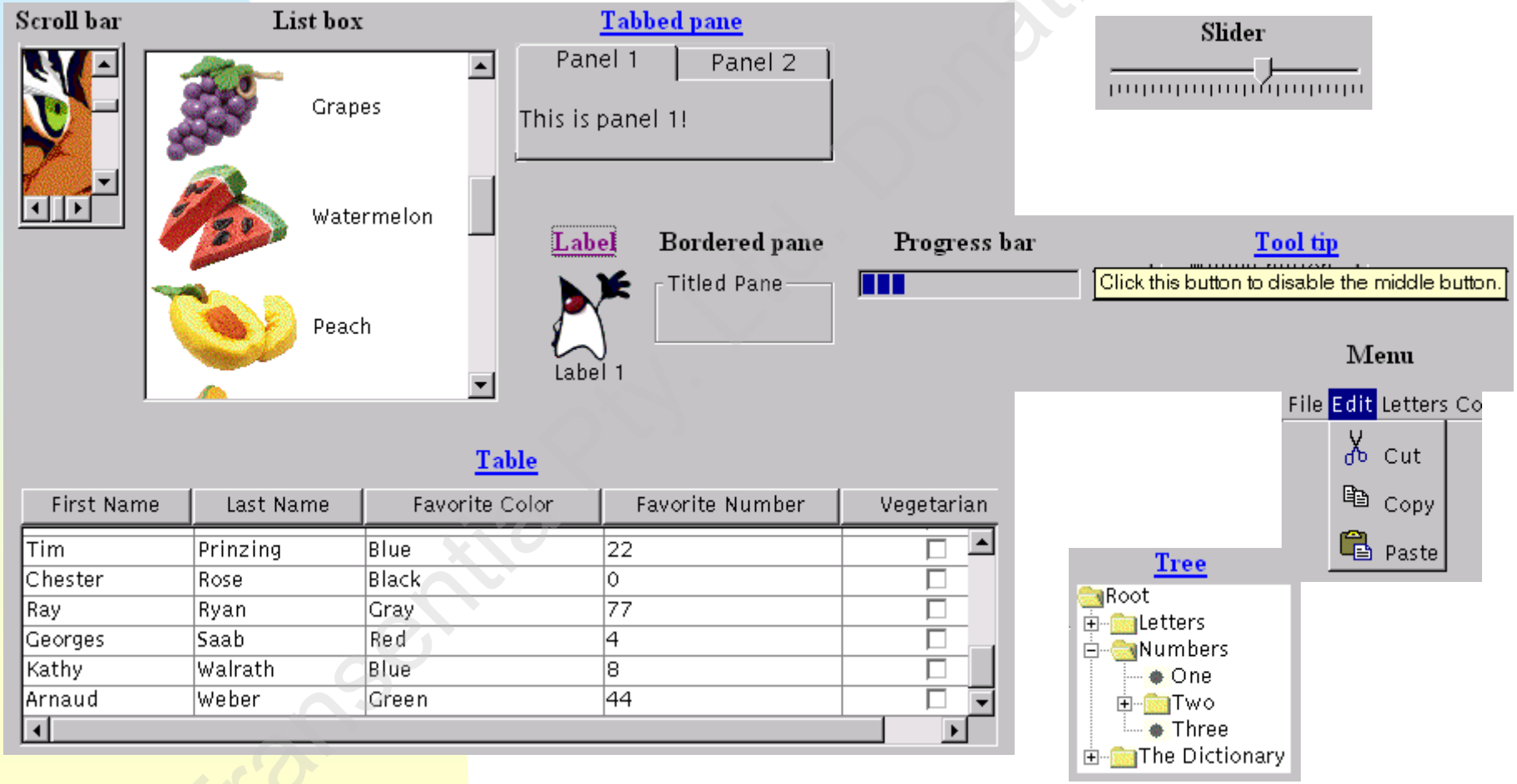
- all JComponents are subclasses of java.awt.Container
  - all can display icons & text
- built-in support for double-buffering
  - may give better performance
- nine border styles
  - can be applied to any JComponent
  - can be transparent
  - BorderFactory class vends border instances
- new mutually-exclusive checkbox menu class
- any component can have an associated tooltip
- JToolBar can float
- extensive new text-handling package

Java Foundation Classes

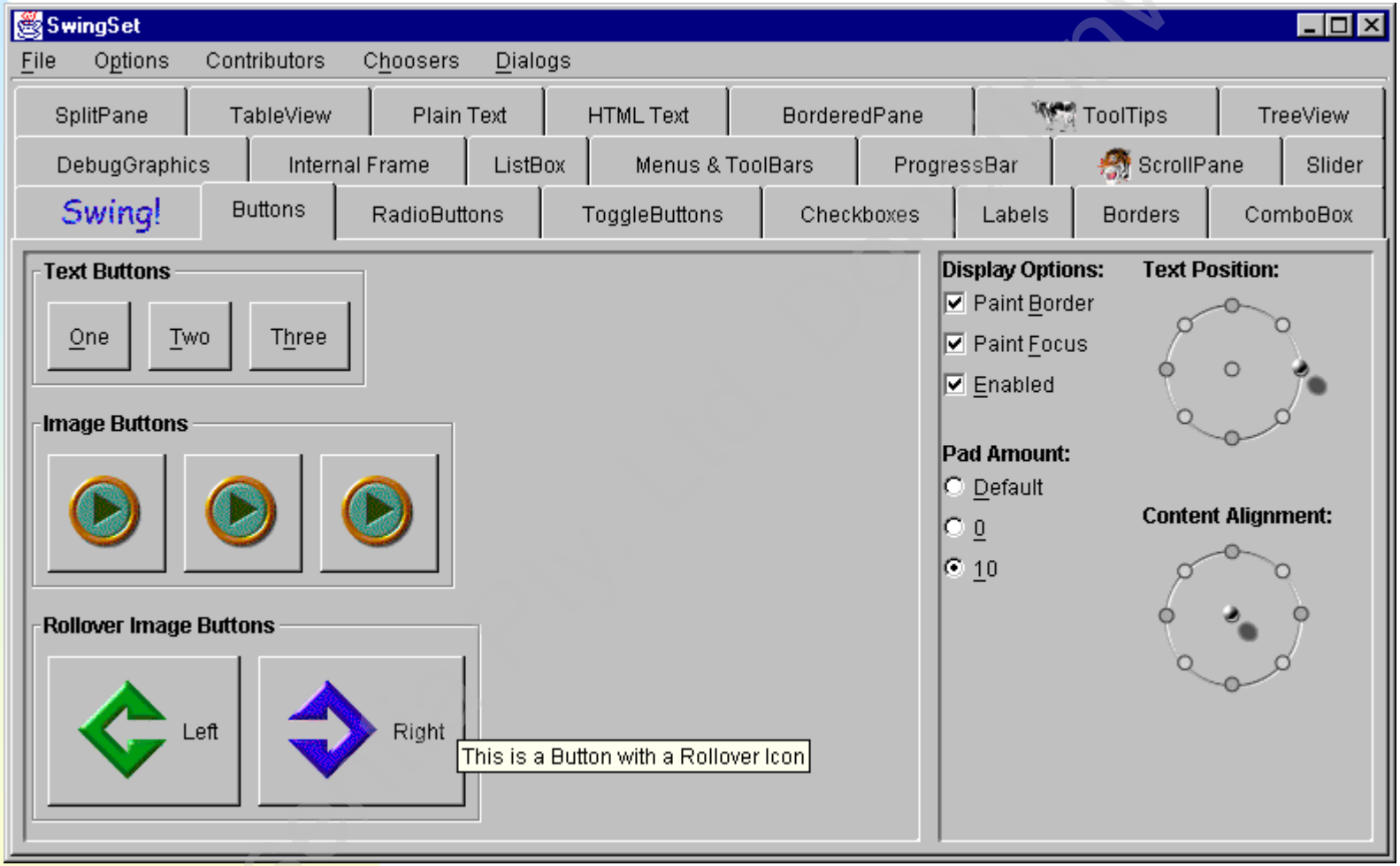
- windows are *not* lightweight components
  - affects JFrame, JWindow, JDialog, etc.
  - can't be arbitrary shape, transparent, etc.
- content frames
  - supplied by all major containers
  - components added to content frame, not container
- JLayeredPane supports 5 *depth ranges*: default, palette, modal, popup, drag
  - used by JFrame
  - efficient menu handling
  - also makes it possible to have overlapping components
- JOptionPane for confirmation dialogs, etc.

Java Foundation Classes

'swing' component gallery

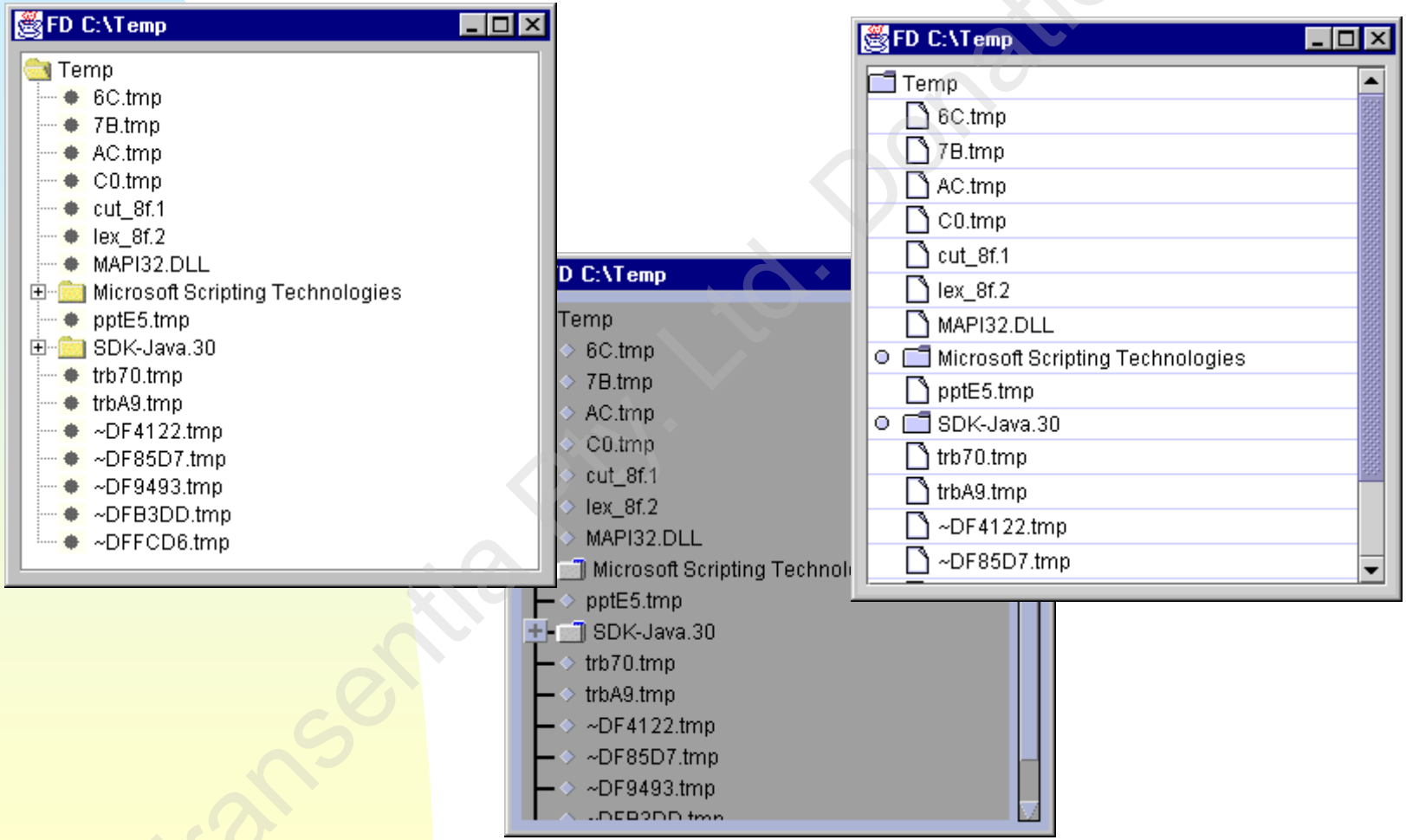


Java Foundation Classes



Java Foundation Classes

an example: file/directory tree lister



Java Foundation Classes

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.tree.*;
import javax.swing.*;
import javax.swing.event.*;

public class FD
  {
  private static DefaultMutableTreeNode lister (File path)
    {
    DefaultMutableTreeNode thisNode = null;
    try
      {
      String name = path.getName ();
      if ("".equals (name))
        name = path.getAbsolutePath ();
      thisNode = new DefaultMutableTreeNode (name);
      String [] list = path.list ();
      for (int i = 0; i < list.length; i ++)
        {
        String s = list [i]; File f = new File (path, s);
        if (f.isDirectory ())
          thisNode.add (lister (f));
        else
          thisNode.add (new DefaultMutableTreeNode (s));
        }
      }
    catch (final NullPointerException npe)
      { /* lack of filesystem permissions ... Ignore */ }
    catch (final Exception e)
      { e.printStackTrace (); }
    return (thisNode);
    }
```

(continued …)

Java Foundation Classes

(… continued)

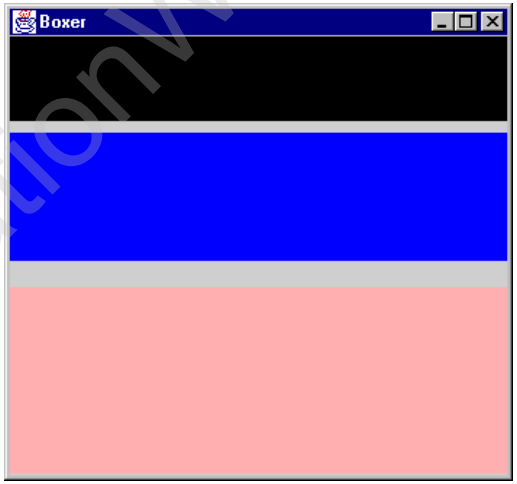
```
public static void main (String [] args)
    {
    try
      {
      UIManager.setLookAndFeel (UIManager.getSystemLookAndFeelClassName ());
      JFrame frame = new JFrame ("FD " + args [0]);
      frame.addWindowListener
        (
        new WindowAdapter ()
          {
          public void windowClosing (WindowEvent e)
            { System.exit (0); }
          }
        );
      DefaultMutableTreeNode root = lister (new File (args [0]));
      JScrollPane sp = new JScrollPane ();
      sp.setPreferredSize (new Dimension (300, 300));
      sp.getViewport ().add (new JTree (root));
      frame.getContentPane ().add (sp);
      frame.pack ();
      frame.show ();
      }
    catch (UnsupportedLookAndFeelException e)
      { /* SQUELCH! */ }
    }
  }
```

Java Foundation Classes

- layout control
  - some new layout managers
    - OverlayLayout
      - simple, small-footprint manager
      - centres an object, uses preferredSize
    - ScrollPaneLayout, ViewportLayout
      - used by JScrollPane and JViewport classes
      - not typically used independently
    - BoxLayout
      - used by Box component (c.f. Panel)
      - glue (stretchy space), struts, rigid areas, fillers affect layout
      - *"...gives an effect similar to GridbagLayout, without the complexity."*

Java Foundation Classes

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

public class Boxer extends JFrame
  {
  public Boxer ()
    {
    setTitle ("Boxer");
    setSize (new Dimension (350, 300));
    Container c = getContentPane ();
    c.setLayout (new BoxLayout (c, BoxLayout.YAXIS));

    c.add (new BobsJPanel (50, Color.black));
    c.add (Box.createVerticalGlue ());
    c.add (new BobsJPanel (80, Color.blue));
    Dimension minSize = new Dimension (Short.MAX_VALUE, 0),
              prefSize = new Dimension (Short.MAX_VALUE, 10),
              maxSize = new Dimension (Short.MAX_VALUE, Short.MAX_VALUE);
    c.add (new Box.Filler (minSize, prefSize, maxSize));
    c.add (new BobsJPanel (120, Color.pink));
    }

  public static void main (String [] args)
    {
    Boxer b = new Boxer ();
    b.addWindowListener
      (
      new WindowAdapter ()
        {
        public void windowClosing (WindowEvent e)
          { System.exit (0); }
        }
      );
    b.show ();
    }
  }
```




```
class BobsJPanel extends JPanel
  {
  public BobsJPanel (int height, Color c)
    {
    setPreferredSize (new Dimension (0, height));
    setBackground (c);
    }
  }
```

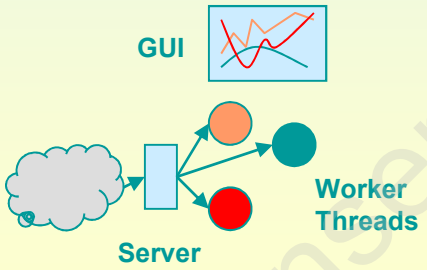
Java Foundation Classes

- events & interfaces
  - package com.sun.java.swing.event
  - same listener scheme as AWT 1.1 and JavaBeans
  - some interesting listeners:

    CellEditorListener, DocumentListener, HyperlinkListener, ListDataListener, ListSelectionListener, MenuListener, PopupMenuListener, TableColumnModelListener, TableModelListener, TreeExpansionListener, TreeModelListener, TreeSelectionListener, UndoableEditListener
  - note: no adapter classes *(as of 1.2ß4, at least)*

Java Foundation Classes

- threading considerations
  - swing not designed to allow multiple threads to manipulate a GUI
    - typically should only access GUI in event-dispatching thread (paint, actionPerformed, etc. are executed by this thread)
    - not always possible!
    - SwingUtilities class to the rescue…
      - invokeLater and invokeAndWait methods
      - queue requests for code to be executed by the event-dispatching thread (effectively as normal events)




```
SwingUtilities.invokeLater
  (
  new Runnable ()
    {
    public void run () { doSomeGUIWork (); }
    }
  );
```

Java Foundation Classes

- Timer class
  - helps with timed actions
- SwingWorker class
  - may make threading easier
    - not in swing proper (downloadable from Java site)
    - provided as convenience
    - SwingWorker's get method starts thread and returns object returned by user's construct method

```
final SwingWorker worker =
  new SwingWorker ()
    {
    public Object construct ()
      {
      return new expensiveDialogComponent ();
      }
    }

Joptionpane.showMessagedialog (f, worker.get ());
```

Java Foundation Classes

- Undo/Redo support
  - javax.swing.undo package

Java Foundation Classes

- drag & drop
  - java.awt.dnd package
  - built on existing java.awt.datatransfer work
  - D&D between two Java applications (even on different VMs) also Java and native applications
  - DragSource instantiated in response to a GUI/other ‘gesture.’ DragGestureListener and DragSourceListener interfaces are important
  - DropTargets encapsulate all of the platform-specific handling of the D&D protocol. Associated with *java.awt.Component*. Provides DropTargetListener interface to facilitate GUI feedback.

Java Foundation Classes


```
// Dnd.java - test drag and drop
import java.awt.*;
import java.awt.event.*;

public class Dnd
  {
  public static void main (String [] args)
    {
    Frame f = new Frame("Dnd");
    Panel p = new Panel ();
    p.add (new DraggableLabel ("One", Color.pink));
    p.add (new DraggableLabel ("Two", Color.red));
    f.add (BorderLayout.NORTH, p);
    f.add (BorderLayout.CENTER, new DropableLabel ("Drag stuff onto here", Color.blue));
    f.setSize (300, 150);
    f.addWindowListener
      (
      new WindowAdapter ()
        {
        public void windowClosing(WindowEvent e) { System.exit(0); }
        }
      );
    f.setVisible (true);
    }
  }
```

Java Foundation Classes

```
// DraggableLabel.java
import java.awt.*;
import java.awt.dnd.*;
import java.awt.datatransfer.*;

public class DraggableLabel extends Label
  implements DragSourceListener, DragGestureListener
  {
  DragSource dragSource;

  public DraggableLabel (String s, Color c)
    {
    super (s);
    setBackground (c);
    dragSource = new DragSource ();
    dragSource.createDefaultDragGestureRecognizer (this,
                                                   DnDConstants.ACTION_COPY,
                                                   this);
    }
  public void dragGestureRecognized (DragGestureEvent e)
    {
    StringSelection text = new StringSelection (getText ());
    dragSource.startDrag (e, DragSource.DefaultCopyDrop, text, this);
    }
  public void dragDropEnd (DragSourceDropEvent e) { }
  public void dragEnter (DragSourceDragEvent e) { }
  public void dragExit (DragSourceEvent e) { }
  public void dragOver (DragSourceDragEvent e) { }
  public void dropActionChanged (DragSourceDragEvent e) { }
}
```

Java Foundation Classes

```
// DropableLabel.java
import java.awt.*;
import java.awt.dnd.*;
import java.awt.datatransfer.*;
public class DropableLabel extends Label implements DropTargetListener
  {
  private DropTarget dropTarget;
  private Color c;
  public DropableLabel (String s, Color c)
    {
    super (s); setAlignment (Label.CENTER);
    setBackground (this.c = c); dropTarget = new DropTarget (this, this);
    }
  public void dragEnter (DropTargetDragEvent e) { e.acceptDrag (DnDConstants.ACTION_COPY); }
  public void dragExit (DropTargetEvent e) { setBackground (c); }
  public void dragOver (DropTargetDragEvent e) { setBackground (Color.gray); }
  public void drop (DropTargetDropEvent e)
    {
    try
      {
      Transferable tr = e.getTransferable ();
      if (tr.isDataFlavorSupported (DataFlavor.stringFlavor))
        {
        e.acceptDrop (DnDConstants.ACTION_COPY_OR_MOVE);
        setText ((String) tr.getTransferData (DataFlavor.stringFlavor));
        e.getDropTargetContext ().dropComplete (true);
        }
      else
        e.rejectDrop ();
      }
    catch (Exception e1) { e1.printStackTrace(); e.rejectDrop(); }
    finally { setBackground (c); }
    }
  public void dropActionChanged (DropTargetDragEvent e) { }
  }
```