Nick Parlante

Swing 2

Overview

Today we'll work on quick coverage of Swing

Topics

- JFrame/layouts/JComponent (on previous handout)
- Listeners
- MVC -- JTable

JFrame

- Install components in the content pane
- pack() invokes the layout managers
- setVisible(true) brings on screen

Working With Layouts

Layouts: Flow, Box, Border

setPreferredSize()

- Send to a JComponent before layout to indicate its size preference

JPanel

- JPanel can hold components and have its own layout manager. Use JPanel to put multiple components in one area of a layout.

Drawing

paintComponent(Graphics g)

- This is the notification sent to a component when it needs to draw itself.
- Use getWidth(), getHeight(), etc. to see how big you are
- Use the passed in Graphics object for drawing
- Note the "passive" drawing style

Object State -> Pixels

- paintComponent() maps the object state to pixels on screen. It should not change the object.

Clipping

Draw region

- System maintains a "clipping region" -- drawing is clipped so that only pixels inside the region are changed. When paintComponent() is sent, the system first establishes a clipping region where it wants the

drawing to happen. This can be a performance optimization if only a small area needs to be drawn.

getGraphics() -- NO

It's possible to get a Graphics object and start drawing. However, this is not consistent with the paintComponent() drawing style, so it should be avoided. getGraphics() is for people who do not really understand how Swing drawing is supposed to work.

repaint()

90% automatic
repaint() = draw request
asynchronous
"up to date" model of repaint
repaint setter style

Listeners

Source: buttons, controls, etc.
Destination: your object
Notification message

1. Listener Interface

e.g. ActionListener

2. Notification Prototype

public void actionPerformed(ActionEvent e)

3. source.addXXX(dest)

Register the dest as a listener to the source
e.g. button.addActionListener(dest)
dest implements ActionListener

4. Event -> Notification

Source iterates through listeners
Sends each the notification
e.g. actionPerformed() executes on each listener

1. dest Implements Intf

Could do this, but using an inner class is easier

2. Create inner class to be the dest

Call new to create one and pass it to addXXX

Sortof like a LISP lambda

An executable thing you can pass around

3. Anonymous inner class

Most convenient

Create on the fly

```
button = new JButton("Beep");
box.add(button);
button.addActionListener(
   new ActionListener() {
      public void actionPerformed(ActionEvent e) {
         Toolkit.getDefaultToolkit().beep();
      }
   }
);
```

Anonymous Inner Class Access

1. see outer ivars -- > yes
2. see outer local vars --> no

   This makes sense -- that activation record is gone by the time the thing runs
3. see **final** local vars --> yes (handy!)

e.g. JSlider /ChangeListener

Ctor JSlider (min, max, current)

slider.getValue();

ChangeListener -- listen for slider changes

stateChanged(ChangeEvent e) -- the notification

e.getSource() -- get the slider pointer from the notification

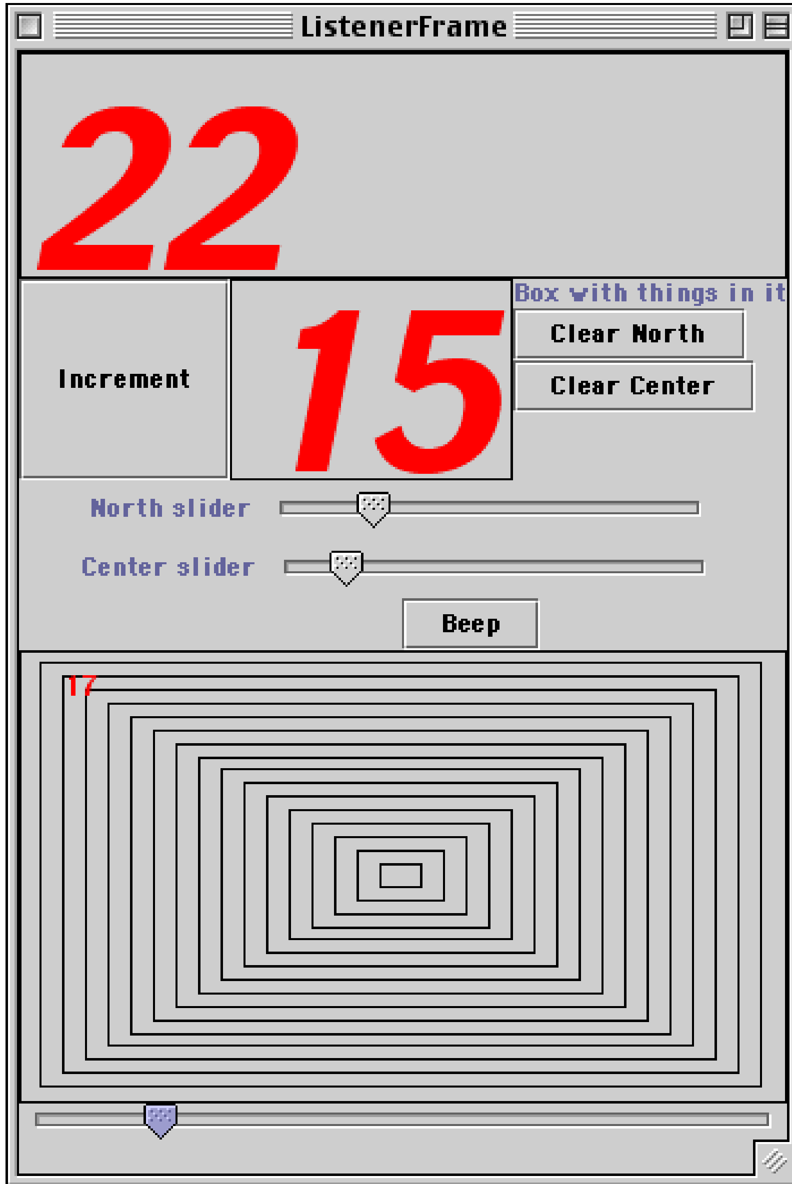
```
slider.addChangeListener(
   new ChangeListener() {
      public void stateChanged(ChangeEvent e) {
         JSlider s = (JSlider) e.getSource();
         // do something with s.getValue()
      }
```

```
      }
    );
```

Widget Example

```
// Demonstrates paintComponent() and setter/repaint() style
public class Widget extends JComponent {
   private int count;

   private static Font font = null;

   public Widget(int width, int height) {
      super();
      setPreferredSize(new Dimension(width, height));

      count = 0;
   }

   /*
    Typical setter -- calls repaint to signal the system.
   */
   public void setCount(int newCount) {
      if (newCount!=count) {
         count = newCount;
         repaint();
      }
   }

   public void increment() {
      setCount(count+1);
   }

   /*
    Draw ourselves with a big font (see the Font class).
   */
   public void paintComponent(Graphics g) {
      // typical "debug rect" around our bounds just to have
      // something show up
      g.drawRect(0, 0, getWidth()-1, getHeight()-1);

      // trick: lazy evaluation of font object
      if (font==null) font = new Font("DIALOG", Font.ITALIC, 96);

      g.setFont(font);
      g.setColor(Color.red);
      g.drawString(Integer.toString(count), 4, getHeight()-4);
   }
}
```

Listener Example




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

/*
 Demonstrates bringing up a frame with a bunch of controls in it.
 Demonstrates using buttons and sliders with  annonymous inner class listeners.

 The Widget class displays a number using a large font, and responds to the
 increment() and setCount() messages.
```

```
*/
public class ListenerFrame extends JFrame {
   private Widget center;
   private Widget north;

   public ListenerFrame() {
      super("ListenerFrame");

      JComponent container = (JComponent) getContentPane();

      // Put in a border layout
      container.setLayout(new BorderLayout());

      // Put a couple widgets directly in the border layout
      north = new Widget(100, 100);
      container.add(north, BorderLayout.NORTH);

      center = new Widget(100, 100);
      container.add(center, BorderLayout.CENTER);


      JButton button;

      // Put a button in the west, and connect it to the center
      button = new JButton("Increment");
      container.add(button, BorderLayout.WEST);
      button.addActionListener(
         new ActionListener() {
            public void actionPerformed(ActionEvent e) {
               // note: we can access ivars of our "outer" object
               center.increment();
            }
         }
      );


      // Create a vertical box, put it in the east, put things in it
      Box box = Box.createVerticalBox();
      container.add(box, BorderLayout.EAST);

      box.add(new JLabel("Box with things in it"));
      button = new JButton("Clear North");
      box.add(button);
      button.addActionListener(
         new ActionListener() {
            public void actionPerformed(ActionEvent e) {
               north.setCount(0);
            }
         }
      );
      button = new JButton("Clear Center");
      box.add(button);
      button.addActionListener(
         new ActionListener() {
            public void actionPerformed(ActionEvent e) {
               center.setCount(0);
            }
         }
```

```
);

// Create a vertical box, put it in the south
// Put panels in it which group things
box = Box.createVerticalBox();
container.add(box, BorderLayout.SOUTH);

JPanel panel = new JPanel();
box.add(panel);
panel.add(new JLabel("North slider"));

// note: store slider as "final" variable so inner class
// can see it
final JSlider slider = new JSlider(0, 100, 0);  // (min, max, current)
panel.add(slider);

slider.addChangeListener(
   new ChangeListener() {
      public void stateChanged(ChangeEvent e) {
         // note: can access final stack var "slider"
         north.setCount(slider.getValue());
      }
   }
);


panel = new JPanel();
box.add(panel);
panel.add(new JLabel("Center slider"));
JSlider slider2 = new JSlider(0, 100, 0);
panel.add(slider2);

slider2.addChangeListener(
   new ChangeListener() {
      public void stateChanged(ChangeEvent e) {
         JSlider s = (JSlider) e.getSource();
         center.setCount(s.getValue());
      }
   }
);

button = new JButton("Beep");
box.add(button);
button.addActionListener(
   new ActionListener() {
      public void actionPerformed(ActionEvent e) {
         Toolkit.getDefaultToolkit().beep();
      }
   }
);

// Put in the XComponent we used as an earlier example
final XComponent xcomp = new XComponent(200,200);
box.add(xcomp);
JSlider slider3 = new JSlider(0, 100, 0);
box.add(slider3);
slider3.addChangeListener(
   new ChangeListener() {
```

```
            public void stateChanged(ChangeEvent e) {
               // note: another way to get a pointer to the source
               JSlider s = (JSlider) e.getSource();
               xcomp.setCount(s.getValue());
            }
         }
      );


      // Quit on window close
      addWindowListener(
         new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
               System.exit(0);
            }
         }
      );
      // in 1.2 this works: frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      pack();
      setVisible(true);

   }

   public static void main(String[] args) {
      new ListenerFrame();
   }
}
```

Model / View / Controller

- Design
  - A decomposition strategy where "presentation" is separated from data maintenance
  - Smalltalk idea
  - Controller is often combined with View, so have Model and View/Controller
- Web Version
  - Shopping cart model is on the server
  - The view is the HTML in front of you
  - Form submit = send transaction to the model, it computes the new state, sends you back a new view

Model

- "data model"
- Storage, not presentation

- Isolate data model operations
  - cut/copy/paste, File Saving, undo, networked data -- these can be expressed just on the model which simplifies things

View

- Presentation
- Gets all data from model
- Edits events piped to model
  - Edit events happen in the view (keyboard, mouse gestures) -- these get messaged to the model which does the actual data maintenance

Controller

- The logic that glues things together.
- Usually, the controller is implemented in the view.

Model Role

- Respond to getData() to provide data
- Respond to SetData() to change data
- Manage a list of listeners
- When receiving a setData() to change the data, notify the listeners of the change (fireXXXChanged)
  - Change notifications express the different changes possible on the model. (cell edited, row deleted, ...)
  - Iterate through the listeners and tell each about the change.

View Role

- Have pointer to model
- Don't store any data
- Send getData() to model to get data as needed
- User edit operations (clicking, typing) in the UI map to setData() messages sent to model
- Register as a listener to model to get change notifications
- Register as a listener to the model(respond to listener notifications)
- On change notification, consider doing a getData() to get the new values.

Advantage: Good Old Modularity

- 2 small problems vs. 1 big problem
- Provides a natural decomposition "pattern"

- You will get used to the MVC decomposition. Other Java programmers will also. It ends up providing a common, understood language.

Isolate coding problems in a smaller domain

- Can solve GUI problems just in the GUI domain, the storage etc. is all quite separate. e.g. don't worry about file saving when implementing scrolling.
- e.g. undo() can just be implemented on the model -- it has to interact with far fewer lines of code than if it were implemented on top of some sort of model+view system

Other MVC Advantages

50% done for you

- Use, say, the view off the shelf, but plug in your own model.
- Or vice-versa.

Multiple views on one model (networked)

Swing Table Classes

JTable -- view

Uses a TableModel for storage

TableModel -- Interface

The messages the define a table model -- the abstraction is a rectangular area of cells.

getValueAt(), setValueAt, getRowCount(), getColumnCount(), ...

TableModelListener -- Interface

Defines the one method tableChanged()

If you want to listen to a TableModel to hear about its changes, implement this interface.

```
public interface TableModelListener extends java.util.EventListener
{
    /**
     * This fine grain notification tells listeners the exact range
     * of cells, rows, or columns that changed.
     */
    public void tableChanged(TableModelEvent e);
}
```

AbstractTableModel

- Implements TableModel 50%
- Provides helper utilities for things not directly related to storage
  - addTableModelListener(), removeTableModelListener(), ...
- fireXXXChanged() convenience methods
  - These iterate over the listeners and send the appropriate notification
  - fireTableCellUpdated(row, col)
  - fireTableRowDeleted(row)
  - etc.
- getValueAt() and setValueAt() are missing

DefaultTableModel

- extends AbstractTableModel
- Complete implementation with Vector

BasicTableModel Code Points

- A complete implementation of TableModel using ArrayList
- getValueAt()
  - Pulls data out of the ArrayList of ArrayList implementation
- setValueAt()
  - Changes the data model and uses fireTableXXX (below) to notify the listeners
- AbstractTableModel
  - Has routine code in it to manage listeners -- add and remove.
  - Has fireTableXXX() methods that notify the listeners -- BasicTableModel uses these to tell the listeners about changes.

1. Passive Example

1. Table View points to model
2. View does model.getXXX to get data to display

2. Row Add Example

1. Add row button wired to the model
2. Model changes its state
3. Model does fireRowAdded() which sends notification to each listener
4. Listeners get the notification, call getData() as needed

3. Edit Example

1. Table View1 points to model for its data and listens for changes
2. Table View2 also points to the model and listens for changes
3. User clicks/edits data in View1
4. View1 does a model.setXXX to make the change

5. Model does a fireDataChanged() -- notifies both listeners
6. Both views get the notification of change, update their display (getXXX) if necessary

View2 can be smart if View1 changes a row that View2 is not currently scrolled to see.

Scenario: Model Substitution

Have some 2-d data. Want to present it in a 2-d GUI. Wrap your data up so that it responds to getColumnCount(), getDataAt(), etc....

Build a JTable, passing it pointer to your object as the data model and voila. The scrolling, the GUI, etc. etc. is all done by JTable.

JTable

The (simple) component side -- make a JTable on a TableModel, and it just works...

```
model = new BasicTableModel();

// Make a table on the model
table = new JTable(model);
table.setAutoResizeMode(JTable.AUTO_RESIZE_ALL_COLUMNS);
JScrollPane scrollpane = new JScrollPane(table);
scrollpane.setPreferredSize(new Dimension(300,200));
container.add(scrollpane, BorderLayout.CENTER);
```

BasicTableModel

Demonstrates a complete implementation of TableModel -- stores the data and generates fireXXXChanged() notifications where necessary.

```
class BasicTableModel extends AbstractTableModel {
   private ArrayList names;
   private ArrayList data;


   public BasicTableModel() {
      super();

      names = new ArrayList();
      data = new ArrayList();
   }


   // Basic Model overrides
   public String getColumnName(int col) {
```

```
   return (String) names.get(col);
}
public int getColumnCount() { return(names.size()); }
public int getRowCount() { return(data.size()); }
public Object getValueAt(int row, int col) {
   ArrayList rowList = (ArrayList) data.get(row);
   String result = null;
   if (col<rowList.size()) {
      result = (String)rowList.get(col);
   }

   // _apparently_ it's ok to return null for a "blank" cell
   return(result);
}


// Support writing
public boolean isCellEditable(int row, int col) { return true; }
public void setValueAt(Object value, int row, int col) {
   ArrayList rowList = (ArrayList) data.get(row);

   if (col>=rowList.size()) {
      while (col>=rowList.size()) rowList.add(null);
   }

   rowList.set(col, value);

   fireTableCellUpdated(row, col);
}


public void addColumn(String name) {
   names.add(name);
   fireTableStructureChanged();
   /*
    At present, TableModelListener does not have a more specific
    notification for changing the number of columns.
   */
}


public int addRow() {
   // Create a new row with nothing in it
   ArrayList row = new ArrayList();
   return(addRow(row));
}


public int addRow(ArrayList row) {
   data.add(row);
   fireTableRowsInserted(data.size()-1, data.size()-1);
   return(data.size() -1);
}


public void deleteRow(int row) {
   if (row == -1) return;
```

```
      data.remove(row);
      fireTableRowsDeleted(row, row);
   }


   /*
    Utility.
    Change a tab-delimited line into an ArrayList.
    This is much more complex than necessary because
    StringTok can't deal with two tabs next to each other.
    StringTokenizer is a good example of inadequate docs for a fairly
    complex class.
   */
   private static ArrayList stringToList(String string) {
      StringTokenizer tokenizer = new StringTokenizer(string, "\t", true);
      ArrayList row = new ArrayList();
      String elem = null;
      String last = null;
      while(tokenizer.hasMoreTokens()) {
         last = elem;
         elem = tokenizer.nextToken();
         if (!elem.equals("\t")) row.add(elem);
         else if (last.equals("\t")) row.add("");
      }
      if (elem.equals("\t")) row.add("");

      return(row);
   }


   /*
    Load the whole model from a file.
   */
   public void loadFile(File file) {
      try {
         FileReader fileReader = new FileReader(file);
         BufferedReader bufferedReader = new BufferedReader(fileReader);

         ArrayList first = stringToList(bufferedReader.readLine());
         names = first;

         String line;
         data = new ArrayList();
         while ((line = bufferedReader.readLine()) != null) {
            data.add(stringToList(line));
         }

         fireTableStructureChanged();
      }
      catch (IOException e) {
         System.err.println(e.getMessage());
      }
   }

}
```