

Graphical User Interfaces are Trees

CS1316: Representing
Structure and Behavior

Story

- Building a Graphical User Interface in Java
 - Adding pieces to a JFrame
 - Buttons, panels, etc.
- Constructing a GUI is building a tree
- Layout managers
 - Layout managers are GUI tree renderers
- Making GUIs *do* something
 - Listeners
- Building a musical instrument

Old style Java: Abstract Window Toolkit - AWT

- Original Graphical User Interface (GUI) Classes
 - Container Objects
 - Frame - Main window with title and border.
 - Panel - groups components
 - Canvas - create custom components
 - Input and Output Classes
 - Label - not editable text
 - Button - pushing fires an event
 - Checkboxes and Radio Buttons
 - TextField - input and output of text
 - TextArea - input and output of multiple lines of text
 - List - Select one or more items from a displayed list
 - Choice - Select one from a drop down list

Swing - javax.swing

- Replacements for most AWT components
 - JButton - Button (images and text)
 - JFrame - Frame (main window)
 - JPanel - Panel (container)
- New GUI components
 - Trees - JTree
 - Split pane - JSplitPane
 - Table - JTable
- Supports multiple looks and feels
 - Java - also called metal, Windows, Mac, Motif

Swing Top-Level Containers

- JFrame - main window with title, maybe a menu bar, and the ability to minimize, maximize, and close the window



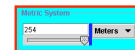
- JApplet - main window for an applet. Inherits from java.applet.Applet



- JDialog - pop-up window for simple communication with the user
 - Like the JFileChooser

Swing General Containers

- JPanel - group components



- JScrollPane - add scroll bars to a component



- JSplitPane - display two separate panes



Working with a JFrame

- Pass the title when you create it
`JFrame frame = new JFrame("FrameDemo");`
- Add components to the content pane
`JLabel label = new JLabel("Hello World");`
`frame.getContentPane().add(label, BorderLayout.CENTER);`
- Set the size of the JFrame
`frame.pack(); // as big as needs to be to display contents`
- Display the JFrame
`frame.setVisible(true); // display the frame`

JFrame Options

- When creating a GUI application
 - Have your main class inherit from JFrame
 - So it is a JFrame
 - Or have your main class inherit from JPanel
 - And create a JFrame in the main method
 - Create the main class object
 - Add the main class object to the content pane of the JFrame
- If your class inherits from JPanel
 - It can be reused in another application
 - Even an applet

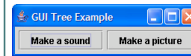
GUI Tree Class



```
Welcome to DrJava.  
> GUItree gt = new  
GUItree();
```

```
/**  
 * A GUI that has various components in it, to demonstrate  
 * UI components and layout managers (rendering)  
 **/  
import javax.swing.*; // Need this to reach Swing components  
  
public class GUItree extends JFrame {  
  
    public GUItree(){  
        super("GUI Tree Example");  
  
        /* Put in a panel with a label in it */  
        JPanel panel1 = new JPanel();  
        this.getContentPane().add(panel1);  
        JLabel label = new JLabel("This is panel 1!");  
        panel1.add(label);  
  
        /* Put in another panel with two buttons in it  
        JPanel panel2 = new JPanel();  
        this.getContentPane().add(panel2);  
        JButton button1 = new JButton("Make a sound");  
        panel2.add(button1);  
        JButton button2 = new JButton("Make a picture");  
        panel2.add(button2);*/  
  
        this.pack();  
        this.setVisible(true);  
    }  
}
```

Whole GUI Tree

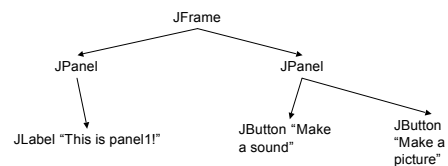


```
public GUItree(){  
    super("GUI Tree Example");  
  
    /* Put in a panel with a label in it */  
    JPanel panel1 = new JPanel();  
    this.getContentPane().add(panel1);  
    JLabel label = new JLabel("This is panel 1!");  
    panel1.add(label);  
  
    /* Put in another panel with two buttons in it */  
    JPanel panel2 = new JPanel();  
    this.getContentPane().add(panel2);  
    JButton button1 = new JButton("Make a  
    sound");  
    panel2.add(button1);  
    JButton button2 = new JButton("Make a  
    picture");  
    panel2.add(button2);  
  
    this.pack();  
    this.setVisible(true);  
}
```

Where'd panel1 go?

- It's there, but the current rendering is smashing it all together.

GUItree is a tree

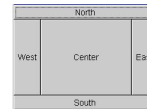


Layout Managers are renderers

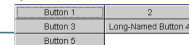
- How are the components assigned a position and size?
 - `setLayout(null)` - the programmer must give all components a size and position
 - `setBounds(topLeftX,topLeftY,width,height);`
- Better: Use a Layout Manager!
 - Arranges the components in a container and sets their size as well
 - Handles when the main window is resized
 - The programmer just adds the components to the container

Layouts - Flow, Border, Grid

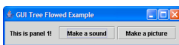
- Flow Layout - left to right, no extra space
- Border Layout - Center item gets extra space



- Grid Layout - same size components



Flowed GUItree



```
> GUItreeFlowed gtf
= new
GUItreeFlowed();
```

```
/**
 * A GUI that has various components in it, to demonstrate
 * UI components and layout managers (rendering)
 */
import javax.swing.*; // Need this to reach Swing components
import java.awt.*; // Need this to reach FlowLayout

public class GUItreeFlowed extends JFrame {

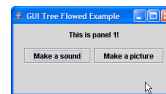
    public GUItreeFlowed(){
        super("GUI Tree Flowed Example");

        this.getContentPane().setLayout(new FlowLayout());
        /* Put in a panel with a label in it */
        JPanel panel1 = new JPanel();
        this.getContentPane().add(panel1);
        JLabel label = new JLabel("This is panel 1!");
        panel1.add(label);

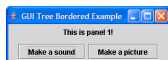
        /* Put in another panel with two buttons in it */
        JPanel panel2 = new JPanel();
        this.getContentPane().add(panel2);
        JButton button1 = new JButton("Make a sound");
        panel2.add(button1);
        JButton button2 = new JButton("Make a picture");
        panel2.add(button2);

        this.pack();
        this.setVisible(true);
    }
}
```

Automatically reflows when resized



Bordered GUItree



```
> GUItreeBordered gtb =
new GUItreeBordered();
```

```
/**
 * A GUI that has various components in it, to demonstrate
 * UI components and layout managers (rendering)
 */
import javax.swing.*; // Need this to reach Swing components
import java.awt.*; // Need this to reach BorderLayout

public class GUItreeBordered extends JFrame {

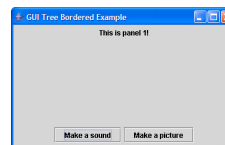
    public GUItreeBordered(){
        super("GUI Tree Bordered Example");

        this.getContentPane().setLayout(new BorderLayout());
        /* Put in a panel with a label in it */
        JPanel panel1 = new JPanel();
        this.getContentPane().add(panel1, BorderLayout.NORTH);
        JLabel label = new JLabel("This is panel 1!");
        panel1.add(label);

        /* Put in another panel with two buttons in it */
        JPanel panel2 = new JPanel();
        this.getContentPane().add(panel2, BorderLayout.SOUTH);
        JButton button1 = new JButton("Make a sound");
        panel2.add(button1);
        JButton button2 = new JButton("Make a picture");
        panel2.add(button2);

        this.pack();
        this.setVisible(true);
    }
}
```

Resizing now follows borders

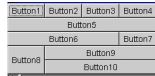


Other Layouts - None, GridBag, Card

- None (null) - programmer specified



- GridBag - flexible grid



- Card - one card shown at a time



BoxLayout

- Two types
 - Horizontal - `BoxLayout.X_AXIS`
 - Vertical - `BoxLayout.Y_AXIS`
- Can use rigidAreas to leave a set amount of space between components
 - `Box.createRigidArea(new Dimension(0,5));`
- Can use horizontal and/or vertical glue to take up extra space
 - `Box.createHorizontalGlue();`



Boxed GUItree

BoxLayout is weird—it takes the pane as an input, and whether you want vertical or horizontal (Y or X_AXIS) “boxing”

```

/**
 * A GUI that has various components in it, to demonstrate
 * UI components and layout managers (rendering)
 */
import javax.swing.*; // Need this to reach Swing components

public class GUItreeBoxed extends JFrame {

    public GUItreeBoxed(){
        super("GUI Tree Boxed Example");

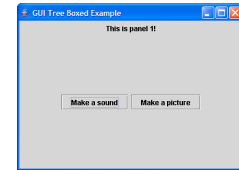
        this.getContentPane().setLayout(new
            BoxLayout(this.getContentPane(),
                BoxLayout.Y_AXIS));

        /* Put in a panel with a label in it */
        JPanel panel1 = new JPanel();
        this.getContentPane().add(panel1);
        JLabel label = new JLabel("This is panel 1!");
        panel1.add(label);

        /* Put in another panel with two buttons in it */
        JPanel panel2 = new JPanel();
        this.getContentPane().add(panel2);
        JButton button1 = new JButton("Make a sound");
        panel2.add(button1);
        JButton button2 = new JButton("Make a picture");
        panel2.add(button2);

        this.pack();
        this.setVisible(true);
    }
}
    
```

Differs in resizing behavior: It's all in boxes (not borders)

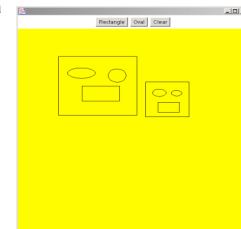


Which Layout to Use?

- An applet or application can have multiple panels (JPanel) and have a different layout in each panel.
 - Panels can be inside of other panels.
- If you want components to not use extra space and stay centered then use `FlowLayout()`
- Or use `BorderLayout` and put one component that uses all extra space in the center.
- Use a `Box` and line up components vertically or horizontally
- For the most control use `null` layout.
 - Much like `LayeredSceneElement!`

Nested Panel Example

- Often an application uses a `BorderLayout`
 - Main panel in Center
 - Other panels in North, South, West, and East as needed
 - Using `FlowLayout` or `Box`
- In the application at right
 - The main panel is in the center
 - The button panel is in the north
 - Using `FlowLayout`



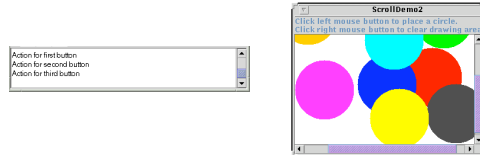
An Cavalcade of Swing Components

- Next few slides show you some of the *many* user interface components in Swing.
- You don't have to know *all* of these!
 - They're here for your benefit.
- Wait a few slides, and we'll go through how to use basic buttons and text.




Swing JScrollPane

- JScrollPane - adds scroll bars to component


```
textArea = new JTextArea(5, 30);
JScrollPane scrollPane = new JScrollPane(textArea);
contentPane.add(scrollPane, BorderLayout.CENTER);
```



Swing Special Purpose Containers

- JTabbedPane - display contents of current tab
 
- JToolBar - groups buttons with icons
 
- JOptionPane - display dialog box
 
- JInternalFrame - inside frames
 

Swing Text Components

- JLabel - not editable text and/or image



```
JLabel firstNameLabel = new JLabel("Label 5", dukeIcon);
```


- JTextField - one line text entry and/or display


```
JTextField nameField = new JTextField(40);
String name = nameField.getText();
```


- JPasswordField - hides typed characters


```
JPasswordField passField = new JPasswordField(8);
String password = passField.getPassword();
```


- JTextArea - multi-line text entry and/or display



```
JTextArea commentArea = new JTextArea(2,30);
String comment = commentArea.getText();
commentArea.setText(comment);
```




Swing List Components

- JList - displays a list of items and user may select one or more


```
Color colors[] = {"Black", "Blue", "Green"};
JList colorList = new JList(colors);
colorList.setVisibleRowCount(2);
String color = colorList.getSelectedValue();
```


- JComboBox - drop down list with selected displayed, can set up for text entry too



```
JComboBox colorBox = new JComboBox(colorList);
String currColor = colorBox.getSelectedItem();
```




Swing Slider and Progress Bar

- JSlider - show a value in a range or pick a value from a continuous range


```
s = new JSlider(100, 1000, 400);
s.setPaintTicks(true);
s.setMajorTickSpacing(100);
s.getValue(); // get the current value from a slider
```


- JProgressBar - used to show how long a user needs to wait yet.


```
progressBar = new JProgressBar(JProgressBar.HORIZONTAL, 0, text.length());
```



Color Chooser

- JColorChooser - use to pick a color
 - Use the static method `showDialog` and pass it the parent component, title, and current color


```
Color newColor = JColorChooser.showDialog(
    parentComponent, title, selColor);
```
 - Example


```
Color newColor = JColorChooser.showDialog(
    this, "Pick a new background
    color", this.getBackground());
```



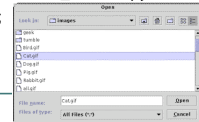
File Chooser

- JFileChooser - use to pick a file


```
// create the file chooser
final JFileChooser fc = new JFileChooser();

// display the chooser as a dialog and get the return value
int returnVal = fc.showOpenDialog(frame);

// if the return value shows that the user selected a file
if (returnVal == JFileChooser.APPROVE_OPTION) {
    File file = fc.getSelectedFile();
}
```

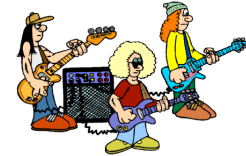


Key to Interactive User Interfaces: Events

- An event is an object that represents an action:
 - user clicks the mouse
 - user presses a key on the keyboard
 - user closes a window
- In Swing, objects add or implement *listeners* for events.
 - Listeners are *interfaces*.
 - Interfaces are not classes: They define functionality that other classes implement.
 - It's a contract that certain functionality will be provided.

Events and Listeners

- Say you want to know when your favorite band will be giving a tour in your city
- You might sign-up to be notified and give your e-mail address
 - Your name and e-mail is added to a list
- When the event is scheduled in your city
 - You will be notified via e-mail that the tour is coming



Events and Listeners

Event	Listener	Example
ActionEvent	ActionListener	Button Pushed
AdjustmentEvent	AdjustmentListener	Move a scrollbar
FocusEvent	FocusListener	Tab into a text area
ItemEvent	ItemListener	Checkbox checked
KeyEvent	KeyListener	Keystroke occurred in a component
MouseEvent	MouseListener	Mouse button click
MouseEvent	MouseMotionListener	Mouse moves or drags
TextEvent	TextListener	A text's component text changed
WindowEvent	WindowListener	Window was closed

Adapters

- An adapter is an abstract class that provides empty implementations for a listener interface.
 - You can *inherit* from an adapter and only *override* the methods you want to handle.

```
class MyMouseListener extends MouseAdapter
{
    /** Method to handle the click of a mouse */
    public void mouseClicked(MouseEvent e)
    { ... }
}
```

Named Inner Classes

- In Swing, you can use *inner classes* which are classes declared inside another class.

```
public class ClassName
{
    attributes
    constructors
    methods
    // named inner class
    class MyMouseAdapter extends MouseAdapter
    {
        methods
    }
}
```

Anonymous Inner Classes

- You can create a new listener in place with an anonymous inner class

```
b.addFocusListener(new FocusListener () {
    public void focusGained (FocusEvent evt) {
        ...
    }
    public void focusLost(FocusEvent evt) {
        ...
    }
});
```

Interactive GUItree: Starting out

```
/**
 * A GUI that has various components in it, to demonstrate
 * UI components and layout managers (rendering).
 * Now with Interactivity!
 */
import javax.swing.*; // Need this to reach Swing components
import java.awt.*; // Need this to reach FlowLayout
import java.awt.event.*; // Need this for listeners and events

public class GUItreeInteractive extends JFrame {

    public GUItreeInteractive(){
        super("GUI Tree Interactive Example");

        this.getContentPane().setLayout(new FlowLayout());
        /* Put in a panel with a label in it */
        JPanel panel1 = new JPanel();
        this.getContentPane().add(panel1);
        JLabel label = new JLabel("This is panel 1!");
        panel1.add(label);
    }
}
```

Interactive GUItree: First button

```
/* Put in another panel with two buttons in it */
JPanel panel2 = new JPanel();
this.getContentPane().add(panel2);
JButton button1 = new JButton("Make a sound");
button1.addActionListener(
    new ActionListener() { // Here's the listener
        // Here's the method we're overriding
        public void actionPerformed(ActionEvent e) {
            Sound s = new Sound(FileChooser.getMediaPath("warble-h.wav"));
            s.play();
        }
    }
);
panel2.add(button1);
```

Interactive GUItree: Second button

```
JButton button2 = new JButton("Make a picture");
button2.addActionListener(
    new ActionListener() { // Here's the listener
        // Here's the method we're overriding
        public void actionPerformed(ActionEvent e) {
            Picture p = new Picture(FileChooser.getMediaPath("shops.jpg"));
            p.show();
        }
    }
);
panel2.add(button2);

this.pack();
this.setVisible(true);
}
```

An inner class can access *instance variables* of the outer class, but not local variables of the method.

Example: A Rhythm Constructing Tool

- Take a name of a sound to add to a root
 - Weave a number of times
 - Repeat a number times
- Play the result



Building the RhythmTool class

```
/**
 * A Rhythm-constructing tool
 **/
import javax.swing.*; // Need this to reach Swing components
import java.awt.*; // Need this to reach FlowLayout
import java.awt.event.*; // Need this for listeners and events

public class RhythmTool extends JFrame {
    /* Base of sound that we're creating */
    public SoundElement root;
    /* Sound that we're creating to add in. */
    public SoundElement newSound;
    /* Declare these here so we can reach them inside listeners */
    private JTextField filename;
    private JTextField count;
    int num;
}
```

Each of the values that we'll access from *inside* the listeners must be declared as *instance variables (fields)* of the tools.

Starting the Window (JFrame)

```
public RhythmTool(){
    super("Rhythm Tool");

    root = new SoundElement(new Sound(1)); // Nearly empty sound
    newSound = new SoundElement(new Sound(1)); // Ditto

    // Layout for the window overall
    this.getContentPane().setLayout(new BorderLayout());
}
```

Filename field

```
/* First panel has new sound field */
JPanel panel1 = new JPanel();
// Put panel one at the top
this.getContentPane().add(panel1, BorderLayout.NORTH);
// Create a space for entering a new sound filename
filename = new JTextField("soundfilename.wav");
filename.addActionListener(
    new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            /* When hit return in filename field,
             * create a new sound with that name.
             * Printing is for debugging purposes.
             **/
            newSound = new SoundElement(
                new Sound(
                    FileChooser.getMediaPath(filename.getText())));
            System.out.println("New sound from "+
                FileChooser.getMediaPath(filename.getText()));
        }
    }
);
panel1.add(filename);
```

Number field

```
/* Put in another panel with number field
 * and repeat & weave buttons */
JPanel panel2 = new JPanel();
// This layout is for the PANEL, not the WINDOW
panel2.setLayout(new BorderLayout());
// Add to MIDDLE of WINDOW

this.getContentPane().add(panel2, BorderLayout.CENTER);
// Add a field for arguments for Repeat and Weave
count = new JTextField("10");
num = 10; // Default value
count.addActionListener(
    new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            // Here's how we convert a string to a number
            num = Integer.parseInt(count.getText());
        }
    }
);
// Add to top of panel
panel2.add(count, BorderLayout.NORTH);
```

Repeat button

```
// Now do the Repeat button
JButton button1 = new JButton("Repeat");
button1.addActionListener(
    new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            // Repeat the number of times specified
            root.repeatNext(newSound, num);
        }
    }
);
// Add to RIGHT of PANEL
panel2.add(button1, BorderLayout.EAST);
```

Weave button

```
// Now do the Weave button
JButton button2 = new JButton("Weave");
button2.addActionListener(
    new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            // We'll weave 10 copies in
            // every num times
            root.weave(newSound, 10, num);
        }
    }
);
// Add to LEFT of PANEL
panel2.add(button2, BorderLayout.WEST);
```


Play Button (and end)

```
/* Put in another panel with the Play button */
JPanel panel3 = new JPanel();
// Put in bottom of WINDOW
this.getContentPane().add(panel3, BorderLayout.SOUTH);
JButton button3 = new JButton("Play");
button3.addActionListener(
    new ActionListener() {
        // If this gets triggered, play the composed sound
        public void actionPerformed(ActionEvent e) {
            root.playFromMeOn();
        }
    }
);
panel3.add(button3); // No layout manager here

this.pack();
this.setVisible(true);
}
```