Tutorial 2: Java Swing

ECSE 321: Intro to Software Engineering
Electrical and Computer Engineering

McGill University

Winter 2009

Contents

1 Overview

1.1 What is Swing?

What is Swing?

- **Swing is a set of program components for Java programmers that provide the ability to create graphical user interface (GUI) components,**
- **Replaces the *Abstract Window Toolkit* or *AWT* as of Java 1.1**
- **Some of the features include:**
  - **Lightweight. Not built on native window-system windows.**
  - **Much bigger set of built-in controls. Trees, image buttons, tabbed panes, sliders, toolbars, color choosers, tables, text areas to display HTML or RTF, etc.**
  - **Much more customizable; Can change border, text alignment, or add image to almost any control.**
  - **Can change look and feel at runtime, or design own look and feel.**
  - **Model-View-Controller architecture lets you change the internal data representation (lists, trees, tables).**

1.3 Using Swing

How do we use Swing?

- **Swing provides many standard GUI components such as buttons, lists, menus, and text areas, which you combine to create your program's GUI.**
- **Swing components start with the letter J; JFrame, JButton, etc.**
- **Use *containers* and *layout managers* to create windows.**
- **Use *components* and *event handlers* for user interaction.**
- `import javax.swing.*;`

2 Containers

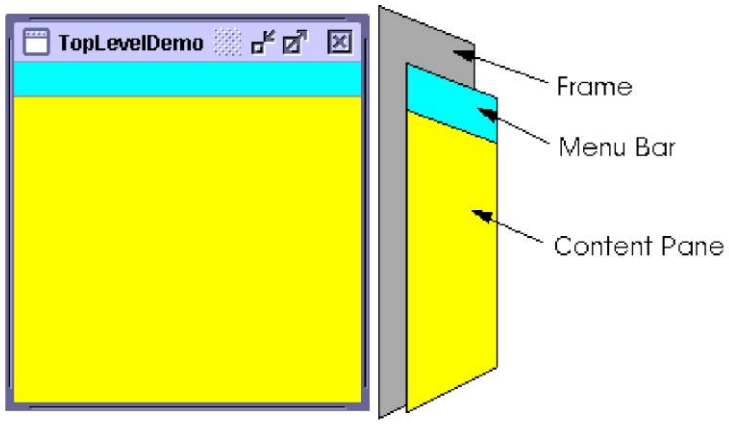
2.1 Top Level Containers

Swing Containers

- **Every Java program that has a GUI has at least one top-level container.**
- **Swing provides containers such as windows and tool bars.**
  - **JFrame, JDialog**
  - **JPanel, JTabbedPane, JScrollPane, JInternalFrame**

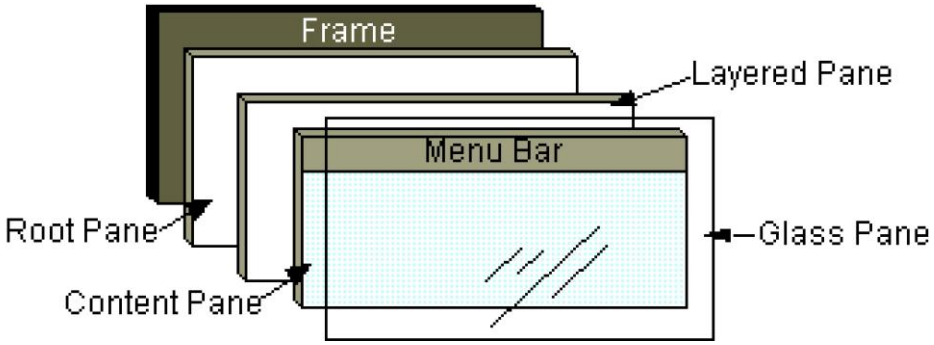
Top-Level Containers I

- **Every GUI component must be part of a containment hierarchy. .**
- **Each GUI component can be contained only once.**
- **Each top-level container has a content pane.**
- **You can optionally add a menu bar to a top-level container.**



Top-Level Containers II

- **The root pane manages the content pane and the menu bar, along with a couple of other contain-ers.**
- **The layered pane directly contains the menu bar and content pane**
- **The glass pane is often used to intercept input events occurring over the top-level container, and can also be used to paint over multiple components.**

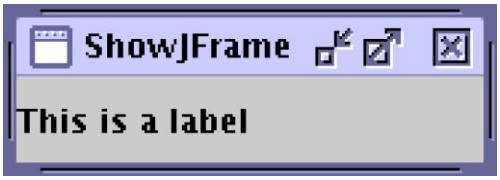


Frames

- **A JFrame is a window that has decorations such as a border, a title and buttons for closing and iconifying the window.**
- **The decorations on a frame are platform dependent.**
- **Use** `JFrame.getContentPane.add()` **to add components.**

- `Pack()` **sets the framesize based on the preferred sizes of sub-components.**

```
import javax.swing.*;

class ShowFrame {

   public static void main(String args[]) {
          JFrame frame = new JFrame("ShowJFrame");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      JLabel label = new JLabel("This is a label");
      frame.getContentPane().add(label);
      frame.pack();
      frame.setVisible(true);
   }
}
```



2.2 Intermediate Containers

Intermediate Level Containers

- **Also known as *panels* or *panes*.**
- **Simplify the positioning of other components:**
  - `JPanel`
- **Play a visible and interactive role:**
  - `JScrollPane`
  - `JTabbedPane`

JPanel

```
import javax.swing.*;

class ShowPanel {
   public static void main(String args[]) {
      JFrame frame = new JFrame("ShowJPanel");
      JPanel panel = new JPanel();
      JLabel label = new JLabel(
            "This is a label with some text in it.");
      JButton button = new JButton("Click Me");
      panel.add(label);
      panel.add(button);
      frame.getContentPane().add(panel);
      frame.pack();
      frame.setVisible(true);
   }
}
```

JScrollPane

- **A `JScrollPane` provides a scrollable view of a component.**

```
import javax.swing.*;

class ShowScrollPane {

   public static void main(String args[]) {
      JFrame frame = new JFrame("ShowScrollPane");
      JPanel panel = new JPanel();
      JLabel label = new JLabel(
            "This is a label with some text in it.");
      JButton button = new JButton("Click Me");
      panel.add(label);
      panel.add(button);
      JScrollPane sp = new JScrollPane(panel); frame
      .getContentPane().add(sp);
      frame.pack();
      frame.setVisible(true);
   }
}
```

2.3 Layout Management

Layout Management

- **The process of determining the size and position of components.**
- **Layout management can be done using *absolute* positioning.**
  - **Difficult and will cause major headaches.**
- **Better to use layout mangers:**
  - **Components can provide size and position hints to layout managers.**
  - `setPreferredSize`, `setMinimumSize`, `setMaximumSize`

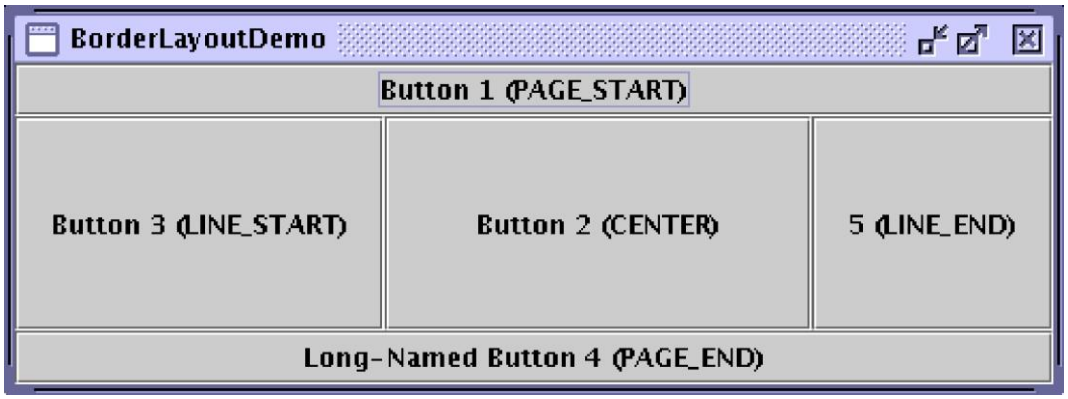
Available Layout Managers

- **Swing provides us with several layout managers:**
  - `BorderLayout`
  - `BoxLayout`
  - `CardLayout`
  - `FlowLayout`
  - `GridBagLayout`
  - `GridLayout`
  - `SpringLayout`

BorderLayout

- **Every content pane is initialized to use a** `BorderLayout`**.**
- **A BorderLayout places components in up to five areas: top, bottom, left, right, and center.**
- **When you resize a frame, the center portion get squeezed/expanded.**

```
pane . add(button , BorderLayout .CENTER) ; pane .
add(button , BorderLayout .LINE_START) ; pane .
add(button , BorderLayout .PAGE_END) ; pane .
add(button , BorderLayout .LINE_END) ;
```



FlowLayout

- `FlowLayout` **is the default layout manager for every** `JPanel`**.**
- **Lays out components in a single row, starting a new row if its container isn't sufficiently wide.**

GridBagLayout I

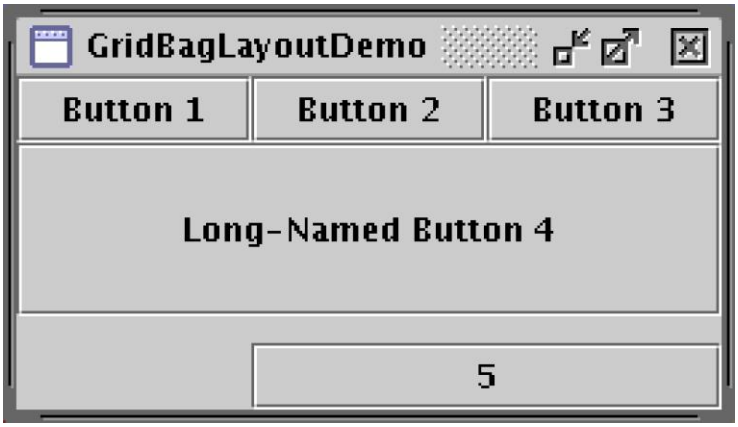
- `GridBagLayout` is a sophisticated, flexible layout manager.
- It aligns components by placing them within a grid of cells,
  - A component can span more than one cell.
  - The rows in the grid can have different heights
  - Grid columns can have different widths.

GridBagLayout II

- Use `GridBagContraints` to tell the layout manager how to handle components.
  - *gridx, gridy:* Specify the row and column at the upper left of the component.
  - *gridwidth, gridheight:* Specify the number of columns (for gridwidth) or rows (for gridheight) in the component's display area.
  - *anchor:* Used when the component is smaller than its display area to determine where (within the area) to place the component.
  - *weightx, weighty:* Weights are used to determine how to distribute space among columns this is important for specifying resizing behavior.
  - See docs for more info

GridBagLayout III

```
pane . setLayout(new GridBagLayout ( ) ) ;
GridBagConstraints c = new GridBagConstraints ( ) ; button
= new JButton ( "Long−Named Button 4" ) ;
c . ipady = 40;
c . weightx = 0.0; c .
gridwidth = 3; c .
gridx = 0;
c . gridy = 1;
pane . add(button , c ) ;
```



Layout Tips

- When building a GUI, *don't* use a single `JPanel` to hold everything.
  - Partition your GUI into several smaller panels to create a hierarchy
- Use `BorderLayout` for your top-level panel/pane.
- `FlowLayout`, `GridBagLayout` should be enough to handle all your needs.
- If you're having a hard time laying out your GUI, you probably didn't partion your components well.

2.5 JDialog and JOptionPane

JDialog I

- Every dialog is dependent on a frame
- Destroying a frame destroys all its dependent dialogs.
- When the frame is iconified, its dependent dialogs disappear from the screen.
- When the frame is deiconified, its dependent dialogs return to the screen.
- A dialog can be modal. When a modal dialog is visible it blocks user input to all other windows in the program.

JDialog II

- Swing provides several standard dialogs

  – `JFileChooser` –

  `JProgressBar` –

  `JColorChooser`

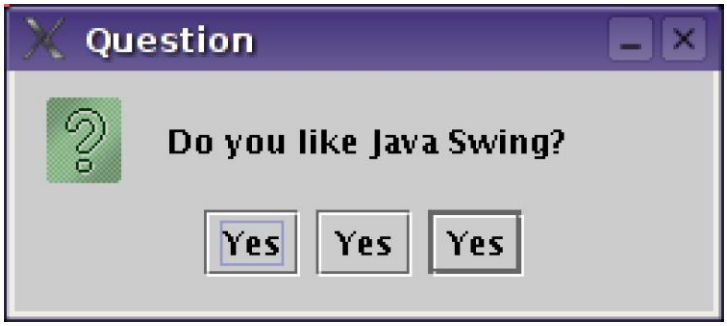
- Can create custom dialogs using `JDialog`, but for most applications, `JOptionPane` is sufficient.

JOptionPane

- The JOptionPane class can be used to create simple modal dialogs.
- Icons, title, text and buttons can be customized.

```
import javax.swing.*;

class ShowDialog {

   public static void main(String args[]) {

      JFrame frame = new JFrame();
      Object[] options = {"Yes", "Yes", "Yes"};

      int n = JOptionPane.showOptionDialog(
            frame,
            "Do you like Java Swing?",
            "Question",
```

```
            JOptionPane .YES_NO_CANCEL_OPTION,
            JOptionPane .QUESTION_MESSAGE, null ,

            options ,
            options [2]);

        System . exit (0);
    }
}
```



3 Components and Events

3.1 Overview

Components and Events I

- Swing provides many components which allow the user to interact with a program.
  - JButton, JToggleButton, JCheckBox, JRadioButton
  - JList, JComboBox, JTextField, JTextArea, JTable, JTree
  - JFileChooser, JColorChooser, JSlider, JProgressBar, JPasswordField

Components and Events II

- Every time a user types a character or pushes a mouse button, an event occurs.
- Any object can be notified of an event by registering as an event listener on the appropriate event source.
- Multiple listeners can register to be notified of events of a particular type from a particular source.
- A single listener can be registered with many sources.

3.3 Event Listeners

Event Listener Interfaces

- `ActionListener`: One method to receive action events.
- `FocusListener`: Gain/loss of keyboard focus.
- `ItemListener`: The state of an item changes.

- `KeyListener`: **Key is pressed, released or typed.**
- `MouseListener`: **Mouse is pressed, released, clicked over a component.**
- `MouseMotionListener`: **Cursor moves over a component.**

Implementing an Event Handler

- **Implement a listener interface or extend a class that implements a listener interface.**
- **Register an instance of the event handler class as a listener upon one or more components.**
- **Implement the methods in the listener interface to handle the event.**

ActionListener Interface

- **Action listeners are the easiest and most common event handlers to implement.**
- **We only need to override one method to handle events.**

```
public interface ActionListener {

   void actionPerformed(ActionEvent e ) ;

}
```

ActionListener Example

- **Make a program that has a label and two buttons.**
  - **The label displays an integer.**
  - **One button increments the integer**
  - **One button decrements.**
- **How would you do this?**

ActionListener Example II

- **Two inner classes implement the** `ActionListener` **interface.**

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

class Action1 extends JPanel { int
   count = 0;
   JLabel label = new JLabel ( "0" , JLabel .CENTER) ;
```

```
    public  Action1 ()  {
        JButton inc = new JButton ( "+" ) ;
        JButton dec = new JButton ( "−" ) ;
        inc . addActionListener (new IncListener ( ) ) ; dec .
        addActionListener  (new  DecListener  ( )  )  ;  this  .
        setLayout(new BorderLayout ( ) ) ;
        this . add( inc , BorderLayout .LINE_START) ;
        this . add( label ) ;
        this . add(dec,   BorderLayout .LINE_END) ;
    }

    class IncListener implements ActionListener { public void
        actionPerformed(ActionEvent e) {
            label . setText ( ""  + (++count ) ) ;
        }
    }
```

```
    class DecListener implements ActionListener { public
        void actionPerformed(ActionEvent e) {
            label . setText ( ""  + (−−count ) ) ;
        }
    }

    public static void main( String args [ ] ) { JFrame .
        setDefaultLookAndFeelDecorated( true ) ; JFrame
        frame = new JFrame( "Inc /Dec" ) ;
        JPanel panel = new Action1 ( ) ; frame .
        setDefaultCloseOperation (
            JFrame .EXIT_ON_CLOSE) ;
        frame . getContentPane ( ) . add(panel ) ;
        frame . pack ( ) ;
        frame . setVisible ( true ) ;
    }
}
```

ActionListener Example II

- **A single instance of `IncDecListener` is registered to both buttons.**

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

class Action2 extends JPanel { int
    count = 0;
    JLabel  label  = new  JLabel ( "0" ,  JLabel .CENTER) ;

    public  Action2 ()  {
        JButton inc = new JButton ( "+" ) ;
        JButton dec = new JButton ( "−" ) ;
        ActionListener al = new IncDecListener ( ) ; inc .
        setActionCommand( "inc" ) ;
        inc . addActionListener ( al ) ;
```

```
        dec . setActionCommand( "dec" ) ;
        dec . addActionListener ( al ) ;
        this . setLayout(new  BorderLayout ( ) ) ;
        this . add( inc ,   BorderLayout .LINE_START) ;
        this . add( label ) ;
        this . add(dec,   BorderLayout .LINE_END) ;
    }
```

```
    class  IncDecListener  implements  ActionListener  {
        public  void  actionPerformed(ActionEvent  e)  {
            String  s  =  e .getActionCommand ( ) ;
            i f (s . equals( "inc" ))  {
                count++;
            } else   i f (s . equals( "dec" ))       {
                count−−;
            }
            label . setText ( ""  +  count ) ;
        }
    }

    public   static   void  main( String          args [ ] )  {
        JFrame . setDefaultLookAndFeelDecorated( true ) ;
        JFrame  frame  = new  JFrame( "Inc /Dec" ) ;
        JPanel  panel  = new  Action2 ( ) ;
        frame . setDefaultCloseOperation (
            JFrame .EXIT_ON_CLOSE) ;
        frame . getContentPane ( ) . add(panel ) ;
        frame . pack ( ) ;
        frame . setVisible ( true ) ;
    }
}
```

4 Look and Feel

4.1 Look and Feel

Look and Feel in Java

- **Can change the look and feel of Java programs.**
- **A look and feel can be provided via a JAR file.**
- **The standard cross-platform look and feel is called *Metal*.**
- **Java 1.4.2 introduces two look and feels**
  - **– *GTK+* is cross-platform and many themes are available. –**
  - ***Microsoft Windows XP* works only on Windows.**
- **If no LAF is specified, the UI manager uses the LAF specified by the *swing.defaultlaf* property.**

4.2 Metal

Metal Look and Feel I

- `UIManager.setLookAndFeel(UIManager .getCrossPlatformLookAndFeelClassName());`

Metal Look and Feel II

- **Can use the default Java window decorations.**
- `JFrame.setDefaultLookAndFeelDecorated(true);`

4.3 GTK

The Gimp Toolkit

- **The *Gimp Toolkit* (GTK+) is a multi-platform widget toolkit for creating graphical user interfaces.**
- **There are *many* themes freely available on the net.**
- **Links:**
  - http://www.gtk.org
  - art.gnome.org/themes/gtk2
  - http://themes.freshmeat.net/browse/958

The GTK Look and Feel

- `UIManager.setLookAndFeel( "com.sun.java.swing.plaf.gtk.GTKLookAndFeel");`
- `java -Dswing.gtkthemefile= "./GTK2-Glossy¯P/Glossy P/gtk-2.0/gtkrc" LookAndFeel`

Look and Feel Source Code

```
import javax . swing . * ;

class LookAndFeel {

   public   static  void  main( String  args [ ] ) {
      try  {
           UIManager. setLookAndFeel( "com. sun . java . swing . plaf . gtk .GTKLookAndFeel" ) ;
      } catch (Exception e) {
           e . printStackTrace ( ) ;
      }
      JFrame frame = new JFrame( "Metal" ) ;
      JPanel  rootPanel = new JPanel ( ) ;
      JPanel  panel1 = new JPanel ( ) ;
      panel1 . add(new JButton ( "Button" ) ) ;
      panel1 . add(new JLabel ( "Label" ) ) ;
      panel1 . add(new JTextArea( "Text  Area" ) ) ;
      JPanel  panel2 = new JPanel ( ) ;
      panel2 . add(new JCheckBox( "Check" ) ) ;
      panel2 . add(new JRadioButton( "Option" ) ) ;
      rootPanel . add(panel1 ) ;
      rootPanel . add(panel2 ) ;

      frame . setDefaultCloseOperation (JFrame .EXIT_ON_CLOSE) ;
      frame . getContentPane ( ) . add( rootPanel ) ;
      frame . pack ( ) ;
      frame . setVisible ( true ) ;
   }
}
```

Final Words

- **The preceding slides gave a *brief* overview of Java Swing**
- **Visit** http://java.sun.com/docs/books/tutorial/index.html **for *much* more info.**