Java GUI Libraries

Swing Programming

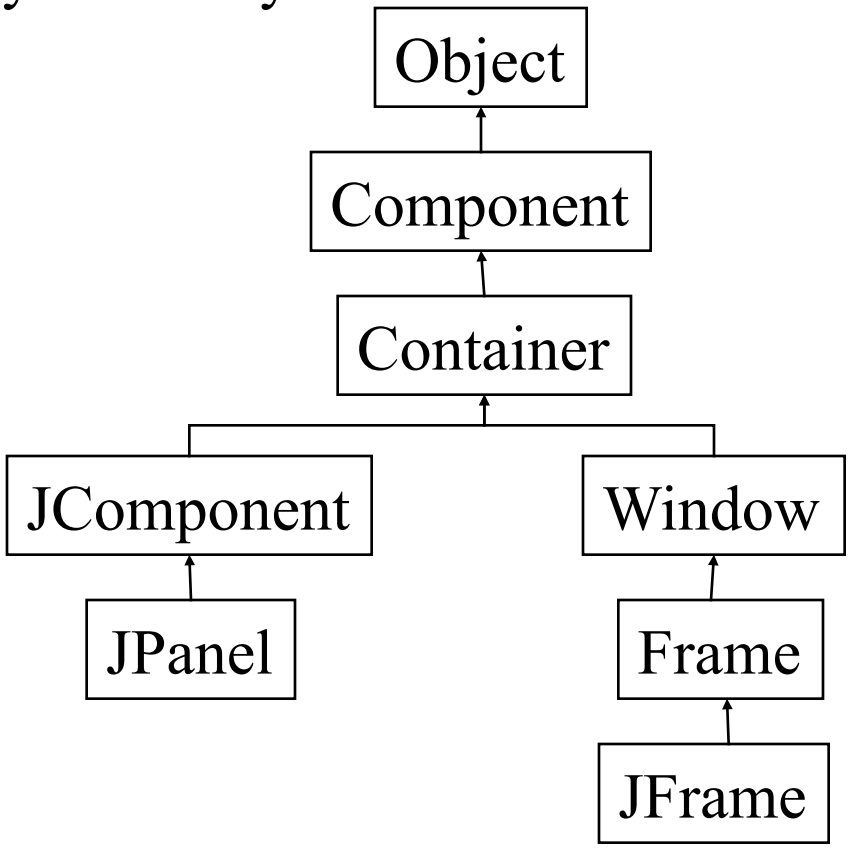
Swing Components

- Swing is a collection of libraries that contains primitive *widgets* or *controls* used for designing *Graphical User Interfaces (*GUIs).

- Commonly used classes in javax.swing package:
  - JButton, JTextBox, JTextArea, JPanel, JFrame, JMenu, JSlider, JLabel, JIcon, …
  - There are many, many such classes to do anything imaginable with GUIs
  - Here we only study the basic architecture and do simple examples

Swing components, cont.

- Each component is a Java class with a fairly extensive inheritency hierarchy:



Using Swing Components

- Very simple, just create object from appropriate class – examples:
  - JButton but = new JButton();
  - JTextField text = new JTextField();
  - JTextArea text = new JTextArea();
  - JLabel lab = new JLabel();
- Many more classes. Don't need to know every one to get started.
- See ch. 9 Hortsmann

Adding components

- Once a component is created, it can be added to a container by calling the container's **add** method:

```
Container cp = getContentPane();
cp.add(new JButton("cancel"));
cp.add(new JButton("go"));
```

This is required

How these are laid out is determined by the layout manager.

Laying out components

- Not so difficult but takes a little practice
- Do not use absolute positioning – not very portable, does not resize well, etc.

Laying out components

- Use layout managers – basically tells form how to align components when they're added.
- Each Container has a layout manager associated with it.
- A JPanel is a Container – to have different layout managers associated with different parts of a form, tile with JPanels and set the desired layout manager for each JPanel, then add components directly to panels.

Layout Managers

- Java comes with 7 or 8. Most common and easiest to use are
  - FlowLayout
  - BorderLayout
  - GridLayout
- Using just these three it is possible to attain fairly precise layout for most simple applications.

Setting layout managers

- Very easy to associate a layout manager with a component. Simply call the **setLayout** method on the Container:

```
JPanel p1 = new JPanel();
p1.setLayout(new FlowLayout(FlowLayout.LEFT));

JPanel p2 = new JPanel();
p2.setLayout(new BorderLayout());
```

As Components are added to the container, the layout manager determines their size and positioning.

Event handling

What are events?

- All components can listen for one or more *events*.
- Typical examples are:
  - Mouse movements
  - Mouse clicks
  - Hitting any key
  - Hitting return key
  - etc.
- Telling the GUI what to do when a particular event occurs is the role of the event handler.

ActionEvent

- In Java, most components have a special event called an *ActionEvent*.
- This is loosely speaking the most common or canonical event for that component.
- A good example is a click for a button.
- To have any component listen for ActionEvents, you must register the component with an ActionListener. e.g.
  - button.addActionListener(new MyAL());

Delegation, cont.

- This is referred to as the Delegation Model.
- When you register an ActionListener with a component, you must pass it the class which will handle the event – that is, do the work when the event is triggered.
- For an ActionEvent, this class must implement the ActionListener interface.
- This is simple a way of guaranteeing that the actionPerformed method is defined.

actionPerformed

- The actionPerformed method has the following signature:

  void actionPerformed(ActionEvent)
- The object of type ActionEvent passed to the event handler is used to query information about the event.
- Some common methods are:
  - getSource()
    - object reference to component generating event
  - getActionCommand()
    - some text associated with event (text on button, etc).

actionPerformed, cont.

- These methods are particularly useful when using one eventhandler for multiple components.

Other Events

- You're not limited to **ActionListener**
- Each type of event represented by a class
- Component responds to an event by making an event object and calling each "listener" registered for that event
- An event listener **implements** a particular listener interface using an inner class
- **addXXXListener( )** adds a listener to your component, **removeXXXListener( )** un-registers it

Event, listener interface and add- and remove-methods	Components supporting this event
ActionEvent **ActionListener** **addActionListener( )** **removeActionListener( )**	**JButton, JList, JTextField, JMenuItem** and its derivatives including **JCheckBoxMenuItem, JMenu,** and **JpopupMenu**.
AdjustmentEvent **AdjustmentListener** **addAdjustmentListener( )** **removeAdjustmentListener( )**	**JScrollbar** and anything you create that implements the **Adjustable** interface.
ComponentEvent **ComponentListener** **addComponentListener( )** **removeComponentListener( )**	***Component** and its derivatives, including **JButton, JCanvas, JCheckBox, JComboBox, Container, JPanel, JApplet, JScrollPane, Window, JDialog, JFileDialog, JFrame, JLabel, JList, JScrollbar, JTextArea,** and **JTextField**.

ContainerEvent **ContainerListener** **addContainerListener( )** **removeContainerListener( )**	**Container** and its derivatives, including **JPanel, JApplet, JScrollPane, Window, JDialog, JFileDialog,** and **JFrame.**
FocusEvent **FocusListener** **addFocusListener( )** **removeFocusListener( )**	**Component** and derivatives*.
TextEvent **TextListener** **addTextListener( )** **removeTextListener( )**	Anything derived from **JTextComponent,** including **JTextArea** and **JTextField**.
KeyEvent **KeyListener** **addKeyListener( )** **removeKeyListener( )**	**Component** and derivatives*.

MouseEvent (for both clicks and motion) **MouseListener** **addMouseListener( )** **removeMouseListener( )**	**Component** and derivatives*.
MouseEvent (for both clicks and motion) **MouseMotionListener** **addMouseMotionListener( )** **removeMouseMotionListener( )**	**Component** and derivatives*.
WindowEvent **WindowListener** **addWindowListener( )** **removeWindowListener( )**	**Window** and its derivatives, including **JDialog, JFileDialog,** and **JFrame**.
ItemEvent **ItemListener** **addItemListener( )** **removeItemListener( )**	**JCheckBox, JCheckBoxMenuItem, JComboBox, JList,** and anything that implements the **ItemSelectable** interface.

Listener interface w/ adapter	Methods in interface
ActionListener	actionPerformed(ActionEvent)
AdjustmentListener	adjustmentValueChanged(AdjustmentEvent)
ComponentListener ComponentAdapter	componentHidden(ComponentEvent) componentShown(ComponentEvent) componentMoved(ComponentEvent) componentResized(ComponentEvent)
ContainerListener ContainerAdapter	componentAdded(ContainerEvent) componentRemoved(ContainerEvent)
FocusListener FocusAdapter	focusGained(FocusEvent) focusLost(FocusEvent)
KeyListener KeyAdapter	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)

MouseListener MouseAdapter	mouseClicked(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mousePressed(MouseEvent) mouseReleased(MouseEvent)
MouseMotionListener MouseMotionAdapter	mouseDragged(MouseEvent) mouseMoved(MouseEvent)
WindowListener WindowAdapter	windowOpened(WindowEvent) windowClosing(WindowEvent) windowClosed(WindowEvent) windowActivated(WindowEvent) windowDeactivated(WindowEvent) windowIconified(WindowEvent) windowDeiconified(WindowEvent)
ItemListener	itemStateChanged(ItemEvent)

Simplest GUI

```
import javax.swing.JFrame;
class SimpleGUI extends JFrame{
        SimpleGUI(){
            setSize(400,400); //set frames size in pixels
            setDefaultCloseOperation(EXIT_ON_CLOSE);
            show();
        }

        public static void main(String[] args){
            SimpleGUI gui = new SimpleGUI();
            System.out.println(“main thread coninues”);
        }
}
```

Another Simple GUI

```
import javax.swing.*;
class SimpleGUI extends JFrame{
          SimpleGUI(){
            setSize(400,400); //set frames size in pixels
            setDefaultCloseOperation(EXIT_ON_CLOSE);
            JButton but1 = new JButton(“Click me”);
             Container cp = getContentPane();//must do this
            cp.add(but1);
              show();
        }

        public static void main(String[] args){
            SimpleGUI gui = new SimpleGUI();
            System.out.println(“main thread coninues”);
        }}
```

Add Layout Manager

```
import javax.swing.*; import java.awt.*;
class SimpleGUI extends JFrame{
        SimpleGUI(){
          setSize(400,400); //set frames size in pixels
          setDefaultCloseOperation(EXIT_ON_CLOSE);
          JButton but1 = new JButton("Click me");
           Container cp = getContentPane();//must do this
          cp.setLayout(new FlowLayout(FlowLayout.CENTER);
            cp.add(but1);
            show();
       }

       public static void main(String[] args){
          SimpleGUI gui = new SimpleGUI();
          System.out.println("main thread coninues");
       }}
```

Add call to event handler

```
import javax.swing.*; import java.awt.*;
class SimpleGUI extends JFrame{
        SimpleGUI(){
          setSize(400,400); //set frames size in pixels
          setDefaultCloseOperation(EXIT_ON_CLOSE);
          JButton but1 = new JButton(“Click me”);
           Container cp = getContentPane();//must do this
          cp.setLayout(new FlowLayout(FlowLayout.CENTER);
            but1.addActionListener(new MyActionListener());
            cp.add(but1);
            show();
      }
        public static void main(String[] args){
          SimpleGUI gui = new SimpleGUI();
          System.out.println(“main thread coninues”);
      }}
```

Event Handler Code

```
class MyActionListener implements ActionListener{
   public void actionPerformed(ActionEvent ae){
        JOptionPane.showMessageDialog("I got clicked", null);
   }



}
```

Add second button/event

```
class SimpleGUI extends JFrame{
        SimpleGUI(){
            /* .... */
            JButton but1 = new JButton(“Click me”);
            JButton but2 = new JButton(“exit”);
            MyActionListener al = new MyActionListener();
            but1.addActionListener(al);
            but2.addActionListener(al);
            cp.add(but1);
             cp.add(but2);
            show();
        }
}
```

How to distinguish events –Less good way

```
class MyActionListener implents ActionListener{
   public void actionPerformed(ActionEvent ae){
      if (ae.getActionCommand().equals(“Exit”){
          System.exit(1);
       }
       else if (ae.getActionCommand().equals(“Click me”){
           JOptionPane.showMessageDialog(null, “I’m clicked”)
       }


}
```

Good way

```
class MyActionListener implents ActionListener{
   public void actionPerformed(ActionEvent ae){
      if (ae.getSource()  ==  but2){
          System.exit(1);
       }
       else if (ae.getSource() == but1){
           JOptionPane.showMessageDialog(null, “I’m clicked”
       }
}
```

Question: How are but1, but2 brought into scope to do this?
Question: Why is this better?

Putting it all together

- See LoginForm.java example in class notes