Java II

Walter Wesley
UCSD Extension
Information Technology and
Software Engineering

Module 5

Introduction to the Swing GUI Framework

Sample Swing Application

Further Swing Resources

Introduction to the Swing GUI Framework

- The Swing Graphical User Interface (GUI) framework facilititates the development of GUI applications.
- This is an introduction to Swing. Swing is a large topic, and it will be covered in greater depth in Java III.
- The best way to learn Swing is to jump in and start using it.
- The following slides present a basic Swing application.
- Although these slides are copyrighted, please feel free to do whatever you like with the source code that follows.
- The source code will be posted separately, so you can download and experiment with it.
- Further resource links are provided at the very end.

DrawControlApp.java

```
// DrawControlApp.java
import java.awt.Color;
import javax.swing.JFrame;

public class DrawControlApp
{
   public static void main( String args[] )
   {
      JFrame frame = new ControlFrame( "Controlling Multimedia Projects..." );
   }
}
```

- Our ***main*** method is much smaller in this example.
- This is an opportunity for us to play with inheritance.
- So, we define our own specialized ***JFrame***, the ***ControlFrame***.

DrawControlPanel.java

```
// DrawControlPanel.java
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JPanel;

public class DrawControlPanel extends JPanel
{
  private Color fillColor = Color.CYAN;
  private int ovalWidth = 90;

  public DrawControlPanel()
  {
    setSize( 200, 200 );
  }

  public void paintComponent( Graphics g )
  {
    super.paintComponent( g );  // invoke the superclass paint Component
    this.setBackground( Color.WHITE );
    g.setColor( fillColor );
    g.fillOval( 50, 50, ovalWidth, 60 );
  }

  void setFillColor(Color fillColor)
  {
    this.fillColor = fillColor;
  }

  {
  void setOvalWidth(int ovalWidth)
    this.ovalWidth = ovalWidth;
  }

  int getOvalWidth()
  {
    return ovalWidth;
  }
}
```

- But first, let's look at our ***specialized JPanel***.
- A ***JPanel*** is a Swing component that assists in organizing other components—think of it as ***a visual container***.
- Here we are specializing our own ***JPanel*** through inheritance; a DrawControlFrame ***extends JPanel***.

ControlFrame.java

```
// ControlFrame.java
import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.Color;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JMenuBar;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.ItemListener;
import java.awt.event.ItemEvent;
import javax.swing.JOptionPane;
import javax.swing.JSlider;
import javax.swing.SwingConstants;
import javax.swing.event.ChangeListener;
import javax.swing.event.ChangeEvent;
import java.awt.BorderLayout;
import javax.swing.JTextField;
import javax.swing.JLabel;
import javax.swing.JButton;
import javax.swing.JOptionPane;


public class ControlFrame extends JFrame
{
```

- Here our ***ControlFrame*** class ***extends JFrame***.
- We could reduce the number of imports by importing entire packages with a wildcard specifier (e.g., ***import javax.swing.*;***).
- However, it is often a good idea to import each class explicitly. That way, you can see up front which classes are used to implement the current class.

ControlFrame.java

```
{
  private final JPanel calcPanel;
  private JPanel mainPanel;
  private JTextField xValTextField;
  private JTextField yValTextField;
  private JSlider widthJSlider;
  private JButton calcJButton;

  private JLabel calcJLabel;
  private String yStr;

  private String xStr;
  {
    super( title );
  public ControlFrame(String title)
    mainPanel.setSize(200, 250);

    mainPanel = new JPanel( new BorderLayout() );
    calcPanel.setSize(200, 200);

    calcPanel = new JPanel( new FlowLayout() );
    drawPanel.setSize(200, 200);

    final DrawControlPanel drawPanel = new DrawControlPanel();

    JMenu fileMenu = new JMenu( "File" );
```

- We have several ***Swing*** components as ***object fields***, starting with ***mainPanel*** and ending with ***calcButton***.
- The ***final*** keyword means that once a value is assigned to the object field, ***it cannot be changed***.
- The need for the use of the final keyword will be explained later.
- A ***JSlider*** is an input control component. By sliding a ***thumb*** along a slide-bar, the user is able ***to select from a range of integer values***.
- A ***JTextField*** accepts ***text input*** from the user.
- ***JLabels*** display textual information to the user.
- A ***JButton*** can be clicked by the user.

ControlFrame.java

```
{
  private final JPanel calcPanel;
  private JPanel mainPanel;
  private JTextField xValTextField;
  private JTextField yValTextField;
  private JSlider widthJSlider;
  private JButton calcJButton;

  private JLabel calcJLabel;
  private String yStr;

  private String xStr;
  {
    super( title );
  public ControlFrame(String title)
    mainPanel.setSize(200, 250);

    mainPanel = new JPanel( new BorderLayout() );
    calcPanel.setSize(200, 200);

    calcPanel = new JPanel( new FlowLayout() );
    drawPanel.setSize(200, 200);

    final DrawControlPanel drawPanel = new DrawControlPanel();

    JMenu fileMenu = new JMenu( "File" );
```

- The two ***String object fields***, ***xStr*** and ***yStr***, are there to hold onto the text entered by the user into the ***JTextFields***.
- ***ControlFrame*** accepts a ***String*** parameter, which it then passes to its ***Frame superclass***.
- We create two ***JPanels*** and set their size (width, height) in pixels.
- As we create the ***JPanels***, we create and pass in a ***Layout Manager*** object.
- A ***Layout Manager*** controls how the visual components added to the ***JPanel*** will be laid out.
- A ***BorderLayout*** lays items out into ***North***, ***South***, ***East***, ***West***, and ***Center*** areas.
- A ***FlowLayout*** lays items out f***rom left to right***, ***wrapping around*** like words in a sentence.

ControlFrame.java

```
    this.setContentPane( mainPanel );
    fileMenu.setMnemonic( 'F' );
    JMenuItem aboutItem = new JMenuItem( "About..." );
    JMenu fileMenu = new JMenu( "File" );
    fileMenu.add( aboutItem );
    aboutItem.addActionListener(
    aboutItem.setMnemonic( 'A' );
      {
        public void actionPerformed( ActionEvent event )
      new ActionListener()  // Beginning of anonymous inner class
          JOptionPane.showMessageDialog( ControlFrame.this,
                              "This application provides enhanced\n
control over multimedia projects.",
        {
        }
     }  // End of anonymous inner class
                              "About", JOptionPane.PLAIN_MESSAGE );

    final JMenuBar bar = new JMenuBar();  // Create a JMenuBar
so we can attach menus to it.
    );
    bar.add( fileMenu );  // Add the file menu to the JMenuBar.

    setJMenuBar( bar );  // Attach the JMenuBar to the ControlFrame.
    colorMenu.setMnemonic( 'C' );
```

- We set our ***mainPanel*** to be our ***content pane***.
- The ***content pane*** is the ***top-level container*** of our ***JFrame*** (i.e., ***this***).
- We create a ***JMenu*** object, with the ***String*** “File”. This will be our File menu.
- The ***setMnemonic*** method establishes ***quick-access keys***. The designated character is underlined within the menu. By pressing ***Alt*** and the character at the same time, ***the user can select the menu or menu item via the keyboard***.
- We then create a ***JMenuItem*** object, with the ***String*** “About...”. This will be an item that can be selected from the File menu.
- With the ***addActionListener*** method, we ***register*** a parameter that is known as an ***anonymous inner class*** object—in this way we specify ***how to respond to the aboutItem menu item selection event***.

ControlFrame.java

```
    this.setContentPane( mainPanel );
    fileMenu.setMnemonic( 'F' );
    JMenuItem aboutItem = new JMenuItem( "About..." );
    JMenu fileMenu = new JMenu( "File" );
    fileMenu.add( aboutItem );
    aboutItem.addActionListener(
    aboutItem.setMnemonic( 'A' );
      {
        public void actionPerformed( ActionEvent event )
      new ActionListener()  // Beginning of anonymous inner class
         JOptionPane.showMessageDialog( ControlFrame.this,
                              "This application provides enhanced\n
control over multimedia projects.",
        {
        }
     }  // End of anonymous inner class
                              "About", JOptionPane.PLAIN_MESSAGE );

    final JMenuBar bar = new JMenuBar();  // Create a JMenuBar
so we can attach menus to it.
    );
    bar.add( fileMenu );  // Add the file menu to the JMenuBar.

    setJMenuBar( bar );  // Attach the JMenuBar to the ControlFrame.
    colorMenu.setMnemonic( 'C' );
```

- If the user selects the ***aboutItem*** menu item, we use the ***JOptionPane*** class to launch a ***message dialog***.
- A ***JMenu*** must be added (***registered***) to a ***JMenuBar***, and the ***JMenuBar*** must be added to the ***JFrame***, otherwise none of it will be displayed to the user.
- ***JMenuBar*** has an add method, allowing you to add ***JMenu*** objects.
- ***JFrame*** has a ***setJMenuBar*** method, which, as you might guess, allows you to set the ***JMenuBar*** for the ***JFrame***.
- We then create an additional Color menu.

ControlFrame.java

```
colorMenu.setMnemonic( 'C' );
final JMenu colorMenu = new JMenu( "Color" );
JMenuItem redItem = new JMenuItem( "Red" );
colorMenu.add( redItem );

 new ActionListener()  // Beginning of anonymous inner class
 {
redItem.addActionListener(
   {
     drawPanel.setFillColor( Color.RED );
   public void actionPerformed( ActionEvent event )
   }
 }  // End of anonymous inner class
    repaint();

JMenuItem blueItem = new JMenuItem( "Blue" );
);
blueItem.addActionListener(
 new ActionListener()  // Beginning of anonymous inner class
colorMenu.add( blueItem );
   public void actionPerformed( ActionEvent event )
   {
 {
    repaint();
   }
     drawPanel.setFillColor( Color.BLUE );
);
```

- After creating the Color menu, we create a series of menu items that we then add to the Color menu.
- Each menu item merely sets the associated fill color for the ***DrawControlPanel*** object.
- Notice the repetition of the coding pattern.
- This code could be improved by using arrays to associate ***Strings*** to ***Color*** objects, allowing iteration across the arrays, executing the coding pattern within a loop instead of broken out and repeated as we see here.
- However, our current approach makes it easier to focus on the usage details regarding ***JMenuItems***.

ControlFrame.java

```
JMenuItem calcPanelItem = new JMenuItem( "Calculate" );
calcPanelItem.setMnemonic( 'C' );
fileMenu.add( calcPanelItem );
  new ActionListener()
calcPanelItem.addActionListener(
    public void actionPerformed( ActionEvent event )
    {
  {
      mainPanel.remove( drawPanel );
      mainPanel.remove( widthJSlider );
      bar.remove( colorMenu );
      yValTextField.setText("");
      calcJLabel.setText( "" );
      xValTextField.setText("");
      validate();
      repaint();
      mainPanel.add( calcPanel, BorderLayout.CENTER );
  }
);
    }
JMenuItem drawPanelItem = new JMenuItem( "DrawPanel" );
drawPanelItem.setMnemonic( 'D' );

drawPanelItem.addActionListener(
  new ActionListener()
fileMenu.add( drawPanelItem );
```

- Here we create a Calculate menu item to be placed in the File menu.
- Within the ***actionPerformed event-handling*** method, we respond to the menu item selection ***event*** by...
  - Removing the Color menu, if it is on the menu bar.
  - Removing ***drawPanel*** and ***widthJSlider***, if they had been previously added to ***mainPanel***.
  - Setting our ***JTextField*** instance variables to hold null ***Strings***.
  - Setting ***calcJLabel*** to display a null ***String***.
  - Add ***calcPanel*** to the center of ***mainPanel***.
  - Force the layout manager to perform a fresh layout again (via ***validate();***).
  - Force any graphics to be freshly rendered (via ***repaint();***).

ControlFrame.java

```
JMenuItem calcPanelItem = new JMenuItem( "Calculate" );
calcPanelItem.setMnemonic( 'C' );
fileMenu.add( calcPanelItem );
  new ActionListener()
calcPanelItem.addActionListener(
    public void actionPerformed( ActionEvent event )
    {
  {
      mainPanel.remove( drawPanel );
      mainPanel.remove( widthJSlider );
      bar.remove( colorMenu );
      yValTextField.setText("");
      calcJLabel.setText( "" );
      xValTextField.setText("");
      validate();
      repaint();
      mainPanel.add( calcPanel, BorderLayout.CENTER );
  }
);
    }
JMenuItem drawPanelItem = new JMenuItem( "DrawPanel" );
drawPanelItem.setMnemonic( 'D' );

drawPanelItem.addActionListener(
  new ActionListener()
fileMenu.add( drawPanelItem );
```

- Before we move on, this is where we can see the need for declaring variables like ***bar*** and ***colorMenu*** to be ***final***.
- Remember that ***final*** means that once the variable is initialized, ***it's value cannot change***.
- The ***ActionListener event-handling*** object does not get created until the first time that the Calculate menu item is selected.
- That means that the main method will have already completed and exited!
- ***The main method thread will have terminated, but the Swing event-dispatch thread will continue.***
- Any local variables that an event-handling object depends upon must be bound as constants by declaring them as final.

ControlFrame.java

```
JMenuItem drawPanelItem = new JMenuItem( "DrawPanel" );
drawPanelItem.setMnemonic( 'D' );
fileMenu.add( drawPanelItem );
drawPanelItem.addActionListener(
  {
  new ActionListener()
    {
      mainPanel.remove( calcPanel );
    public void actionPerformed( ActionEvent event )
      drawPanel.setBackground( Color.WHITE );
      mainPanel.add( drawPanel, BorderLayout.CENTER );
      bar.add( colorMenu );
      validate();
      repaint();
      mainPanel.add( widthJSlider, BorderLayout.SOUTH );
  }
);
    }
JMenuItem exitItem = new JMenuItem( "Exit" );
exitItem.setMnemonic( 'x' );

exitItem.addActionListener(
  new ActionListener()
fileMenu.add( exitItem );
    public void actionPerformed( ActionEvent event )
    {
  {
```

- Here we create a DrawPanel menu item to be placed in the File menu.
- Within the ***actionPerformed event-handling*** method, we respond to the menu item selection ***event*** by...
  - Removing ***calcPanel***, if it had been previously added to ***mainPanel***.
  - Adding the Color menu.
  - Setting the ***drawPanel*** background to white.
  - Adding ***drawPanel*** to the center of ***mainPanel***.
  - Adding ***widthJSlider*** to the south of ***mainPanel***.
  - Force the layout manager to perform a fresh layout again (via ***validate();***).
  - Force any graphics to be freshly rendered (via ***repaint();***).

ControlFrame.java

```
   JMenuItem exitItem = new JMenuItem( "Exit" );
   exitItem.setMnemonic( 'x' );
   fileMenu.add( exitItem );
     new ActionListener()
   exitItem.addActionListener(
      public void actionPerformed( ActionEvent event )
      {
    {
      }
    }
       System.exit( 0 );

   widthJSlider = new JSlider( SwingConstants.HORIZONTAL,
0, 100, drawPanel.getOvalWidth() );
   );
   widthJSlider.setPaintTicks( true );

   widthJSlider.setMajorTickSpacing( 10 );
    new ChangeListener()
    {
   widthJSlider.addChangeListener(
      {
        drawPanel.setOvalWidth( widthJSlider.getValue() );
      public void stateChanged( ChangeEvent e )
      }
    }
```

- Here we create an Exit menu item to be placed in the File menu.
- Within the ***actionPerformed event-handling*** method, we respond to the menu item selection ***event*** by exiting the application.
- We create a horizontal ***JSlider***, initializing it to the current value of ***drawPanel's*** oval width.
- We set the spacing of the ***JSlider's*** major ticks to intervals of 10.
- We ensure that the tick marks are painted.
- We ***register*** an ***event-handler*** to the ***JSlider***, ensuring that any change to slider bar causes a corresponding update to the width of the oval, and that the oval is freshly rendered again.

ControlFrame.java

```
xValTextField = new JTextField( 3 );
xValTextField.addActionListener(
  new ActionListener()
    public void actionPerformed( ActionEvent event )
  {
      xStr = event.getActionCommand();
    }
    {
);

  }

yValTextField = new JTextField( 3 );
calcPanel.add( xValTextField );
  new ActionListener()
  {
yValTextField.addActionListener(
    {
      yStr = event.getActionCommand();
    public void actionPerformed( ActionEvent event )
  }
);
    }
calcPanel.add( yValTextField );
```

- The ***xValTextField JTextField*** object is created, allowing for a width of 3 spaces.
- An ***event-handler*** is created, ensuring that we respond to the user's ***<ENTER> key*** by getting the text ***String*** entered and assigning it to ***xStr***.
- We add ***xValTextField*** to ***calcPanel***.
- The above actions are repeated, but this time for ***yValTextField*** and ***yStr***.

ControlFrame.java

```
   calcJButton = new JButton( "Calculate" );
   calcJButton.addActionListener(
    new ActionListener()
    {
     public void actionPerformed( ActionEvent event )
      try {
        int x = Integer.parseInt( xStr );
     {
        int result = x + y;
        calcJLabel.setText(xStr + " + " + yStr + " = " + result);
        int y = Integer.parseInt( yStr );
       catch (NumberFormatException e) {
        JOptionPane.showMessageDialog( ControlFrame.this,
"You must enter a valid number and then <ENTER> for each textbox!",
"Invalid Input", JOptionPane.ERROR_MESSAGE );
       }
       }
     }
        e.printStackTrace();
   );
   calcPanel.add( calcJButton );
    }
   calcJLabel = new JLabel();
   calcPanel.add( calcJLabel, BorderLayout.CENTER );

   setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
```

- We instantiate a ***JButton***, with a ***String*** of “Calculate”.
- We ***register*** an ***event-handler***, ensuring that when the user clicks the ***calcJButton*** button...
  - We convert ***xStr*** to an ***int***, and assign it to ***x***.
  - We convert ***yStr*** to an ***int***, and assign it to ***y***.
  - We calculate the sum of ***x*** and ***y***.
  - We set ***calcJLabel*** to display a summation formula.
- Our ***try...catch exception handling*** construct ensures that, if the user does not correctly enter ***x*** and ***y*** values, we launch a ***MessageDialog***.
- We add ***calcJButton*** to ***calcPanel***.

ControlFrame.java

```
        }
        catch (NumberFormatException e) {
          JOptionPane.showMessageDialog( ControlFrame.this,
"You must enter a valid number and then <ENTER> for each textbox!", "Invalid
Input", JOptionPane.ERROR_MESSAGE );
          e.printStackTrace();
        }
      }
    );
     }

    calcPanel.add( calcJButton );
    calcJLabel = new JLabel();
    calcPanel.add( calcJLabel, BorderLayout.CENTER );

    setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

    setVisible( true );
    setSize( 200, 250 );
    validate();
  }
}
```

- We complete the definition of the ControlFrame class by...
  - Creating the calcJLabel JLabel and adding it to the center of calcPanel.
  - Setting the close operation to exit (this ensures that the application terminates when the user clicks on the “x” in the upper-right corner of the Jframe).
  - Setting the pixel size of the JFrame.
  - Making the JFrame visible.
  - Forcing the layout manager to do a fresh layout.

Futher Java Swing Resources

- The Java Tutorials: Creating a GUI with JFC/Swing (The Swing Tutorial)

  http://java.sun.com/docs/books/tutorial/uiswing/

- The Swing Tutorial

  http://www.javabeginner.com/java-swing/java-swing-tutorial

- SwingWiki: Java Swing Developer Wiki

  http://www.swingwiki.org/

- The SwingSet2 Demo

  This is included within the JDK. Once you identify your JDK installation directory, you will find the demo at:

  <JDK Install Directory>/demo/jfc/SwingSet2

  To extract the .java files from the jar, type the following at the command line:

  jar xf SwingSet2.jar