Swing User Interfaces

Contents

1. Introduction to Swing
2. A worked example

Demo project: DemoSwingUI

2

1. Introduction to Swing

- What is Swing?
- Swing features
- Design patterns used in Swing

What is Swing?

- Swing is a standard (and recommended) Java library for creating user interfaces
  - A vast improvement on the Abstract Windowing Toolkit (AWT), the original GUI library available in JDK 1.0
  - Swing provides more useful controls, and better performance
  - Swing does not use any native code (AWT does...)

Swing Features

- Many components and containers, located in the `javax.swing` package (and other related packages)
  - For example, `javax.swing.JButton`, `javax.swing.JFrame`
- Pluggable look-and-feel
  - Windows look-and-feel
  - Motif look-and-feel
  - Metal look-and-feel
- Plus additional features…
  - High-quality Java 2D graphics and images
  - Drag-and-drop
  - Accessibility API

5

Design Patterns Used in Swing

- Swing uses various design patterns
- Model-View-Controller (MVC)
  - Separate the data (model), from its onscreen appearance (view), from the code that links the two together (controller)
  - For example, `JTable` uses MVC
- Observer-Observable
  - Event-handling
  - For example, when you click a `JButton` ('observable') an `ActionEvent` is raised; listeners ('observers') implement the `ActionListener` interface

6

2. A Worked Example

- Getting started
- Choosing the look and feel
- Setting up a top-level container
- Setting up components
- Adding components to the frame
- Handling events
- Displaying dialog boxes

Getting Started

- The core Swing classes are located in the `javax.swing` package

```
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JButton;
import javax.swing.UIManager;
… etc…
```

- We'll create a standalone Swing application
  - See `SimpleSwingDemo.java`

```
public class SimpleSwingDemo {

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

  public SimpleSwingDemo() {
    // For this example, we'll put all the interesting code here in the constructor
    …
  }
}
```

8

Choosing the Look and Feel

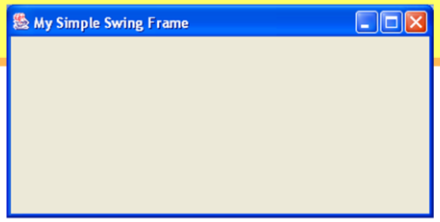
- Swing lets you choose a look and feel for your program

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
…

System.out.print("Choose look-and-feel [windows, motif, metal, default] ");
String in = br.readLine();

if (in.equals("windows")) {

    UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");

} else if (in.equals("motif")) {

    UIManager.setLookAndFeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel");

} else if (in.equals("motif")) {

    UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");

} else {

    // Set the cross-platform look and feel
    UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
}
```

Setting Up a Top-Level Container

- Swing applications have at least one top-level Swing container
  - `JFrame`, `JDialog`, or `JApplet`
- The sample application has a single `JFrame`

```
// Create the Jframe.
JFrame frame = new JFrame("My Simple Swing Frame");

// Set the size of the frame (width, height).
frame.setSize(400,200);

// Ensure the window is closed properly on exit.
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

// Make the frame visible.
frame.setVisible(true);
```



10

Setting Up Components

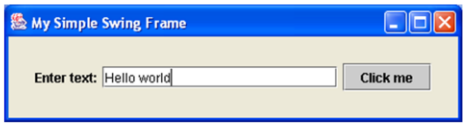
- Swing defines a gamut of component classes
  - `JLabel`, `JTextField`, `JButton`, `JTextArea`, etc.
- Components are typically contained in a `JPanel`

```
JLabel     label     = new JLabel("Enter text:");
JTextField textField = new JTextField(20);
JButton    button    = new JButton("Click me");

// Create the JPanel.
JPanel pane = new JPanel();

// Set an internal border (top, bottom, left, right) for the JPanel.
pane.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));

// Add components to the JPanel.
pane.add(label);
pane.add(textField);
pane.add(button);
```

Adding Components to the Frame

- Add the `JPanel` to the `JFrame`'s content pane
  - Call `pack()` instead of `setSize()` on the `JFrame`

```
// Code, as before…

// Add the JPanel to the frame, and then display the JFrame.
frame.getContentPane().add(pane);
frame.setVisible();
```



12

Handling Events

- Use the familiar approach to event-handling
  - Implement the appropriate `XxxxListener` interface
  - Implement the methods declared in the `XxxxListener` interface
  - Call `addXxxxListener()` on the source object
- For example, to handle button click events:

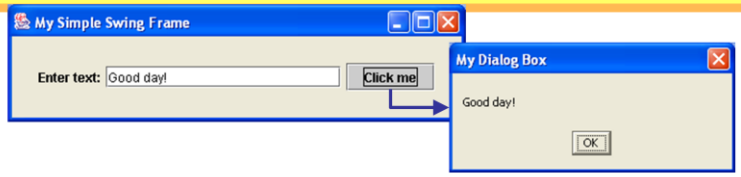
```
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
…

public class SimpleSwingDemo implements ActionListener {

    public SimpleSwingDemo() {
        …
        button.addActionListener(this);
    }

    public void actionPerformed(ActionEvent evt) {
        // Event handler code…
    }
}
```

13

Displaying Dialog Boxes

- Swing provides several ways to display dialog boxes
  - Use `JOptionPane` to create simple, standard dialogs
  - Use `JFileChooser` to display a file-chooser dialog
  - Use `JColorChooser` to display a colour-chooser dialog
  - Use `ProgressMonitor` to display a progress-indicator dialog
  - Use `JDialog` to display custom dialog boxes
- For example, to display a simple dialog box:

```
public void actionPerformed(ActionEvent evt) {
    JOptionPane.showMessageDialog(frame,
                                  textField.getText(),
                                  "My Dialog Box",
                                  JOptionPane.PLAIN_MESSAGE);
}
```



14

Any Questions?

15