Window Interfaces Using Swing

Chapter 12
Reminders

• Project 7 due Nov 17 @ 10:30 pm
• Project 6 grades released: regrades due by next Friday (11-18-2005) at midnight
• Windowing systems that interact with users often are called GUIs.
• A GUI accepts information from a user and makes it available to the program for processing.
• Most of the interaction is graphical in nature.
Event-Driven Programming

• Most GUI programs involve events and event handlers.

• A GUI event is an object that represents some action such as clicking the mouse, dragging the mouse, pressing a keyboard key, clicking the close-window button on a window, etc.

• When an object generates an event, it is said to fire the event.
Event-Driven Programming, cont.

- Objects that can fire events have one or more *listener* objects.
- The programmer chooses which event-firing objects have listeners.
- *Event handlers* are programmer-defined methods associated with the listener objects that determine what happens when events are detected by their associated listener(s).
Layout Managers

• The objects that you add to a container class are arranged by an object known as a layout manager.

• A layout manager is added using method `setLayout`, which is a method of every container class.

• syntax

```java
Container_Object.setLayout(new Layout_Manager_Class(Any_Parameters));
```
Border Layout, cont.

- A BorderLayout manager can place a component into any of five regions.
- Regions which are unused give up their space to `BorderLayout.CENTER`.

```
+----------------+    +----------------+    +----------------+
| BorderLayout.NORTH | = | BorderLayout.CENTER | + | BorderLayout.EAST |
|-----------------+    +----------------+    +----------------+
| BorderLayout.WEST |    | BorderLayout.CENTER |    | BorderLayout.EAST |
|-------------------+    +-----------------+    +------------------+
| BorderLayout.SOUTH |    | BorderLayout.CENTER |    | BorderLayout.EAST |
```

Flow Layout

• The simplest layout manager is the `FlowLayout` manager.

• Components are added and arranged one after another, left to right, until a row is filled. Then components are added to the next row in the same manner.

• Each row is centered in its container.
Grid Layout

• A GridLayout manager arranges components in a grid of rows and columns.

• example

```java
aContainer.setLayout(new GridLayout(2,3));
```
Grid Layout, cont.

- Each entry has the same size.
- Rows are filled one at a time, top to bottom, and from left to right within each row.
Default Layout Managers

• When a layout manager is not added explicitly, a default layout manager is provided.

• The default manager for the content pane of a JFrame is BorderLayout.

• The default manager for a JPanel is FlowLayout.
Buttons

• A *button* is a GUI component that looks like a button and does something when it is clicked using a mouse.
• Like a label, a button is created and added to a container.
• Unlike a label, a button can fire an event and the event can cause a GUI to perform some action.
Adding Buttons

• A button is created using

```
JButton Button_Name = new JButton("Button_Label");
```

• A button is added to a container using

```
Container_Name.add(Button_Name);
```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

/**
 * Simple demonstration of putting buttons in a JFrame.
 */
public class ButtonDemo extends JFrame implements ActionListener
{
    public static final int WIDTH = 300;
    public static final int HEIGHT = 200;

    /**
     * Creates and displays a window of the class ButtonDemo.
     */
    public static void main(String[] args)
    {
        ButtonDemo buttonGui = new ButtonDemo();
        buttonGui.setVisible(true);
    }
    public ButtonDemo()
    {
        setSize(WIDTH, HEIGHT);
        addWindowListener(new WindowDestroyer());
        setTitle("Button Demo");
        Container contentPane = getContentPane();
        contentPane.setBackground(Color.BLUE);

        contentPane.setLayout(new FlowLayout());
        JButton stopButton = new JButton("Red");
        stopButton.addActionListener(this);
        contentPane.add(stopButton);

        JButton goButton = new JButton("Green");
        goButton.addActionListener(this);
        contentPane.add(goButton);

        public void actionPerformed(ActionEvent e)
        {
            Container contentPane = getContentPane();

            if (e.getActionCommand().equals("Red"))
                contentPane.setBackground(Color.RED);
            else if (e.getActionCommand().equals("Green"))
                contentPane.setBackground(Color.GREEN);
            else
                System.out.println("Error in button interface.");
        }
    }
}

Display 12.14
A GUI with Buttons Added
Adding Buttons, cont.

Resulting GUI

Display 12.14
A GUI with Buttons Added
Action Listeners and Action Events

• For each button, the GUI needs to
  – register (specify) the listener object(s).
  – define the methods to be invoked when an event is fired.

• For a statement such as
  ```java
  stopButton.addActionListener(this);
  ```
  the class `ButtonDemo` is itself the listener class.
Action Listeners and Action Events, cont.

- Different kinds of components require different kinds of listener classes.
- Buttons fire *action events* which are handled by *action listeners*.
- An action listener is an object of type `ActionListener`, and `ActionListener` is an *interface* (discussed later).
Action Listeners and Action Events, cont.

- To make a class into an `ActionListener`
  - add `implements ActionListener` to the heading of the class definition
  - define a method named `ActionPerformed`.
  - register the `ActionListener` object with the component that will fire the event using the method `addActionListener`
  - (A component may register with more than one listener.)
Action Listeners and Action Events, cont.

- Any class can be an action listener class.
Buttons and an Action Listener

- `stopButton`
- `goButton`

These buttons fire action events, named `e` in our code.

- `buttonGui`

This is the action listener for both buttons.

When action event `e` is fired, the effect is `buttonGui.actionPerformed(e);`

Display 12.15
Buttons and an Action Listener
The **actionPerformed** Method

- An `actionListener` class must have a method named `actionPerformed` that has one parameter of type `ActionEvent`.

- **syntax**

  ```java
  public void actionPerformed(ActionEvent e)
  {
      Code_for_Actions_Performed
  }
  ```
Interfaces

• An *interface* is a property of a class that states what methods the class must define.

• *ActionListener* is an interface.

• A class which satisfies the requirements of an interface *implements the interface*.

• A class can define methods in addition to the methods required by the interface.
Interfaces, cont.

• To implement an interface, a class must
  – include the phrase `implements Interface_Name`
    at the start of the class definition
  – implement all the method headings listed in
    the definition of the interface.
Multiple Interfaces

• A class which implements multiple interfaces lists the names of all the interfaces, separated by commas.

implments First_Interface_Name,
    Second_Interface_Name, ..., 
    Last_Interface_Name

• The class must implement all the methods of all the listed interfaces.
Method `setActionCommand`

- We can think of the method invocation `e.getActionCommand()` as returning the string written on the button.
- In fact, this method invocation returns a string known as the *action command* for the button.
- A different action command can be specified for the button.
Method `setActionCommand`, cont.

- example

  ```java
  JButton stopButton = new JButton("Red");
  stopButton.setActionCommand("Stop");
  ```

- This permits the same string to be written on two different buttons, but with the two buttons distinguished from one another by the program.
Method `setActionCommand`, cont.

- Every object that fires an action event has an associated string known as the *action command* for that component.
- `e.getActionCommand()` returns the action command for the component that fired `e`.
- The default action command for a button is the string written on it.
- Method `setActionCommand` can be used to change the action command for the object.
The JPanel Class

• A GUI can be organized hierarchically, with window-like containers inside other window-like containers.

• Class JPanel is a simple container that does little more than hold components.

• Components can be placed in a JPanel which can be placed in another JPanel, … which can be placed in a JFrame.
The JPanel Class, cont.

- To place two components in `BorderLayout.SOUTH` for example, simply place the two components in a panel and place the panel in the `BorderLayout.SOUTH` position.
- The panel has its own layout manager.
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

/**
 * Simple demonstration of putting buttons in a panel.
 */
public class PanelDemo extends JFrame implements ActionListener {
    public static final int WIDTH = 300;
    public static final int HEIGHT = 200;

    public static void main(String[] args) {
        PanelDemo guiWithPanel = new PanelDemo();
        guiWithPanel.setVisible(true);
    }

    public PanelDemo() {
        setSize(WIDTH, HEIGHT);
        addWindowListener(new WindowDestroyer());
        setTitle("Panel Demonstration");
        Container contentPane = getContentPane();
        contentPane.setBackground(Color.BLUE);
        contentPane.setLayout(new BorderLayout());

        JPanel buttonPanel = new JPanel();
        buttonPanel.setBackground(Color.WHITE);
        buttonPanel.setLayout(new FlowLayout());

        JButton stopButton = new JButton("Red");
        stopButton.setBackground(Color.RED);
        stopButton.addActionListener(this);
        buttonPanel.add(stopButton);

        JButton goButton = new JButton("Green");
        goButton.setBackground(Color.GREEN);
        goButton.addActionListener(this);
        buttonPanel.add(goButton);

        contentPane.add(buttonPanel, BorderLayout.SOUTH);
    }

    public void actionPerformed(ActionEvent e) {
        Container contentPane = getContentPane();

        if (e.getActionCommand().equals("Red"))
            contentPane.setBackground(Color.RED);
        else if (e.getActionCommand().equals("Green"))
            contentPane.setBackground(Color.GREEN);
        else
            System.out.println("Error in button interface.");
    }
}

Display 12.17
Putting the Buttons in a Panel
The JPanel Class, cont.

Resulting GUI

Display 12.17
Putting the Buttons in a Panel
The Container Class

- **Class Container** is a predefined class.
- An object of a class which descends from class **Container** is called a *container class* and can have components added to it.
- **Class JFrame** is a descendent of class **Container**, permitting any **JFrame** object to hold labels, buttons, panels, and other components.
Adding Components

• To add a component to a JFrame, use method `getContentPane` to obtain the content pane, and the use method `add` with the content pane as the calling object.

• example

```java
Container contentPane = getContentPane();
Jlabel label = new Jlabel(“Click Here”);
contentPane.add(label);
```
Adding Components, cont.

• For other container classes, add components by using method `add` directly with an object of the container class.

• example

```java
JPanel buttonPanel = new JPanel();
JButton stopButton =
    new JButton(“Stop”);
buttonPanel.add(stopButton);
```
Text Areas and Text Fields

• class MemoSaver

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

public class MemoSaver extends JFrame implements ActionListener {
    public static final int WIDTH = 600;
    public static final int HEIGHT = 300;
    public static final int LINES = 10;
    public static final int CHAR_PER_LINE = 40;

    private JTextArea theText;
    private String memo1 = "No Memo 1."
    private String memo2 = "No Memo 2."

    public MemoSaver() {
        setSize(WIDTH, HEIGHT);
        addWindowListener(new WindowDestroyer());
        setTitle("Memo Saver");
        Container contentPane = getContentPane();
        contentPane.setLayout(new BorderLayout());
    }

    JPanel buttonPanel = new JPanel();
    buttonPanel.setBackground(Color.WHITE);
    buttonPanel.setLayout(new FlowLayout());
    JButton memo1Button = new JButton("Save Memo 1");
    memo1Button.addActionListener(this);
    buttonPanel.add(memo1Button);
    JButton memo2Button = new JButton("Save Memo 2");
    memo2Button.addActionListener(this);
    buttonPanel.add(memo2Button);
    JButton clearButton = new JButton("Clear");
    clearButton.addActionListener(this);
    buttonPanel.add(clearButton);
    JButton get1Button = new JButton("Get Memo 1");
    get1Button.addActionListener(this);
    buttonPanel.add(get1Button);
    JButton get2Button = new JButton("Get Memo 2");
    get2Button.addActionListener(this);
    buttonPanel.add(get2Button);
    contentPane.add(buttonPanel, BorderLayout.SOUTH);

    JPanel textPanel = new JPanel();
    textPanel.setBackground(Color.BLUE);
    contentPane.add(textPanel, BorderLayout.CENTER);

    // Implementation of ActionListener

    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == memo1Button) {
            // Action for memo1Button
        } else if (e.getSource() == memo2Button) {
            // Action for memo2Button
        } else if (e.getSource() == clearButton) {
            // Action for clearButton
        } else if (e.getSource() == get1Button) {
            // Action for get1Button
        } else if (e.getSource() == get2Button) {
            // Action for get2Button
        }
    }
}
```

Display 12.19
A GUI with a Text Area
Text Areas and Text Fields,

• class MemoSaver, cont.

```java
theText = new JTextArea(LINES, CHAR_PER_LINE);
theText.setBackground(Color.WHITE);
textPanel.add(theText);
contentPane.add(textPanel, BorderLayout.CENTER);
}
```

```java
public void actionPerformed(ActionEvent e)
{
  String actionCommand = e.getActionCommand();
  if (actionCommand.equals("Save Memo 1"))
    memo1 = theText.getText();
  else if (actionCommand.equals("Save Memo 2"))
    memo2 = theText.getText();
  else if (actionCommand.equals("Clear"))
    theText.setText("");

  else if (actionCommand.equals("Get Memo 1"))
    theText.setText(memo1);
  else if (actionCommand.equals("Get Memo 2"))
    theText.setText(memo2);
  else
    theText.setText("Error in memo interface");
}
```

```java
public static void main(String[] args)
{
  MemoSaver guiMemo = new MemoSaver();
  guiMemo.setVisible(true);
}
```

Display 12.19

A GUI with a Text Area
Text Areas and Text Fields, cont.

Resulting GUI

Display 12.19
A GUI with a Text Area
Text Areas and Text Fields, cont.

• An object of class `JTextArea` can have a size consisting of a specified number of lines and a specified number of characters per line.

• example

  ```java
  JTextArea someText = new JTextArea(10, 30);
  ```
Text Areas and Text Fields, cont.

• An object of class `JTextField` can have a size consisting of a specified number of characters.

• example

  ```java
  JTextField name = new JTextField(10);
  ```
Text Areas and Text Fields, cont.

- The number of characters (per line) is not absolute, but represents the space needed for one \( m \) character.
Line Wrapping in Text Areas

• Method `setLineWrap` sets the line wrapping policy for a `JTextArea` object.

• example

  ```java
  theText.setLineWrap(true);
  ```

• If the argument is set to `false`, extra characters will be on the same line, but will not be visible.
Read-Only Text Components

• To specify that a user cannot write in a JTextArea or a JTextField, use method setEditable.

  example

  theText.setEditable(false);

• A JTextArea or a JTextField can be made editable subsequently using, for example

  theText.setEditable(true);
Inputting and Outputting Numbers

• Input provided using a JTextArea object or JTextField object is received as a string.
• When numeric input is needed, the string must be converted to a number.
• To output a number using a GUI constructed with Swing, the number must be converted to a string.
• All input typed by the user is string output, and all displayed output is string output.
• To convert a string to an integer, use, for example

        Integer.parseInt("42");
    or
        Integer.parseInt(ioField.getText());

or, to eliminate whitespace before or after the input, use

        Integer.parseInt
            (ioField.getText().trim());
Inputting and Outputting Numbers, cont.

- To input numbers of type double, use
  ```java
  Double.parseDouble((ioField.getText().trim()));
  ```
- Analogous conversions can be done with classes `Long` and `Float`. 
Inputting and Outputting Numbers, cont.

• To write numeric output to a `JTextArea` or a `JTextField`, use method `toString`.

• examples

```java
Integer.toString(sum);
Double.toString(average);
ioField.setText(Integer.toString(sum));
```
Catching a `NumberFormatException`

- A GUI, such as class `Adder`, has no control over what the user enters in the text field. The user might enter commas, or even alphabetic characters, resulting in a `NumberFormatException`, which leaves the GUI in an unpredictable state.
Catching a `NumberFormatException`,

- A `NumberFormatException` can be caught, and the user can be asked to reenter the number.
Summary

• You have learned the basics of event-driven programming.
• You have designed and coded a simple GUI with buttons and text.
• You have learned about several Swing-related classes.