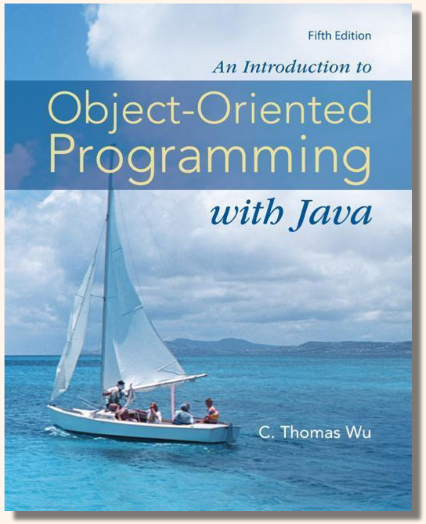


Chapter 14

GUI and Event-Driven Programming

Objectives

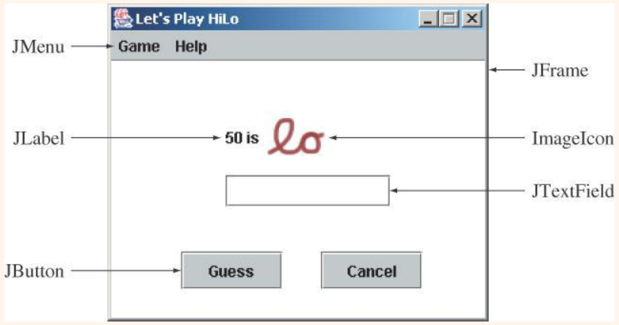
- After you have read and studied this chapter, you should be able to
  - Define a subclass of JFrame to implement a customized frame window.
  - Write event-driven programs using Java's delegation-based event model
  - Arrange GUI objects on a window using layout managers and nested panels
  - Write GUI application programs using JButton, JLabel, ImageIcon, JTextField, JTextArea, JCheckBox, JRadioButton, JComboBox, JList, and JSlider objects from the javax.swing package
  - Write GUI application programs with menus
  - Write GUI application programs that process mouse events

Graphical User Interface

- In Java, GUI-based programs are implemented by using classes from the **javax.swing** and **java.awt** packages.
- The Swing classes provide greater compatibility across different operating systems. They are fully implemented in Java, and behave the same on different operating systems.

Sample GUI Objects

- Various GUI objects from the **javax.swing** package.

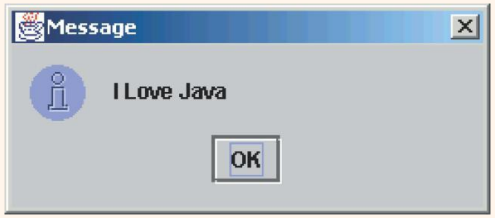


JOptionPane

- Using the JOptionPane class is a simple way to display the result of a computation to the user or receive an input from the user.
- We use the showMessageDialog class method for output.
- We use the showInputDialog class method for input. This method returns the input as a String value so we need to perform type conversion for input of other data types

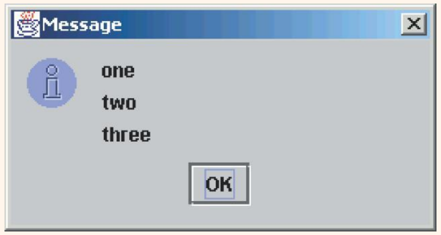
Using JOptionPane for Output

```
import javax.swing.*;
. . .

JOptionPane.showMessageDialog( null, "I Love Java" );
```



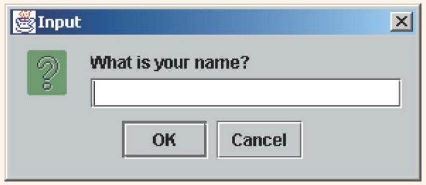
Using JOptionPane for Output - 2

```
import javax.swing.*;
. . .

JOptionPane.showMessageDialog( null, "one\ntwo\nthree" );
//place newline \n to display multiple lines of output
```



JOptionPane for Input

```
import javax.swing.*;
. . .

String inputstr =

JOptionPane.showInputDialog( null, "What is your name?" );
```

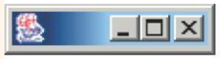


Subclassing **JFrame**

- To create a customized frame window, we define a subclass of the **JFrame** class.
- The **JFrame** class contains rudimentary functionalities to support features found in any frame window.

Creating a Plain **JFrame**

```
import javax.swing.*;
class Ch7DefaultJFrame {
    public static void main( String[] args ) {
        JFrame defaultJFrame;
        defaultJFrame = new JFrame();
        defaultJFrame.setVisible(true);
    }
}
```



You may not notice this frame window on the screen at first because it is so small. Look carefully at the top left corner of the screen.

Creating a Subclass of **JFrame**

- To define a subclass of another class, we declare the subclass with the reserved word **extends**.

```
import javax.swing.*;

class Ch7JFrameSubclass1 extends JFrame {
    . . .
}
```

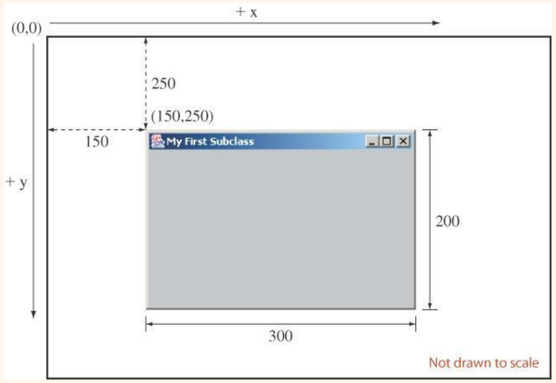
Customizing Ch14JFrameSubclass1

- An instance of Ch14JFrameSubclass1 will have the following default characteristics:
  - The title is set to **My First Subclass**.
  - The program terminates when the close box is clicked.
  - The size of the frame is 300 pixels wide by 200 pixels high.
  - The frame is positioned at screen coordinate (150, 250).
- These properties are set inside the default constructor.

Source File: `Ch14JFrameSubclass1.java`

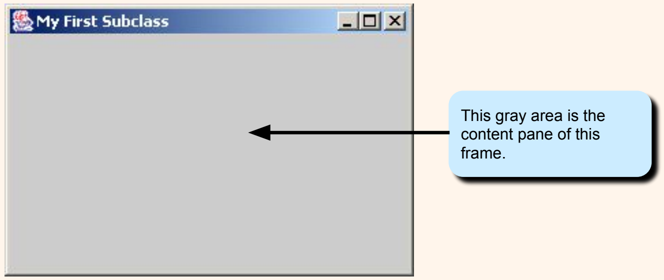
Displaying Ch14JFrameSubclass1

- Here's how a **Ch14JFrameSubclass1** frame window will appear on the screen.



The Content Pane of a Frame

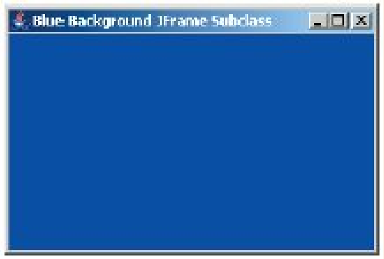
- The content pane is where we put GUI objects such as buttons, labels, scroll bars, and others.
- We access the content pane by calling the frame's getContentPane method.



Changing the Background Color

- Here's how we can change the background color of a content pane to blue:

```
Container contentPane = getContentPane();
contentPane.setBackground(Color.BLUE);
```



Source File: Ch14JFrameSubclass2.java

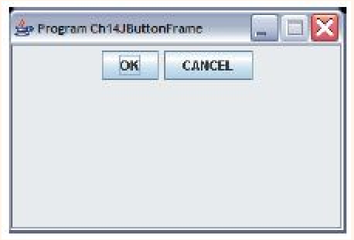
Placing GUI Objects on a Frame

- There are two ways to put GUI objects on the content pane of a frame:
  - Use a *layout manager*
    - FlowLayout
    - BorderLayout
    - GridLayout
  - Use *absolute positioning*
    - null layout manager

Placing a Button

- A JButton object a GUI component that represents a pushbutton.
- Here's an example of how we place a button with FlowLayout.

```
contentPane.setLayout(
            new FlowLayout());
okButton
      = new JButton("OK");
cancelButton
      = new JButton("CANCEL");
contentPane.add(okButton);
contentPane.add(cancelButton);
```



Event Handling

- An action involving a GUI object, such as clicking a button, is called an *event.*
- The mechanism to process events is called *event handling*.
- The event-handling model of Java is based on the concept known as the *delegation-based event model.*
- With this model, event handling is implemented by two types of objects:
  - event source objects
  - event listener objects

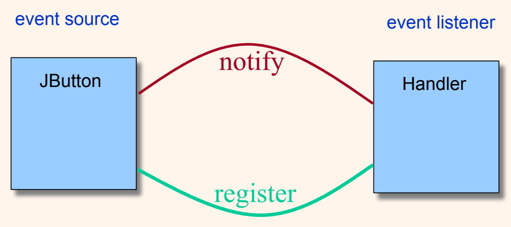
Event Source Objects

- An event source is a GUI object where an event occurs. We say an event source generates events.
- Buttons, text boxes, list boxes, and menus are common event sources in GUI-based applications.
- Although possible, we do not, under normal circumstances, define our own event sources when writing GUI-based applications.

Event Listener Objects

- An event listener object is an object that includes a method that gets executed in response to the generated events.
- A listener must be associated, or registered, to a source, so it can be notified when the source generates events.

Connecting Source and Listener

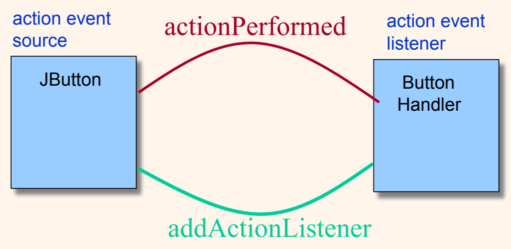


A listener must be registered to a event source. Once registered, it will get notified when the event source generates events.

Event Types

- Registration and notification are specific to event types
  - Mouse listener handles mouse events
  - Item listener handles item selection events
  - and so forth
- Among the different types of events, the action event is the most common.
  - Clicking on a button generates an action event
  - Selecting a menu item generates an action event
  - and so forth
- Action events are generated by action event sources and handled by action event listeners.

Handling Action Events



```
JButton button = new JButton("OK");
ButtonHandler handler = new ButtonHandler( );

button.addActionListener(handler);
```

The Java Interface

- A Java interface includes only constants and abstract methods.
- An abstract method has only the method header, or prototype. There is no method body. You cannot create an instance of a Java interface.
- A Java interface specifies a behavior.
- A class implements an interface by providing the method body to the abstract methods stated in the interface.
- Any class can implement the interface.

ActionListener Interface

- When we call the addActionListener method of an event source, we must pass an instance of a class that implements the **ActionListener interface**.
- The ActionListener interface includes one method named actionPerformed.
- A class that implements the ActionListener interface must therefore **provide the method body** of actionPerformed.
- Since actionPerformed is the method that will be called when an action event is generated, this is the place where we put a code we want to be executed in response to the generated events.

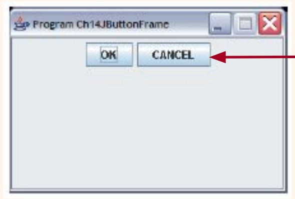
The ButtonHandler Class

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

class ButtonHandler implements ActionListener {
    . . .
    public void actionPerformed(ActionEvent event) {
        JButton clickedButton = (JButton) event.getSource();

        JRootPane rootPane = clickedButton.getRootPane( );
        Frame     frame    = (JFrame) rootPane.getParent();

        frame.setTitle("You clicked " + clickedButton.getText());
    }
}
```

Container as Event Listener

- Instead of defining a separate event listener such as ButtonHandler, it is much more common to have an object that contains the event sources be a listener.
  - Example: We make this frame a listener of the action events of the buttons it contains.

event listener



event source

Ch14JButtonFrameHandler

```
. . .
class Ch14JButtonFrameHandler extends JFrame
                         implements ActionListener {
   . . .
   public void actionPerformed(ActionEvent event) {
        JButton clickedButton
             = (JButton) event.getSource();

        String  buttonText = clickedButton.getText();

        setTitle("You clicked " + buttonText);
   }
}
```

GUI Classes for Handling Text

- The Swing GUI classes **JLabel**, **JTextField**, and **JTextArea** deal with text.
- A **JLabel** object displays uneditable text (or image).
- A **JTextField** object allows the user to enter a single line of text.
- A **JTextArea** object allows the user to enter multiple lines of text. It can also be used for displaying multiple lines of uneditable text.

JTextField

- We use a JTextField object to accept a single line to text from a user. An action event is generated when the user presses the ENTER key.
- The getText method of JTextField is used to retrieve the text that the user entered.

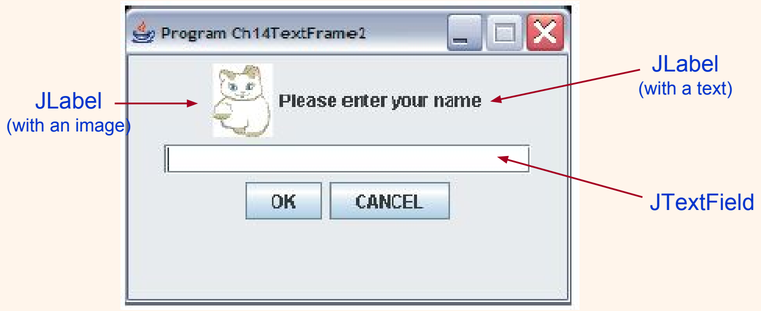
```
JTextField input = new JTextField( );
input.addActionListener(eventListener);
contentPane.add(input);
```

JLabel

- We use a JLabel object to display a label.
- A label can be a text or an image.
- When creating an image label, we pass ImageIcon object instead of a string.

```
JLabel textLabel = new JLabel("Please enter your name");
contentPane.add(textLabel);

JLabel imgLabel = new JLabel(new ImageIcon("cat.gif"));
contentPane.add(imgLabel);
```

Ch14TextFrame2



JTextArea

- We use a JTextArea object to display or allow the user to enter multiple lines of text.
- The setText method assigns the text to a JTextArea, replacing the current content.
- The append method appends the text to the current text.

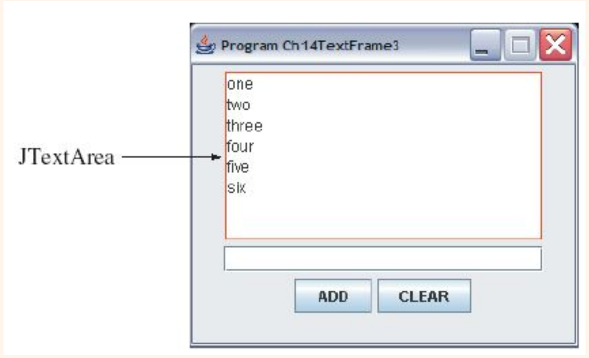
```
JTextArea textArea
      = new JTextArea( );
. . .
textArea.setText("Hello\n");
textArea.append("the lost ");
textArea.append("world");
```

```
Hello
the lost world
```

JTextArea

Ch14TextFrame3

- The state of a **Ch14TextFrame3** window after six words are entered.



```
one
two
three
four
five
six
```

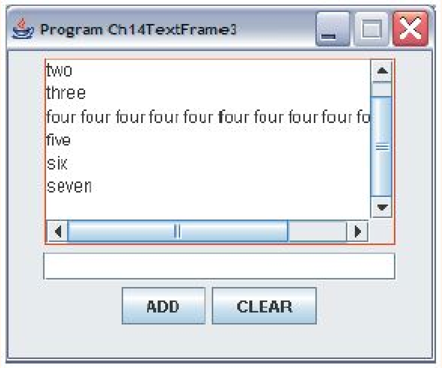
Adding Scroll Bars to JTextArea

- By default a JTextArea does not have any scroll bars. To add scroll bars, we place a JTextArea in a JScrollPane object.

```
JTextArea   textArea    = new JTextArea();
. . .
JScrollPane scrollText = new JScrollPane(textArea);
. . .
contentPane.add(scrollText);
```

Ch14TextFrame3 with Scroll Bars

- A sample Ch14TextFrame3 window when a JScrollPane is used.



```
two
three
four four four four four four four four four fo
five
six
seven
```

Layout Managers

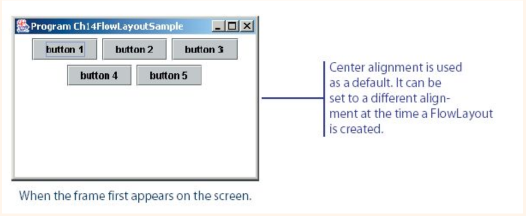
- The layout manager determines how the GUI components are added to the container (such as the content pane of a frame)
- Among the many different layout managers, the common ones are
  - FlowLayout (see Ch14FlowLayoutSample.java)
  - BorderLayout (see Ch14BorderLayoutSample.java)
  - GridLayout (see Ch14GridLayoutSample.java)

FlowLayout

- In using this layout, GUI componentsare placed in left-to-right order.
  - When the component does not fit on the same line, left-to-right placement continues on the next line.
- As a default, components on each line are centered.
- When the frame containing the component is resized, the placement of components is adjusted accordingly.

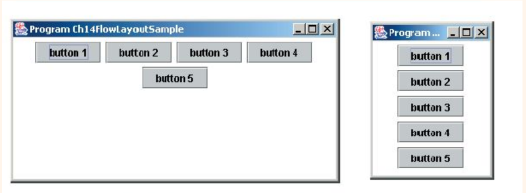
FlowLayout Sample

This shows the placement of five buttons by using FlowLayout.



Center alignment is used as a default. It can be set to a different alignment at the time a FlowLayout is created.

When the frame first appears on the screen.



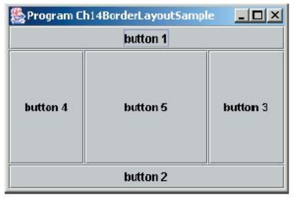
After the frame's width is widened and shortened.

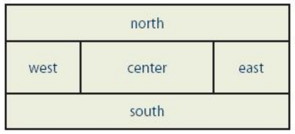
BorderLayout

- This layout manager divides the container into five regions: center, north, south, east, and west.
- The north and south regions expand or shrink in height only
- The east and west regions expand or shrink in width only
- The center region expands or shrinks on both height and width.
- Not all regions have to be occupied.

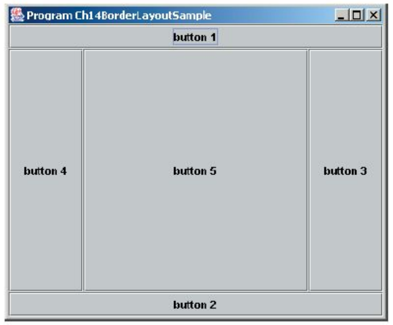
BorderLayout Sample

When the frame first appears on the screen.





After the frame is resized.

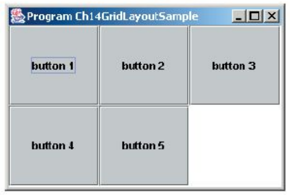


GridLayout

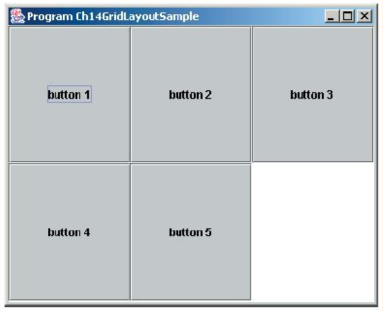
- This layout manager placesGUI components on equal-size N by M grids.
- Components are placed in top-to-bottom, left-to-right order.
- The number of rows and columns remains the same after the frame is resized, but the width and height of each region will change.

GridLayout Sample

When the frame first appears on the screen.



After the frame is resized.



Nesting Panels

- It is possible, but very difficult, to place all GUI components on a single JPanel or other types of containers.
- A better approach is to use multiple panels, placing panels inside other panels.
- To illustrate this technique, we will create two sample frames that contain nested panels.
- Ch14NestedPanels1.java provides the user interface for playing Tic Tac Toe.
- Ch14NestedPanels2.java provides the user interface for playing HiLo.

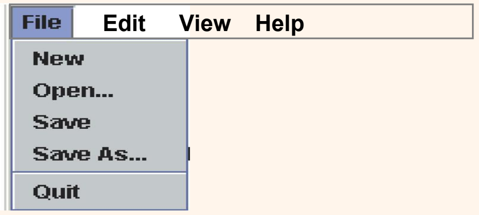
Other Common GUI Components

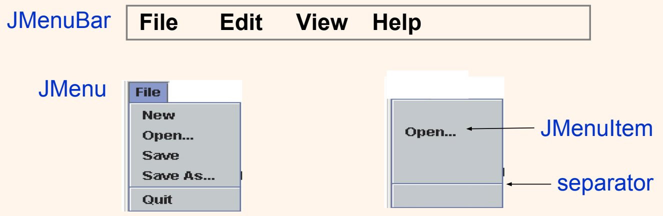
- JCheckBox
  - see Ch14JCheckBoxSample1.java and Ch14JCheckBoxSample2.java
- JRadioButton
  - see Ch14JRadioButtonSample.java
- JComboBox
  - see Ch14JComboBoxSample.java
- JList
  - see Ch14JListSample.java
- JSlider
  - see Ch14JSliderSample.java

Menus

- The javax.swing package contains three menu-related classes: JMenuBar, JMenu, and JMenuItem.
- JMenuBar is a bar where the menus are placed. There is one menu bar per frame.
- JMenu (such as File or Edit) is a group of menu choices. JMenuBar may include many JMenu objects.
- JMenuItem (such as Copy, Cut, or Paste) is an individual menu choice in a JMenu object.
- Only the JMenuItem objects generate events.

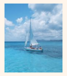
Menu Components





Sequence for Creating Menus

1. Create a JMenuBar object and attach it to a frame.
2. Create a JMenu object.
3. Create JMenuItem objects and add them to the JMenu object.
4. Attach the JMenu object to the JMenuBar object.



Handling Mouse Events

- Mouse events include such user interactions as
  - moving the mouse
  - dragging the mouse (moving the mouse while the mouse button is being pressed)
  - clicking the mouse buttons.
- The MouseListener interface handles mouse button
  - mouseClicked, mouseEntered, mouseExited, mousePressed, and mouseReleased
- The MouseMotionListener interface handles mouse movement
  - mouseDragged and mouseMoved.
- See Ch14TrackMouseFrame and Ch14SketchPad