Ch 3-4: GUI Basics

Java Software Solutions
Foundations of Program Design
Sixth Edition

by
Lewis & Loftus
GUI Components

- A GUI component is an object that represents a screen element such as a button or a text field.
- GUI-related classes are defined primarily in the `java.awt and the javax.swing packages`.
- The Abstract Windowing Toolkit (AWT) was the original Java GUI package.
- The Swing package provides additional and more versatile components.
GUI Containers

- A *GUI container* is a component that is used to hold and organize other components.

- A *frame* is a container that is used to display a GUI-based Java application.

- A frame is displayed as a separate window with a title bar – it can be repositioned and resized on the screen as needed.

- A *panel* is a container that cannot be displayed on its own but is used to organize other components.

- A panel must be added to another container to be displayed.
GUI Containers

- A GUI container can be classified as either heavyweight or lightweight
  - A **heavyweight container** is one that is managed by the underlying operating system
  - A **lightweight container** is managed by the Java program itself
- Occasionally this distinction is important
- A frame is a heavyweight container and a panel is a lightweight container
Labels

- A *label* is a GUI component that displays a line of text.
- Labels are usually used to display information or identify other components in the interface.
- Let's look at a program that organizes two labels in a panel and displays that panel in a frame.
- This program is not interactive, but the frame can be repositioned and resized.

See Authority.java
Containers that contain other components make up the containment hierarchy of an interface.

This hierarchy can be as intricate as needed to create the visual effect desired.

The following example nests two panels inside a third panel – note the effect this has as the frame is resized.

See NestedPanels.java
Images

- Images are often used in programs with a graphical interface.
- Java can manage images in both JPEG and GIF formats.
- As we've seen, a `JLabel` object can be used to display a line of text.
- It can also be used to display an image using the `ImageIcon` class:
  ```java
  ImageIcon icon = new ImageIcon("devil.gif");
  label1 = new JLabel("Devil Left", icon, SwingConstants.CENTER);
  ```
- That is, a label can be composed of text, and image, or both at the same time.
- And we can set the position of the image relative to the text.
Some objects contain information that determines how the object should be represented visually.

Most GUI components are graphical objects.

We can have some effect on how components get drawn.

We did this in Chapter 2 when we defined the `paint` method of an applet.

Let's look at some other examples of graphical objects.

Coming up: Smiling Face Example.
Graphical User Interfaces

• A Graphical User Interface (GUI) in Java is created with at least three kinds of objects:
  ◦ components
  ◦ events
  ◦ listeners

• We've previously discussed components, which are objects that represent screen elements
  ◦ labels, buttons, text fields, menus, etc.

• Some components are containers that hold and organize other components
  ◦ frames, panels, applets, dialog boxes

Coming up: Events
Events

- An event is an object that represents some activity to which we may want to respond.

- For example, we may want our program to perform some action when the following occurs:
  - the mouse is moved
  - the mouse is dragged
  - a mouse button is clicked
  - a graphical button is clicked
  - a keyboard key is pressed
  - a timer expires

- Events often correspond to user actions, but not always.

Coming up: Events and Listeners
Events and Listeners

- The Java standard class library contains several classes that represent typical events

- Components, such as a graphical button, generate (or fire) an event when it occurs

- A *listener* object "waits" for an event to occur and responds accordingly

- We can design listener objects to take whatever actions are appropriate when an event occurs
Component

Event

Listener

A component object may generate an event

A corresponding listener object is designed to respond to the event

When the event occurs, the component calls the appropriate method of the listener, passing an object that describes the event

Coming up: GUI Development
Generally we use components and events that are predefined by classes in the Java class library

Therefore, to create a Java program that uses a GUI we must:

1. instantiate and set up the necessary components
2. implement listener classes for any events we care about
3. establish the relationship between listeners and components that generate the corresponding events

Let's now explore some new components and see how this all comes together
Buttons

- A push button is a component that allows the user to initiate an action by pressing a graphical button using the mouse.
- A push button is defined by the JButton class.
- It generates an action event.
- The PushCounter example displays a push button that increments a counter each time it is pushed.

See PushCounter
PushCounterPanel.java
Push Counter Example

- The components of the GUI are the button, a label to display the counter, a panel to organize the components, and the main frame.

- The `PushCounterPanel` class represents the panel used to display the button and label.

- The `PushCounterPanel` class is derived from `JPanel` using inheritance.

- The constructor of `PushCounterPanel` sets up the elements of the GUI and initializes the counter to zero.
The PushCounterPanel also serves as the listener for the button events.

This is done by implementing the “ActionListener” interface:

```java
public class PushCounterPanel extends JPanel implements ActionListener
```

Implementing the interface ActionListener says

- This class will define all methods defined in the ActionListener interface
- Anywhere you need an ActionListener, you can use an instance of this class. It “is” an ActionListener as well as a JPanel from Java’s point of view
Listener classes are written by implementing a listener interface.

An interface is a list of methods that the implementing class must define.

The only method in the ActionListener interface is the actionPerformed method.

The Java class library contains interfaces for many types of events.

We discuss interfaces in more detail in Chapter 6.

Coming up: Push Counter Example
Push Counter Example

- **The PushCounterPanel constructor:**
  - establishes the relationship between the button and the listener by the call to `addActionListener`

- When the user presses the button, the button component creates an `ActionEvent` object and calls the `actionPerformed` method of the listener

- The `actionPerformed` method increments the counter and resets the text of the label

Coming up: Text Fields
Text Fields

- Let's look at another GUI example that uses another type of component
  - A *text field* allows the user to enter one line of input
  - If the cursor is in the text field, the text field component generates an action event when the enter key is pressed

See
textfieldExample/Fahrenheit.java
textFieldExample/FahrenheitPanel.java
Fahrenheit Example

- Like the PushCounter example, the GUI is set up in a separate panel class

- The FahrenheitPanel constructor instantiates the listener and adds it to the text field

- When the user types a temperature and presses enter, the text field generates the action event and calls the actionPerformed method of the listener

- The actionPerformed method computes the conversion and updates the result label

Challenge: Can you make it dynamically update the temp as you type (maybe a KeyListener?)