Chapter 17

Swing I

Slides prepared by Rose Williams, *Binghamton University*

Kenrick Mock, *University of Alaska Anchorage*

PEARSON ALWAYS LEARNING

Introduction to Swing

- A *GUI (graphical user interface)* is a windowing system that interacts with the user
- The Java *AWT (Abstract Window Toolkit)* package is the original Java package for creating *GUIs*
- The Swing package is an improved version of the AWT
  - However, it does not completely replace the AWT
  - Some AWT classes are replaced by Swing classes, but other AWT classes are needed when using Swing
- Swing GUIs are designed using a form of object-oriented programming known as *event-driven programming*

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Events

- *Event-driven programming* is a programming style that uses a signal-and-response approach to programming
- An *event* is an object that acts as a signal to another object know as a *listener*
- The sending of an event is called *firing the event*
  - The object that fires the event is often a GUI component, such as a button that has been clicked

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Listeners

- A listener object performs some action in response to the event
  - A given component may have any number of listeners
  - Each listener may respond to a different kind of event, or multiple listeners might may respond to the same events

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Exception Objects

- An exception object is an event
  - The throwing of an exception is an example of firing an event
- The listener for an exception object is the `catch` block that catches the event

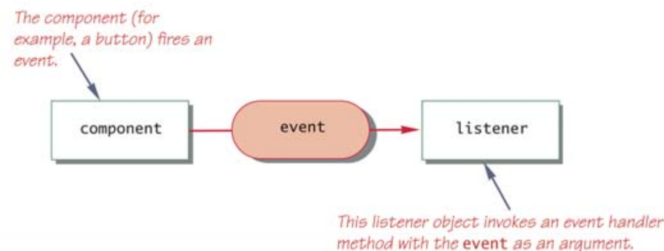
Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Event Handlers

- A listener object has methods that specify what will happen when events of various kinds are received by it
  - These methods are called *event handlers*
- The programmer using the listener object will define or redefine these event-handler methods

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Event Firing and an Event Listener



Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Event-Driven Programming

- Event-driven programming is very different from most programming seen up until now
  - So far, programs have consisted of a list of statements executed in order
  - When that order changed, whether or not to perform certain actions (such as repeat statements in a loop, branch to another statement, or invoke a method) was controlled by the logic of the program

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Event-Driven Programming

- In event-driven programming, objects are created that can fire events, and listener objects are created that can react to the events
- The program itself no longer determines the order in which things can happen
  - Instead, the events determine the order

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Event-Driven Programming

- In an event-driven program, the next thing that happens depends on the event that occurs
- In particular, *methods are defined that will never be explicitly invoked in any program*
  - Instead, methods are invoked automatically when an event signals that the method needs to be called

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A Simple Window

- A simple window can consist of an object of the `JFrame` class
  - A `JFrame` object includes a border and the usual three buttons for minimizing, changing the size of, and closing the window
  - The `JFrame` class is found in the `javax.swing` package

```
JFrame firstWindow = new JFrame();
```

- A `JFrame` can have components added to it, such as buttons, menus, and text labels
  - These components can be programmed for action

```
firstWindow.add(endButton);
```

  - It can be made visible using the `setVisible` method

```
firstWindow.setVisible(true);
```

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A First Swing Demonstration (Part 1 of 4)

Display 17.2 **A First Swing Demonstration Program**

```
1  import javax.swing.JFrame;
2  import javax.swing.JButton;

3  public class FirstSwingDemo
4  {
5      public static final int WIDTH = 300;
6      public static final int HEIGHT = 200;

7      public static void main(String[] args)
8      {
9          JFrame firstWindow = new JFrame();
10         firstWindow.setSize(WIDTH, HEIGHT);
```

This program is not typical of the style we will use in Swing programs.

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A First Swing Demonstration (Part 2 of 4)

Display 17.2 A First Swing Demonstration Program

```
11          firstWindow.setDefaultCloseOperation(
12                              JFrame.DO_NOTHING_ON_CLOSE);

13          JButton endButton = new JButton("Click to end program.");
14          EndingListener buttonEar = new EndingListener();
15          endButton.addActionListener(buttonEar);
16          firstWindow.add(endButton);

17          firstWindow.setVisible(true);
18      }
19  }
```

This is the file `FirstSwingDemo.java`*.*

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A First Swing Demonstration (Part 3 of 4)

Display 17.2 A First Swing Demonstration Program

```
1  import java.awt.event.ActionListener;
2  import java.awt.event.ActionEvent;

3  public class EndingListener implements ActionListener
4  {
5      public void actionPerformed(ActionEvent e)
6      {
7          System.exit(0);
8      }
9  }
```

This is the file `EndingListener.java`*.*

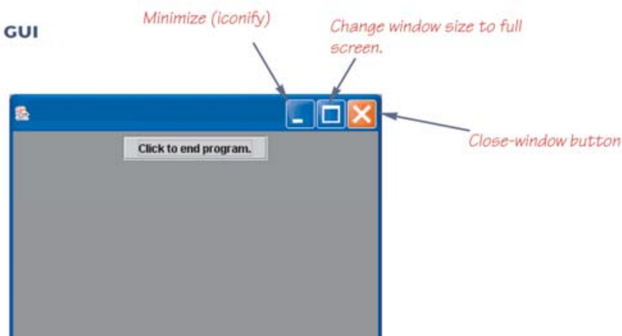
(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A First Swing Demonstration (Part 4 of 4)

Display 17.2 A First Swing Demonstration Program

RESULTING GUI



Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Some Methods in the Class `JFrame` (Part 1 of 3)

Display 17.3 Some Methods in the Class `JFrame`

The class `JFrame` is in the `javax.swing` package.

`public JFrame()`

Constructor that creates an object of the class `JFrame`.

`public JFrame(String title)`

Constructor that creates an object of the class `JFrame` with the title given as the argument.

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Some Methods in the Class **JFrame** (Part 2 of 3)

Display 17.3 **Some Methods in the Class JFrame**

```
public void setDefaultCloseOperation(int operation)
```

Sets the action that will happen by default when the user clicks the close-window button. The argument should be one of the following defined constants:

JFrame.DO_NOTHING_ON_CLOSE: Do nothing. The JFrame does nothing, but if there are any registered window listeners, they are invoked. (Window listeners are explained in Chapter 19.)
JFrame.HIDE_ON_CLOSE: Hide the frame after invoking any registered WindowListener objects.
JFrame.DISPOSE_ON_CLOSE: Hide and *dispose* the frame after invoking any registered window listeners. When a window is disposed it is eliminated but the program does not end. To end the program, you use the next constant as an argument to setDefaultCloseOperation.
JFrame.EXIT_ON_CLOSE: Exit the application using the System exit method. (Do not use this for frames in applets. Applets are discussed in Chapter 18.)

If no action is specified using the method setDefaultCloseOperation, then the default action taken is JFrame.HIDE_ON_CLOSE.
Throws an IllegalArgumentException if the argument is not one of the values listed above.[2]
Throws a SecurityException if the argument is JFrame.EXIT_ON_CLOSE and the Security Manager will not allow the caller to invoke System.exit. (You are not likely to encounter this case.)

```
public void setSize(int width, int height)
```

Sets the size of the calling frame so that it has the width and height specified. Pixels are the units of length used.

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Some Methods in the Class **JFrame** (Part 3 of 3)

Display 17.3 **Some Methods in the Class JFrame**

```
public void setTitle(String title)
```

Sets the title for this frame to the argument string.

```
public void add(Component componentAdded)
```

Adds a component to the JFrame.

```
public void setLayout(LayoutManager manager)
```

Sets the layout manager. Layout managers are discussed later in this chapter.

```
public void setJMenuBar(JMenuBar menubar)
```

Sets the menubar for the calling frame. (Menus and menu bars are discussed later in this chapter.)

```
public void dispose()
```

Eliminates the calling frame and all its subcomponents. Any memory they use is released for reuse. If there are items left (items other than the calling frame and its subcomponents), then this does not end the program. (The method dispose is discussed in Chapter 19.)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Pixels and the Relationship between Resolution and Size

- A *pixel* is the smallest unit of space on a screen
  - Both the size and position of Swing objects are measured in pixels
  - The more pixels on a screen, the greater the screen resolution
- A high-resolution screen of fixed size has many pixels
  - Therefore, each one is very small
- A low-resolution screen of fixed size has fewer pixels
  - Therefore, each one is much larger
- Therefore, a two-pixel figure on a low-resolution screen will look larger than a two-pixel figure on a high-resolution screen

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Pitfall: Forgetting to Program the Close-Window Button

- The following lines from the **FirstSwingDemo** program ensure that when the user clicks the *close-window button*, nothing happens

```
firstWindow.setDefaultCloseOperation(
                    JFrame.DO_NOTHING_ON_CLOSE);
```

- If this were not set, the default action would be **JFrame.HIDE_ON_CLOSE**
  - This would make the window invisible and inaccessible, but would not end the program
  - Therefore, given this scenario, there would be no way to click the "Click to end program" button
- Note that the close-window and other two accompanying buttons are part of the **JFrame** object, and not separate buttons

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Buttons

- A *button* object is created from the class **`JButton`** and can be added to a **`JFrame`**
  - The argument to the **`JButton`** constructor is the string that appears on the button when it is displayed

```
JButton endButton = new
            JButton("Click to end program.");
firstWindow.add(endButton);
```

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Action Listeners and Action Events

- Clicking a button fires an event
- The event object is "sent" to another object called a listener
  - This means that a method in the listener object is invoked automatically
  - Furthermore, it is invoked with the event object as its argument
- In order to set up this relationship, a GUI program must do two things
  1. It must specify, for each button, what objects are its listeners, i.e., it must register the listeners
  2. It must define the methods that will be invoked automatically when the event is sent to the listener

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Action Listeners and Action Events

```
EndingListener buttonEar = new
                   EndingListener());
endButton.addActionListener(buttonEar);
```

- Above, a listener object named **`buttonEar`** is created and registered as a listener for the button named **`endButton`**
  - Note that a button fires events known as *action events*, which are handled by listeners known as *action listeners*

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Action Listeners and Action Events

- Different kinds of components require different kinds of listener classes to handle the events they fire
- An action listener is an object whose class implements the **`ActionListener`** interface
  - The **`ActionListener`** interface has one method heading that must be implemented

```
public void actionPerformed(ActionEvent e)
```

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Action Listeners and Action Events

```
public void actionPerformed(ActionEvent e)
{
  System.exit(0);
}
```

- The `EndingListener` class defines its `actionPerformed` method as above
  - When the user clicks the `endButton`, an action event is sent to the action listener for that button
  - The `EndingListener` object `buttonEar` is the action listener for `endButton`
  - The action listener `buttonEar` receives the action event as the parameter `e` to its `actionPerformed` method, which is automatically invoked
  - Note that `e` must be received, even if it is not used

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Pitfall: Changing the Heading for `actionPerformed`

- When the `actionPerformed` method is implemented in an action listener, its header must be the one specified in the `ActionListener` interface
  - It is already determined, and may not be changed
  - Not even a throws clause may be added

```
public void actionPerformed(ActionEvent e)
```

- The only thing that can be changed is the name of the parameter, since it is just a placeholder
  - Whether it is called `e` or something else does not matter, as long as it is used consistently within the body of the method

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Tip: Ending a Swing Program

- GUI programs are often based on a kind of infinite loop
  - The windowing system normally stays on the screen until the user indicates that it should go away
- If the user never asks the windowing system to go away, it will never go away
- In order to end a GUI program, `System.exit` must be used when the user asks to end the program
  - It must be explicitly invoked, or included in some library code that is executed
  - Otherwise, a Swing program will not end after it has executed all the code in the program

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A Better Version of Our First Swing GUI

- A better version of `FirstWindow` makes it a derived class of the class `JFrame`
  - This is the normal way to define a windowing interface
- The constructor in the new `FirstWindow` class starts by calling the constructor for the parent class using `super();`
  - This ensures that any initialization that is normally done for all objects of type `JFrame` will be done
- Almost all initialization for the window `FirstWindow` is placed in the constructor for the class
- Note that this time, an anonymous object is used as the action listener for the `endButton`

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

The Normal Way to Define a **JFrame** (Part 1 of 4)

Display 17.4 **The Normal Way to Define a JFrame**

```
1   import javax.swing.JFrame;
2   import javax.swing.JButton;

3   public class FirstWindow extends JFrame
4   {
5       public static final int WIDTH = 300;
6       public static final int HEIGHT = 200;

7       public FirstWindow()
8       {
9           super();
10          setSize(WIDTH, HEIGHT);

11          setTitle("First Window Class");
```

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

The Normal Way to Define a **JFrame** (Part 2 of 4)

Display 17.4 **The Normal Way to Define a JFrame**

```
12          setDefaultCloseOperation(
13                          JFrame.DO_NOTHING_ON_CLOSE);

14          JButton endButton = new JButton("Click to end program.");
15          endButton.addActionListener(new EndingListener());
16          add(endButton);
17      }
18  }
```

The class `EndingListener` *is defined in Display 17.2.*

This is the file `FirstWindow.java`.

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

The Normal Way to Define a **JFrame** (Part 3 of 4)

Display 17.4 **The Normal Way to Define a JFrame**

This is the file `DemoWindow.java`.

```
1   public class DemoWindow
2   {
3       public static void main(String[] args)
4       {
5           FirstWindow w = new FirstWindow();
6           w.setVisible(true);
7       }
8   }
```

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

The Normal Way to Define a **JFrame** (Part 4 of 4)

Display 17.4 **The Normal Way to Define a JFrame**

RESULTING GUI

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Labels

- A *label* is an object of the class **`JLabel`**
  - Text can be added to a **`JFrame`** using a label
  - The text for the label is given as an argument when the **`JLabel`** is created
  - The label can then be added to a **`JFrame`**

```
JLabel greeting = new JLabel("Hello");
add(greeting);
```

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Color

- In Java, a *color* is an object of the class **`Color`**
  - The class **`Color`** is found in the **`java.awt`** package
  - There are constants in the **`Color`** class that represent a number of basic colors
- A **`JFrame`** can not be colored directly
  - Instead, a program must color something called the *content pane* of the **`JFrame`**
  - Since the content pane is the "inside" of a **`JFrame`**, coloring the content pane has the effect of coloring the inside of the **`JFrame`**
  - Therefore, the background color of a **`JFrame`** can be set using the following code:

```
getContentPane().setBackground(Color);
```

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

The Color Constants

Display 17.5 **The Color Constants**

Color.BLACK	Color.MAGENTA
Color.BLUE	Color.ORANGE
Color.CYAN	Color.PINK
Color.DARK_GRAY	Color.RED
Color.GRAY	Color.WHITE
Color.GREEN	Color.YELLOW
Color.LIGHT_GRAY	

The class `Color` is in the `java.awt` package.

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A **`JFrame`** with Color (Part 1 of 4)

Display 17.6 **A JFrame with Color**

```
1  import javax.swing.JFrame;
2  import javax.swing.JLabel;
3  import java.awt.Color;

4  public class ColoredWindow extends JFrame
5  {
6      public static final int WIDTH = 300;
7      public static final int HEIGHT = 200;

8      public ColoredWindow(Color theColor)
9      {
10         super("No Charge for Color");
11         setSize(WIDTH, HEIGHT);
12         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A **JFrame** with Color (Part 2 of 4)

Display 17.6 **A JFrame with Color**

```
13          getContentPane().setBackground(theColor);
14          JLabel aLabel = new JLabel("Close-window button works.");
15          add(aLabel);
16      }
17      public ColoredWindow()
18      {
19          this(Color.PINK);
20      }
21  }
```

This is an invocation of the other constructor.

This is the file ColoredWindow.java.

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A **JFrame** with Color (Part 3 of 4)

Display 17.6 **A JFrame with Color**

```
1   import java.awt.Color;
2   public class DemoColoredWindow
3   {
4       public static void main(String[] args)
5       {
6           ColoredWindow w1 = new ColoredWindow();
7           w1.setVisible(true);
8           ColoredWindow w2 = new ColoredWindow(Color.YELLOW);
9           w2.setVisible(true);
10      }
11  }
```

This is the file ColoredWindow.java.

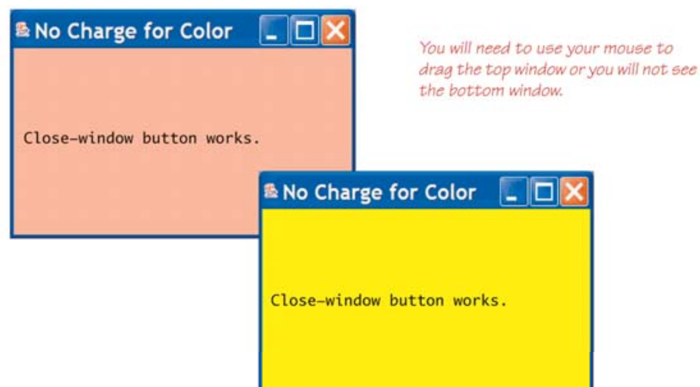
(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A **JFrame** with Color (Part 4 of 4)

Display 17.6 **A JFrame with Color**

RESULTING GUI



You will need to use your mouse to drag the top window or you will not see the bottom window.

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Containers and Layout Managers

- Multiple components can be added to the content pane of a **`JFrame`** using the **`add`** method
  - However, the **`add`** method does not specify how these components are to be arranged
- To describe how multiple components are to be arranged, a *layout manager* is used
  - There are a number of layout manager classes such as **`BorderLayout`**, **`FlowLayout`**, and **`GridLayout`**
  - If a layout manager is not specified, a default layout manager is used

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Border Layout Managers

- A **BorderLayout** manager places the components that are added to a **JFrame** object into five regions
  - These regions are: **BorderLayout.NORTH**, **BorderLayout.SOUTH**, **BorderLayout.EAST**, **BorderLayout.WEST**, and **BorderLayout.Center**
- A **BorderLayout** manager is added to a **JFrame** using the **setLayout** method
  - For example:

```
setLayout(new BorderLayout());
```

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

The BorderLayout Manager (Part 1 of 4)

Display 17.7 **The BorderLayout Manager**

```
1  import javax.swing.JFrame;
2  import javax.swing.JLabel;
3  import java.awt.BorderLayout;

4  public class BorderLayoutJFrame extends JFrame
5  {
6      public static final int WIDTH = 500;
7      public static final int HEIGHT = 400;

8      public BorderLayoutJFrame()
9      {
10         super("BorderLayout Demonstration");
11         setSize(WIDTH, HEIGHT);
12         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

The BorderLayout Manager (Part 2 of 4)

Display 17.7 **The BorderLayout Manager**

```
13         setLayout(new BorderLayout());

14         JLabel label1 = new JLabel("First label");
15         add(label1, BorderLayout.NORTH);

16         JLabel label2 = new JLabel("Second label");
17         add(label2, BorderLayout.SOUTH);

18         JLabel label3 = new JLabel("Third label");
19         add(label3, BorderLayout.CENTER);
20     }
21 }
```

This is the file BorderLayoutJFrame.java.

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

The BorderLayout Manager (Part 3 of 4)

Display 17.7 **The BorderLayout Manager**

This is the file BorderLayoutDemo.java.

```
1  public class BorderLayoutDemo
2  {
3      public static void main(String[] args)
4      {
5          BorderLayoutJFrame gui = new BorderLayoutJFrame();
6          gui.setVisible(true);
7      }
8  }
```

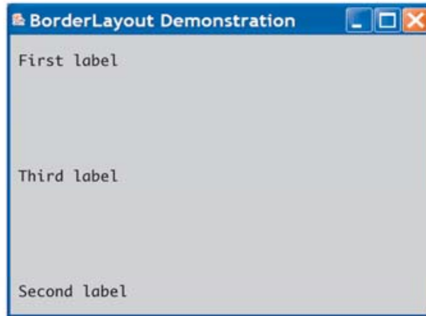
(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

The **BorderLayout** Manager (Part 4 of 4)

Display 17.7 The BorderLayout Manager

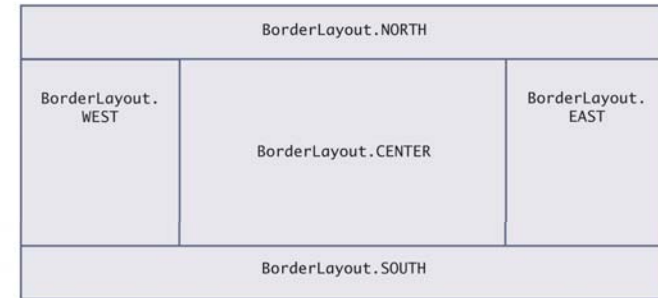
RESULTING GUI



Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

BorderLayout Regions

Display 17.8 BorderLayout Regions



Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Border Layout Managers

- The previous diagram shows the arrangement of the five border layout regions
  - Note: None of the lines in the diagram are normally visible
- When using a `BorderLayout` manager, the location of the component being added is given as a second argument to the `add` method

```
add(label1, BorderLayout.NORTH);
```

  - Components can be added in any order since their location is specified

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Flow Layout Managers

- The `FlowLayout` manager is the simplest layout manager

```
setLayout(new FlowLayout());
```

  - It arranges components one after the other, going from left to right
  - Components are arranged in the order in which they are added
- Since a location is not specified, the `add` method has only one argument when using the `FlowLayoutManager`

```
add.(label1);
```

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Grid Layout Managers

- A **GridLayout** manager arranges components in a two-dimensional grid with some number of rows and columns

```
setLayout(new GridLayout(rows, columns));
```

  - Each entry is the same size
  - The two numbers given as arguments specify the number of rows and columns
  - Each component is stretched so that it completely fills its grid position
  - Note: None of the lines in the diagram are normally visible

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Grid Layout Managers

- When using the **GridLayout** class, the method **add** has only one argument

```
add(label1);
```

  - Items are placed in the grid from left to right
  - The top row is filled first, then the second, and so forth
  - Grid positions may not be skipped
- Note the use of a **main** method in the GUI class itself in the following example
  - This is often a convenient way of demonstrating a class

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

The **GridLayout** Manager (Part 1 of 4)

Display 17.9 **The GridLayout Manager**

```
1   import javax.swing.JFrame;
2   import javax.swing.JLabel;
3   import java.awt.GridLayout;

4   public class GridLayoutJFrame extends JFrame
5   {
6       public static final int WIDTH = 500;
7       public static final int HEIGHT = 400;

8       public static void main(String[] args)
9       {
10          GridLayoutJFrame gui = new GridLayoutJFrame(2, 3);
11          gui.setVisible(true);
12      }
```

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

The **GridLayout** Manager (Part 2 of 4)

Display 17.9 **The GridLayout Manager**

```
13      public GridLayoutJFrame(int rows, int columns )
14      {
15          super();
16          setSize(WIDTH, HEIGHT);
17          setTitle("GridLayout Demonstration");
18          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19          setLayout(new GridLayout(rows, columns ));

20          JLabel label1 = new JLabel("First label");
21          add(label1);
```

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

The GridLayout Manager (Part 3 of 4)

Display 17.9 **The GridLayout Manager**

```
22          JLabel label2 = new JLabel("Second label");
23          add(label2);

24          JLabel label3 = new JLabel("Third label");
25          add(label3);

26          JLabel label4 = new JLabel("Fourth label");
27          add(label4);

28          JLabel label5 = new JLabel("Fifth label");
29          add(label5);
30      }
31  }
```

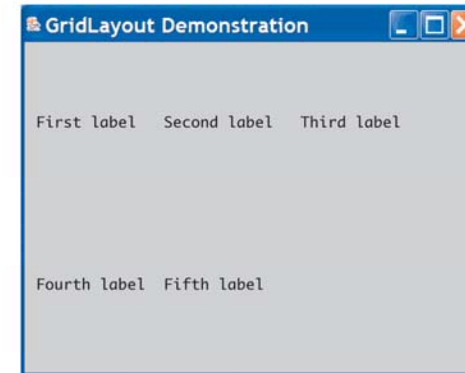
(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

The GridLayout Manager (Part 4 of 4)

Display 17.9 **The GridLayout Manager**

RESULTING GUI



Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Some Layout Managers

Display 17.10 **Some Layout Managers**

LAYOUT MANAGER	DESCRIPTION
These layout manager classes are in the java.awt package.	
FlowLayout	Displays components from left to right in the order in which they are added to the container.
BorderLayout	Displays the components in five areas: north, south, east, west, and center. You specify the area a component goes into in a second argument of the add method.
GridLayout	Lays out components in a grid, with each component stretched to fill its box in the grid.

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Panels

- A GUI is often organized in a hierarchical fashion, with containers called *panels* inside other containers
- A panel is an object of the **JPanel** class that serves as a simple container
  - It is used to group smaller objects into a larger component (the panel)
  - One of the main functions of a **JPanel** object is to subdivide a **JFrame** or other container

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Panels

- Both a **JFrame** and each panel in a **JFrame** can use different layout managers
  - Additional panels can be added to each panel, and each panel can have its own layout manager
  - This enables almost any kind of overall layout to be used in a GUI

```
setLayout(new BorderLayout());
JPanel somePanel = new JPanel();
somePanel.setLayout(new FlowLayout());
```

- Note in the following example that panel and button objects are given color using the **setBackground** method without invoking **getContentPane**
  - The **getContentPane** method is only used when adding color to a **JFrame**

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Using Panels (Part 1 of 8)

Display 17.11 **Using Panels**

```
1  import javax.swing.JFrame;
2  import javax.swing.JPanel;
3  import java.awt.BorderLayout;
4  import java.awt.GridLayout;
5  import java.awt.FlowLayout;
6  import java.awt.Color;
7  import javax.swing.JButton;
8  import java.awt.event.ActionListener;
9  import java.awt.event.ActionEvent;

10 public class PanelDemo extends JFrame implements ActionListener
11 {
12     public static final int WIDTH = 300;
13     public static final int HEIGHT = 200;
```

In addition to being the GUI class, the class PanelDemo is the action listener class. An object of the class PanelDemo is the action listener for the buttons in that object.

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Using Panels (Part 2 of 8)

Display 17.11 **Using Panels**

```
14     private JPanel redPanel;
15     private JPanel whitePanel;
16     private JPanel bluePanel;

17     public static void main(String[] args)
18     {
19         PanelDemo gui = new PanelDemo();
20         gui.setVisible(true);
21     }

22     public PanelDemo()
23     {
24         super("Panel Demonstration");
25         setSize(WIDTH, HEIGHT);
26         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
27         setLayout(new BorderLayout());
```

We made these instance variables because we want to refer to them in both the constructor and the method actionPerformed.

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Using Panels (Part 3 of 8)

Display 17.11 **Using Panels**

```
28         JPanel biggerPanel = new JPanel();
29         biggerPanel.setLayout(new GridLayout(1, 3));

30         redPanel = new JPanel();
31         redPanel.setBackground(Color.LIGHT_GRAY);
32         biggerPanel.add(redPanel);

33         whitePanel = new JPanel();
34         whitePanel.setBackground(Color.LIGHT_GRAY);
35         biggerPanel.add(whitePanel);
```

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Using Panels (Part 4 of 8)

Display 17.11 **Using Panels**

```
36          bluePanel = new JPanel();
37          bluePanel.setBackground(Color.LIGHT_GRAY);
38          biggerPanel.add(bluePanel);

39          add(biggerPanel, BorderLayout.CENTER);

40          JPanel buttonPanel = new JPanel();
41          buttonPanel.setBackground(Color.LIGHT_GRAY);
42          buttonPanel.setLayout(new FlowLayout());

43          JButton redButton = new JButton("Red");
44          redButton.setBackground(Color.RED);
45          redButton.addActionListener(this);
46          buttonPanel.add(redButton);
```

An object of the class PanelDemo *is the action listener for the buttons in that object.*

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Using Panels (Part 5 of 8)

Display 17.11 **Using Panels**

```
47          JButton whiteButton = new JButton("White");
48          whiteButton.setBackground(Color.WHITE);
49          whiteButton.addActionListener(this);
50          buttonPanel.add(whiteButton);

51          JButton blueButton = new JButton("Blue");
52          blueButton.setBackground(Color.BLUE);
53          blueButton.addActionListener(this);
54          buttonPanel.add(blueButton);

55          add(buttonPanel, BorderLayout.SOUTH);
56      }
```

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Using Panels (Part 6 of 8)

Display 17.11 **Using Panels**

```
57      public void actionPerformed(ActionEvent e)
58      {
59          String buttonString = e.getActionCommand();

60          if (buttonString.equals("Red"))
61              redPanel.setBackground(Color.RED);
62          else if (buttonString.equals("White"))
63              whitePanel.setBackground(Color.WHITE);
64          else if (buttonString.equals("Blue"))
65              bluePanel.setBackground(Color.BLUE);
66          else
67              System.out.println("Unexpected error.");
68      }
69  }
```

(continued)

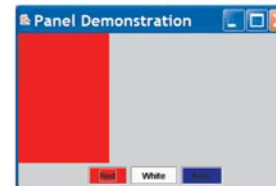
Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Using Panels (Part 7 of 8)

Display 17.11 **Using Panels**

RESULTING GUI (When first run)

RESULTING GUI (After clicking Red button)



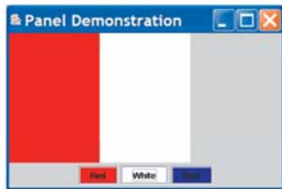
(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

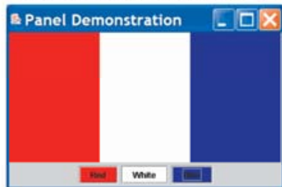
Using Panels (Part 8 of 8)

Display 17.11 **Using Panels**

RESULTING GUI (After clicking White button)



RESULTING GUI (After clicking Blue button)



Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

The `Container` Class

- Any class that is a descendent class of the class Container is considered to be a container class
  - The `Container` class is found in the `java.awt` package, not in the Swing library
- Any object that belongs to a class derived from the `Container` class (or its descendents) can have components added to it
- The classes `JFrame` and `JPanel` are descendent classes of the class `Container`
  - Therefore they and any of their descendents can serve as a container

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

The `JComponent` Class

- Any descendent class of the class `JComponent` is called a *component class*
  - Any `JComponent` object or *component* can be added to any container class object
  - Because it is derived from the class `Container`, a `JComponent` can also be added to another `JComponent`

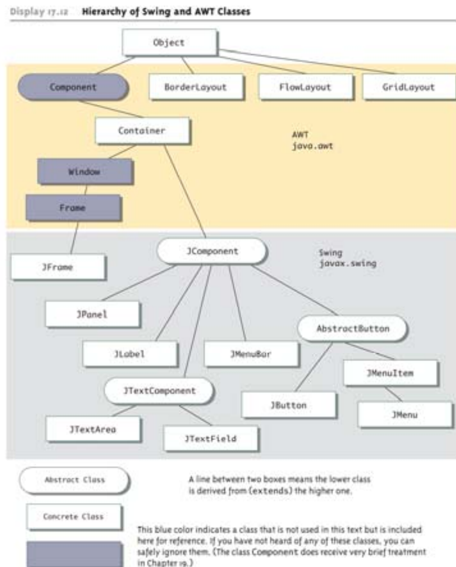
Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Objects in a Typical GUI

- Almost every GUI built using Swing container classes will be made up of three kinds of objects:
  1. The container itself, probably a panel or window-like object
  2. The components added to the container such as labels, buttons, and panels
  3. A layout manager to position the components inside the container

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Hierarchy of Swing and AWT Classes



Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Tip: Code a GUI's Look and Actions Separately

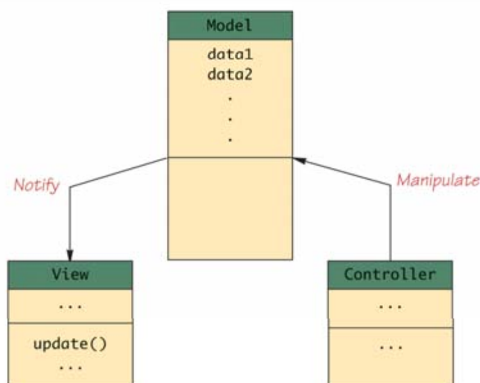
- The task of designing a Swing GUI can be divided into two main subtasks:
  1. Designing and coding the appearance of the GUI on the screen
  2. Designing and coding the actions performed in response to user actions
- In particular, it is useful to implement the **actionPerformed** method as a *stub*, until the GUI looks the way it should

```
public void actionPerformed(ActionEvent e)
{}
```

- This philosophy is at the heart of the technique used by the *Model-View-Controller* pattern

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

The Model-View-Controller Pattern



Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Menu Bars, Menus, and Menu Items

- A *menu* is an object of the class **JMenu**
- A choice on a menu is called a *menu item*, and is an object of the class **JMenuItem**
  - A menu can contain any number of menu items
  - A menu item is identified by the string that labels it, and is displayed in the order to which it was added to the menu
- The **add** method is used to add a menu item to a menu in the same way that a component is added to a container object

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Menu Bars, Menus, and Menu Items

- The following creates a new menu, and then adds a menu item to it

```
JMenu diner = new
        JMenu("Daily Specials");
JMenuItem lunch = new
        JMenuItem("Lunch Specials");
lunch.addActionListener(this);
diner.add(lunch);
```

  – Note that the `this` parameter has been registered as an action listener for the menu item

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Nested Menus

- The class `JMenu` is a descendent of the `JMenuItem` class
  – Every `JMenu` can be a menu item in another menu
  – Therefore, menus can be nested
- Menus can be added to other menus in the same way as menu items

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Menu Bars and `JFrame`

- A *menu bar* is a container for menus, typically placed near the top of a windowing interface
- The `add` method is used to add a menu to a menu bar in the same way that menu items are added to a menu

```
JMenuBar bar = new JMenuBar();
bar.add(diner);
```

- The menu bar can be added to a `JFrame` in two different ways
  1. Using the `setJMenuBar` method
     `setJMenuBar(bar);`
  2. Using the `add` method – which can be used to add a menu bar to a `JFrame` or any other container

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A GUI with a Menu (Part 1 of 8)

Display 17.14 **A GUI with a Menu**

```
1  import javax.swing.JFrame;
2  import javax.swing.JPanel;
3  import java.awt.GridLayout;
4  import java.awt.Color;
5  import javax.swing.JMenu;
6  import javax.swing.JMenuItem;
7  import javax.swing.JMenuBar;
8  import java.awt.event.ActionListener;
9  import java.awt.event.ActionEvent;
```

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A GUI with a Menu (Part 2 of 8)

Display 17.14 **A GUI with a Menu**

```
10  public class MenuDemo extends JFrame implements ActionListener
11  {
12      public static final int WIDTH = 300;
13      public static final int HEIGHT = 200;

14      private JPanel redPanel;
15      private JPanel whitePanel;
16      private JPanel bluePanel;

17      public static void main(String[] args)
18      {
19          MenuDemo gui = new MenuDemo();
20          gui.setVisible(true);
21      }
```

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A GUI with a Menu (Part 3 of 8)

Display 17.14 **A GUI with a Menu**

```
22      public MenuDemo()
23      {
24          super("Menu Demonstration");
25          setSize(WIDTH, HEIGHT);
26          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
27          setLayout(new GridLayout(1, 3));

28          redPanel = new JPanel();
29          redPanel.setBackground(Color.LIGHT_GRAY);
30          add(redPanel);

31          whitePanel = new JPanel();
32          whitePanel.setBackground(Color.LIGHT_GRAY);
33          add(whitePanel);
```

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A GUI with a Menu (Part 4 of 8)

Display 17.14 **A GUI with a Menu**

```
34          bluePanel = new JPanel();
35          bluePanel.setBackground(Color.LIGHT_GRAY);
36          add(bluePanel);

37          JMenu colorMenu = new JMenu("Add Colors");

38          JMenuItem redChoice = new JMenuItem("Red");
39          redChoice.addActionListener(this);
40          colorMenu.add(redChoice);

41          JMenuItem whiteChoice = new JMenuItem("White");
42          whiteChoice.addActionListener(this);
43          colorMenu.add(whiteChoice);
```

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A GUI with a Menu (Part 5 of 8)

Display 17.14 **A GUI with a Menu**

```
44          JMenuItem blueChoice = new JMenuItem("Blue");
45          blueChoice.addActionListener(this);
46          colorMenu.add(blueChoice);

47          JMenuBar bar = new JMenuBar();
48          bar.add(colorMenu);
49          setJMenuBar(bar);
50      }
```

The definition of **actionPerformed** *is identical to the definition given in Display 17.11 for a similar GUI using buttons instead of menu items.*

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A GUI with a Menu (Part 6 of 8)

Display 17.14 **A GUI with a Menu**

```
51     public void actionPerformed(ActionEvent e)
52     {
53         String buttonString = e.getActionCommand();

54         if (buttonString.equals("Red"))
55             redPanel.setBackground(Color.RED);
56         else if (buttonString.equals("White"))
57             whitePanel.setBackground(Color.WHITE);
58         else if (buttonString.equals("Blue"))
59             bluePanel.setBackground(Color.BLUE);
60         else
61             System.out.println("Unexpected error.");
62     }
63 }
```

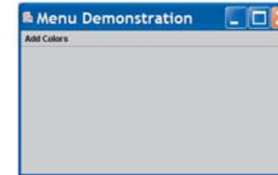
(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

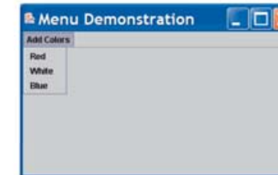
A GUI with a Menu (Part 7 of 8)

Display 17.14 **A GUI with a Menu**

Resulting GUI



Resulting GUI (after clicking Add Colors in the menu bar)



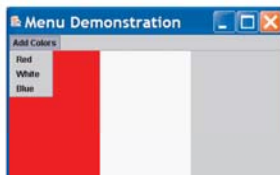
(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

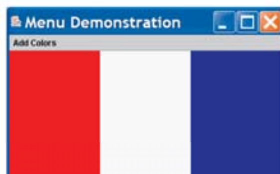
A GUI with a Menu (Part 8 of 8)

Display 17.14 **A GUI with a Menu**

Resulting GUI (after choosing Red and White on the menu)



Resulting GUI (after choosing all the colors on the menu)



Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

The `AbstractButton` and `Dimension` Classes

- The classes `JButton` and `JMenuItem` are derived classes of the abstract class named `AbstractButton`
  - All of their basic properties and methods are inherited from the class `AbstractButton`
- Objects of the `Dimension` class are used with buttons, menu items, and other objects to specify a size
  - The `Dimension` class is in the package `java.awt`
    `Dimension(int width, int height)`
  - Note: `width` and `height` parameters are in pixels

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

The `setActionCommand` Method

- When a user clicks a button or menu item, an event is fired that normally goes to one or more action listeners
  - The action event becomes an argument to an `actionPerformed` method
  - This action event includes a `String` instance variable called the *action command* for the button or menu item
  - The default value for this string is the string written on the button or the menu item
  - This string can be retrieved with the `getActionCommand` method

```
e.getActionCommand()
```

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

The `setActionCommand` Method

- The `setActionCommand` method can be used to change the action command for a component
  - This is especially useful when two or more buttons or menu items have the same default action command strings

```
JButton nextButton = new JButton("Next");
nextButton.setActionCommand("Next Button");

JMenuItem choose = new JMenuItem("Next");
choose.setActionCommand("Next Menu Item");
```

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Some Methods in the Class `AbstractButton` (Part 1 of 3)

Display 17.15 **Some Methods in the Class** `AbstractButton`

The abstract class `AbstractButton` is in the `javax.swing` package.
All of these methods are inherited by both of the classes `JButton` and `JMenuItem`.

`public void setBackground(Color theColor)`

Sets the background color of this component.

`public void addActionListener(ActionListener listener)`

Adds an `ActionListener`.

`public void removeActionListener(ActionListener listener)`

Removes an `ActionListener`.

`public void setActionCommand(String actionCommand)`

Sets the action command.

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Some Methods in the Class `AbstractButton` (Part 2 of 3)

Display 17.15 **Some Methods in the Class** `AbstractButton`

`public String getActionCommand()`

Returns the action command for this component.

`public void setText(String text)`

Makes text the only text on this component.

`public String getText()`

Returns the text written on the component, such as the text on a button or the string for a menu item.

`public void setPreferredSize(Dimension preferredSize)`

Sets the preferred size of the button or label. Note that this is only a suggestion to the layout manager. The layout manager is not required to use the preferred size. The following special case will work for most simple situations. The `int` values give the width and height in pixels.

```
public void setPreferredSize(
                   new Dimension(int width, int height))
```

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Some Methods in the Class `AbstractButton` (Part 3 of 3)

Display 17.15 **Some Methods in the Class AbstractButton**

```
public void setMaximumSize(Dimension maximumSize)
```

Sets the maximum size of the button or label. Note that this is only a suggestion to the layout manager. The layout manager is not required to respect this maximum size. The following special case will work for most simple situations. The int values give the width and height in pixels.

```
public void setMaximumSize(
                    new Dimension(int width, int height))
```

```
public void setMinimumSize(Dimension minimumSize)
```

Sets the minimum size of the button or label. Note that this is only a suggestion to the layout manager. The layout manager is not required to respect this minimum size.
Although we do not discuss the Dimension class, the following special case is intuitively clear and will work for most simple situations. The int values give the width and height in pixels.

```
public void setMinimumSize(
                    new Dimension(int width, int height))
```

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Listeners as Inner Classes

- Often, instead of having one action listener object deal with all the action events in a GUI, a separate **`ActionListener`** class is created for each button or menu item
  - Each button or menu item has its own unique action listener
  - There is then no need for a multiway if-else statement
- When this approach is used, each class is usually made a private inner class

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Listeners as Inner Classes (Part 1 of 6)

Display 17.16 **Listeners as Inner Classes**

```
    <Import statements are the same as in Display 17.14.>

1  public class InnerListenersDemo extends JFrame
2  {
3      public static final int WIDTH = 300;
4      public static final int HEIGHT = 200;

5      private JPanel redPanel;
6      private JPanel whitePanel;
7      private JPanel bluePanel;
```

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Listeners as Inner Classes (Part 2 of 6)

Display 17.16 **Listeners as Inner Classes**

```
8      private class RedListener implements ActionListener
9      {
10         public void actionPerformed(ActionEvent e)
11         {
12             redPanel.setBackground(Color.RED);
13         }
14     } //End of RedListener inner class

15     private class WhiteListener implements ActionListener
16     {
17         public void actionPerformed(ActionEvent e)
18         {
19             whitePanel.setBackground(Color.WHITE);
20         }
21     } //End of WhiteListener inner class
```

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Listeners as Inner Classes (Part 3 of 6)

Display 17.16 **Listeners as Inner Classes**

```
22     private class BlueListener implements ActionListener
23     {
24         public void actionPerformed(ActionEvent e)
25         {
26             bluePanel.setBackground(Color.BLUE);
27         }
28     } //End of BlueListener inner class

29     public static void main(String[] args)
30     {
31         InnerListenersDemo gui = new InnerListenersDemo();
32         gui.setVisible(true);
33     }
```

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Listeners as Inner Classes (Part 4 of 6)

Display 17.16 **Listeners as Inner Classes**

The resulting GUI is the same as in Display 17.14.

```
34     public InnerListenersDemo()
35     {
36         super("Menu Demonstration");
37         setSize(WIDTH, HEIGHT);
38         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
39         setLayout(new GridLayout(1, 3));

40         redPanel = new JPanel();
41         redPanel.setBackground(Color.LIGHT_GRAY);
42         add(redPanel);

43         whitePanel = new JPanel();
44         whitePanel.setBackground(Color.LIGHT_GRAY);
45         add(whitePanel);
```

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Listeners as Inner Classes (Part 5 of 6)

Display 17.16 **Listeners as Inner Classes**

```
46         bluePanel = new JPanel();
47         bluePanel.setBackground(Color.LIGHT_GRAY);
48         add(bluePanel);

49         JMenu colorMenu = new JMenu("Add Colors");

50         JMenuItem redChoice = new JMenuItem("Red");
51         redChoice.addActionListener(new RedListener());
52         colorMenu.add(redChoice);
```

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Listeners as Inner Classes (Part 6 of 6)

Display 17.16 **Listeners as Inner Classes**

```
53         JMenuItem whiteChoice = new JMenuItem("White");
54         whiteChoice.addActionListener(new WhiteListener());
55         colorMenu.add(whiteChoice);

56         JMenuItem blueChoice = new JMenuItem("Blue");
57         blueChoice.addActionListener(new BlueListener());
58         colorMenu.add(blueChoice);

59         JMenuBar bar = new JMenuBar();
60         bar.add(colorMenu);
61         setJMenuBar(bar);
62     }

63 }
```

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Text Fields

- A *text field* is an object of the class `JTextField`
  - It is displayed as a field that allows the user to enter a single line of text

```
private JTextField name;
. . .
name = new JTextField(NUMBER_OF_CHAR);
```

  - In the text field above, at least `NUMBER_OF_CHAR` characters can be visible

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Text Fields

- There is also a constructor with one additional `String` parameter for displaying an initial `String` in the text field

```
JTextField name = new JTextField(
               "Enter name here.", 30);
```

- A Swing GUI can read the text in a text field using the `getText` method

```
String inputString = name.getText();
```

- The method `setText` can be used to display a new text string in a text field

```
name.setText("This is some output");
```

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A Text Field (Part 1 of 7)

Display 17.17 **A Text Field**

```
1  import javax.swing.JFrame;
2  import javax.swing.JTextField;
3  import javax.swing.JPanel;
4  import javax.swing.JLabel;
5  import javax.swing.JButton;
6  import java.awt.GridLayout;
7  import java.awt.BorderLayout;
8  import java.awt.FlowLayout;
9  import java.awt.Color;
10 import java.awt.event.ActionListener;
11 import java.awt.event.ActionEvent;
```

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A Text Field (Part 2 of 7)

Display 17.17 **A Text Field**

```
12 public class TextFieldDemo extends JFrame
13                          implements ActionListener
14 {
15     public static final int WIDTH = 400;
16     public static final int HEIGHT = 200;
17     public static final int NUMBER_OF_CHAR = 30;

18     private JTextField name;


19     public static void main(String[] args)
20     {
21         TextFieldDemo gui = new TextFieldDemo();
22         gui.setVisible(true);
23     }
```

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A Text Field (Part 3 of 7)

Display 17.17 **A Text Field**

```
24      public TextFieldDemo()
25      {
26          super("Text Field Demo");
27          setSize(WIDTH, HEIGHT);
28          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
29          setLayout(new GridLayout(2, 1));

30          JPanel namePanel = new JPanel();
31          namePanel.setLayout(new BorderLayout());
32          namePanel.setBackground(Color.WHITE);

33          name = new JTextField(NUMBER_OF_CHAR);
```

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A Text Field (Part 4 of 7)

Display 17.17 **A Text Field**

```
34          namePanel.add(name, BorderLayout.SOUTH);
35          JLabel nameLabel = new JLabel("Enter your name here:");
36          namePanel.add(nameLabel, BorderLayout.CENTER);

37          add(namePanel);

38          JPanel buttonPanel = new JPanel();
39          buttonPanel.setLayout(new FlowLayout());
40          buttonPanel.setBackground(Color.PINK);
41          JButton actionButton = new JButton("Click me");
42          actionButton.addActionListener(this);
43          buttonPanel.add(actionButton);

44          JButton clearButton = new JButton("Clear");
45          clearButton.addActionListener(this);
46          buttonPanel.add(clearButton);
```

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A Text Field (Part 5 of 7)

Display 17.17 **A Text Field**

```
47          add(buttonPanel);
48      }

49      public void actionPerformed(ActionEvent e)
50      {
51          String actionCommand = e.getActionCommand();

52          if (actionCommand.equals("Click me"))
53              name.setText("Hello " + name.getText());
54          else if (actionCommand.equals("Clear"))
55              name.setText("");
56          else
57              name.setText("Unexpected error.");
58      }

59  }
```

This sets the text field equal to the empty string, which makes it blank.

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A Text Field (Part 6 of 7)

Display 17.17 **A Text Field**

Resulting GUI (When program is started and a name entered)

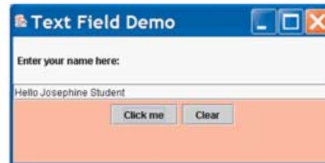
(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A Text Field (Part 7 of 7)

Display 17.17 **A Text Field**

RESULTING GUI (After clicking the "Click me" button)



Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Text Areas

- A *text area* is an object of the class `JTextArea`
  - It is the same as a text field, except that it allows multiple lines
  - Two parameters to the `JTextArea` constructor specify the minimum number of lines, and the minimum number of characters per line that are guaranteed to be visible

```
JTextArea theText = new JTextArea(5,20);
```

  - Another constructor has one addition `String` parameter for the string initially displayed in the text area

```
JTextArea theText = new JTextArea(
               "Enter\ntext here." 5, 20);
```

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Text Areas

- The line-wrapping policy for a `JTextArea` can be set using the method `setLineWrap`
  - The method takes one `boolean` type argument
  - If the argument is `true`, then any additional characters at the end of a line will appear on the following line of the text area
  - If the argument is `false`, the extra characters will remain on the same line and not be visible

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

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Text Fields and Text Areas

- A `JTextField` or `JTextArea` can be set so that it can not be changed by the user

```
theText.setEditable(false);
```

  - This will set `theText` so that it can only be edited by the GUI program, not the user
  - To reverse this, use true instead (this is the default)

```
theText.setEditable(true);
```

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Tip: Labeling a Text Field

- In order to label one or more text fields:
  - Use an object of the class `JLabel`
  - Place the text field(s) and label(s) in a `JPanel`
  - Treat the `JPanel` as a single component

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Numbers of Characters Per Line

- The number of characters per line for a `JTextField` or `JTextArea` object is the number of *em* spaces
- An *em space* is the space needed to hold one uppercase letter `M`
  - The letter `M` is the widest letter in the alphabet
  - A line specified to hold 20 `M`'s will almost always be able to hold more than 20 characters

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Tip: Inputting and Outputting Numbers

- When attempting to input numbers from any Swing GUI, input text must be converted to numbers
  - If the user enters the number `42` in a `JTextField`, the program receives the string `"42"` and must convert it to the integer `42`
- The same thing is true when attempting to output a number
  - In order to output the number `42`, it must first be converted to the string `"42"`

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

The Class `JTextComponent`

- Both `JTextField` and `JTextArea` are derived classes of the abstract class `JTextComponent`
- Most of their methods are inherited from `JTextComponent` and have the same meanings
  - Except for some minor redefinitions to account for having just one line or multiple lines

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Some Methods in the Class `JTextComponent` (Part 1 of 2)

Display 17.18 **Some Methods in the Class `JTextComponent`**

All these methods are inherited by the classes `JTextField` and `JTextArea`.
The abstract class `JTextComponent` is in the package `javax.swing.text`. The classes `JTextField` and `JTextArea` are in the package `javax.swing`.

```
public String getText()
```
Returns the text that is displayed by this text component.

```
public boolean isEditable()
```
Returns `true` if the user can write in this text component. Returns `false` if the user is not allowed to write in this text component.

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Some Methods in the Class `JTextComponent` (Part 2 of 2)

Display 17.18 **Some Methods in the Class `JTextComponent`**

```
public void setBackground(Color theColor)
```
Sets the background color of this text component.

```
public void setEditable(boolean argument)
```
If `argument` is `true`, then the user is allowed to write in the text component. If `argument` is `false`, then the user is not allowed to write in the text component.

```
public void setText(String text)
```
Sets the text that is displayed by this text component to be the specified `text`.

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A Swing Calculator

- A GUI for a simple calculator keeps a running total of numbers
  - The user enters a number in the text field, and then clicks either + or –
  - The number in the text field is then added to or subtracted from the running total, and displayed in the text field
  - This value is kept in the instance variable **`result`**
  - When the GUI is first run, or when the user clicks the **`Reset`** button, the value of **`result`** is set to zero

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A Swing Calculator

- If the user enters a number in an incorrect format, then one of the methods throws a **`NumberFormatException`**
  - The exception is caught in the catch block inside the **`actionPerformed`** method
  - Note that when this exception is thrown, the value of the instance variable **`result`** is not changed

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A Simple Calculator (Part 1 of 11)

Display 17.19 **A Simple Calculator**

```
1  import javax.swing.JFrame;
2  import javax.swing.JTextField;
3  import javax.swing.JPanel;
4  import javax.swing.JLabel;
5  import javax.swing.JButton;
6  import java.awt.BorderLayout;
7  import java.awt.FlowLayout;
8  import java.awt.Color;
9  import java.awt.event.ActionListener;
10 import java.awt.event.ActionEvent;
```

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A Simple Calculator (Part 2 of 11)

Display 17.19 **A Simple Calculator**

```
11 /**
12  A simplified calculator.
13  The only operations are addition and subtraction.
14 */
15 public class Calculator extends JFrame
16                         implements ActionListener
17 {
18     public static final int WIDTH = 400;
19     public static final int HEIGHT = 200;
20     public static final int NUMBER_OF_DIGITS = 30;

21     private JTextField ioField;
22     private double result = 0.0;

23     public static void main(String[] args)
24     {
25         Calculator aCalculator = new Calculator();
26         aCalculator.setVisible(true);
27     }
```

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A Simple Calculator (Part 3 of 11)

Display 17.19 **A Simple Calculator**

```
28     public Calculator()
29     {
30         setTitle("Simplified Calculator");
31         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
32         setSize(WIDTH, HEIGHT);
33         setLayout(new BorderLayout());

34         JPanel textPanel = new JPanel();
35         textPanel.setLayout(new FlowLayout());
36         ioField =
37              new JTextField("Enter numbers here.", NUMBER_OF_DIGITS);
38         ioField.setBackground(Color.WHITE);
39         textPanel.add(ioField);
40         add(textPanel, BorderLayout.NORTH);
```

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A Simple Calculator (Part 4 of 11)

Display 17.19 **A Simple Calculator**

```
41         JPanel buttonPanel = new JPanel();
42         buttonPanel.setBackground(Color.BLUE);
43         buttonPanel.setLayout(new FlowLayout());

44         JButton addButton = new JButton("+");
45         addButton.addActionListener(this);
46         buttonPanel.add(addButton);
47         JButton subtractButton = new JButton("-");
48         subtractButton.addActionListener(this);
49         buttonPanel.add(subtractButton);
50         JButton resetButton = new JButton("Reset");
51         resetButton.addActionListener(this);
52         buttonPanel.add(resetButton);

53        add(buttonPanel, BorderLayout.CENTER);
54     }
```

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A Simple Calculator (Part 5 of 11)

Display 17.19 **A Simple Calculator**

```
55      public void actionPerformed(ActionEvent e)
56      {
57          try
58          {
59              assumingCorrectNumberFormats(e);
60          }
61          catch (NumberFormatException e2)
62          {
63              ioField.setText("Error: Reenter Number.");
64          }
65      }
```

A NumberFormatException does not need to be declared or caught in a catch block.

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A Simple Calculator (Part 6 of 11)

Display 17.19 **A Simple Calculator**

```
66      //Throws NumberFormatException.
67      public void assumingCorrectNumberFormats(ActionEvent e)
68      {
69          String actionCommand = e.getActionCommand();

70          if (actionCommand.equals("+"))
71          {
72              result = result + stringToDouble(ioField.getText());
73              ioField.setText(Double.toString(result));
74          }
75          else if (actionCommand.equals("-"))
76          {
77              result = result - stringToDouble(ioField.getText());
78              ioField.setText(Double.toString(result));
```

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A Simple Calculator (Part 7 of 11)

Display 17.19 **A Simple Calculator**

```
79          }
80          else if (actionCommand.equals("Reset"))
81          {
82              result = 0.0;
83              ioField.setText("0.0");
84          }
85          else
86              ioField.setText("Unexpected error.");
87      }
```

(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A Simple Calculator (Part 8 of 11)

Display 17.19 **A Simple Calculator**

```
88      //Throws NumberFormatException.
89      private static double stringToDouble(String stringObject)
90      {
91          return Double.parseDouble(stringObject.trim());
92      }

93  }
```

(continued)

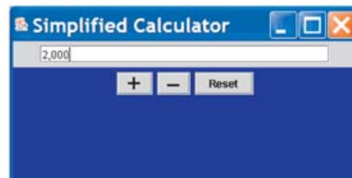
Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

A Simple Calculator (Part 9 of 11)

Display 17.19 **A Simple Calculator**

RESULTING GUI (When started)

RESULTING GUI (After entering 2,000)



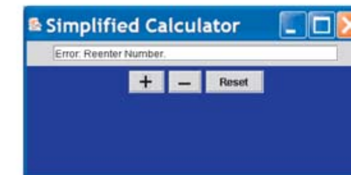
(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

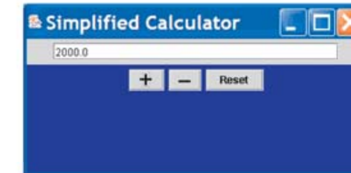
A Simple Calculator (Part 10 of 11)

Display 17.19 **A Simple Calculator**

RESULTING GUI (After clicking +)



RESULTING GUI (After entering 2000 and clicking +)



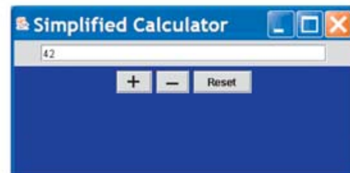
(continued)

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

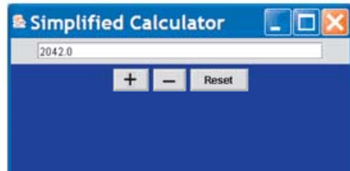
A Simple Calculator (Part 11 of 11)

Display 17.19 **A Simple Calculator**

RESULTING GUI (After entering 42)



RESULTING GUI (After clicking +)



Copyright © 2012 Pearson Addison-Wesley. All rights reserved.

Uncaught Exceptions

- In a Swing program, throwing an uncaught exception does not end the GUI
  - However, it may leave it in an unpredictable state
- It is always best to catch any exception that is thrown even if all the catch block does is output an error message, or ask the user to reenter some input

Copyright © 2012 Pearson Addison-Wesley. All rights reserved.