CHAPTER

12

GUI Basics

Objectives

- To distinguish between Swing and AWT (§12.2).
- To describe the Java GUI API hierarchy (§12.3).
- To create user interfaces using frames, panels, and simple GUI components (§12.4).
- To understand the role of layout managers and use the `FlowLayout`, `GridLayout`, and `BorderLayout` managers to lay out components in a container (§12.5).
- To use `JPanel` to group components in a subcontainer (§12.6).
- To create objects for colors using the `Color` class (§12.7).
- To create objects for fonts using the `Font` class (§12.8).
- To apply common features such as borders, tool tips, fonts, and colors on Swing components (§12.9).
- To decorate the border of GUI components (§12.9).
- To create image icons using the `ImageIcon` class (§12.10).
- To create and use buttons using the `JButton` class (§12.11).
- To create and use check boxes using the `JCheckBox` class (§12.12).
- To create and use radio buttons using the `JRadioButton` class (§12.13).
- To create and use labels using the `JLabel` class (§12.14).
- To create and use text fields using the `JTextField` class (§12.15).

12.1 Introduction

Key Point

Java GUI is an excellent pedagogical tool for learning object-oriented programming.

The design of the API for Java GUI programming is an excellent example of how the object-oriented principle is applied. This chapter serves two purposes. First, it presents the basics of Java GUI programming. Second, it uses GUI to demonstrate OOP. Specifically, this chapter introduces the framework of the Java GUI API and discusses GUI components and their relationships, containers and layout managers, colors, fonts, borders, image icons, and tool tips. It also introduces some of the most frequently used GUI components.

12.2 Swing vs. AWT

Key Point

AWT GUI components are replaced by more versatile and stable Swing GUI components.

We used simple GUI examples to demonstrate OOP in Section 8.6.3, Displaying GUI Components. We used the GUI components such as `JButton`, `JLabel`, `JTextField`, `JRadioButton`, and `JComboBox`. Why do the GUI component classes have the prefix *J*? Instead of `JButton`, why not name it simply `Button`? In fact, there is a class already named `Button` in the `java.awt` package.

AWT

When Java was introduced, the GUI classes were bundled in a library known as the *Abstract Windows Toolkit (AWT)*. AWT is fine for developing simple graphical user interfaces, but not for developing comprehensive GUI projects. In addition, AWT is prone to platform-specific bugs. The AWT user-interface components were replaced by a more robust, versatile, and flexible library known as *Swing components*. Swing components are painted directly on canvases using Java code, except for components that are subclasses of `java.awt.Window` or `java.awt.Panel`, which must be drawn using native GUI on a specific platform. Swing components depend less on the target platform and use less of the native GUI resource. For this reason, Swing components that don't rely on native GUI are referred to as *lightweight components*, and AWT components are referred to as *heavyweight components*.

Swing components

lightweight component

heavyweight component

why prefix J?

To distinguish new Swing component classes from their AWT counterparts, the Swing GUI component classes are named with a prefixed *J*. Although AWT components are still supported in Java, it is better to learn how to program using Swing components, because the AWT user-interface components will eventually fade away. This book uses Swing GUI components exclusively.

Check Point

MyProgrammingLab™

12.1 Why are the Swing GUI classes named with the prefix *J*?

12.2 Explain the difference between AWT GUI components and Swing GUI components.

12.3 The Java GUI API

Key Point

The GUI API contains classes that can be classified into three groups: component classes, container classes, *and* helper classes.

The hierarchical relationships of the Java GUI API are shown in Figure 12.1. Recall that the triangular arrow denotes the inheritance relationship, the diamond denotes the composition relationship, and the filled diamond denotes the exclusive composition relationship. The object composition relationship was introduced in Section 10.7.

component class

container class

helper class

The subclasses of `Component` are called *component classes* for creating the user interface. The classes, such as `JFrame`, `JPanel`, and `JApplet`, are called *container classes* used to contain other components. The classes, such as `Graphics`, `Color`, `Font`, `FontMetrics`, and `Dimension`, are called *helper classes* used to support GUI components.

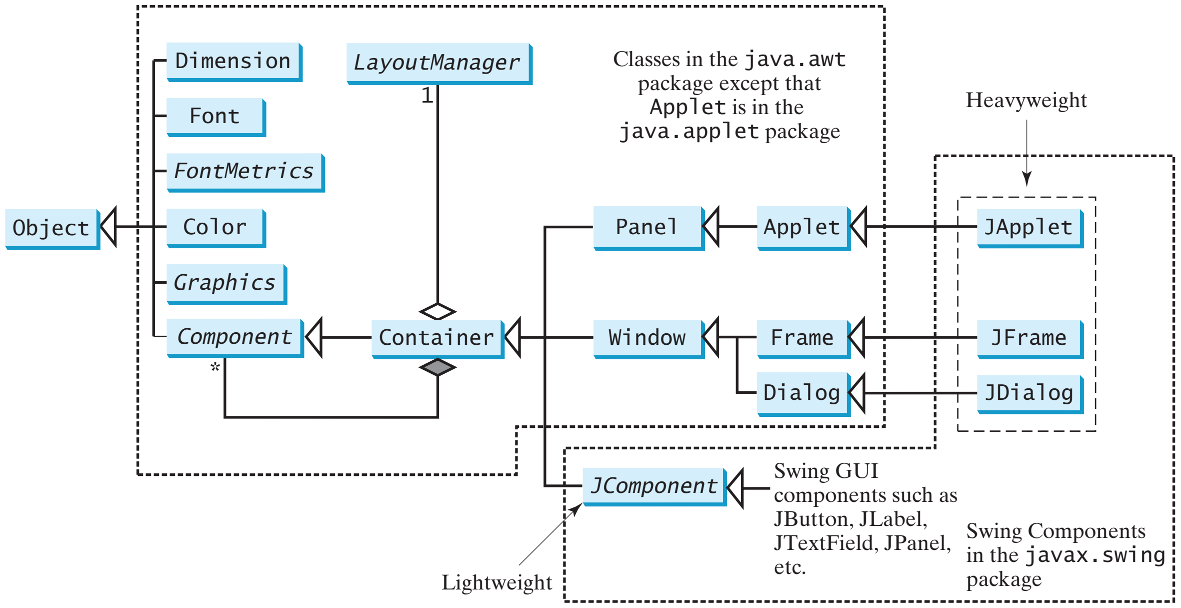

Dimension
LayoutManager
1
Classes in the java.awt package except that Applet is in the java.applet package
Font
FontMetrics
Heavyweight
Object
Color
Panel
Applet
JApplet
Graphics
Component
Container
Window
Frame
JFrame
*
Dialog
JDialog
JComponent
Swing GUI components such as JButton, JLabel, JTextField, JPanel, etc.
Swing Components in the javax.swing package
Lightweight

FIGURE 12.1 Java GUI programming utilizes the classes shown in this hierarchical diagram.

Note
The JFrame, JApplet, JDialog, and JComponent classes and their subclasses are grouped in the javax.swing package. Applet is in the java.applet class. All the other classes in Figure 12.1 are grouped in the java.awt package.

12.3.1 Component Classes

An instance of Component can be displayed on the screen. Component is the root class of all the user-interface classes including container classes, and JComponent is the root class of all the lightweight Swing components. Both Component and JComponent are abstract classes (abstract classes will be introduced in Chapter 15). For now, all you need to know is that abstract classes are the same as classes except that you cannot create instances using the new operator. For example, you cannot use new JComponent() to create an instance of JComponent. However, you can use the constructors of concrete subclasses of JComponent to create JComponent instances. It is important to become familiar with the class inheritance hierarchy. For example, the following statements all display true:

abstract class

```
JButton jbtOK = new JButton("OK");
System.out.println(jbtOK instanceof JButton);
System.out.println(jbtOK instanceof JComponent);
System.out.println(jbtOK instanceof Container);
System.out.println(jbtOK instanceof Component);
System.out.println(jbtOK instanceof Object);
```

12.3.2 Container Classes

An instance of Container can hold instances of Component. A container is called a *top-level container* if it can be displayed without being embedded in another container. Window, Frame, Dialog, JFrame, and JDialog are top-level containers. Window, Panel, Applet, Frame, and Dialog are the container classes for AWT components. To work with Swing components, use Container, JFrame, JDialog, JApplet, and JPanel, as described in Table 12.1.

top-level container

Table 12.1 GUI Container Classes

Container Class	*Description*
`java.awt.Container`	is used to hold components. Frames, panels, and applets are its subclasses.
`javax.swing.JFrame`	is a top-level container for holding other Swing user-interface components in Java GUI applications.
`javax.swing.JPanel`	is an invisible container for grouping user-interface components. Panels can be nested. You can place panels inside another panel. `JPanel` is also often used as a canvas to draw graphics.
`javax.swing.JApplet`	is a base class for creating a Java applet using Swing components.
`javax.swing.JDialog`	is a popup window generally used as a temporary window to receive additional information from the user or to provide notification to the user.

12.3.3 GUI Helper Classes

The helper classes, such as `Graphics`, `Color`, `Font`, `FontMetrics`, `Dimension`, and `LayoutManager`, are not subclasses of `Component`. They are used to describe the properties of GUI components, such as graphics context, colors, fonts, and dimension, as described in Table 12.2.

Table 12.2 GUI Helper Classes

Helper Class	*Description*
`java.awt.Graphics`	is an abstract class that provides the methods for drawing strings, lines, and simple shapes.
`java.awt.Color`	deals with the colors of GUI components. For example, you can specify background or foreground colors in components like `JFrame` and `JPanel`, or you can specify colors of lines, shapes, and strings in drawings.
`java.awt.Font`	specifies fonts for the text and drawings on GUI components. For example, you can specify the font type (e.g., SansSerif), style (e.g., bold), and size (e.g., 24 points) for the text on a button.
`java.awt.FontMetrics`	is an abstract class used to get the properties of the fonts.
`java.awt.Dimension`	encapsulates the width and height of a component (in integer precision) in a single object.
`java.awt.LayoutManager`	specifies how components are arranged in a container.

Note

The helper classes are in the `java.awt` package. The Swing components do not replace all the classes in the AWT, only the AWT GUI component classes (e.g., `Button`, `TextField`, `TextArea`). The AWT helper classes are still useful in GUI programming.

Check Point
MyProgrammingLab™

12.3 Which class is the root of the Java GUI component classes? Is a container class a subclass of `Component`? Which class is the root of the Swing GUI component classes?

12.4 Which of the following statements have syntax errors?

```
Component c1 = new Component();
JComponent c2 = new JComponent();
Component c3 = new JButton();
JComponent c4 = new JButton();
```

12.4 Frames

A frame is a window for holding other GUI components.

Key Point

To create a user interface, you need to create either a frame or an applet to hold the user-interface components. This section introduces frames. Creating Java applets will be introduced in Chapter 18.

12.4.1 Creating a Frame

To create a frame, use the **JFrame** class, as shown in Figure 12.2.

javax.swing.JFrame	
+JFrame()	Creates a default frame with no title.
+JFrame(title: String)	Creates a frame with the specified title.
+setSize(width: int, height: int): void	Sets the size of the frame.
+setLocation(x: int, y: int): void	Sets the upper-left-corner location of the frame.
+setVisible(visible: boolean): void	Sets true to display the frame.
+setDefaultCloseOperation(mode: int): void	Specifies the operation when the frame is closed.
+setLocationRelativeTo(c: Component): void	Sets the location of the frame relative to the specified component. If the component is null, the frame is centered on the screen.
+pack(): void	Automatically sets the frame size to hold the components in the frame.

FIGURE 12.2 The ***JFrame*** class is used to create a window for displaying GUI components.

The program in Listing 12.1 creates a frame.

LISTING 12.1 MyFrame.java

```java
 1  import javax.swing.JFrame;
 2
 3  public class MyFrame {
 4    public static void main(String[] args) {
 5      JFrame frame = new JFrame("MyFrame"); // Create a frame
 6      frame.setSize(400, 300); // Set the frame size
 7      frame.setLocationRelativeTo(null); // Center a frame
 8      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 9      frame.setVisible(true); // Display the frame
10    }
11  }
```

import package
create frame
set size
center frame
close upon exit
display the frame

The frame is not displayed until the **frame.setVisible(true)** method is invoked. **frame.setSize(400, 300)** specifies that the frame is **400** pixels wide and **300** pixels high. If the **setSize** method is not used, the frame will be sized to display just the title bar. Since the **setSize** and **setVisible** methods are both defined in the **Component** class, they are inherited by the **JFrame** class. Later you will see that these methods are also useful in many other subclasses of **Component**.

When you run the **MyFrame** program, a window will be displayed on the screen (see Figure 12.3a).

Invoking **setLocationRelativeTo(null)** (line 7) centers the frame on the screen. Invoking **setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)** (line 8) tells the program to terminate when the frame is closed. If this statement is not used, the program does not terminate when the frame is closed. In that case, you have to stop the program by pressing *Ctrl+C* at the DOS prompt window in Windows or stop the process by using the kill command

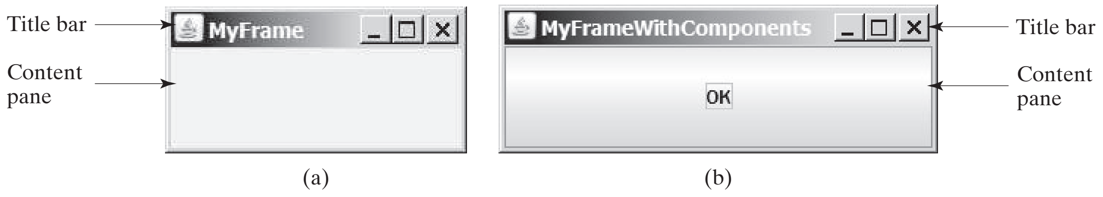


FIGURE 12.3 (a) The program creates and displays a frame with the title `MyFrame`. (b) An OK button is added to the frame.

in UNIX. If you run the program from an IDE such as Eclipse or NetBeans, you need to click the red *Terminate* button in the Console pane to stop the program.

pixel and resolution

Note

Recall that a pixel is the smallest unit of space available for drawing on the screen. You can think of a pixel as a small rectangle and think of the screen as paved with pixels. The *resolution* specifies the number of pixels in horizontal and vertical dimensions of the screen. The more pixels the screen has, the higher the screen's resolution. The higher the resolution, the finer the detail you can see.

`setSize` before centering

Note

You should invoke the `setSize(w, h)` method before invoking `setLocationRelativeTo(null)` to center the frame.

12.4.2 Adding Components to a Frame

The frame shown in Figure 12.3a is empty. Using the `add` method, you can add components to the frame, as shown in Listing 12.2.

LISTING 12.2 MyFrameWithComponents.java

create a button
add to frame
set size
exit upon closing window
center the frame
set visible

```java
import javax.swing.*;

public class MyFrameWithComponents {
  public static void main(String[] args) {
    JFrame frame = new JFrame("MyFrameWithComponents");

    // Add a button to the frame
    JButton jbtOK = new JButton("OK");
    frame.add(jbtOK);

    frame.setSize(400, 300);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setVisible(true);
  }
}
```

Each `JFrame` contains a *content pane*, which is an instance of `java.awt.Container`. The GUI components such as buttons are placed in the content pane in a frame. In earlier versions of Java, you had to use the `getContentPane` method in the `JFrame` class to return the content pane of the frame, then invoke the content pane's `add` method to place a component in the content pane, as follows:

```java
java.awt.Container container = frame.getContentPane();
container.add(jbtOK);
```

This was cumbersome. Versions of Java since Java 5 allow you to place components in the content pane by invoking a frame's **add** method, as follows:

```
frame.add(jbtOK);
```

This feature is called *content-pane delegation*. Strictly speaking, a component is added to the content pane of a frame. For simplicity, we say that a component is added to a frame.

content-pane delegation

In Listing 12.2, an object of **JButton** was created using **new JButton("OK")**, and this object was added to the content pane of the frame (line 9).

The **add(Component comp)** method defined in the **Container** class adds an instance of **Component** to the container. Since **JButton** is a subclass of **Component**, an instance of **JButton** is also an instance of **Component**. To remove a component from a container, use the **remove** method. The following statement removes the button from the container:

```
container.remove(jbtOK);
```

When you run the program **MyFrameWithComponents**, the window will be displayed as in Figure 12.3b. The button is always centered in the frame and occupies the entire frame no matter how you resize it. This is because components are put in the frame by the content pane's layout manager, and the default layout manager for the content pane places the button in the center. In the next section, you will use several different layout managers to place components in the desired locations.

12.5 How do you create a frame? How do you set the size for a frame? How do you add components to a frame? What would happen if the statements **frame.setSize(400, 300)** and **frame.setVisible(true)** were swapped in Listing 12.2?

Check Point
MyProgrammingLab™

12.5 Layout Managers

Each container contains a layout manager, which is an object responsible for laying out the GUI components in the container.

Key Point

In many other window systems, the user-interface components are arranged by using hard-coded pixel measurements. For example, when placing a button at location (**10, 10**) in a window using hard-coded pixel measurements, the user interface might look fine on one system but be unusable on another. Java's *layout managers* provide a level of abstraction that automatically maps your user interface on all window systems.

layout manager

The Java GUI components are placed in containers, where they are arranged by the container's layout manager. In the preceding program, you did not specify where to place the *OK* button in the frame, but Java knows where to place it, because the layout manager works behind the scenes to place components in the correct locations. A layout manager is created using a layout manager class.

Layout managers are set in containers using the **setLayout(aLayoutManager)** method. For example, you can use the following statements to create an instance of ***X*Layout** and set it in a container:

```
LayoutManager layoutManager = new XLayout();
container.setLayout(layoutManager);
```

This section introduces three basic layout managers: **FlowLayout**, **GridLayout**, and **BorderLayout**.

12.5.1 FlowLayout

VideoNote
Use FlowLayout

FlowLayout is the simplest layout manager. The components are arranged in the container from left to right in the order in which they were added. When one row is filled, a new row is started. You can specify the way the components are aligned by using one of three constants: FlowLayout.RIGHT, FlowLayout.CENTER, or FlowLayout.LEFT. You can also specify the gap between components in pixels. The class diagram for FlowLayout is shown in Figure 12.4.

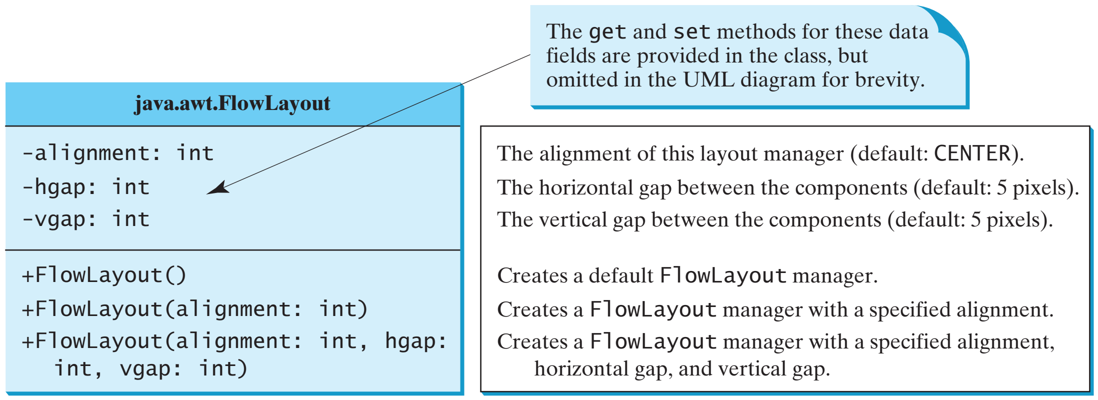

The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.
java.awt.FlowLayout
-alignment: int
-hgap: int
-vgap: int
+FlowLayout()
+FlowLayout(alignment: int)
+FlowLayout(alignment: int, hgap: int, vgap: int)
The alignment of this layout manager (default: CENTER).
The horizontal gap between the components (default: 5 pixels).
The vertical gap between the components (default: 5 pixels).
Creates a default FlowLayout manager.
Creates a FlowLayout manager with a specified alignment.
Creates a FlowLayout manager with a specified alignment, horizontal gap, and vertical gap.

FIGURE 12.4 FlowLayout lays out components row by row.

Listing 12.3 gives a program that demonstrates flow layout. The program adds three labels and text fields to the frame with a FlowLayout manager, as shown in Figure 12.5.

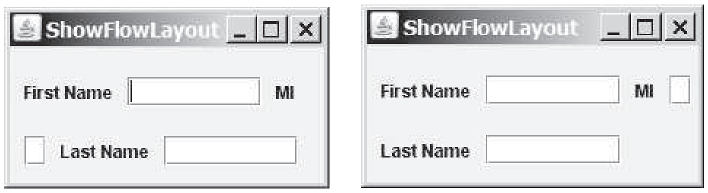
LISTING 12.3 ShowFlowLayout.java

extends JFrame
set layout
add label
add text field

```java
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JFrame;
import java.awt.FlowLayout;

public class ShowFlowLayout extends JFrame {
  public ShowFlowLayout() {
    // Set FlowLayout, aligned left with horizontal gap 10
    // and vertical gap 20 between components
    setLayout(new FlowLayout(FlowLayout.LEFT, 10, 20));

    // Add labels and text fields to the frame
    add(new JLabel("First Name"));
    add(new JTextField(8));
    add(new JLabel("MI"));
    add(new JTextField(1));
    add(new JLabel("Last Name"));
    add(new JTextField(8));
  }

  /** Main method */
  public static void main(String[] args) {
    ShowFlowLayout frame = new ShowFlowLayout();
    frame.setTitle("ShowFlowLayout");
    frame.setSize(200, 200);
26      frame.setLocationRelativeTo(null); // Center the frame
27      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
28      frame.setVisible(true);
29    }
30  }
```

create frame

set visible



(a) (b)

FIGURE 12.5 The components are added by the `FlowLayout` manager to fill in the rows in the container one after another.

This example creates a program using a style different from the programs in the preceding section, where frames were created using the `JFrame` class. This example creates a class named `ShowFlowLayout` that extends the `JFrame` class (line 6). The `main` method in this program creates an instance of `ShowFlowLayout` (line 23). The constructor of `ShowFlowLayout` constructs and places the components in the frame. This is the preferred style of creating GUI applications—for three reasons:

- Creating a GUI application means creating a frame, so it is natural to define a frame to extend `JFrame`.
- The frame may be further extended to add new components or functions.
- The class can be easily reused. For example, you can create multiple frames by creating multiple instances of the class.

Using one style consistently makes programs easy to read. From now on, most of the GUI main classes will extend the `JFrame` class. The constructor of the main class constructs the user interface. The `main` method creates an instance of the main class and then displays the frame.

Will the program work if line 23 is replaced by the following code?

```
JFrame frame = new ShowFlowLayout();
```

Yes. The program will still work because `ShowFlowLayout` is a subclass of `JFrame` and the methods `setTitle`, `setSize`, `setLocationRelativeTo`, `setDefaultCloseOperation`, and `setVisible` (lines 24–28) are all available in the `JFrame` class.

In this example, the `FlowLayout` manager is used to place components in a frame. If you resize the frame, the components are automatically rearranged to fit. In Figure 12.5a, the first row has three components, but in Figure 12.5b, the first row has four components, because the width has been increased.

If you replace the `setLayout` statement (line 10) with `setLayout(new FlowLayout(FlowLayout.RIGHT, 0, 0))`, all the rows of buttons will be right aligned with no gaps.

An anonymous `FlowLayout` object was created in the statement (line 10):

```
setLayout(new FlowLayout(FlowLayout.LEFT, 10, 20));
```

which is equivalent to:

```
FlowLayout layout = new FlowLayout(FlowLayout.LEFT, 10, 20);
setLayout(layout);
```

This code creates an explicit reference to the object `layout` of the `FlowLayout` class. The explicit reference is not necessary, because the object is not directly referenced in the `ShowFlowLayout` class.

Suppose you add the same button to the frame ten times; will ten buttons appear in the frame? No, a GUI component such as a button can be added to only one container and only once in a container. Adding a button to a container multiple times is the same as adding it once.

Note

GUI components cannot be shared by containers, because only one GUI component can appear in only one container at a time. Therefore, the relationship between a component and a container is the composition denoted by a filled diamond, as shown in Figure 12.1.

Caution

Do not forget to put the `new` operator before a layout manager class when setting a layout style—for example, `setLayout(new FlowLayout())`.

Note

The constructor `ShowFlowLayout()` does not explicitly invoke the constructor `JFrame()`, but the constructor `JFrame()` is invoked implicitly. See Section 11.3.2, Constructor Chaining.

12.5.2 `GridLayout`

The `GridLayout` manager arranges components in a grid (matrix) formation. The components are placed in the grid from left to right, starting with the first row, then the second, and so on, in the order in which they are added. The class diagram for `GridLayout` is shown in Figure 12.6.

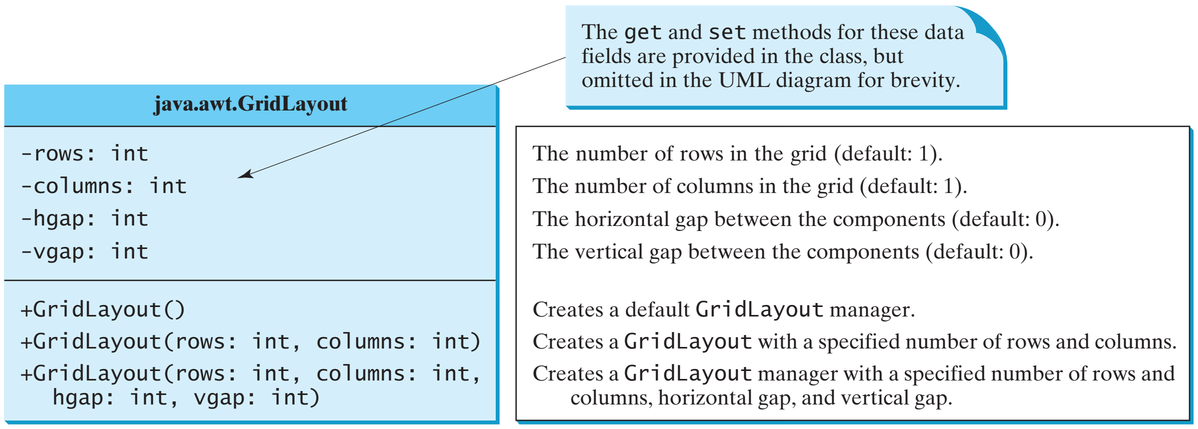

The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.
java.awt.GridLayout
-rows: int
-columns: int
-hgap: int
-vgap: int
+GridLayout()
+GridLayout(rows: int, columns: int)
+GridLayout(rows: int, columns: int, hgap: int, vgap: int)
The number of rows in the grid (default: 1).
The number of columns in the grid (default: 1).
The horizontal gap between the components (default: 0).
The vertical gap between the components (default: 0).
Creates a default GridLayout manager.
Creates a GridLayout with a specified number of rows and columns.
Creates a GridLayout manager with a specified number of rows and columns, horizontal gap, and vertical gap.

FIGURE 12.6 `GridLayout` lays out components in equal-sized cells on a grid.

You can specify the number of rows and columns in the grid. The basic rules are as follows:

- The number of rows or the number of columns can be zero, but not for both. If one is zero and the other is nonzero, the nonzero dimension is fixed, while the zero dimension is determined dynamically by the layout manager. For example, if you specify zero rows and three columns for a grid that has ten components, `GridLayout` creates three fixed columns of four rows, with the last row containing one component. If you specify three rows and zero columns for a grid that has ten components, `GridLayout` creates three fixed rows of four columns, with the last row containing two components.

- If both the number of rows and the number of columns are nonzero, the number of rows is the dominating parameter; that is, the number of rows is fixed, and the layout manager dynamically calculates the number of columns. For example, if you specify three rows and three columns for a grid that has ten components, `GridLayout` creates three fixed rows of four columns, with the last row containing two components.

Listing 12.4 gives a program that demonstrates grid layout. The program is similar to the one in Listing 12.3, except that it adds three labels and three text fields to the frame of `GridLayout` instead of `FlowLayout`, as shown in Figure 12.7.

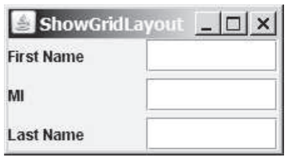


FIGURE 12.7 The `GridLayout` manager divides the container into grids; then the components are added to fill in the cells row by row.

LISTING 12.4 ShowGridLayout.java

```java
 1  import javax.swing.JLabel;
 2  import javax.swing.JTextField;
 3  import javax.swing.JFrame;
 4  import java.awt.GridLayout;
 5
 6  public class ShowGridLayout extends JFrame {
 7    public ShowGridLayout() {
 8      // Set GridLayout, 3 rows, 2 columns, and gaps 5 between
 9      // components horizontally and vertically
10      setLayout(new GridLayout(3, 2, 5, 5));
11
12      // Add labels and text fields to the frame
13      add(new JLabel("First Name"));
14      add(new JTextField(8));
15      add(new JLabel("MI"));
16      add(new JTextField(1));
17      add(new JLabel("Last Name"));
18      add(new JTextField(8));
19    }
20
21    /** Main method */
22    public static void main(String[] args) {
23      ShowGridLayout frame = new ShowGridLayout();
24      frame.setTitle("ShowGridLayout");
25      frame.setSize(200, 125);
26      frame.setLocationRelativeTo(null); // Center the frame
27      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
28      frame.setVisible(true);
29    }
30  }
```

set layout
add label
add text field
create the frame
set visible

If you resize the frame, the layout of the components remains unchanged (i.e., the number of rows and columns does not change, and the gaps don't change either).

All components are given equal size in the container of `GridLayout`.

Replacing the `setLayout` statement (line 10) with `setLayout(new GridLayout(3, 10))` would still yield three rows and *two* columns. The `columns` parameter is ignored because the **rows** parameter is nonzero. The actual number of columns is calculated by the layout manager.

What would happen if the **setLayout** statement (line 10) were replaced with **setLayout(new GridLayout(4, 2))** or with **setLayout(new GridLayout(2, 2))**? Please try it yourself.

Note

In **FlowLayout** and **GridLayout**, the order in which the components are added to the container is important. The order determines the location of the components in the container.

12.5.3 BorderLayout

The **BorderLayout** manager divides a container into five areas: East, South, West, North, and Center. Components are added to a **BorderLayout** by using **add(Component, index)**, where **index** is a constant **BorderLayout.EAST**, **BorderLayout.SOUTH**, **BorderLayout.WEST**, **BorderLayout.NORTH**, or **BorderLayout.CENTER**. The class diagram for **BorderLayout** is shown in Figure 12.8.

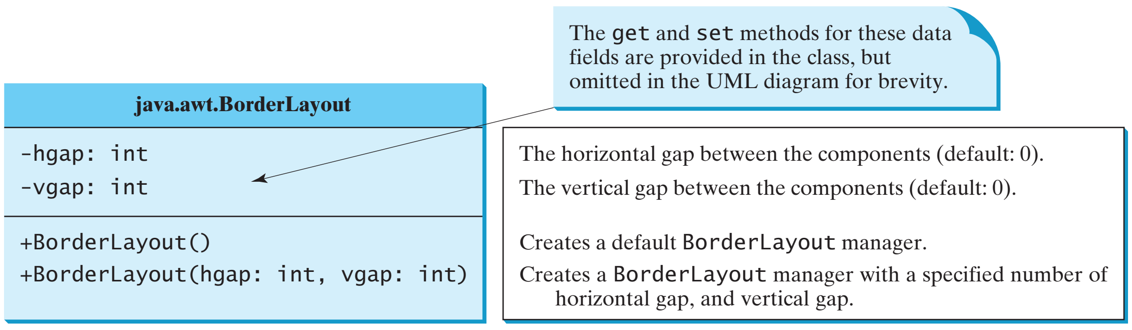

The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.
java.awt.BorderLayout
-hgap: int
-vgap: int
+BorderLayout()
+BorderLayout(hgap: int, vgap: int)
The horizontal gap between the components (default: 0).
The vertical gap between the components (default: 0).
Creates a default BorderLayout manager.
Creates a BorderLayout manager with a specified number of horizontal gap, and vertical gap.

FIGURE 12.8 **BorderLayout** lays out components in five areas.

The components are laid out according to their preferred sizes and their placement in the container. The North and South components can stretch horizontally; the East and West components can stretch vertically; the Center component can stretch both horizontally and vertically to fill any empty space.

Listing 12.5 gives a program that demonstrates border layout. The program adds five buttons labeled **East**, **South**, **West**, **North**, and **Center** to the frame with a **BorderLayout** manager, as shown in Figure 12.9.

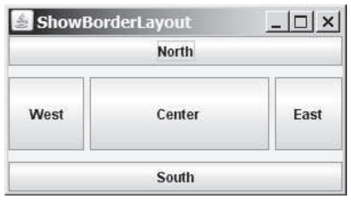


FIGURE 12.9 **BorderLayout** divides the container into five areas, each of which can hold a component.

LISTING 12.5 ShowBorderLayout.java

```java
import javax.swing.JButton;
import javax.swing.JFrame;
import java.awt.BorderLayout;
```

```
 4
 5  public class ShowBorderLayout extends JFrame {
 6    public ShowBorderLayout() {
 7      // Set BorderLayout with horizontal gap 5 and vertical gap 10
 8      setLayout(new BorderLayout(5, 10));
 9
10      // Add buttons to the frame
11      add(new JButton("East"), BorderLayout.EAST);
12      add(new JButton("South"), BorderLayout.SOUTH);
13      add(new JButton("West"), BorderLayout.WEST);
14      add(new JButton("North"), BorderLayout.NORTH);
15      add(new JButton("Center"), BorderLayout.CENTER);
16    }
17
18    /** Main method */
19    public static void main(String[] args) {
20      ShowBorderLayout frame = new ShowBorderLayout();
21      frame.setTitle("ShowBorderLayout");
22      frame.setSize(300, 200);
23      frame.setLocationRelativeTo(null); // Center the frame
24      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
25      frame.setVisible(true);
26    }
27  }
```

set layout

add buttons

create the frame

set visible

The buttons are added to the frame (lines 11–15). Note that the **add** method for **BorderLayout** is different from the one for **FlowLayout** and **GridLayout**. With **BorderLayout**, you specify where to put the components.

It is unnecessary to place components to occupy all the areas. If you remove the East button from the program and rerun it, you will see that the Center button stretches rightward to occupy the East area.

Note

BorderLayout interprets the absence of an index specification as **BorderLayout.CENTER**. For example, **add(component)** is the same as **add(Component, BorderLayout.CENTER)**. If you add two components to a container of **BorderLayout**, as follows,

```
container.add(component1);
container.add(component2);
```

only the last component is displayed.

12.5.4 Properties of Layout Managers

Layout managers have properties that can be changed dynamically.

- **FlowLayout** has **alignment**, **hgap**, and **vgap** properties. You can use the **setAlignment**, **setHgap**, and **setVgap** methods to specify the alignment and the horizontal and vertical gaps.
- **GridLayout** has the **rows**, **columns**, **hgap**, and **vgap** properties. You can use the **setRows**, **setColumns**, **setHgap**, and **setVgap** methods to specify the number of rows, the number of columns, and the horizontal and vertical gaps.
- **BorderLayout** has the **hgap** and **vgap** properties. You can use the **setHgap** and **setVgap** methods to specify the horizontal and vertical gaps.

In the preceding sections an anonymous layout manager is used because the properties of a layout manager do not change once it is created. If you have to change the properties of a layout manager dynamically, the layout manager must be explicitly referenced by a variable. You can then change the properties of the layout manager through the variable. For example, the following code creates a layout manager and sets its properties:

```
// Create a layout manager
FlowLayout flowLayout = new FlowLayout();

// Set layout properties
flowLayout.setAlignment(FlowLayout.RIGHT);
flowLayout.setHgap(10);
flowLayout.setVgap(20);
```

Check Point

MyProgrammingLab™

12.6 Will the program work if `ShowFlowLayout` in line 23 in Listing 12.3 is replaced by `JFrame`?

Will the program work if `ShowGridLayout` in line 23 in Listing 12.4 is replaced by `JFrame`?

Will the program work if `ShowBorderLayout` line 20 in Listing 12.5 is replaced by `JFrame`?

12.7 Why do you need to use layout managers? What is the default layout manager for a frame? How do you add a component to a frame?

12.8 Describe `FlowLayout`. How do you create a `FlowLayout` manager? How do you add a component to a `FlowLayout` container? Is there a limit to the number of components that can be added to a `FlowLayout` container? What are the properties for setting the horizontal and vertical gaps between the components in the container? Can you specify alignment?

12.9 Describe `GridLayout`. How do you create a `GridLayout` manager? How do you add a component to a `GridLayout` container? Is there a limit to the number of components that can be added to a `GridLayout` container? What are the properties for setting the horizontal and vertical gaps between the components in the container?

12.10 Describe `BorderLayout`. How do you create a `BorderLayout` manager? How do you add a component to a `BorderLayout` container? What are the properties for setting the horizontal and vertical gaps between the components in the container?

12.11 The following program is supposed to display a button in a frame, but nothing is displayed. What is the problem?

```
public class Test extends javax.swing.JFrame {
  public Test() {
    add(new javax.swing.JButton("OK"));
  }

  public static void main(String[] args) {
    javax.swing.JFrame frame = new javax.swing.JFrame();
    frame.setSize(100, 200);
    frame.setVisible(true);
  }
}
```

12.6 Using Panels as Subcontainers

Key Point

A container can be placed inside another container. Panels can be used as subcontainers to group GUI components to achieve the desired layout.

VideoNote
Use panels as subcontainers

Suppose that you want to place ten buttons and a text field in a frame. The buttons are placed in grid formation, but the text field is placed on a separate row. It is difficult to achieve the desired look by placing all the components in a single container. With Java GUI programming, you can divide a window into panels. Panels act as subcontainers to group user-interface components. You add the buttons in one panel, then add the panel to the frame.

The Swing version of panel is **JPanel**. You can use **new JPanel()** to create a panel with a default **FlowLayout** manager or **new JPanel(LayoutManager)** to create a panel with the specified layout manager. Use the **add(Component)** method to add a component to the panel. For example, the following code creates a panel and adds a button to it:

```
JPanel p = new JPanel();
p.add(new JButton("OK"));
```

Panels can be placed inside a frame or inside another panel. The following statement places panel **p** in frame **f**:

```
f.add(p);
```

Listing 12.6 gives an example that demonstrates using panels as subcontainers. The program creates a user interface for a microwave oven, as shown in Figure 12.10.

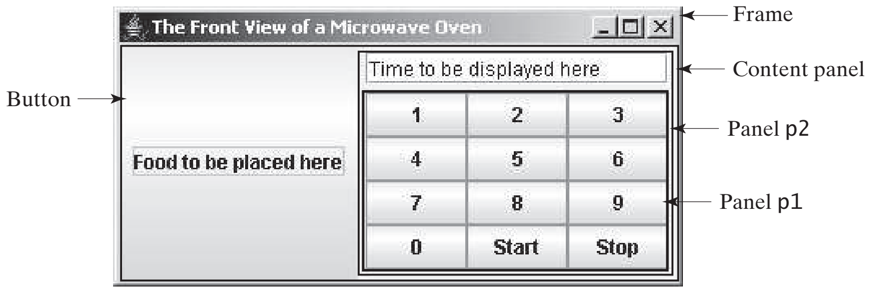


FIGURE 12.10 The program uses panels to organize components.

LISTING 12.6 TestPanels.java

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

public class TestPanels extends JFrame {
  public TestPanels() {
    // Create panel p1 for the buttons and set GridLayout
    JPanel p1 = new JPanel();                                      panel p1
    p1.setLayout(new GridLayout(4, 3));

    // Add buttons to the panel
    for (int i = 1; i <= 9; i++) {
      p1.add(new JButton("" + i));
    }

    p1.add(new JButton("" + 0));
    p1.add(new JButton("Start"));
    p1.add(new JButton("Stop"));

    // Create panel p2 to hold a text field and p1
    JPanel p2 = new JPanel(new BorderLayout());                    panel p2
    p2.add(new JTextField("Time to be displayed here"),
      BorderLayout.NORTH);
    p2.add(p1, BorderLayout.CENTER);

    // Add contents to the frame
    add(p2, BorderLayout.EAST);                                    add p2 to frame
    add(new JButton("Food to be placed here"),
      BorderLayout.CENTER);
  }

31    /** Main method */
32    public static void main(String[] args) {
33      TestPanels frame = new TestPanels();
34      frame.setTitle("The Front View of a Microwave Oven");
35      frame.setSize(400, 250);
36      frame.setLocationRelativeTo(null); // Center the frame
37      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
38      frame.setVisible(true);
39    }
40  }
```

The `setLayout` method is defined in `java.awt.Container`. Since `JPanel` is a subclass of `Container`, you can use `setLayout` to set a new layout manager in the panel (line 8). Lines 7–8 can be replaced by `JPanel p1 = new JPanel(new GridLayout(4, 3))`.

To achieve the desired layout, the program uses panel `p1` of `GridLayout` to group the number buttons, the *Stop* button, and the *Start* button, and panel `p2` of `BorderLayout` to hold a text field in the north and `p1` in the center. The button representing the food is placed in the center of the frame, and `p2` is placed in the east of the frame.

The statement (lines 21–22)

```
p2.add(new JTextField("Time to be displayed here"),
  BorderLayout.NORTH);
```

creates an instance of `JTextField` and adds it to `p2`. `JTextField` is a GUI component that can be used for user input as well as to display values.

superclass `Container`

Note

It is worthwhile to note that the `Container` class is the superclass for GUI component classes, such as `JButton`. Every GUI component is a container. In theory, you could use the `setLayout` method to set the layout in a button and add components to a button, because all the public methods in the `Container` class are inherited by `JButton`, but for practical reasons you should not use buttons as containers.

Check Point

MyProgrammingLab™

12.12 How do you create a panel with a specified layout manager?

12.13 What is the default layout manager for a `JPanel`? How do you add a component to a `JPanel`?

12.14 Can you use the `setTitle` method in a panel? What is the purpose of using a panel?

12.15 Since a GUI component class such as `JButton` is a subclass of `Container`, can you add components to a button?

12.7 The `Color` Class

Key Point

Each GUI component has background and foreground colors. Colors are objects created from the `Color` class.

You can set colors for GUI components by using the `java.awt.Color` class. Colors are made of red, green, and blue components, each represented by an `int` value that describes its intensity, ranging from `0` (darkest shade) to `255` (lightest shade). This is known as the *RGB model*.

You can create a color using the following constructor:

```
public Color(int r, int g, int b);
```

in which `r`, `g`, and `b` specify a color by its red, green, and blue components. For example,

```
Color color = new Color(128, 100, 100);
```

`IllegalArgumentException`

Note

The arguments `r`, `g`, `b` are between `0` and `255`. If a value beyond this range is passed to the argument, an `IllegalArgumentException` will occur.

You can use the `setBackground(Color c)` and `setForeground(Color c)` methods defined in the `java.awt.Component` class to set a component's background and foreground colors. Here is an example of setting the background and foreground of a button:

```
JButton jbtOK = new JButton("OK");
jbtOK.setBackground(color);
jbtOK.setForeground(new Color(100, 1, 1));
```

Alternatively, you can use one of the 13 standard colors (`BLACK`, `BLUE`, `CYAN`, `DARK_GRAY`, `GRAY`, `GREEN`, `LIGHT_GRAY`, `MAGENTA`, `ORANGE`, `PINK`, `RED`, `WHITE`, and `YELLOW`) defined as constants in `java.awt.Color`. The following code, for instance, sets the foreground color of a button to red:

```
jbtOK.setForeground(Color.RED);
```

Check Point
MyProgrammingLab™

12.16 How do you create a color? What is wrong about creating a `Color` using `new Color(400, 200, 300)`? Which of two colors is darker, `new Color(10, 0, 0)` or `new Color(200, 0, 0)`?

12.17 How do you create a `Color` object with a random color?

12.18 How do you set a button object `jbtOK` with blue background?

12.8 The Font Class

Key Point

Each GUI component has the font property. Fonts are objects created from the `Font` class.

You can create a font using the `java.awt.Font` class and set fonts for the components using the `setFont` method in the `Component` class.

The constructor for `Font` is:

```
public Font(String name, int style, int size);
```

You can choose a font name from `SansSerif`, `Serif`, `Monospaced`, `Dialog`, and `DialogInput`, choose a style from `Font.PLAIN` (0), `Font.BOLD` (1), `Font.ITALIC` (2), and `Font.BOLD` + `Font.ITALIC` (3), and specify a font size of any positive integer. For example, the following statements create two fonts and set one font to a button.

```
Font font1 = new Font("SansSerif", Font.BOLD, 16);
Font font2 = new Font("Serif", Font.BOLD + Font.ITALIC, 12);

JButton jbtOK = new JButton("OK");
jbtOK.setFont(font1);
```

Tip

find available fonts

If your system supports other fonts, such as "Times New Roman," you can use the font to create a `Font` object. To find the fonts available on your system, you need to obtain an instance of `java.awt.GraphicsEnvironment` using its static method `getLocalGraphicsEnvironment()`. `GraphicsEnvironment` is an abstract class that describes the graphics environment on a particular system. You can use its `getAllFonts()` method to obtain all the available fonts on the system and its `getAvailableFontFamilyNames()` method to obtain the names of all the available fonts. For example, the following statements print all the available font names in the system:

```
GraphicsEnvironment e =
  GraphicsEnvironment.getLocalGraphicsEnvironment();
String[] fontnames = e.getAvailableFontFamilyNames();

for (int i = 0; i < fontnames.length; i++)
  System.out.println(fontnames[i]);
```

Check Point
MyProgrammingLab™

12.19 How do you create a `Font` object with font name `Courier`, size `20`, and style `bold`?

12.20 How do you find all available fonts on your system?

12.9 Common Features of Swing GUI Components

Key Point

GUI components have common features. They are defined in the superclasses `Component`, `Container`, *and* `JComponent`.

VideoNote
Use Swing common properties

So far in this chapter you have used several GUI components (e.g., `JFrame`, `Container`, `JPanel`, `JButton`, `JLabel`, and `JTextField`). Many more GUI components will be introduced in this book. It is important to understand the common features of Swing GUI components. The `Component` class is the root for all GUI components and containers. All Swing GUI components (except `JFrame`, `JApplet`, and `JDialog`) are subclasses of `JComponent`, as shown in Figure 12.1. Figure 12.11 lists some frequently used methods in `Component`, `Container`, and `JComponent` for manipulating properties such as font, color, mouse cursor, size, tool tip text, and border.

`Component`
`Container`
`JComponent`
tool tip

A *tool tip* is text displayed on a component when you move the mouse onto the component. It is often used to describe the function of a component.

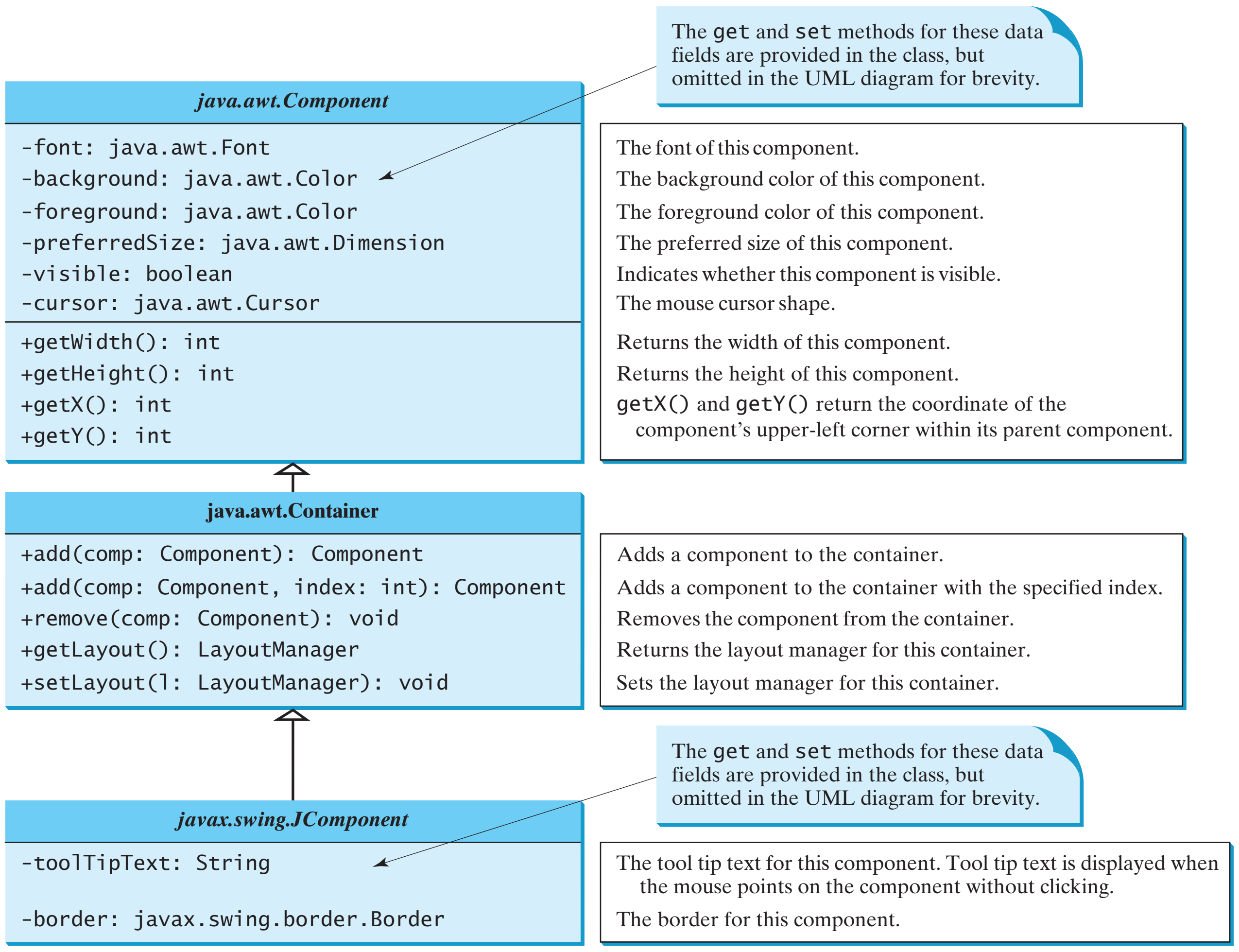

The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.
java.awt.Component
-font: java.awt.Font
-background: java.awt.Color
-foreground: java.awt.Color
-preferredSize: java.awt.Dimension
-visible: boolean
-cursor: java.awt.Cursor
+getWidth(): int
+getHeight(): int
+getX(): int
+getY(): int
The font of this component.
The background color of this component.
The foreground color of this component.
The preferred size of this component.
Indicates whether this component is visible.
The mouse cursor shape.
Returns the width of this component.
Returns the height of this component.
getX() and getY() return the coordinate of the component's upper-left corner within its parent component.
java.awt.Container
+add(comp: Component): Component
+add(comp: Component, index: int): Component
+remove(comp: Component): void
+getLayout(): LayoutManager
+setLayout(l: LayoutManager): void
Adds a component to the container.
Adds a component to the container with the specified index.
Removes the component from the container.
Returns the layout manager for this container.
Sets the layout manager for this container.
The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.
javax.swing.JComponent
-toolTipText: String
-border: javax.swing.border.Border
The tool tip text for this component. Tool tip text is displayed when the mouse points on the component without clicking.
The border for this component.

FIGURE 12.11 All the Swing GUI components inherit the public methods from `Component`, `Container`, and `JComponent`.

You can set a border for any object of the **JComponent** class. Swing has several types of borders. To create a titled border, use **new TitledBorder(String title)**. To create a line border, use **new LineBorder(Color color, int width)**, where **width** specifies the thickness of the line.

border

Listing 12.7 is an example to demonstrate Swing common features. The example creates a panel **p1** to hold three buttons (line 8) and a panel **p2** to hold two labels (line 26), as shown in Figure 12.12. The background of the button **jbtLeft** is set to white (line 12) and the foreground of the button **jbtCenter** is set to green (line 13). The tool tip of the button **jbtRight** is set in line 14. Titled borders are set on panels **p1** and **p2** (lines 18, 37) and line borders are set on the labels (lines 33–34).

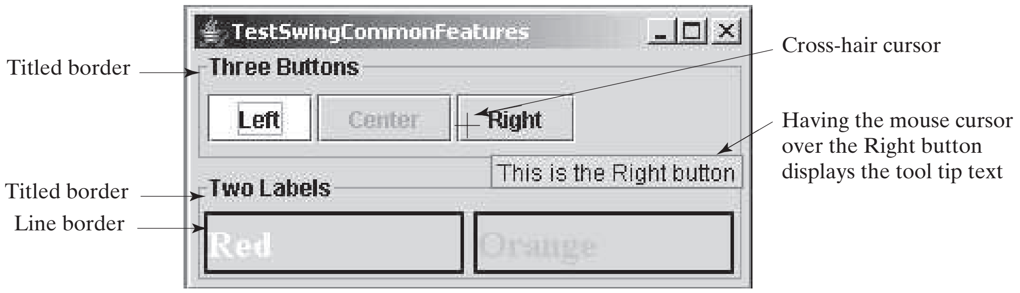


FIGURE 12.12 The font, color, border, and tool tip text are set in the message panel.

The mouse cursor is set to the cross-hair shape in **p1** (line 19). The **Cursor** class contains the constants for specifying the cursor shape such as **DEFAULT_CURSOR**(↖), **CROSSHAIR_CURSOR** (+), **HAND_CURSOR**(☝), **MOVE_CURSOR**(✥), **TEXT_CURSOR**(I), and so on. A **Cursor** object for the cross-hair cursor is created using **new Cursor(Cursor.CROSSHAIR_CURSOR)** (line 19) and this cursor is set for **p1**. Note that the default mouse cursor is still used in **p2**, because the program does not explicitly set a mouse cursor for **p2**.

mouse cursor

LISTING 12.7 TestSwingCommonFeatures.java

```
 1  import java.awt.*;
 2  import javax.swing.*;
 3  import javax.swing.border.*;
 4
 5  public class TestSwingCommonFeatures extends JFrame {
 6    public TestSwingCommonFeatures() {
 7      // Create a panel to group three buttons
 8      JPanel p1 = new JPanel(new FlowLayout(FlowLayout.LEFT, 2, 2));
 9      JButton jbtLeft = new JButton("Left");
10      JButton jbtCenter = new JButton("Center");
11      JButton jbtRight = new JButton("Right");
12      jbtLeft.setBackground(Color.WHITE);
13      jbtCenter.setForeground(Color.GREEN);
14      jbtRight.setToolTipText("This is the Right button");
15      p1.add(jbtLeft);
16      p1.add(jbtCenter);
17      p1.add(jbtRight);
18      p1.setBorder(new TitledBorder("Three Buttons"));
19      p1.setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
20
21      // Create a font and a line border
22      Font largeFont = new Font("TimesRoman", Font.BOLD, 20);
23      Border lineBorder = new LineBorder(Color.BLACK, 2);
```

set background
set foreground
set tool tip text
set titled border
set mouse cursor
create a font
create a border

set foreground

set font

set line border

set titled border

```
24
25      // Create a panel to group two labels
26      JPanel p2 = new JPanel(new GridLayout(1, 2, 5, 5));
27      JLabel jlblRed = new JLabel("Red");
28      JLabel jlblOrange = new JLabel("Orange");
29      jlblRed.setForeground(Color.RED);
30      jlblOrange.setForeground(Color.ORANGE);
31      jlblRed.setFont(largeFont);
32      jlblOrange.setFont(largeFont);
33      jlblRed.setBorder(lineBorder);
34      jlblOrange.setBorder(lineBorder);
35      p2.add(jlblRed);
36      p2.add(jlblOrange);
37      p2.setBorder(new TitledBorder("Two Labels"));
38
39      // Add two panels to the frame
40      setLayout(new GridLayout(2, 1, 5, 5));
41      add(p1);
42      add(p2);
43    }
44
45    public static void main(String[] args) {
46      // Create a frame and set its properties
47      JFrame frame = new TestSwingCommonFeatures();
48      frame.setTitle("TestSwingCommonFeatures");
49      frame.setSize(300, 150);
50      frame.setLocationRelativeTo(null); // Center the frame
51      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
52      frame.setVisible(true);
53    }
54  }
```

property default values

Note

The same property may have different default values in different components. For example, the **visible** property in **JFrame** is **false** by default, but it is **true** in every instance of **JComponent** (e.g., **JButton** and **JLabel**) by default. To display a **JFrame**, you have to invoke **setVisible(true)** to set the **visible** property **true**, but you don't have to set this property for a **JButton** or a **JLabel**, because it is already **true**. To make a **JButton** or a **JLabel** invisible, you can invoke **setVisible(false)**. Please run the program and see the effect after inserting the following two statements in line 38:

```
jbtLeft.setVisible(false);
jlblRed.setVisible(false);
```

Check Point

12.21 How do you set background color, foreground color, font, and tool tip text on a Swing GUI component?

12.22 Why is the tool tip text not displayed in the following code?

```
 1  import javax.swing.*;
 2
 3  public class Test extends JFrame {
 4    private JButton jbtOK = new JButton("OK");
 5
 6    public static void main(String[] args) {
 7      // Create a frame and set its properties
 8      JFrame frame = new Test();
 9      frame.setTitle("Logic Error");
10      frame.setSize(200, 100);
11      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
    frame.setVisible(true);
  }

  public Test() {
    jbtOK.setToolTipText("This is a button");
    add(new JButton("OK"));
  }
}
```

12.23 Show the output of the following code:

```
import javax.swing.*;

public class Test {
  public static void main(String[] args) {
    JButton jbtOK = new JButton("OK");
    System.out.println(jbtOK.isVisible());

    JFrame frame = new JFrame();
    System.out.println(frame.isVisible());
  }
}
```

12.24 What happens if you add a button to a container several times, as shown below? Does it cause syntax errors? Does it cause runtime errors?

```
JButton jbt = new JButton();
JPanel panel = new JPanel();
panel.add(jbt);
panel.add(jbt);
panel.add(jbt);
```

12.10 Image Icons

Key Point

Image icons can be displayed in many GUI components. Image icons are objects created using the `ImageIcon` class.

An icon is a fixed-size picture; typically it is small and used to decorate components. Images are normally stored in image files. Java currently supports three image formats: GIF (Graphics Interchange Format), JPEG (Joint Photographic Experts Group), and PNG (Portable Network Graphics). The image file names for these types end with **.gif**, **.jpg**, and **.png**, respectively. If you have a bitmap file or image files in other formats, you can use image-processing utilities to convert them into the GIF, JPEG, or PNG format for use in Java.

To display an image icon, first create an `ImageIcon` object using `new javax.swing.ImageIcon(filename)`. For example, the following statement creates an icon from an image file **us.gif** in the `image` directory under the current class path:

image-file format

```
ImageIcon icon = new ImageIcon("image/us.gif");
```

create `ImageIcon`

image/us.gif is located in **c:\book\image\us.gif**. The back slash (`\`) is the Windows file path notation. In UNIX, the forward slash (`/`) should be used. In Java, the forward slash (`/`) is used to denote a relative file path under the Java classpath (e.g., **image/us.gif**, as in this example).

file path character

Tip

File names are not case sensitive in Windows but are case sensitive in UNIX. To enable your programs to run on all platforms, name all the image files consistently, using lowercase.

naming files consistently

An image icon can be displayed in a label or a button using **new JLabel(imageIcon)** or **new JButton(imageIcon)**. Listing 12.8 demonstrates displaying icons in labels and buttons. The example creates two labels and two buttons with icons, as shown in Figure 12.13.

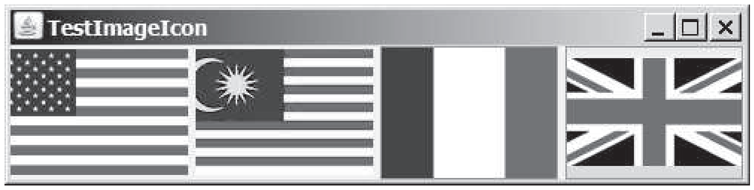


FIGURE 12.13 The image icons are displayed in labels and buttons.

LISTING 12.8 TestImageIcon.java

create image icons

a label with image

a button with image

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

public class TestImageIcon extends JFrame {
  private ImageIcon usIcon = new ImageIcon("image/us.gif");
  private ImageIcon myIcon = new ImageIcon("image/my.jpg");
  private ImageIcon frIcon = new ImageIcon("image/fr.gif");
  private ImageIcon ukIcon = new ImageIcon("image/uk.gif");

  public TestImageIcon() {
    setLayout(new GridLayout(1, 4, 5, 5));
    add(new JLabel(usIcon));
    add(new JLabel(myIcon));
    add(new JButton(frIcon));
    add(new JButton(ukIcon));
  }

  /** Main method */
  public static void main(String[] args) {
    TestImageIcon frame = new TestImageIcon();
    frame.setTitle("TestImageIcon");
    frame.setSize(200, 200);
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }
}
```

sharing borders and icons

Note

Borders and icons can be shared. Thus, you can create a border or icon and use it to set the **border** or **icon** property for any GUI component. For example, the following statements set a border **b** for the panels **p1** and **p2**:

```java
p1.setBorder(b);
p2.setBorder(b);
```

The following statements set an icon in the buttons **jbt1** and **jbt2**:

```java
jbt1.setIcon(icon);
jbt2.setIcon(icon);
```

Tip

A splash screen is an image that is displayed while the application is starting up. If your program takes a long time to load, you may display a splash screen to alert the user. For example, the following command:

splash screen

```
java -splash:image/us.gif TestImageIcon
```

displays an image while the program TestImageIcon is being loaded.

Check Point

MyProgrammingLab™

12.25 How do you create an ImageIcon from the file **image/us.gif** in the class directory?

12.26 Will the following code display three buttons? Will the buttons display the same icon?

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

public class Test extends JFrame  {
  public static void main(String[] args) {
    // Create a frame and set its properties
    JFrame frame = new Test();
    frame.setTitle("ButtonIcons");
    frame.setSize(200, 100);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }

  public Test() {
    ImageIcon usIcon = new ImageIcon("image/us.gif");
    JButton jbt1 = new JButton(usIcon);
    JButton jbt2 = new JButton(usIcon);

    JPanel p1 = new JPanel();
    p1.add(jbt1);

    JPanel p2 = new JPanel();
    p2.add(jbt2);

    JPanel p3 = new JPanel();
    p2.add(jbt1);

    add(p1, BorderLayout.NORTH);
    add(p2, BorderLayout.SOUTH);
    add(p3, BorderLayout.CENTER);
  }
}
```

12.27 Can a border or an icon be shared by GUI components?

12.11 JButton

Key Point

To create a push button, use the JButton *class.*

We have used JButton in the examples to demonstrate the basics of GUI programming. This section will introduce more features of JButton. The following sections will introduce GUI components JCheckBox, JRadioButton, JLabel, JTextField, and JPasswordField. More GUI components such as JTextArea, JComboBox, JList, JScrollBar, and JSlider will be introduced in Chapter 17. The relationship of these classes is pictured in Figure 12.14.

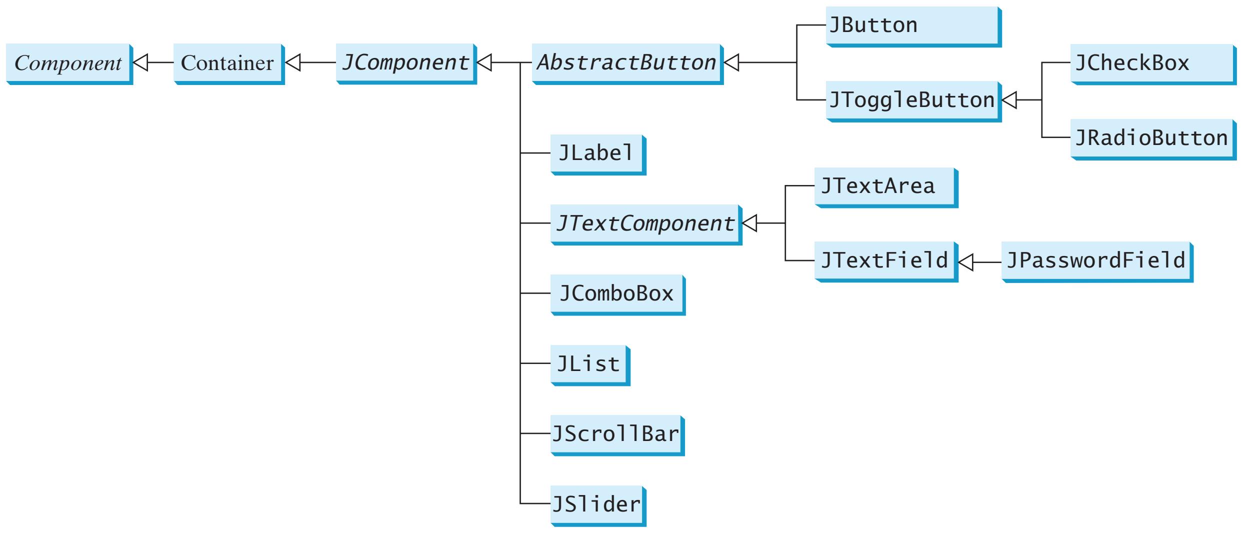


Figure 12.14 These Swing GUI components are frequently used to create user interfaces.

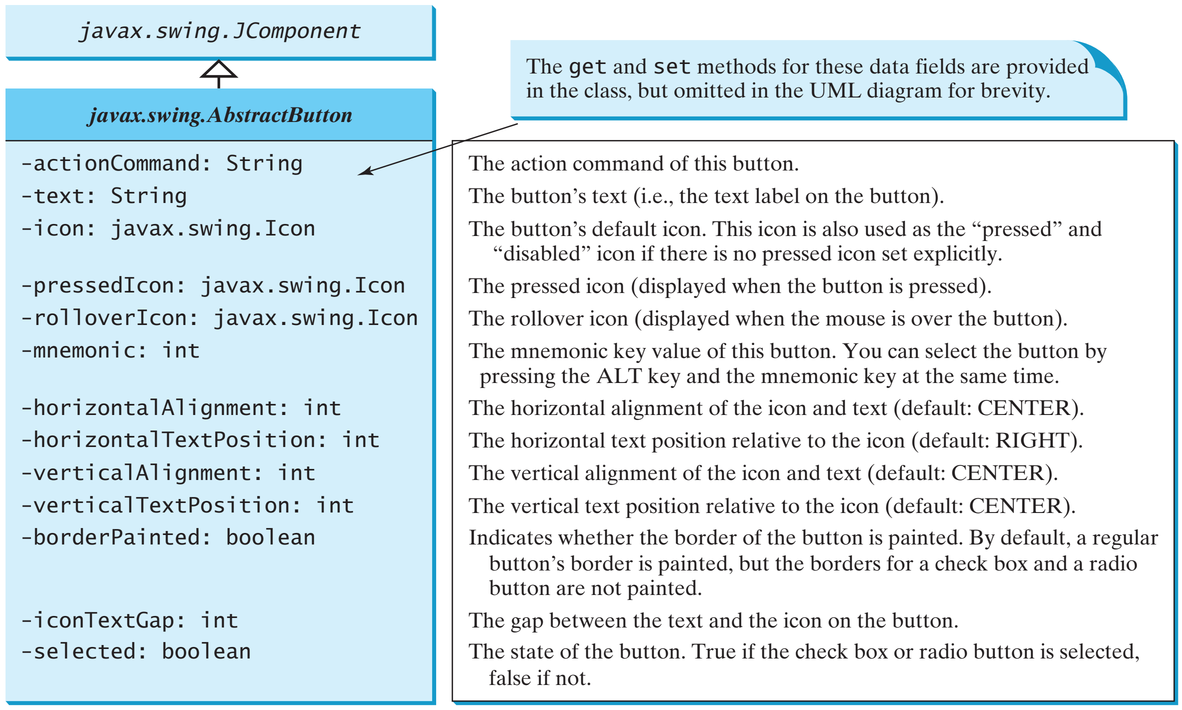
naming convention for components

Note

Throughout this book, the prefixes `jbt`, `jchk`, `jrb`, `jlbl`, `jtf`, `jpf`, `jta`, `jcbo`, `jlst`, `jscb`, and `jsld` are used to name reference variables for `JButton`, `JCheckBox`, `JRadioButton`, `JLabel`, `JTextField`, `JPasswordField`, `JTextArea`, `JComboBox`, `JList`, `JScrollBar`, and `JSlider`.

`AbstractButton`

A *button* is a component that triggers an action when clicked. Swing provides regular buttons, toggle buttons, check box buttons, and radio buttons. The common features of these buttons are defined in `javax.swing.AbstractButton`, as shown in Figure 12.15.


javax.swing.JComponent
javax.swing.AbstractButton
The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

-actionCommand: String	The action command of this button.
-text: String	The button's text (i.e., the text label on the button).
-icon: javax.swing.Icon	The button's default icon. This icon is also used as the "pressed" and "disabled" icon if there is no pressed icon set explicitly.
-pressedIcon: javax.swing.Icon	The pressed icon (displayed when the button is pressed).
-rolloverIcon: javax.swing.Icon	The rollover icon (displayed when the mouse is over the button).
-mnemonic: int	The mnemonic key value of this button. You can select the button by pressing the ALT key and the mnemonic key at the same time.
-horizontalAlignment: int	The horizontal alignment of the icon and text (default: CENTER).
-horizontalTextPosition: int	The horizontal text position relative to the icon (default: RIGHT).
-verticalAlignment: int	The vertical alignment of the icon and text (default: CENTER).
-verticalTextPosition: int	The vertical text position relative to the icon (default: CENTER).
-borderPainted: boolean	Indicates whether the border of the button is painted. By default, a regular button's border is painted, but the borders for a check box and a radio button are not painted.
-iconTextGap: int	The gap between the text and the icon on the button.
-selected: boolean	The state of the button. True if the check box or radio button is selected, false if not.

Figure 12.15 `AbstractButton` defines common features of different types of buttons.

JButton inherits AbstractButton and provides several constructors to create buttons, as shown in Figure 12.16.

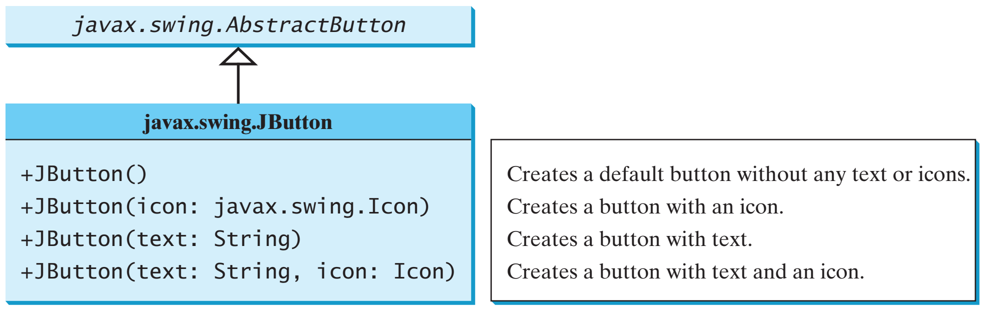

javax.swing.AbstractButton
javax.swing.JButton
+JButton()
+JButton(icon: javax.swing.Icon)
+JButton(text: String)
+JButton(text: String, icon: Icon)
Creates a default button without any text or icons.
Creates a button with an icon.
Creates a button with text.
Creates a button with text and an icon.

FIGURE 12.16 JButton defines a regular push button.

12.11.1 Icons, Pressed Icons, and Rollover Icons

A button has a default icon, a pressed icon, and a rollover icon. Normally you use the default icon, because the other icons are for special effects. A pressed icon is displayed when a button is pressed, and a rollover icon is displayed when the mouse is over the button but not pressed. For example, Listing 12.9 displays the U.S. flag as a regular icon, the Canadian flag as a pressed icon, and the British flag as a rollover icon, as shown in Figure 12.17.

LISTING 12.9 TestButtonIcons.java

```java
import javax.swing.*;

public class TestButtonIcons extends JFrame  {
  public static void main(String[] args) {
    // Create a frame and set its properties
    JFrame frame = new TestButtonIcons();
    frame.setTitle("ButtonIcons");
    frame.setSize(200, 100);
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }

  public TestButtonIcons() {
    ImageIcon usIcon = new ImageIcon("image/usIcon.gif");
    ImageIcon caIcon = new ImageIcon("image/caIcon.gif");
    ImageIcon ukIcon = new ImageIcon("image/ukIcon.gif");

    JButton jbt = new JButton("Click it", usIcon);
    jbt.setPressedIcon(caIcon);
    jbt.setRolloverIcon(ukIcon);

    add(jbt);
  }
}
```

create icons

regular icon
pressed icon
rollover icon

add a button

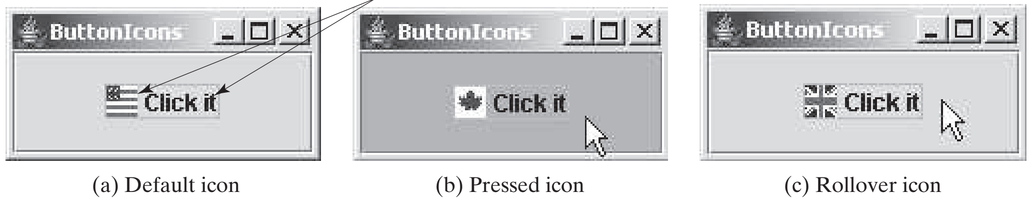


FIGURE 12.17 A button can have several types of icons.

12.11.2 Alignments

horizontal alignment

Horizontal alignment specifies how the icon and text are placed horizontally on a button. You can set the horizontal alignment using `setHorizontalAlignment(int)` with one of the five constants `LEADING`, `LEFT`, `CENTER`, `RIGHT`, or `TRAILING`, as shown in Figure 12.18. At present, `LEADING` and `LEFT` are the same, and `TRAILING` and `RIGHT` are the same. Future implementation may distinguish them. The default horizontal alignment is `AbstractButton.CENTER`.

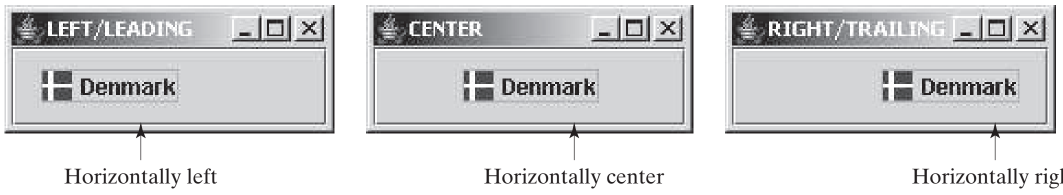


FIGURE 12.18 You can specify how the icon and text are placed on a button horizontally.

vertical alignment

Vertical alignment specifies how the icon and text are placed vertically on a button. You can set the vertical alignment using `setVerticalAlignment(int)` with one of the three constants `TOP`, `CENTER`, or `BOTTOM`, as shown in Figure 12.19. The default vertical alignment is `AbstractButton.CENTER`.

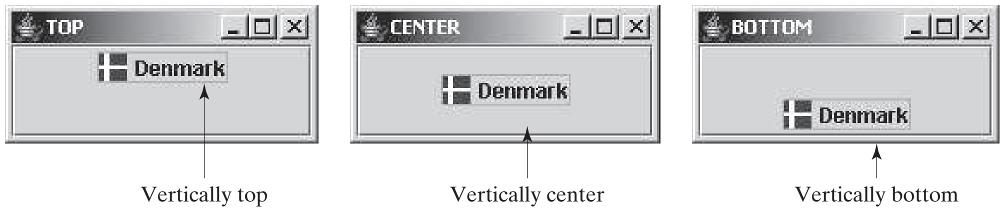


FIGURE 12.19 You can specify how the icon and text are placed on a button vertically.

12.11.3 Text Positions

horizontal text position

Horizontal text position specifies the horizontal position of the text relative to the icon. You can set the horizontal text position using `setHorizontalTextPosition(int)` with one of the five constants `LEADING`, `LEFT`, `CENTER`, `RIGHT`, or `TRAILING`, as shown in Figure 12.20. At present, `LEADING` and `LEFT` are the same, and `TRAILING` and `RIGHT` are the same. Future implementation may distinguish them. The default horizontal text position is `AbstractButton.RIGHT`.

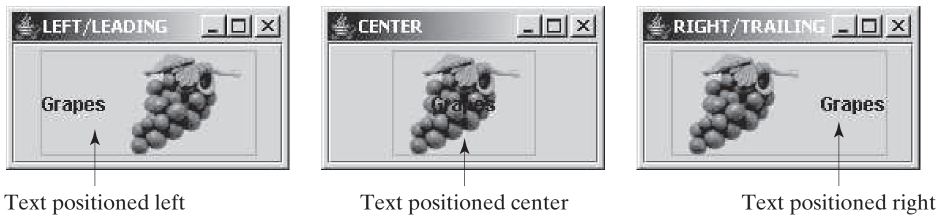


FIGURE 12.20 You can specify the horizontal position of the text relative to the icon.

vertical text position

Vertical text position specifies the vertical position of the text relative to the icon. You can set the vertical text position using `setVerticalTextPosition(int)` with one of the three constants TOP, CENTER, or BOTTOM, as shown in Figure 12.21. The default vertical text position is AbstractButton.CENTER.

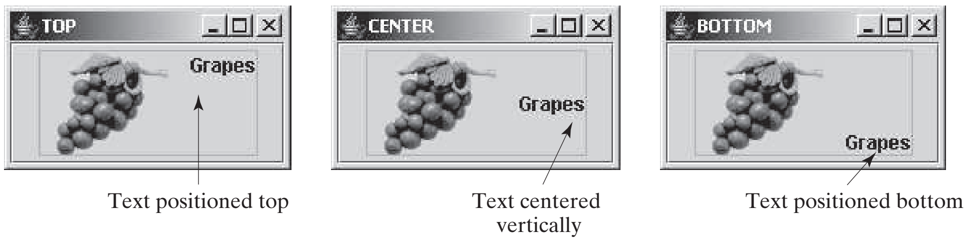


FIGURE 12.21 You can specify the vertical position of the text relative to the icon.

Check Point
MyProgrammingLab™

12.28 How do you create a button with the text OK? How do you change text on a button? How do you set an icon, a pressed icon, and a rollover icon in a button?

12.29 Given a JButton object jbtOK, write statements to set the button's foreground to red, background to yellow, mnemonic to K, tool tip text to Click OK to proceed, horizontal alignment to RIGHT, vertical alignment to BOTTOM, horizontal text position to LEFT, vertical text position to TOP, and icon text gap to 5.

12.30 List at least five properties defined in the AbstractButton class.

12.12 JCheckBox

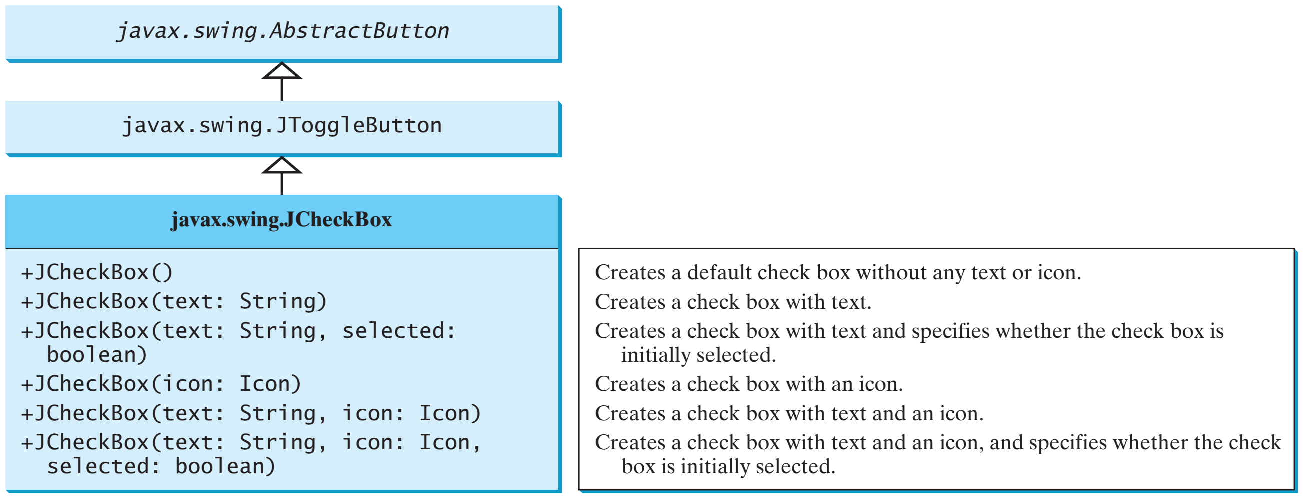
Key Point

To create a check box button, use the JCheckBox class.

toggle button

A *toggle button* is a two-state button like a light switch. JToggleButton *inherits* AbstractButton *and* implements a toggle button. Often JToggleButton's subclasses JCheckBox and JRadioButton are used to enable the user to toggle a choice on or off. This section introduces JCheckBox. JRadioButton will be introduced in the next section.

JCheckBox inherits all the properties from AbstractButton, such as text, icon, mnemonic, verticalAlignment, horizontalAlignment, horizontalTextPosition, verticalTextPosition, and selected, and provides several constructors to create check boxes, as shown in Figure 12.22.


javax.swing.AbstractButton

javax.swing.JToggleButton

javax.swing.JCheckBox

```
+JCheckBox()
+JCheckBox(text: String)
+JCheckBox(text: String, selected:
  boolean)
+JCheckBox(icon: Icon)
+JCheckBox(text: String, icon: Icon)
+JCheckBox(text: String, icon: Icon,
  selected: boolean)
```

Creates a default check box without any text or icon.
Creates a check box with text.
Creates a check box with text and specifies whether the check box is initially selected.
Creates a check box with an icon.
Creates a check box with text and an icon.
Creates a check box with text and an icon, and specifies whether the check box is initially selected.

FIGURE 12.22 JCheckBox defines a check box button.

Here is an example for creating a check box with the text *Student*. Its foreground is **red**, the background is **white**, its mnemonic key is **S**, and it is initially selected.

```
JCheckBox jchk = new JCheckBox("Student", true);
jchk.setForeground(Color.RED);
jchk.setBackground(Color.WHITE);
jchk.setMnemonic('S');
```

mnemonics

The button can also be accessed by using the keyboard mnemonics. Pressing ***Alt+S*** is equivalent to clicking the check box.

isSelected?

To see if a check box is selected, use the `isSelected()` method.

Check Point
MyProgrammingLab™

12.31 How do you create a check box? How do you create a check box with the box checked initially? How do you determine whether a check box is selected?

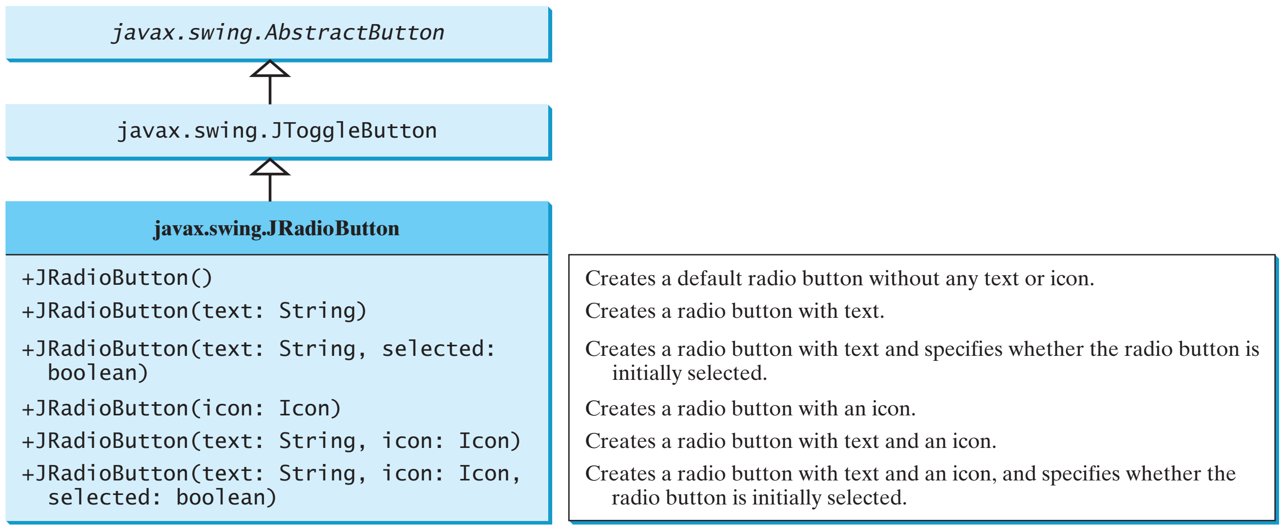
12.13 JRadioButton

Key Point

To create a radio button, use the `JRadioButton` *class.*

Radio buttons, also known as *option buttons*, enable you to choose a single item from a group of choices. In appearance radio buttons resemble check boxes, but check boxes display a square that is either checked or blank, whereas radio buttons display a circle that is either filled (if selected) or blank (if not selected).

`JRadioButton` inherits `AbstractButton` and provides several constructors to create radio buttons, as shown in Figure 12.23. These constructors are similar to the constructors for `JCheckBox`.



+JRadioButton()	Creates a default radio button without any text or icon.
+JRadioButton(text: String)	Creates a radio button with text.
+JRadioButton(text: String, selected: boolean)	Creates a radio button with text and specifies whether the radio button is initially selected.
+JRadioButton(icon: Icon)	Creates a radio button with an icon.
+JRadioButton(text: String, icon: Icon)	Creates a radio button with text and an icon.
+JRadioButton(text: String, icon: Icon, selected: boolean)	Creates a radio button with text and an icon, and specifies whether the radio button is initially selected.

FIGURE 12.23 `JRadioButton` defines a radio button.

Here is an example for creating a radio button with the text Student. The code specifies **red** foreground, **white** background, mnemonic key **S**, and initially selected.

```
JRadioButton jrb = new JRadioButton("Student", true);
jrb.setForeground(Color.RED);
jrb.setBackground(Color.WHITE);
jrb.setMnemonic('S');
```

To group radio buttons, you need to create an instance of **`java.swing.ButtonGroup`** and use the **`add`** method to add them to it, as follows:

```
ButtonGroup group = new ButtonGroup();
group.add(jrb1);
group.add(jrb2);
```

This code creates a radio-button group for the radio buttons **`jrb1`** and **`jrb2`** so that they are selected mutually exclusively. Without grouping, **`jrb1`** and **`jrb2`** would be independent.

Note

`ButtonGroup` is not a subclass of **`java.awt.Component`**, so a **`ButtonGroup`** object cannot be added to a container.

GUI helper class

To see if a radio button is selected, use the **`isSelected()`** method.

12.32 How do you create a radio button? How do you create a radio button with the button selected initially? How do you group radio buttons together? How do you determine whether a radio button is selected?

Check Point

MyProgrammingLab™

12.14 Labels

To create a label, use the **`JLabel`** *class.*

Key Point

A *label* is a display area for a short text, an image, or both. It is often used to label other components (usually text fields). Figure 12.24 lists the constructors and methods in the **`JLabel`** class.

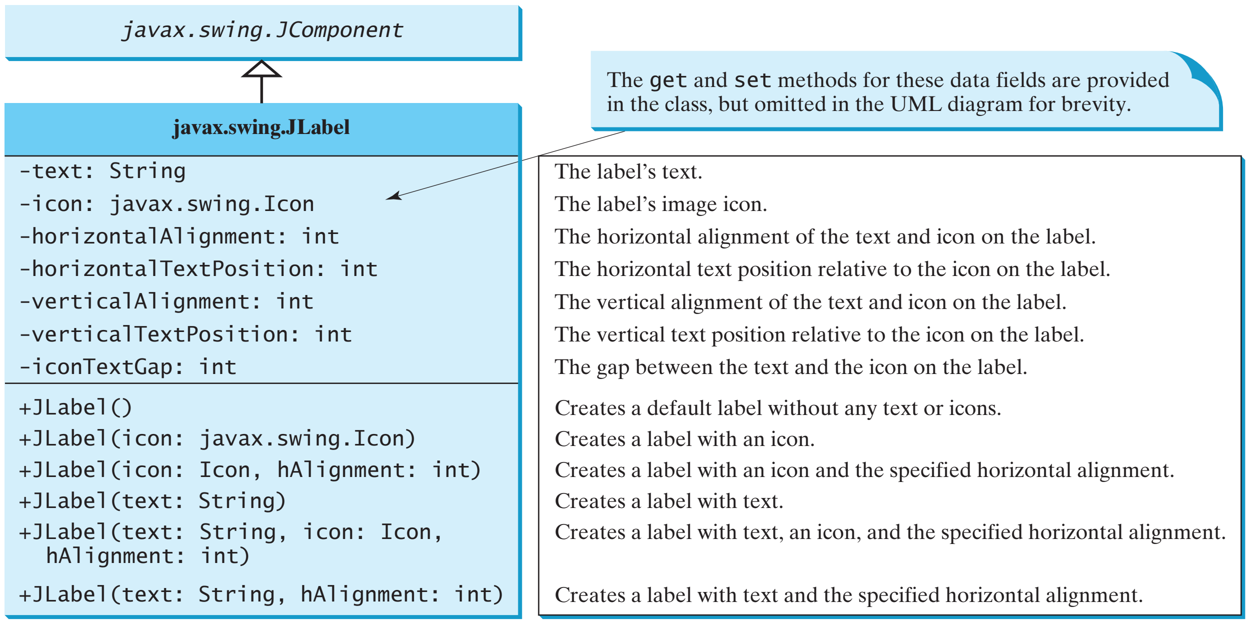
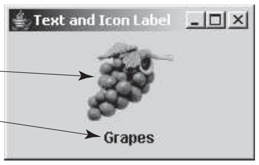

javax.swing.JComponent
javax.swing.JLabel
-text: String
-icon: javax.swing.Icon
-horizontalAlignment: int
-horizontalTextPosition: int
-verticalAlignment: int
-verticalTextPosition: int
-iconTextGap: int
+JLabel()
+JLabel(icon: javax.swing.Icon)
+JLabel(icon: Icon, hAlignment: int)
+JLabel(text: String)
+JLabel(text: String, icon: Icon, hAlignment: int)
+JLabel(text: String, hAlignment: int)
The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.
The label's text.
The label's image icon.
The horizontal alignment of the text and icon on the label.
The horizontal text position relative to the icon on the label.
The vertical alignment of the text and icon on the label.
The vertical text position relative to the icon on the label.
The gap between the text and the icon on the label.
Creates a default label without any text or icons.
Creates a label with an icon.
Creates a label with an icon and the specified horizontal alignment.
Creates a label with text.
Creates a label with text, an icon, and the specified horizontal alignment.
Creates a label with text and the specified horizontal alignment.

FIGURE 12.24 **`JLabel`** displays text or an icon, or both.

`JLabel` inherits all the properties from **`JComponent`** and has many properties similar to the ones in **`JButton`**, such as **`text`**, **`icon`**, **`horizontalAlignment`**, **`verticalAlignment`**, **`horizontalTextPosition`**, **`verticalTextPosition`**, and **`iconTextGap`**. For example, the following code displays a label with text and an icon:

```
// Create an image icon from an image file
ImageIcon icon = new ImageIcon("image/grapes.gif");

// Create a label with a text, an icon,
// with centered horizontal alignment
JLabel jlbl = new JLabel("Grapes", icon, JLabel.CENTER);

//Set label's text alignment and gap between text and icon
jlbl.setHorizontalTextPosition(JLabel.CENTER);
jlbl.setVerticalTextPosition(JLabel.BOTTOM);
jlbl.setIconTextGap(5);
```



Check Point
MyProgrammingLab™

12.33 How do you create a label named `Address`? How do you change the name on a label? How do you set an icon in a label?

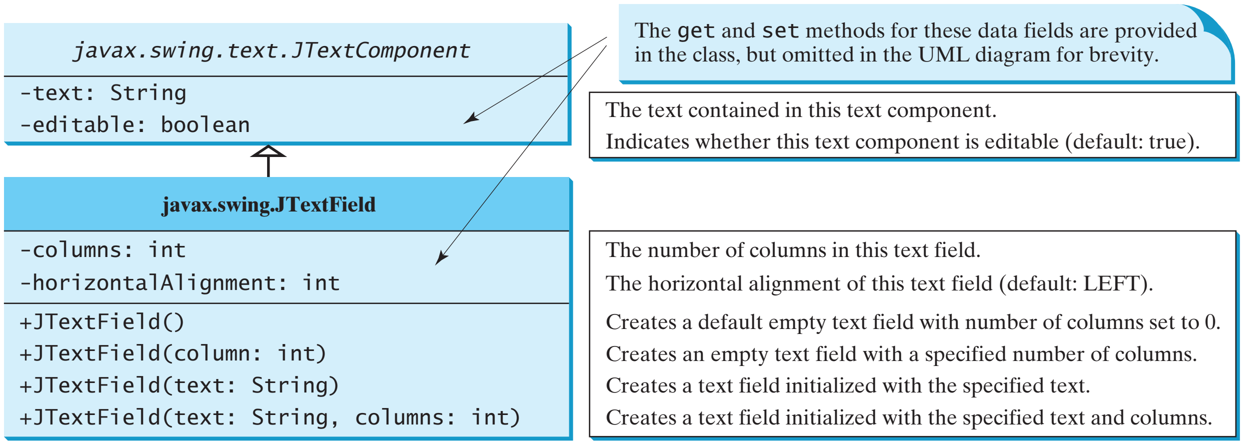
12.34 Given a `JLabel` object `jlblMap`, write statements to set the label's foreground to `red`, background to `yellow`, mnemonic to `M`, tool tip text to `Map image`, horizontal alignment to `RIGHT`, vertical alignment to `BOTTOM`, horizontal text position to `LEFT`, vertical text position to `TOP`, and icon text gap to `5`.

12.15 Text Fields

Key Point

To create a text field, use the `JTextField` *class.*

A *text field* can be used to enter or display a string. `JTextField` is a subclass of `JTextComponent`. Figure 12.25 lists the constructors and methods in `JTextField`.



```
javax.swing.text.JTextComponent
-text: String
-editable: boolean

javax.swing.JTextField
-columns: int
-horizontalAlignment: int
+JTextField()
+JTextField(column: int)
+JTextField(text: String)
+JTextField(text: String, columns: int)
```

The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

The text contained in this text component.
Indicates whether this text component is editable (default: true).

The number of columns in this text field.
The horizontal alignment of this text field (default: LEFT).
Creates a default empty text field with number of columns set to 0.
Creates an empty text field with a specified number of columns.
Creates a text field initialized with the specified text.
Creates a text field initialized with the specified text and columns.

FIGURE 12.25 `JTextField` enables you to enter or display a string.

`JTextField` inherits `JTextComponent`, which inherits `JComponent`. Here is an example of creating a text field with red foreground color and right horizontal alignment:

```
JTextField jtfMessage = new JTextField("T-Storm");
jtfMessage.setForeground(Color.RED);
jtfMessage.setHorizontalAlignment(JTextField.RIGHT);
```

To set new text in a text field, use the `setText(newText)` method. To get the text from a text field, use the `getText()` method.

Note

If a text field is used for entering a password, use `JPasswordField` to replace `JTextField`. `JPasswordField extends JTextField` and hides the input text with echo characters (e.g., ******). By default, the echo character is *. You can specify a new echo character using the `setEchoChar(char)` method.

JPasswordField

Check Point
MyProgrammingLab™

12.35 How do you create a text field with `10` columns and the default text `Welcome to Java`? How do you write the code to check whether a text field is empty?

12.36 How do you create text field for entering passwords?

Key Terms

AWT	446	layout manager	451
component class	446	lightweight component	446
container class	446	Swing components	446
heavyweight component	446	splash screen	467
helper class	446	top-level container	447

Chapter Summary

1. Every container has a *layout manager* that is used to position and place components in the container in the desired locations. Three simple and frequently used layout managers are `FlowLayout`, `GridLayout`, and `BorderLayout`.

2. You can use a `JPanel` as a subcontainer to group components to achieve a desired layout.

3. Use the `add` method to place components in a `JFrame` or a `JPanel`. By default, the frame's layout is `BorderLayout`, and the `JPanel`'s layout is `FlowLayout`.

4. Colors are made of red, green, and blue components, each represented by an unsigned byte value that describes its intensity, ranging from `0` (darkest shade) to `255` (lightest shade). This is known as the *RGB model*.

5. To create a `Color` object, use `new Color(r, g, b)`, in which `r`, `g`, and `b` specify a color by its red, green, and blue components. Alternatively, you can use one of the 13 standard colors (`BLACK`, `BLUE`, `CYAN`, `DARK_GRAY`, `GRAY`, `GREEN`, `LIGHT_GRAY`, `MAGENTA`, `ORANGE`, `PINK`, `RED`, `WHITE`, `YELLOW`) defined as constants in `java.awt.Color`.

6. Every *Swing* GUI component is a subclass of `javax.swing.JComponent`, and `JComponent` is a subclass of `java.awt.Component`. The properties `font`, `background`, `foreground`, `height`, `width`, and `preferredSize` in `Component` are inherited in these subclasses, as are `toolTipText` and `border` in `JComponent`.

7. You can use borders on any Swing components. You can create an image icon using the `ImageIcon` class and display it in a label and a button. Icons and borders can be shared.

8. You can display a text and icon on buttons (`JButton`, `JCheckBox`, `JRadioButton`) and labels (`JLabel`).

9. You can specify the horizontal and vertical text alignment in `JButton`, `JCheckBox`, `JRadioButton`, and `JLabel`, and the horizontal text alignment in `JTextField`.

10. You can specify the horizontal and vertical text position relative to the icon in `JButton`, `JCheckBox`, `JRadioButton`, and `JLabel`.

Test Questions

Do the test questions for this chapter online at www.cs.armstrong.edu/liang/intro9e/test.html.

MyProgrammingLab™

Programming Exercises

download image files

Note
The image icons used in the exercises can be obtained from www.cs.armstrong.edu/liang/intro9e/book.zip under the image folder.

Sections 12.2–12.6

12.1 (*Use the `FlowLayout` manager*) Write a program that meets the following requirements (see Figure 12.26):

- Create a frame and set its layout to `FlowLayout`.
- Create two panels and add them to the frame.
- Each panel contains three buttons. The panel uses `FlowLayout`.

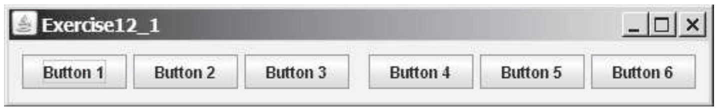


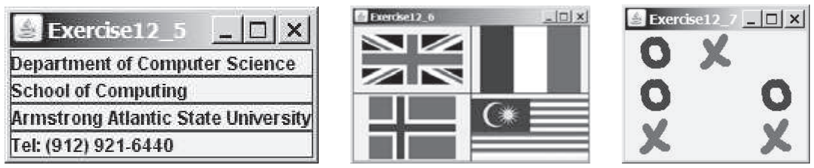
Figure 12.26 Exercise 12.1 places the first three buttons in one panel and the other three buttons in another panel.

12.2 (*Use the `BorderLayout` manager*) Rewrite the preceding program to create the same user interface, but instead of using `FlowLayout` for the frame, use `BorderLayout`. Place one panel in the south of the frame and the other in the center.

12.3 (*Use the `GridLayout` manager*) Rewrite Programming Exercise 12.1 to add six buttons into a frame. Use a `GridLayout` of two rows and three columns for the frame.

12.4 (*Use `JPanel` to group buttons*) Rewrite Programming Exercise 12.1 to create the same user interface. Instead of creating buttons and panels separately, define a class that extends the `JPanel` class. Place three buttons in your panel class, and create two panels from the user-defined panel class.

12.5 (*Display labels*) Write a program that displays four lines of text in four labels, as shown in Figure 12.27a. Add a line border around each label.



Department of Computer Science
School of Computing
Armstrong Atlantic State University
Tel: (912) 921-6440

(a) (b) (c)

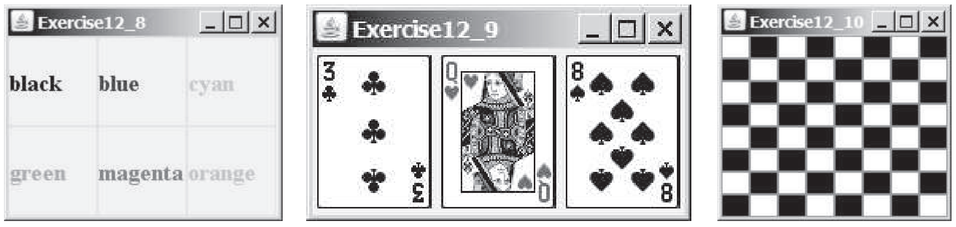
Figure 12.27 (a) Exercise 12.5 displays four labels. (b) Exercise 12.6 displays four icons. (c) Exercise 12.7 displays a tic-tac-toe board with image icons in labels.

Sections 12.7–12.15

12.6 (*Display icons*) Write a program that displays four icons in four labels, as shown in Figure 12.27b. Add a line border around each label.

****12.7** (*Game: display a tic-tac-toe board*) Display a frame that contains nine labels. A label may display an image icon for X or an image icon for O, as shown in Figure 12.27c. What to display is randomly decided. Use the `Math.random()` method to generate an integer `0` or `1`, which corresponds to displaying an X or O image icon. These images are in the files **x.gif** and **o.gif**.

***12.8** (*Swing common features*) Display a frame that contains six labels. Set the background of the labels to white. Set the foreground of the labels to black, blue, cyan, green, magenta, and orange, respectively, as shown in Figure 12.28a. Set the border of each label to a line border with the color yellow. Set the font of each label to Times Roman, bold, and 20 pixels. Set the text and tool tip text of each label to the name of its foreground color.



(a) (b) (c)

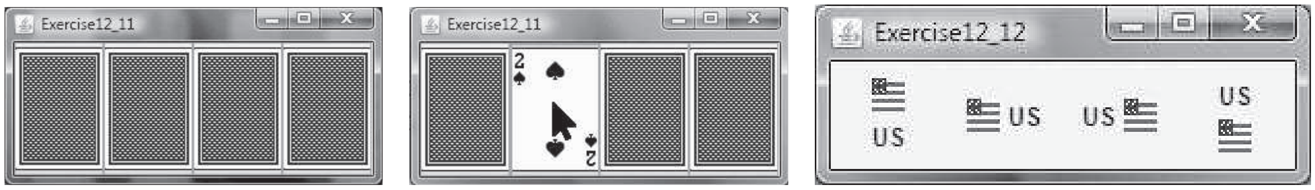
FIGURE 12.28 (a) Six labels are placed in the frame. (b) Three cards are randomly selected. (c) A checkerboard is displayed using buttons.

***12.9** (*Game: display three cards*) Display a frame that contains three labels. Each label displays a card, as shown in Figure 12.28b. The card image files are named **1.png**, **2.png**, . . ., **54.png** (including jokers) and stored in the **image/card** directory. All three cards are distinct and selected randomly.

***12.10** (*Game: display a checkerboard*) Write a program that displays a checkerboard in which each white and black cell is a `JButton` with a background black or white, as shown in Figure 12.28c.

VideoNote
Display a checkerboard

***12.11** (*Game: display four cards*) Use the same cards from Exercise 12.9 to display a frame that contains four buttons. All buttons have the same icon from **backCard.png**, as shown in Figure 12.29a. The pressed icons are four cards randomly selected from the 54 cards in a deck, as shown in Figure 12.29b.



(a) (b) (c)

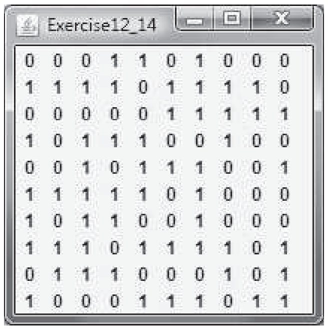
FIGURE 12.29 (a) The four buttons have the same icon. (b) Each button's pressed icon is randomly picked from the deck. (c) The image icons and texts are displayed in four labels.

12.12 (*Use labels*) Write a program that displays the image icon and the text in four labels, as shown Figure 12.29c.

12.13 (*Display 54 cards*) Expand Exercise 12.9 to display all 54 cards in 54 labels, nine per row.

VideoNote
Display a random matrix

***12.14** (*Display random 0 or 1*) Write a program that displays a 10-by-10 square matrix, as shown in Figure 12.30. Each element in the matrix is **0** or **1**, randomly generated. Display each number centered in a label.



```
0  0  0  1  1  0  1  0  0  0
1  1  1  1  0  1  1  1  1  0
0  0  0  0  0  1  1  1  1  1
1  0  1  1  1  0  0  1  0  0
0  0  1  0  1  1  1  0  0  1
1  1  1  1  1  0  1  0  0  0
1  0  1  1  0  0  1  0  0  0
1  1  1  0  1  1  1  1  0  1
0  1  1  1  0  0  0  1  0  1
1  0  0  0  1  1  1  0  1  1
```

FIGURE 12.30 The program randomly generates 0s and 1s.