Module 12

Building Java GUIs Using the Swing API

Objectives

Upon completion of this module, you should be able to:

- Describe the JFC Swing technology
- Define Swing
- Identify the Swing packages
- Describe the GUI building blocks: containers, components, and layout managers
- Examine top-level, general-purpose, and special-purpose properties of container
- Examine components
- Examine layout managers
- Describe the Swing single-threaded model
- Build a GUI using Swing components

Additional Resources

Additional resources – The following references provide additional information on the topics described in this module:

Using all the functionality of JFC/Swing technology requires a lot more practice and study. Some references for further study of JFC/Swing technology are:

- JFC/Swing technology tutorial in the free Java tutorials, located at the following URL:

  `http://java.sun.com/docs/books/tutorial/`
- The code of the `SwingSet` demo
- The API documentation

  Start by studying the `javax.swing` package, and let that study lead you as necessary to the sub-packages.
- Arnold, Gosling, Holmes. *The Java Programming Language, Fourth Edition*. Prentice-Hall. 2005.
- Zakhour, Hommel, Royal, Rabinovitch, Risser, Hoeber. *The Java Tutorial: A Short Course on the Basics, Fourth Edition*. Prentice-Hall. 2006.
- Bates, Sierra. *Head First Java, Second Edition*. O'Reilly Media. 2005.

What Are the Java Foundation Classes (JFC)?

JFC, or Java Foundation Classes, is a set of Graphical User Interface (GUI) support packages that are available as part of the Java SE platform and which became core application program interfaces (APIs) in JDK 1.2. As a result of this, some of the boundaries between JFC and core JDK have become blurred, but for the purposes of this course, JFC includes the following features:

- The Swing component set – *Swing* is an enhanced component set that provides replacement components for those in the original AWT and a number of more advanced components.
- 2D graphics – Using the Java 2D API, you can perform advanced drawing, complex color manipulations, shape and transformation (rotate, shear, stretch, and so forth) handling, and treat text as shapes that can be manipulated. 2D graphics are not discussed in this course.
- Pluggable look-and-feel. This feature provides Swing components a choice of look-and-feel. The same program can be rendered in Microsoft Windows, Motif, and Metal look-and-feel formats.
- Accessibility – National governments are increasingly mandating that computer programs used in their departments are accessible to those with disabilities. The Swing component set facilitates such programming by means of the accessibility APIs. It provides interfaces for associative technologies, such as screen readers, screen magnifiers, audible text readers (speech processing), and so on.
- Drag-and-drop – GUI-based data transfer, both between elements of one program and between different programs, has been a feature of modern GUI systems for a number of years. This type of transfer falls into two forms from the user's point of view. These are cut-and-paste and drag-and-drop. The Java JDK releases include support for both these communications media.
- Internationalization – The JFC classes support different character sets such as Japanese, Chinese, and Korean. This allows developers to build applications that can interact with users worldwide in their own languages.

What Is Swing?

Swing is an enhanced component set that provides replacement components for those in the original AWT and a number of more advanced components. These components enable you to create user interfaces with the type of functionality that has become expected in modern applications. Such components include trees, tables, advanced text editors, and tear-off toolbars.

Swing also has special features. For example, using Swing, you can write a program that adopts either the *look-and-feel* of the host platform or that uses a common look-and-feel written especially for the Java programming language (Metal). In fact, you can create your own look-and-feel from scratch, or modify an existing one and *plug it in* to your program, either hard-coded or by the user or system administrator selecting a look-and-feel at runtime.

Note – The term look-and-feel occurs frequently in this module. Look refers to appearance of components, while feel refers to the way they react to user actions, such as mouse clicks. Writing a look-and-feel refers to writing the necessary classes to define new appearance and input behavior. Writing a look-and-feel is outside the scope of this module.

Pluggable Look-and-Feel

Pluggable look-and-feel enables developers to build applications that execute on any platform as if they were developed for that specific platform. A program executed in the Microsoft Windows environment appears as if it was developed for this environment; and the same program executed on the UNIX platform appears as if it was developed for the UNIX environment.

Developers can create their own custom Swing components, with any kind of look-and-feel that they choose to design. This increases the consistency of applications and applets deployed across platforms. An entire application's GUI can switch from one look-and-feel to a different one at runtime.

Pluggable look-and-feel provided by the Swing components is facilitated by their underlying architecture. The next section describes the Swing architecture and explains how it facilitates the pluggable look-and-feel.

Swing Architecture

Swing components are designed based on the Model-View-Controller (MVC) architecture. The Swing architecture is not strictly based on the MVC architecture but has its roots in the MVC.

Model-View-Controller Architecture

According to the MVC architecture, a component can be modeled as three separate parts. Figure 12-1 shows the MVC architecture.

- Model – The model stores the data used to define the component.
- View – The view represents the visual display of the component. This display is governed by the data in the model.
- Controller – The controller deals with the behavior of the components when a user interacts with it. This behavior can include any updates to the model or view.

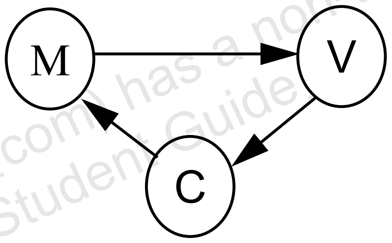

Figure 12-1 Model-View-Controller Architecture

Theoretically, these three types of architecture (Model, View, Controller) should be represented by different class types. But, practically, this turns out to be very difficult because of the dependencies between the view and the controller. The role of the controller is heavily dependent on the implementation of the view, because the user interacts with the view. In other words, it is difficult to write a generic controller, independent of the implementation of the view. This issue is addressed by the separable model architecture.

Separable Model Architecture

The Swing components follow a separable model architecture. In this architecture the view and the controller are merged as a single composite object, because of their tight dependency on each other. The model object is treated as a separate object just like in MVC architecture. Figure 12-2 on page 12-6 shows the separable model architecture.

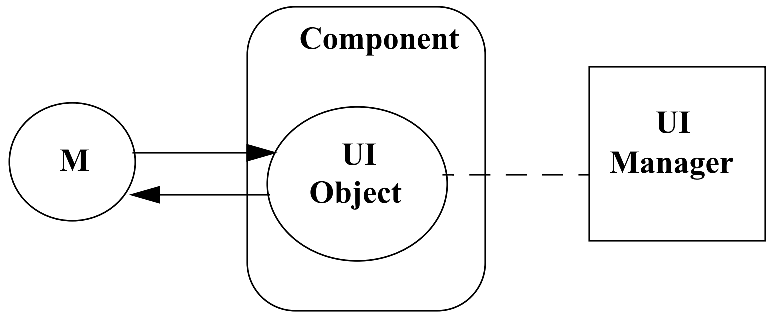

Figure 12-2 Separable Model Architecture

The UI Object in Figure 12-2 is referred to as UI delegate. With this architecture it is possible to delegate a few responsibilities of the component to an independent UI Object. This explains how the pluggable look-and-feel is facilitated by the Swing components. The components make the visual appearance of the components the responsibility of the independent UI Object. So the UI Object of the installed look-and-feel is responsible for the components look-and-feel.

Swing Packages

The Swing API has a rich and convenient set of packages that makes it powerful and flexible. Table 12-1 lists each package name and the purpose of each package.

Table 12-1 Swing Packages

Package Name	**Purpose**
`javax.swing`	Provides a set of *light-weight* components such as, `JButton`, `JFrame`, `JCheckBox`, and much more
`javax.swing.border`	Provides classes and interfaces for drawing specialized borders such as, bevel, etched, line, matte, and more
`javax.swing.event`	Provides support for events fired by Swing components
`javax.swing.undo`	Allows developers to provide support for undo/redo in applications such as text editors
`javax.swing.colorchooser`	Contains classes and interfaces used by the `JColorChooser` component
`javax.swing.filechooser`	Contains classes and interfaces used by the `JFileChooser` component
`javax.swing.table`	Provides classes and interfaces for handling `JTable`
`javax.swing.tree`	Provides classes and interfaces for handling `JTree`
`javax.swing.plaf`	Provides one interface and many abstract classes that Swing uses to provide its pluggable look-and-feel capabilities
`javax.swing.plaf.basic`	Provides user interface objects built according to the `Basic` look-and-feel

Table 12-1 Swing Packages (Continued)

Package Name	Purpose
`javax.swing.plaf.metal`	Provides user interface objects built according to the `Java` look-and-feel
`javax.swing.plaf.multi`	Provides user interface objects that combine two or more look-and-feels
`javax.swing.plaf.synth`	Provides user interface objects for a skinnable look-and-feel in which all painting is delegated
`javax.swing.text`	Provides classes and interfaces that deal with editable and non-editable text components
`javax.swing.text.html`	Provides the class `HTMLEditorKit` and supporting classes for creating HTML text editors
`javax.swing.text.html.parser`	Provides the default HTML parser, along with support classes

Examining the Composition of a Java Technology GUI

A Swing API-based GUI is composed of the following elements.

- Containers

  Containers are on top of the GUI containment hierarchy. All the components in the GUI are added to these containers. JFrame, JDialog, JWindow, and JApplet are the top-level containers.

- Components

  All the GUI components are derived from the JComponent class, for example, JComboBox, JAbstractButton, and JTextComponent.

- Layout Managers

  Layout managers are responsible for laying out components in a container. BorderLayout, FlowLayout, GridLayout are a few examples of the layout managers. There are more sophisticated and complex layout managers that give more control over the GUI.

Figure 12-3 shows the use of components, containers, and layout managers in the composition of the sample Swing user interface.

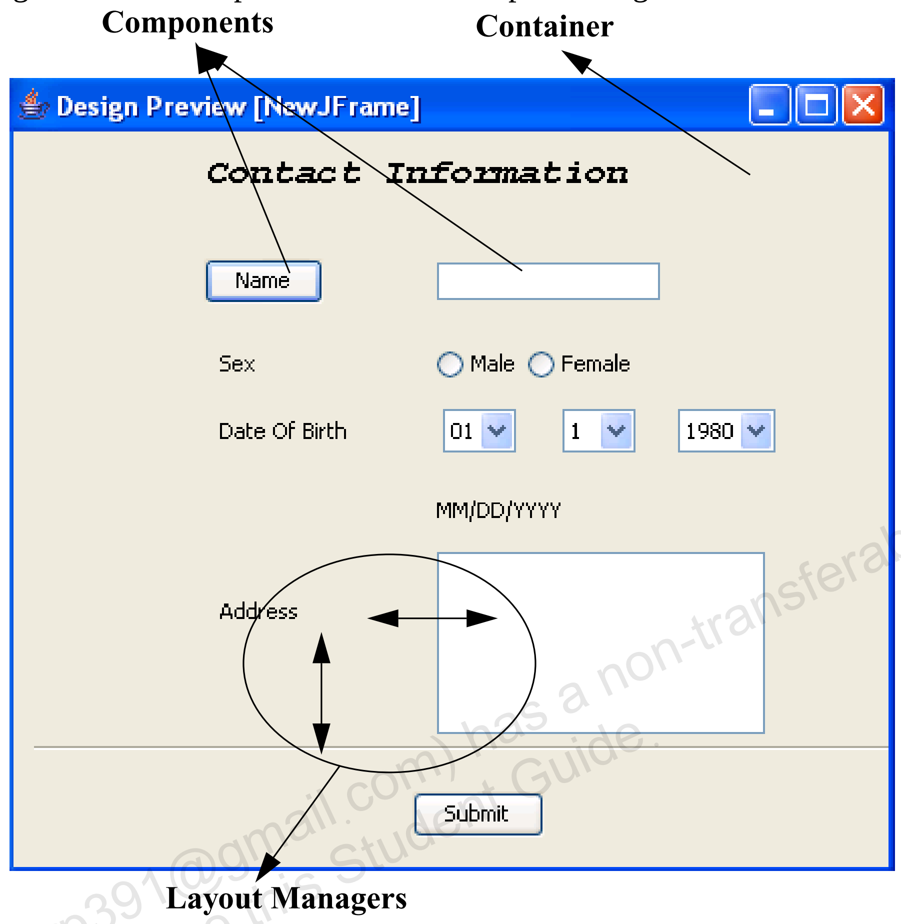

Figure 12-3 GUI Composition

Swing Containers

Swing containers can be classified into three main categories

- Top-level containers
- General-purpose containers
- Special-purpose containers

Top-level Containers

Top-level containers are at the top of the Swing containment hierarchy. There are three top-level Swing containers: `JFrame`, `JWindow`, and `JDialog`. There is also a special class, `JApplet`, which, while not strictly a top-level container, is worth mentioning here because it should be used as the top-level of any applet that uses Swing components. Figure 12-4 shows the inheritance hierarchy of the top-level containers.

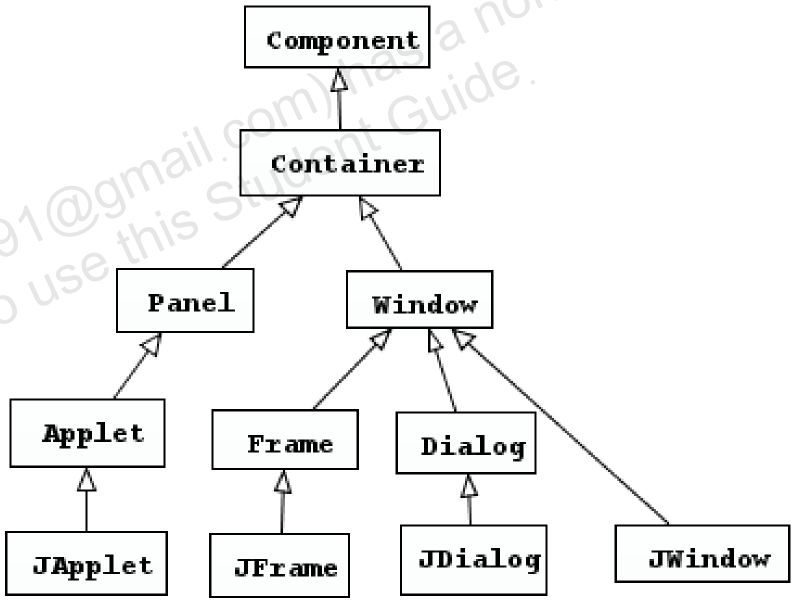

Figure 12-4 Top-Level Container Hierarchy

`JApplet`, `JFrame`, `JDialog`, and `JWindow` classes are directly derived from `Applet`, `Frame`, `Dialog`, and `Window` classes, respectively. It is important to note this because all the other Swing containers and components are derived from `JComponents`.

Swing Components

Swing GUIs use two kinds of classes: GUI classes and non-GUI support classes. The GUI classes are visual and descendants of `JComponent`, and are called `J` classes. The non-GUI classes provide services and perform vital functions for GUI classes; however, they do not produce any visual output

Swing components primarily provide components for text handling, buttons, labels, lists, panes, combo boxes, scroll bars, scroll panes, menus, tables, and trees. Swing components can be broadly classified as follows:

- Buttons
- Text components
- Uneditable information display components
- Menus
- Formatted display components
- Other basic controls

The Swing Component Hierarchy

Figure 12-5 illustrates the hierarchy relationships of the swing components.

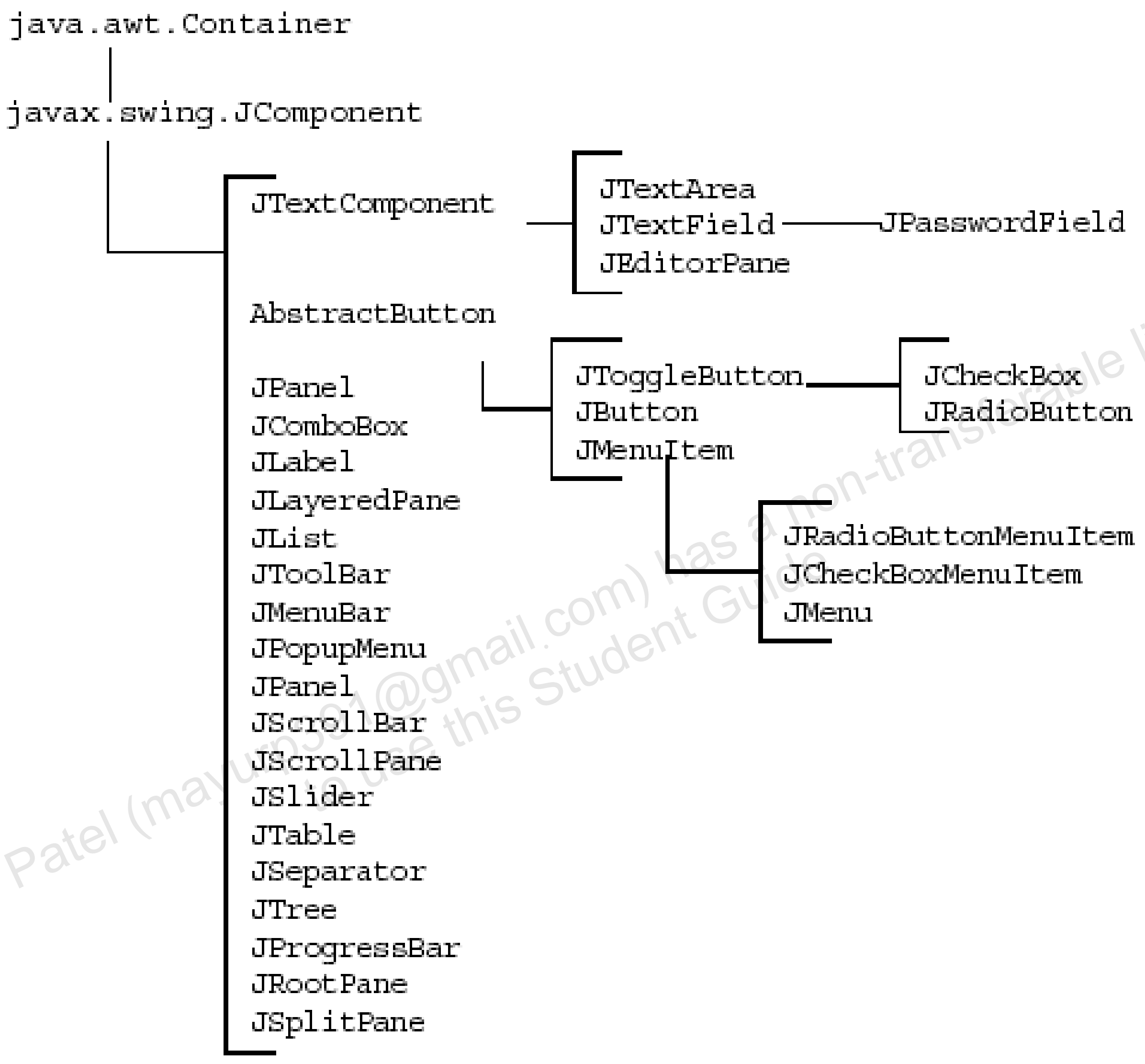

Figure 12-5 Swing Component Hierarchy

Note – The Swing components' event handling classes are examples of non-GUI classes.

Properties of Swing Components

This section describes the properties of Swing components.

Common Component Properties

All the Swing components share some common properties because they all extend the `JComponent` class. Table 12-2 shows a subset of the common properties that all Swing components inherit from `JComponent`.

Table 12-2 Common Component Properties

Property	Methods

Border	`Border getBorder()`
	`void setBorder(Border b)`

Background and foreground color	`void setBackground(Color bg)`
	`void setForeground(Color bg)`

Font	`void setFont(Font f)`
Opaque	`void setOpaque(boolean isOpaque)`

Maximum and minimum size	`void setMaximumSize(Dimension d)`
	`void setMinimumSize(Dimension d)`

Alignment	`void setAlignmentX(float ax)`
	`void setAlignmentY(float ay)`

Preferred size	`void setPreferredSize(Dimension ps)`

Note – Some properties, such as preferred size, are hints to the layout manager. Although a component can provide the layout manager with hints, the layout manager can ignore these hints and use other information to rendering the component.

Component-Specific Properties

This section discusses component properties specifically taking `JComboBox` as an example. `JComboBox` inherits all the properties in `JComponent` and defines more specific properties. Table 12-3 describes some properties specific to `JComboBox`.

Table 12-3 Component Specific Properties

Properties	**Methods**
Maximum row count	`void setMaximumRowCount(int count)`
Model	`void setModel(ComboBoxModel cbm)`
Selected index	`int getSelectedIndex()`
Selected Item	`Object getSelectedItem()`
Item count	`int getItemCount()`
Renderer	`void setRenderer(ListCellRenderer ar)`
Editable	`void setEditable(boolean flag)`

Layout Managers

A layout manager determines the size and position of the components within a container. The alternative to using layout managers is absolute positioning by pixel coordinates. Absolute positioning is achieved through setting a container's layout property to null. Absolute positioning is not platform-portable. Issues such as the sizes of fonts and screens ensure that a layout that is correct based on coordinates can potentially be unusable on other platforms.

Unlike absolute positioning, layout managers have mechanisms to cope with the following situations:

- The resizing of the GUI by the user
- Different fonts and font sizes used by different operating systems or by user customization
- The text layout requirements of different international locales (left-right, right-left, vertical)

To cope with these situations, layout managers lay out components according to a predetermined policy. For example, the policy of the `GridLayout` is to position child components in equal-sized cells, starting at the top left and working left to right, top to bottom until the grid is full. The following sections describe some of the layout managers that you can use. Each section highlights the policy used by the layout manager under discussion.

The `BorderLayout` Layout Manager

`BorderLayout` arranges the components in five different regions: CENTER, NORTH, SOUTH, EAST, and WEST. The border layout manager limits the number of components added to each region to one.

The position of the component should be specified. If no position is specified, by default the component is added to the CENTER. All the extra space left is used by the component in the CENTER.

`BorderLayout` is the default layout for `JFrame`, `JDialog`, and `JApplet`. Figure 12-6 on page 12-17 shows a display using a border layout. The display shows five `JButtons` added to a `JFrame`.

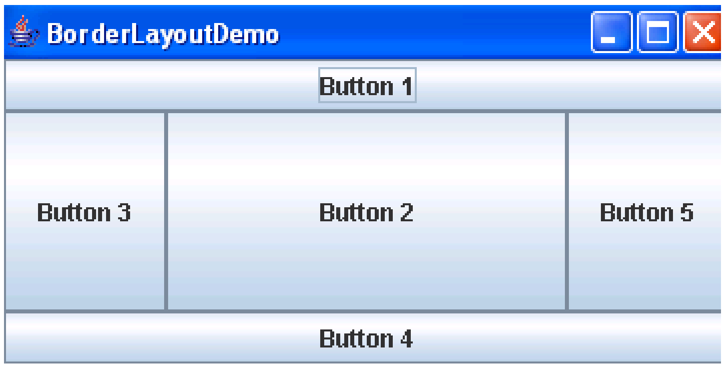

Figure 12-6 The BorderLayout Example

The code below shows the BorderLayout example. It adds five buttons to the JFrame.

Code 12-1 BorderLayout Example

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

public class BorderExample {
  private JFrame f;
  private JButton bn, bs, bw, be, bc;

  public BorderExample() {
    f = new JFrame("Border Layout");
    bn = new JButton("Button 1");
    bc = new JButton("Button 2");
    bw = new JButton("Button 3");
    bs = new JButton("Button 4");
    be = new JButton("Button 5");
  }

  public void launchFrame() {
    f.add(bn, BorderLayout.NORTH);
    f.add(bs, BorderLayout.SOUTH);
    f.add(bw, BorderLayout.WEST);
    f.add(be, BorderLayout.EAST);
    f.add(bc, BorderLayout.CENTER);
    f.setSize(400,200);
    f.setVisible(true);
  }

  public static void main(String args[]) {
        BorderExample guiWindow2 = new BorderExample();
        guiWindow2.launchFrame();
    }
}
```

The `FlowLayout` Layout Manager

`FlowLayout` arranges the components in a row. By default, it arranges the components from `LEFT_TO_RIGHT`. This orientation can be changed using the `ComponentOrientation` property `RIGHT_TO_LEFT`. The vertical and horizontal spacing between the components can be specified. If not, the default vertical and horizontal gap of five units is used. Figure 12-7 shows the flow layout example. Similar to the border layout example, five `JButtons` are added to the `JFrame`.

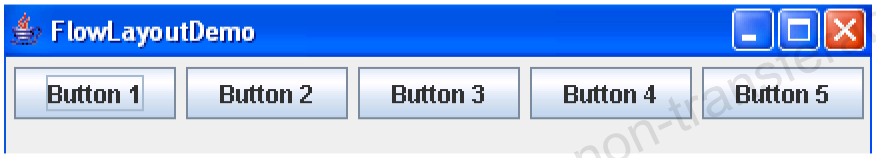

Figure 12-7 The `FlowLayout` Example

The following code shows an example of `FlowLayout`. It adds five buttons to the `JFrame`.

Code 12-2 `FlowLayout` Example

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

public class FlowExample {
    private JFrame f;
    private JButton b1;
    private JButton b2;
    private JButton b3;
    private JButton b4;
    private JButton b5;

    public FlowExample() {
        f = new JFrame("GUI example");
        b1 = new JButton("Button 1");
        b2 = new JButton("Button 2");
        b3 = new JButton("Button 3");
        b4 = new JButton("Button 4");
        b5 = new JButton("Button 5");
    }

    public void launchFrame() {
        f.setLayout(new FlowLayout());
        f.add(b1);
        f.add(b2);
        f.add(b3);
        f.add(b4);
        f.add(b5);
        f.pack();
        f.setVisible(true);
    }

    public static void main(String args[]) {
        FlowExample guiWindow = new FlowExample();
        guiWindow.launchFrame();
    }

} // end of FlowExample class
```

The BoxLayout Layout Manager

BoxLayout arranges the components either vertically or horizontally.

The BoxLayout constructor takes a parameter called axis, where the direction to align the components should be specified. This parameter can take any of the following values

- X_AXIS – Components are arranged horizontally from left to right.
- Y_AXIS – Components are arranged vertically from top to bottom.
- LINE_AXIS – Components are laid in the same direction as words in a line.
- PAGE_AXIS – Components are arranged in the same direction as the lines in a page.

Figure 12-8 shows a box layout example. Similar to the border layout example, five JButtons are added to the JFrame. The parameter used in this example for aligning the components was set to Y-AXIS.

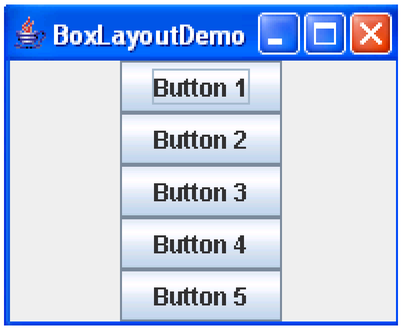

Figure 12-8 The BoxLayout Example

The CardLayout Layout Manager

CardLayout arranges the components as a stack of cards. Each card accepts a single component for display. By making this single component a container, you can display multiple components in a card. Only one card is visible at a time. Figure 12-9 illustrates the use of the card layout manager.

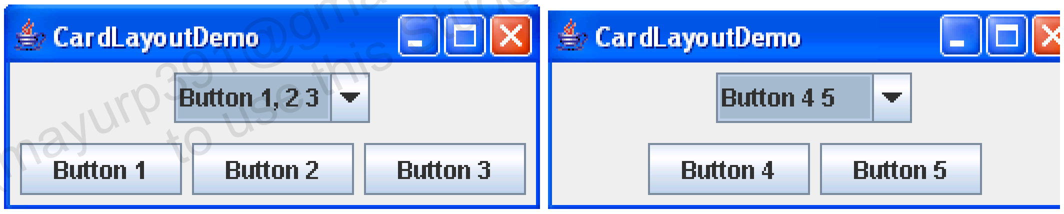

Figure 12-9 The CardLayout Example

In the example shown in Figure 12-9, Button1, Button2, and Button3 are arranged on Card1 and Button4 and Button5 are arranged on Card2. A combo box is used to select the card to be displayed.

The GridLayout Layout Manager

GridLayout arranges the components in rows and columns. Each component occupies the same amount of space in the container. When creating the grid layout, the number of rows and columns should be specified. If not specified, by default, the layout manager creates one row and one column. The vertical gap and the horizontal gap between the components can also be specified. Figure 12-10 illustrates the use of the GridLayout. Similar to the border layout shown in Figure 12-6 on page 12-17, five JButtons are added to the JFrame.

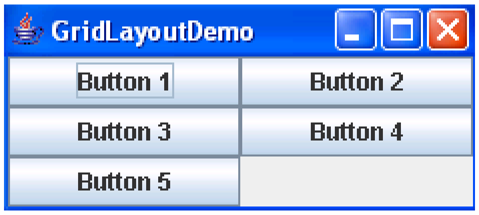

Figure 12-10 The GridLayout Example

The following code shows an example of GridLayout. It adds five buttons to the JFrame.

Code 12-3 GridLayout Example

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

public class GridExample {
  private JFrame f;
  private JButton b1, b2, b3, b4, b5;

  public GridExample() {
    f = new JFrame("Grid Example");
    b1 = new JButton("Button 1");
    b2 = new JButton("Button 2");
    b3 = new JButton("Button 3");
    b4 = new JButton("Button 4");
    b5 = new JButton("Button 5");
  }

  public void launchFrame() {
    f.setLayout (new GridLayout(3,2));

    f.add(b1);
    f.add(b2);
    f.add(b3);
    f.add(b4);
    f.add(b5);

    f.pack();
    f.setVisible(true);
  }

  public static void main(String args[]) {
    GridExample grid = new GridExample();
    grid.launchFrame();
  }
}
```

The GridBagLayout Layout Manager

GridBagLayout arranges the components in rows and columns, similar to grid layout, but provides a wide variety of flexibility options for resizing and positioning the components. This layout is used to design complex GUIs. The constraints on the components are specified using the GridBagConstraints class. Some of the constants in this class are gridwidth, gridheight, gridx, gridy, weightx, and weighty. Figure 12-11 illustrates the use of GridBagLayout. Five JButtons are added to the JFrame. You can notice that the components are of different size and are positioned at very specific locations.

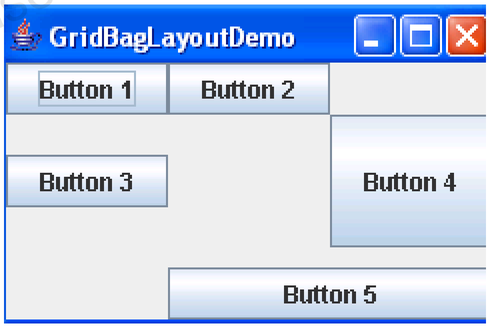

Figure 12-11 The GridBagLayout Example

The GroupLayout Layout Manager

In addition to these layout managers, `GroupLayout` was added to Java SE version 6. This layout manager was added for use by toolmakers. It is the layout manager used in the `NetBeans` IDE GUI builder tool. For more information see:
http://java.sun.com/docs/books/tutorial/uiswing/layout/group.html.

GUI Construction

A Java technology GUI can be created using either of the following techniques:

- Programmatic construction

  This technique uses code to create the GUI. This technique is useful for learning GUI construction. However, it is very laborious for use in production environments.

- Construction using a GUI builder tool

  This technique uses a GUI builder tool to create the GUI. The GUI developer uses a visual approach to drag-and-drop containers and components to a work area. The tool permits the positioning and resizing of containers and components using a pointer device such as a computer mouse. With each step, the tool automatically generates the Java technology classes required to reproduce the GUI.

Programmatic Construction

This sections describes creating a simple GUI that prints Hello World. The code shown in Code 12-4 creates a container `JFrame` with a title HelloWorldSwing. It later adds a `JLabel` with the Accessible Name property set to Hello World.

Code 12-4 The `HelloWorldSwing` Application

```
import javax.swing.*;
public class HelloWorldSwing {
    private static void createAndShowGUI() {
        JFrame frame = new JFrame("HelloWorldSwing");
        //Set up the window.
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JLabel label = new JLabel("Hello World");
        // Add Label
        frame.add(label);
         frame.setSize(300,200);
        // Display Window
        frame.setVisible(true);}

    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            //Schedule for the event-dispatching thread:
            //creating,showing this app's GUI.
```

```
                  public void run() {createAndShowGUI();}
              });
          }
      }
```

Figure 12-12 shows the GUI interface that is generated by the code. The default layout for `JFrame` is `BorderLayout`. So, by default, the component `JLabel` is added to the center position of the container. Also note that the label occupies the entire frame, as the center component occupies all the remaining space in the container in the `BorderLayout`.

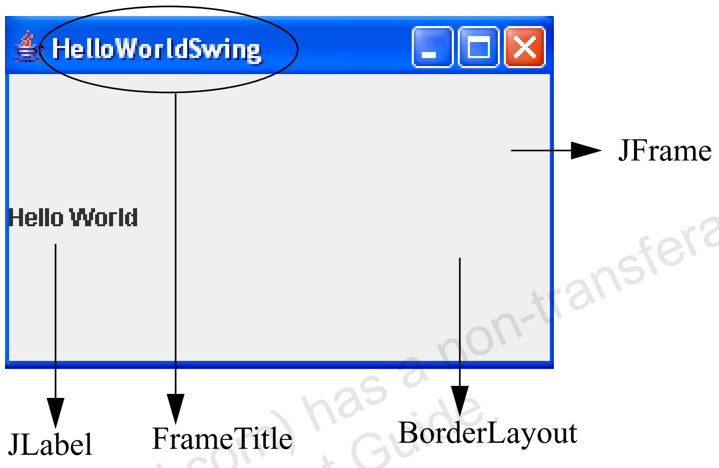

Figure 12-12 The `HelloWorldString` Output

Key Methods

This section explains the key methods used in Code 12-4 on page 12-24. The methods can be divided into two different categories.

1. Methods for setting up the frame and adding a label.

   a. `setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)`: This method defines the behavior of the `JFrame` when a close operation is initiated. There are four possible ways of handling this.

      1. `DO_NOTHING_ON_CLOSE`: Does nothing when the close operation is initiated. This constant is defined in `WindowsConstants`.

      2. `HIDE_ON_CLOSE`: Invokes any `WindowListener` objects and hides the frame. This constant is defined in `WindowConstants`.

3. `DISPOSE_ON_CLOSE`: Invokes any `WindowListener` objects and hides and disposes the frame. This constant is defined in `WindowConstants`.
4. `EXIT_ON_CLOSE`: Exits the application. This constant is defined in the `JFrame` class.

b. `setVisible(true)`: When the `JFrame` is first created, it creates an invisible frame. To make the frame visible, the parameter for the `setVisible` should be set to true. `JFrame` inherits this method from `java.awt.Component`.

c. `add(Component c)`: This method adds the components to the container. `JFrame` inherits this method from `java.awt.Component` class. Five different overloaded methods are defined in the `java.awt.Component`.

2. Methods for making the GUI thread-safe and efficient.

   Several tasks are involved in displaying the GUI efficiently. These tasks can be broadly defined as:

   a. Executing the application code.

      This task involves starting the GUI application and executing the code for rendering the GUI

   b. Handling the events raised from the GUI:

      Several events can be raised by the components in the GUI. For example, when a button is pressed an event is generated. Event listeners should be defined to handle this event. This task dispatches the event to the appropriate listeners that handle the event.

c. Handle some time-consuming processes:

   Several activities might be time-consuming and they can be run in the background so that the GUI would be efficient and responsive. These kinds of activities are handled by this task.

To handle these tasks efficiently, the Swing framework uses threads that are light-weight processes. The tasks described above can be handled by these threads separately and concurrently. The programmer should utilize these threads. The Swing framework provides a collection of utility methods in the `SwingUtilities` class.

- `SwingUtilites.invokeLater(new Runnable())`:

  In the Java programming language, threads are created using the `Runnable` interface. This interface defines a method `run` that should be implemented by all the classes using this interface. The `invokeLater` method schedules the GUI creation task to execute the `run` method asynchronously by the event-handling thread after all the pending events are completed.