Graphical User Interface Programming in Java

by DR DANNY POO
www.DrDannyPoo.com

Graphical User Interface Programming in Java

Community Edition

Danny Poo

Copyright © 2020, Danny Poo.

ALL RIGHTS RESERVED
No part of this work covered by the copyright hereon may be reproduced or used in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, web distribution or information storage and retrieval systems—without the written permission of the author.

PREFACE

Who Should Read This Book

This book teaches the development of graphical user interface design in Java. Readers are assumed to have prior knowledge in Java programming. "Graphical User Interface Programming in Java" contains substantial resources to help readers learn graphical user interface programming in a systematic and fundamentals-first way.

Topics Covered

Topics covered include: the Java Graphical User Interface (GUI) model, windows, frames, layout managers, panels, components, text fields, buttons, lists, dialogs, menus, and event handling in interactive user interface design. This text may be used as a basic text for teaching GUI programming in Java or combined with my other book "Java Programming" for a more complete reading of Java and its technologies.

How This Book Is Organized

This book is organized into nine chapters.

Chapter 1: Java Graphical User Interface Model

The delivery of this book begins with a brief on the Java Graphical User Interface (GUI) Model. The provision of GUI is facilitated by a set of Application Programming Interfaces (APIs) known as the Abstract Window Toolkit (AWT). The AWT is the beginning of GUI programming in Java. Although the introduction of the Java programming architecture proves to be a success in fulfilling the "Write Once, Run Anywhere" promise that Java was first conceptualized, it did not fair so well when it comes to the look-and-feel design of a GUI. An application's GUI developed in a Microsoft Windows environment does not have the Microsoft's GUI look-and-feel; neither does a Java application developed for the Apple IOS's operating environment have a look-and-feel that is of Apple's graphical application. To address this need, the Java Swing APIs were devised and made available in Java 2 (yes, it has been a while since its launch). The design of this book is based largely on the Java Swing APIs.

Chapter 2: Windows, Layout Managers and Panels

Chapter 2 discusses the concept of windows. A window is an opening for users (like you and me) to gain access into an application program. It exists within a windowing system that provides the mechanism for multiple applications to share a computer's graphical display resources simultaneously. With windows, a user can interact with each application and switch from one application to another without the need to re-initialize the application.

Chapter 3: User Interactions and Event Handling

Chapter 3 discusses the event-handling model for managing user interactions with the GUI components. This chapter demonstrates the development of a working calculator using the Java Swing architecture.

Chapter 4: Labels, Buttons and Combo Boxes

Chapter 4 looks at common components used in GUI design. In particular, it examines labels (display areas for short texts, images or both), buttons (components that trigger action events when clicked) and combo-boxes (components that allow users to choose an item from a list).

Chapter 5: Text Components

Chapter 5 highlights the importance of handling text inputs and outputs in GUI design. Text fields and text areas are two text components that are commonly used in graphical user interface design. This chapter discusses the various types of text components supported in the Java Swing. Specifically, they include: text fields, text areas, formatted text fields, password fields, editor pane and text pane.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

Chapter 6: Viewing Contents Through Panes
Chapter 6 elaborates on the concept of pane, a component that is tightly connected with frame (or window). This chapter covers scroll pane (for facilitating scrolling in viewing parts of a content), split pane (for facilitating splitting of a pane into multiple sections), tabbed pane (for facilitating multiple panes for containing groups of components), internal frame (for facilitating a frame within a frame), desktop pane (for facilitating a container for creating virtual desktop), and layered pane (for facilitating the positioning of components in various depth dimensions).

Chapter 7: Editing Styled Texts in Editor Panes
There are two other types of pane supported in the Java Swing: Editor Panes and Option Panes. Editor Panes are used for editing styled texts and are covered in Chapter 7.

Chapter 8: Dialogs
Option Panes are used in standard dialog boxes and are covered in Chapter 8.

Chapter 9: Menus
This edition of the book ends with Chapter 9 when we discuss the concept of menus. As you may be aware, a menu presents a list of functions an application supports and allows users to choose from this list the required function to execute. Concepts discussed include: menu, menu bar, menu item, submenu, and popup menu.

Enjoy!

Dr Danny Poo
www.DrDannyPoo.com

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

ABOUT THE AUTHOR

Dr. Danny Poo brings with him 40 years of Software Engineering and Information Technology and Management experience. A graduate from the University of Manchester Institute of Science and Technology (UMIST), England, Dr. Poo is currently an Associate Professor at the Department of Information Systems and Analytics, National University of Singapore.

A well-known speaker in seminars, Dr. Poo has conducted numerous in-house training and consultancy for organizations, both locally and regionally. His notable teaching credentials include • Data Strategy • Data StoryTelling • Data Visualisation • Big Data Analytics • Machine Learning • Data Management • Data Governance • Data Architecture • Capstone Projects for Business Analytics • Software Engineering • Server-side Systems Design and Development • Information Technology Project Management • Health Informatics • Healthcare Analytics • Health Informatics Leadership.

Dr. Poo has also published extensively in conferences and journals on Software Engineering and Information Management.

Dr. Poo was the founding Director of the Centre for Health Informatics. This Centre provides courses to train healthcare professionals in Health Informatics. Dr. Poo is instrumental in developing curriculum and courses for this Centre. In particular, he has delivered numerous rounds of Health Informatics course since 2012 and has trained as many as 1000 healthcare and IT professionals on this subject. Besides, he teaches a course on Healthcare Analytics to healthcare professionals since it started in May 2015. To date, he has run fourteen 3-full-days-sessions of this course since May 2015. This course continues to receive great interest from participants.

Dr. Poo is the author of four books – "Learn to Program Java", "Learn to Program Enterprise JavaBeans 3.0", "Object-Oriented Programming and Java", and "Learn to Program Java User Interface".

Dr. Poo has also consulted for these companies • Deutsche Bank • Gemplus • Micron • NCR • PIL • PSA • Rhode-Schwarz • Standard Chartered Bank • Singapore Technologies Electronic • Monetary Authority of Singapore (MAS) • Infocomm Development Authority (IDA) • National Library Board (NLB) • Ministry of Manpower (MOM) • Nanyang Technological University (NTU) • Nanyang Polytechnic (NYP) • National University Hospital.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

CHAPTER 1: JAVA GRAPHICAL USER INTERFACE MODEL

Java was born out of a necessity – the necessity to run a computer program on any operating systems. Traditionally, a computer program written for a particular operating system can only be executed on that operating system. For example, suppose you wrote a computer program using the C programming language, when the program is compiled on a Microsoft Windows operating system, that program can only run on a similar Microsoft Windows operating system on another machine. It is not possible for you to run that same program on a computer installed with say, Apple Macintosh operating system.

In 1995, when Sun Microsystems (Note: Sun Microsystems is now part of Oracle) launched Java, programs written in the Java programming language can be executed on any known operating system, so long as a piece of software known as the Java Virtual Machine is installed on the computer. This notion of "Write Once, Run Anywhere" has made Java programs extremely portable. While this is certainly an advantage and a step forward in portable software, the solution is only partial.

The next challenge facing Sun Microsystems software engineers is the windowing system. While you are able to run a Java program on any platform without changing any of the computational code, you will have to rewrite the graphical user interface (GUI) code to match the windowing system of the target operating platform. What is lacking is a solution where a Java GUI component (such as a drop-down list) written for the Microsoft Windows platform, when executed in a Unix environment, should look like a Motif (Note: Motif is the graphical user interface windowing system for Unix systems) drop-down list (and not a Microsoft Windows drop-down list), or when it is run on an Apple Macintosh operating system, should look like a Macintosh drop-down list, etc. To solve this problem, Sun Microsystems produced the Abstract Window Toolkit (AWT).

AWT is a collection of application programming interfaces (APIs) that enables Java programs to integrate into the native desktop window system. AWT forms the base for a richer and more extensible graphical user interface component library with a pluggable look-and-feel.

In the Java 2 (JDK 1.2) release, the component library, known as the Java Swing, was introduced. With Java Swing's pluggable look-and-feel capability, you can design GUI components for one operating system but automatically have the look-and-feel of another operating platform when the same GUI components are run on the target system.

Together, AWT and Java Swing form the core of the Java Foundation Classes (JFC). The latter consists of a set of APIs Java programmers can use to develop their applications.

Besides AWT and Java Swing, the JFC also includes APIs for advanced two-dimensional graphics, imaging, text and printing (the Java 2D), for ensuring an application is accessible to users with disabilities (i.e. accessibility), and for creating applications that can interact with users around the world using the user's own language (i.e. internationalization).

The JFC enables you to build fully functional GUI interfaces that can run on any machine that support the Java 2 platform, including Solaris, Unix, Linux, Microsoft Windows, and the Apple Macintosh operating system.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

1.1 Abstract Window Toolkit (AWT)

The Abstract Window Toolkit (AWT) is Sun Microsystems' attempt at providing GUI programming for any operating system platform in Java. Basically, it consists of a set of APIs that support:

1. The creation of graphical user interface components such as buttons, labels, checkboxes, scrollbars, windows, menus, and etc.
2. Event handling that manages the events fired by some GUI components.
3. Graphics and imaging tools.
4. Layout managers for handling the layouts of components on windows independent of window size and screen resolution.
5. Data transfer such as cut-and-paste utilities through the native operating platform clipboard.

1.1.1 The AWT Packages

The AWT consists of a number of packages:

Package	Provide Interfaces and Classes for
`java.awt`	Creating user interface components and for painting graphics and images.
`java.awt.color`	Color spaces.
`java.awt.datatransfer`	Transferring data between and within applications.
`java.awt.dnd`	Transferring information between two entities logically associated with presentation elements in the graphical user interface.
`java.awt.event`	Dealing with events generated by some GUI components.
`java.awt.font`	Font manipulation.
`java.awt.geom`	Defining and performing operations on objects related to two-dimensional geometry. The classes are commonly known as the Java 2D classes.
`java.awt.im`	The input method framework.
`java.awt.im.spi`	Enabling the development of input methods that can be used with any Java runtime environment.
`java.awt.image`	Creating and modifying images.
`java.awt.image.renderable`	Producing rendering-independent images.
`java.awt.print`	General printing.

1.2 Java Swing

The Java Swing is an extension of the AWT. It implements GUI components found in the AWT (such as buttons, labels, and scrollbars) and other high-level components (such as tree view, list box, and tabbed panes). Implemented entirely in the Java programming language, the Java Swing is based on the JDK 1.1 Lightweight User Interface Framework.

1.2.1 The Java Swing Packages

The Java Swing consists of the following packages:

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

Package	Provide Interfaces and Classes for
javax.swing	Creating user interface components that can work the same on any operating platform.
javax.swing.border	Drawing specialized borders around a Java Swing component.
javax.swing.colorchooser	Choosing color. These interfaces and classes are used by the javax.swing.JColorChooser component.
javax.swing.event	Dealing with events generated by some Java Swing GUI components.
javax.swing.filechooser	Choosing file. These interfaces and classes are used by the javax.swing.JFileChooser component.
javax.swing.plaf	Providing the pluggable look-and-feel capabilities i.e. the same graphical user interface component appears the same in any operating platform using the same code.
javax.swing.plaf.basic	Building user interface objects according to the Basic look-and-feel.
javax.swing.plaf.metal	Building user interface objects according to the default Java look-and-feel codenamed *Metal*.
javax.swing.plaf.multi	Building user interface objects that combine two or more look-and-feels.
javax.swing.plaf.synth	Building user interface objects that has a *Synth* look-and-feel. *Synth* is a skinnable look-and-feel that all painting is delegated.
javax.swing.table	Dealing with by the javax.swing.JTable component.
javax.swing.text	Dealing with editable and non-editable text components.
javax.swing.text.html	Creating HTML text editors. It contains the class HTMLEditorKit.
javax.swing.text.html.parser	Parsing HTML texts. It contains the default HTML parser.
javax.swing.text.rtf	Creating Rich-Text-Format text editors. It contains the class RTFEditorKit.
javax.swing.tree	Dealing with the javax.swing.JTree component.
javax.swing.undo	Developers to provide support for undo/redo in applications such as text editors.

Note that Java Swing packages begin with "javax" and not "java".

1.2.2 The Java Swing GUI Components

The Java Swing includes some GUI components that are similar to those found in the AWT. For example, a button in AWT is implemented using the java.awt.Button class. There is a corresponding class (the javax.swing.JButton class) in the Java Swing for representing a button. Although these two classes achieve the same purpose of creating and manipulating a button, there is a difference in the look-and-feel of the buttons created by these two classes. Figure 1.1 shows two buttons. The button on the left frame is created using the Java Swing button class while the button on the right frame is created using the AWT button class.

FIGURE 1.1: A Java Swing Button and an AWT Button in a Frame

Code 1.1 produces the Java Swing button while Code 1.2 produces the AWT button. We will not explain the code here but we will discuss buttons later in Chapter 4.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

Code 1.1: A Java Swing Button in a Frame

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

class SwingButton extends JFrame {

  // Create button objects
  JButton swingButton = new JButton("Swing Button");

  // Create panel object
  JPanel panel = new JPanel();

  SwingButton() {
    // get content pane of frame
    Container contentPane = getContentPane();

    // add button to panel
    panel.add(swingButton);

    // add panel to contentPane
    contentPane.add(panel);
  }

  public static void main(String argv[]) {

    // Create frame
    SwingButton frame = new SwingButton();
    frame.setTitle("Swing Button Test");
    frame.setLocation(300, 100);

    //Display frame
    frame.setSize(250, 100);
    frame.setVisible(true);
  }
}
```

Code 1.2: An AWT Button in a Frame

```
import java.awt.*;

import java.awt.*;

class AWTButton extends Frame {

  // Create button objects
  Button awtButton = new Button("AWT Button");

  // Create panel object
  Panel panel = new Panel();

  AWTButton() {
    panel.add(awtButton);
    this.add(panel);
  }

  public static void main(String argv[]) {

    // Create frame
    AWTButton frame = new AWTButton();
    frame.setTitle("AWT Button Test");
    frame.setLocation(300, 100);

    //Display frame
    frame.setSize(250, 100);
    frame.setVisible(true);
  }
}
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

With the advent of Java 2, developers are likely to use the Java Swing GUI components. We will therefore focus our discussion on the Java Swing components instead of the AWT components.

1.3 Components and Containers

All graphical user interface objects in Java (be it an object from AWT or Java Swing) are ***components***. A *component* is a graphical and displayable object that is capable of interacting with the user. A typical example of a component is the button which we introduced earlier. Other examples include scrollbars, checkboxes, radio-buttons, text areas, tabbed panes, and etc.

A *container* is a generic AWT component that can contain other AWT components. The class hierarchy in Figure 1.2 shows the relationship between java.awt.Component and java.awt.Container. A Java Swing component (javax.swing.JComponent) is defined as a direct subclass of java.awt.Container suggesting that a Java Swing component can behave as an AWT component and container.

```
java.lang.Object
└──    java.awt.Component
    └──java.awt.Container
        └──    java.swing.JComponent
```

FIGURE 1.2: Class Hierarchy of Container

1.3.1 Containment Hierarchy

You can think of a container as a receptacle for holding objects (or components). To hold objects, a container uses a *list* to keep track of its components which are only displayable on a screen if they are included in a *containment hierarchy*. The latter is a tree-like, hierarchical structure for organizing components. At the root of the containment hierarchy is a *top-level container* (see Figure 1.3).

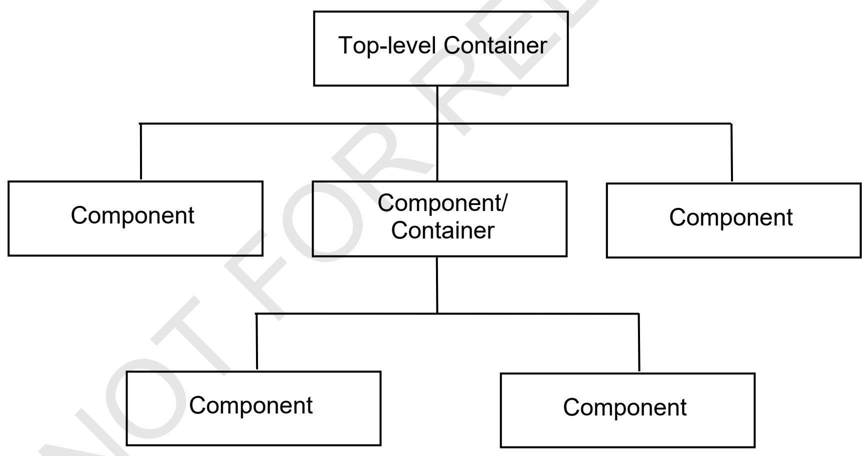


FIGURE 1.3: Containment Hierarchy

There are three top-level containers in the Java Swing: javax.swing.JFrame (Note: A frame is a top-level window with a title and border), javax.swing.JDialog (Note: A dialog is similar to a frame in that it is also a top-level window with a title and border. However, it has an additional characteristic of being able to take in some form of input from the user), and javax.swing.JApplet (Note: An applet is a small program run within another application). The class hierarchy for these classes is given in Figure 1.4, Figure 1.5, and Figure 1.6 respectively. Note that there is an AWT equivalent for each top-level container in Java Swing.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
java.lang.Object
└── java.awt.Component
    └── java.awt.Container
        └── java.awt.Window
            └──java.awt.Frame
                └──javax.swing.JFrame
```

FIGURE 1.4: Class Hierarchy of JFrame

```
java.lang.Object
└── java.awt.Component
    └── java.awt.Container
        └── java.awt.Window
            └──java.awt.Dialog
                └──javax.swing.JDialog
```

FIGURE 1.5: Class Hierarchy of JDialog

```
java.lang.Object
└── java.awt.Component
    └── java.awt.Container
        └── java.awt.Panel
            └──java.applet.Applet
                └──javax.swing.JApplet
```

FIGURE 1.6: Class Hierarchy of JApplet

1.3.2 Content Pane

Each top-level container has a *content pane* for containing GUI components. All components in the content pane are displayable on the screen. A menu bar can also be added within a top-level container but it is placed outside the content pane. For example, in Figure 1.7, a JFrame top-level container is shown with a content pane and a menu bar in a containment hierarchy. The content pane has two components – a button and a label – added into it. As explained earlier, all components (e.g. button, label and menu bar) in the containment hierarchy are displayable on the screen.

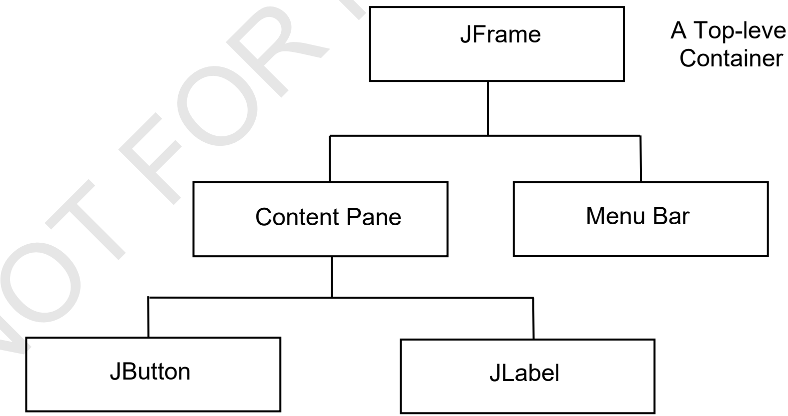


FIGURE 1.7: A JFrame Containment Hierarchy

Figure 1.8 is a picture of a Java Swing frame (the top-level container) created with a menu bar and an empty content pane.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

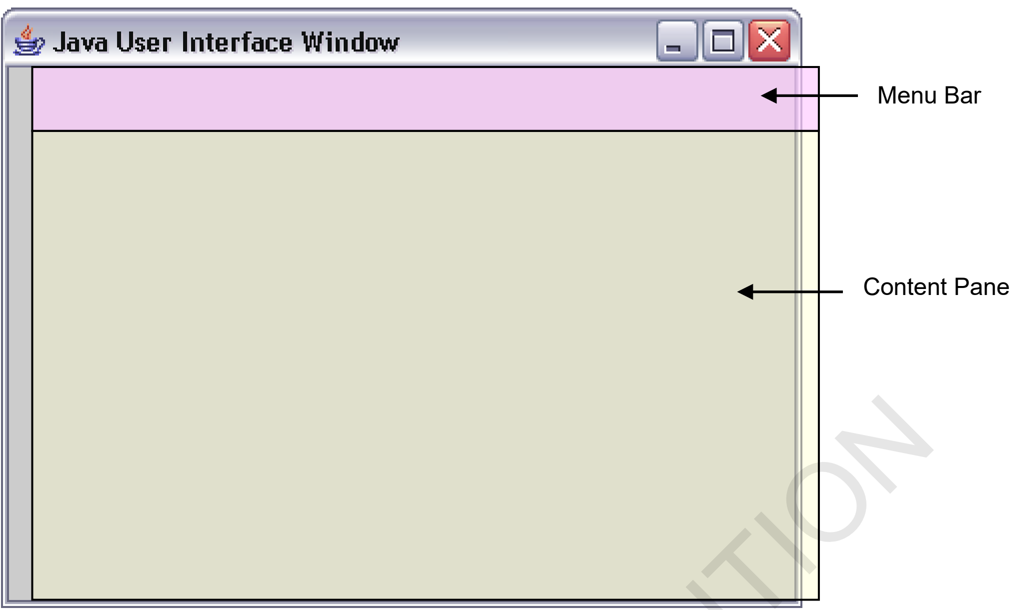


FIGURE 1.8: A Frame with a Menu Bar and Content Pane

1.3.3 Adding a Component into Containers

A GUI component can only be added into one container at any one time. If a GUI component has already been added into a container and if you try to add it into another container, it would be removed from the first container before it is added to the second container. To illustrate, let us suppose we have buttonA added into panel1 and later we add buttonA to panel2; displaying the two panels would reveal that buttonA appears in panel2 but not in panel1.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

Chapter 2: Windows, Layout Managers and Panels

A window is an opening for users (like you and me) to gain access into an application program. It exists within a windowing system that provides the mechanism for multiple applications to share a computer's graphical display resources simultaneously. With windows, a user can interact with each application and switch from one application to another without the need to re-initialize the application. In other words, windows make you do your work a lot more easily.

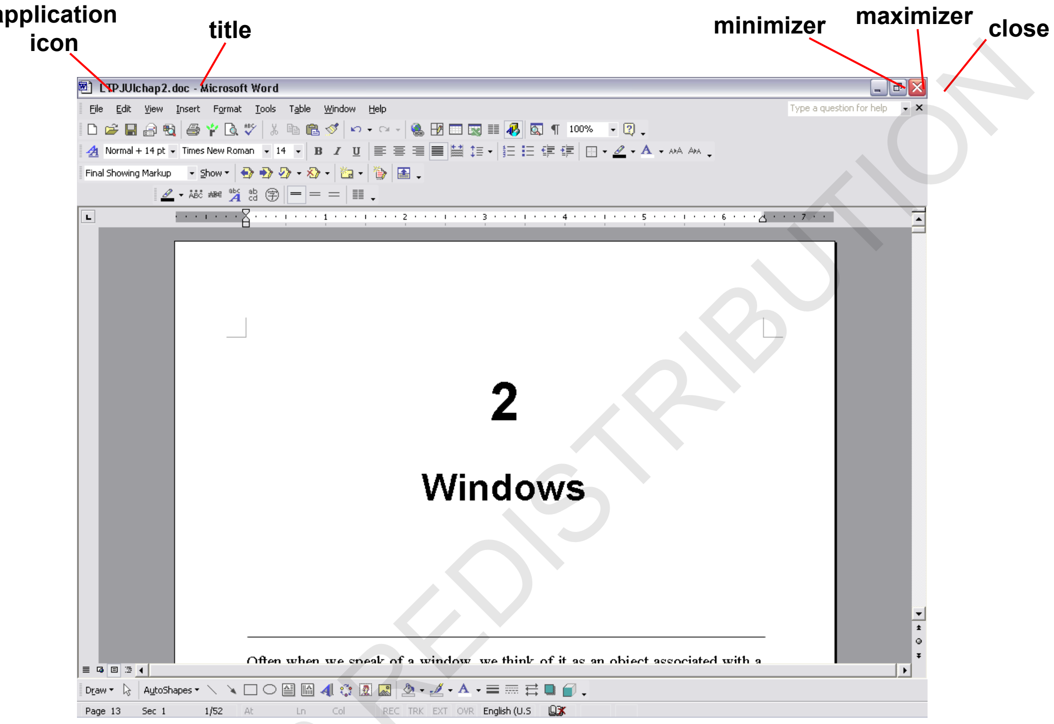


FIGURE 2.1: A Window

Figure 2.1 is an example of the ubiquitous Microsoft Word application window. Using a similar window, this text document is produced. The window in Figure 2.1 contains some common icons that are of interest to us: minimizer, maximizer, close, and application icons. How do we create windows with the Java Swing? We will show you how in this chapter.

NOT FOR REDISTRIBUTION

2.1 Creating Windows (or Frames)

A window in Java can be as simple as the one shown in Figure 2.2 or a bit more decorated as shown in Figure 2.3. Figure 2.2 shows a window with only one component – a button labeled "Close window" and Figure 2.3 shows a window with three window icons, an application icon and a title. Both windows are created using the javax.swing.JFrame class in the Java Swing.

A window is known as a *frame*[1] in the Java Swing. Henceforth, "Window" and "Frame" will be used interchangeably to refer to the same thing in this book.

[1] A frame is a top-level window with a title and border.

Copyright © 2020, Danny Poo.

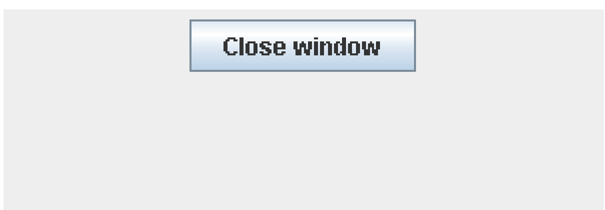


FIGURE 2.2: A Java Window

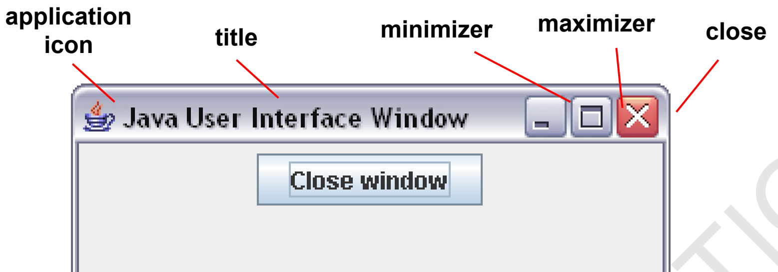


FIGURE 2.3: A More Familiar Java Window

2.1.1 Creating Our First Window

Code 2.1 is our first attempt at creating frames (or windows).

Code 2.1: Frame1

```
import javax.swing.JFrame;

class Frame1 {

  public static void main(String argv[]) {
    JFrame frame = new JFrame("Java User Interface Window");
    frame.setSize(400, 300);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setLocation(300, 100);
    frame.setVisible(true);
  }
}
```

Let us examine Code 2.1:

1. A frame is created from the javax.swing.JFrame class. This class must be imported into the application program. Alternatively, we can import the entire javax.swing package like import javax.swing.*; .
2. Instantiates a new JFrame object with the title “Java User Interface Window”.
3. Four different methods are then called to set some values on the frame. The setSize() method defines the size of the frame (in this case, 400 by 300 pixels). The setDefaultCloseOperation() method sets the operation that will happen by default when the user initiates a “close” [2] on the frame. The operation that we have set is JFrame.EXIT_ON_CLOSE which forces the application to exit using the System exit method.
4. By default, Java aligns all frames to the coordinate (0, 0) – the top left-hand corner of the display screen. For our example, we want the frame to align itself to the coordinate (300, 100). We do this via the setLocation() method. This method is inherited from the javax.awt.Component class. We will discuss the Component class later in the book.
5. A frame by default is invisible. Therefore, if we want the frame to be visible, we have to explicitly set it via the setVisible() method. This method takes in a parameter and the latter must be set to true for the frame to be visible, otherwise, set it to false if you do not want the frame to appear.

[2] When the user clicks on the “close” icon on the top right-hand corner of a window.

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

Figure 2.4 is the result of executing Code 2.1.

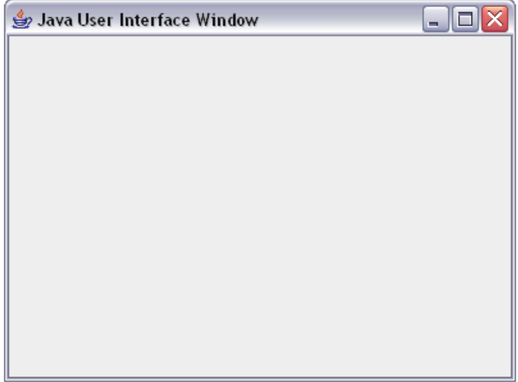


FIGURE 2.4: A Frame as a Window

2.1.2 The JFrame Class

The JFrame class has four constructors as shown in Table 2.1.

TABLE 2.1: JFrame Constructors

No.	Constructor	Description
1	`JFrame()`	Creates a new and invisible JFrame object.
2	`JFrame (GraphicsConfiguration gc)`	Creates a new and invisible JFrame object in the specified GraphicsConfiguration of a screen device and a blank title.
3	**`JFrame (String title)`**	**Creates a new and invisible JFrame object with the specified title.**
4	`JFrame (String title, GraphicsConfiguration gc)`	Creates a new and invisible JFrame object with the specified title and the specified GraphicsConfiguration of a screen device.

Code 2.1 makes use of the third constructor to include a title ("Java User Interface Window") to the frame.

The JFrame class is a subclass of the java.awt.Frame class which is also a subclass of the java.awt.Window class as shown in Figure 2.5. This explains why a frame is also a window.

```
java.lang.Object
└── java.awt.Component
     └── java.awt.Container
          └── java.awt.Window
               └── java.awt.Frame
                    └── javax.swing.JFrame
```

FIGURE 2.5: Class Hierarchy of JFrame

2.2 Adding Components into Windows (or Frames)

The frame that we have created so far is plain and does not provide any utility. Let us now add components into a frame.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

2.2.1 Add a Button

A typical component that we can add into a frame is a button which is a component that allows users to initiate a command when pressed or clicked. It is implemented in the Java Swing by the javax.swing.JButton class. To add a component into a frame, we need to do the following:

1. Get the Content Pane associated with the frame
2. Add the component into the Content Pane

So, to add a button into a frame, we have to:

1. Get the Content Pane associated with the frame
2. Add the button into the Content Pane

A *content pane* can be thought of as a container to store all the displayable components on a frame. There is a content pane for each frame. A default content pane is automatically created and associated with a frame when the latter is instantiated. Besides content panes, we can also add menu bars[3] into frames. Visually, Figure 2.6 shows the relationship among content pane, menu bar and frame.

To illustrate, let us add a button labeled "OK" into a frame. Code 2.2 is an implementation:

1. A javax.swing.JButton object is instantiated. "OK" labels the button.
2. There are two parts in the statement frame.getContentPane().add(button);. The first part frame.getContentPane() gets the content pane while the second part add(button) adds a button into the content pane via the latter's add() method. As mentioned earlier, only items added into a frame's content pane are displayable provided the frame is set to be visible.
3. Note that the statement frame.setVisible(true) is included at the end of the code. The position in which this statement is included in the code is significant. *Any component added into the frame after this statement is not displayable*[4].

Code 2.2: Adding a Button into a Frame

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

class Frame2 {

  public static void main(String argv[]) {
    JFrame frame = new JFrame("Java User Interface Window");
    frame.setSize(400, 300);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setLocation(300, 100);

    // add components
    JButton button = new JButton("OK");
    frame.getContentPane().add(button);

    // show frame
    frame.setVisible(true);
  }
}
```

NOT FOR REDISTRIBUTION

[3] We will discuss Menu Bar and its related entities in Chapter 9.

[4] Readers may want to verify this by shifting `frame.setVisible(true)` to other part of the code above the statement `frame.getContentPane().add(button)`.

Copyright © 2020, Danny Poo.

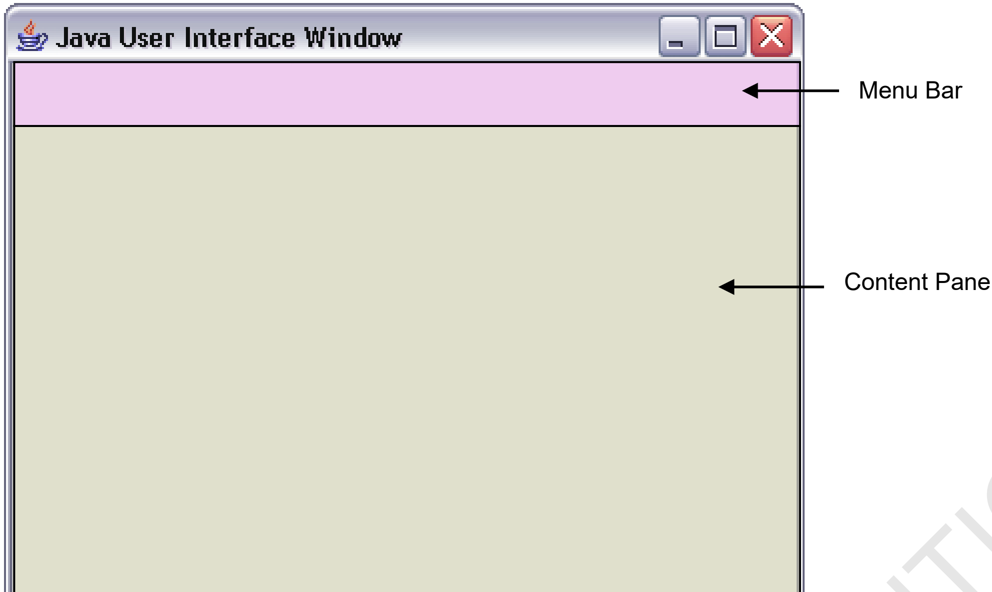


FIGURE 2.6: Content Pane, Menu Bar and Frame

The output from Code 2.2 is shown in Figure 2.7.

Notice the "OK" button fills the entire content pane and since there is no menu bar associated with the frame, the entire frame. Why is that so? Obviously, this is not what we want! How do we remedy this? We will return to this topic when we discuss Panel later in this chapter.

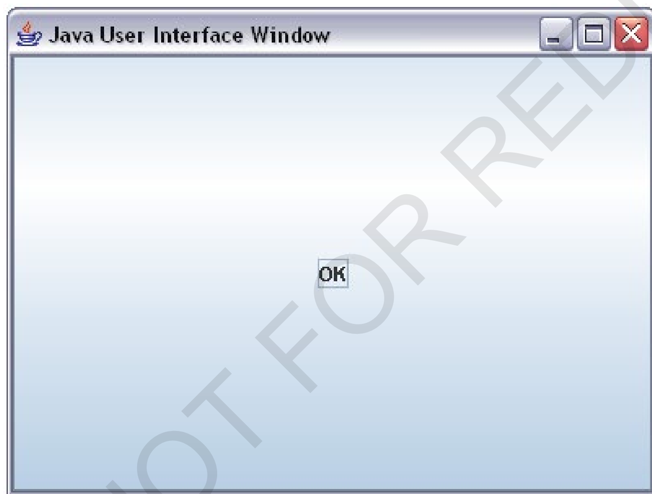


FIGURE 2.7: Frame with a Button

2.2.2 Components and Containers

Every item in a graphical user interface (GUI) is a component by definition. A component may be contained within a container. We have seen earlier that a frame is a container – the containment of its components is realized via its content pane which is a container – as well as a component. By means of the inheritance hierarchy in Figure 2.5, *all containers are components*.

Although a frame, by definition, is a component, it is defined as a top-level container in the Java Swing. Therefore, a frame is generally not used as a component.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

2.2.3 Adding Multiple Components into a Frame

Previously, we added one button into a frame. In the next example, we will add two buttons labeled “Button 1” and “Button 2” into a frame. Code 2.3 illustrates. This code is very similar to Code 2.2. We have basically added some statements to add the additional components (see Lines 10 to 13). The output for Code 2.3 is shown in Figure 2.8.

Code 2.3: Adding Two Buttons

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

class Frame3 {

  public static void main(String argv[]) {
    JFrame frame = new JFrame("Java User Interface Window");
    frame.setSize(400, 300);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setLocation(300, 100);

    // add components
    JButton button1 = new JButton("Button 1");
    frame.getContentPane().add(button1);
    JButton button2 = new JButton("Button 2");
    frame.getContentPane().add(button2);

    // show frame
    frame.setVisible(true);
  }
}
```

Did you notice anything strange about Figure 2.8? The last component added is displayed and any component added earlier (such as Button 1) is not visible and is therefore lost. How could we display all the components added regardless of the sequence the components have been added into the frame? The answer lies in the use of *panel*. A panel is the simplest form of containers for attaching components. As a container, a panel can also contain other panels.

NOT FOR REDISTRIBUTION

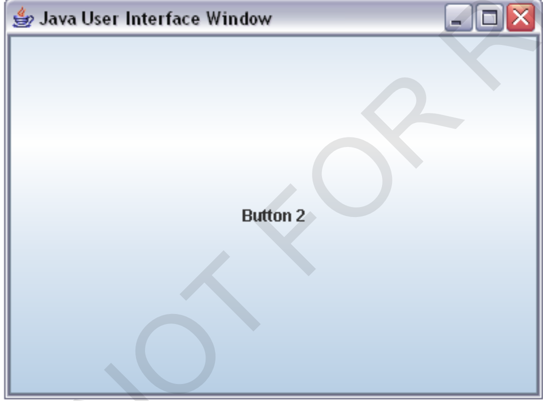


FIGURE 2.8: Adding Two Buttons

2.2.4 Introducing Panel and JPanel

A panel is implemented by the javax.swing.JPanel class. In Code 2.4, a JPanel object is instantiated. It is used to contain the two created buttons. The JPanel object is then added into the frame. The output of Code 2.4 is given in Figure 2.9. We will discuss JPanel in more details later in this chapter.

Copyright © 2020, Danny Poo.

Code 2.4: Using Panel to Contain Components

```
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

class Frame4 {

  public static void main(String argv[]) {
    JFrame frame = new JFrame("Java User Interface Window");
    frame.setSize(400, 300);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setLocation(300, 100);

    // add components
    JButton button1 = new JButton("Button 1");
    JButton button2 = new JButton("Button 2");
    JPanel panel = new JPanel();
    panel.add(button1);
    panel.add(button2);
    frame.getContentPane().add(panel);

    // show frame
    frame.setVisible(true);
  }
}
```

FIGURE 2.9: Using Panel to Contain Components

From this example, it is clear that:

1. By adding a button into a panel, the button does not occupy the entire content pane.
2. By adding buttons into a panel, the buttons are visible and no button is lost.

2.2.5 Creating Your Own Content Pane for a Frame

Instead of using the default pane, developers can create their own Content Pane. The latter has to be manually associated to the frame using the setContentPane() method as shown in Code 2.5. The output from this code is the same as that of Code 2.4 (see Figure 2.9).

Code 2.5: Setting Your Own Content Pane

```
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JComponent;

class Frame5 {

  public static void main(String argv[]) {
    JFrame frame = new JFrame("Java User Interface Window");
    frame.setSize(400, 300);
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setLocation(300, 100);

    // add components
    JButton button1 = new JButton("Button 1");
    JButton button2 = new JButton("Button 2");
    JPanel panel = new JPanel();
    panel.add(button1);
    panel.add(button2);

    //Create and set up the content pane.
    JComponent newContentPane = panel;
    newContentPane.setOpaque(true); //content panes must be opaque
    frame.setContentPane(newContentPane);

    // show frame
    frame.setVisible(true);
  }
}
```

2.2.6 Displaying Components in Frames

Java always display the last component added into a frame. Earlier we saw in Code 2.3 how the second button was the only component displayed in the frame. So, it is the same with panels – only the last panel is displayed if more than one panel is added into a frame. Let us illustrate with an example. Suppose we create four buttons and add them into two panels. The two panels are then added into a frame. As expected, only the second panel is displayed (see Code 2.6 and Figure 2.10).

Code 2.6: Displaying Components in a Frame

```
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;


class Frame6 {

  public static void main(String argv[]) {
    JFrame frame = new JFrame("Java User Interface Window");
    frame.setSize(400, 300);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setLocation(300, 100);

    // add components
    JButton button1 = new JButton("Button 1");
    JButton button2 = new JButton("Button 2");
    JButton button3 = new JButton("Button 3");
    JButton button4 = new JButton("Button 4");
    JPanel  panel1  = new JPanel();
    JPanel  panel2  = new JPanel();
    panel1.add(button1);
    panel1.add(button2);
    panel2.add(button3);
    panel2.add(button4);
    frame.getContentPane().add(panel1);
    frame.getContentPane().add(panel2);

    // show frame
    frame.setVisible(true);
  }
}
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

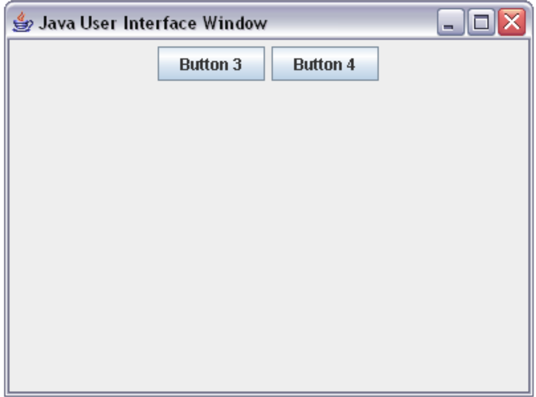


FIGURE 2.10: Displaying Components in a Frame

If we want to display the two panels in the frame, we can either:

1. Create a third panel and include the two panels into the third panel. The third panel is then added into the content pane of the frame; or
2. Use a Layout Manager to position the two panels in the frame's content pane.

We will discuss Layout Manager in the next section.

2.3 Layout Managers

A *layout manager* manages the layout of a container by arranging and resizing the components in the container according to a certain pattern. There are altogether five types of layout managers: FlowLayout, GridLayout, GridBagLayout, BorderLayout and CardLayout. We will discuss FlowLayout, GridLayout and BorderLayout in this section.

2.3.1 Adding Two Panels into a Frame

We will use the BorderLayout layout manager to add two panels and display them in a frame.

Code 2.7: Displaying All Components in a Frame Using Layout Manager

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

class Frame7 {

  public static void main(String argv[]) {
    JFrame frame = new JFrame("Java User Interface Window");
    frame.setSize(400, 300);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setLocation(300, 100);

    // add components
    JButton button1 = new JButton("Button 1");
    JButton button2 = new JButton("Button 2");
    JButton button3 = new JButton("Button 3");
    JButton button4 = new JButton("Button 4");
    JPanel  panel1  = new JPanel();
    JPanel  panel2  = new JPanel();
    panel1.add(button1);
    panel1.add(button2);
    panel2.add(button3);
    panel2.add(button4);
    frame.getContentPane().setLayout(new BorderLayout());
    frame.getContentPane().add(panel1, BorderLayout.NORTH);
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
    frame.getContentPane().add(panel2, BorderLayout.SOUTH);

    // show frame
    frame.setVisible(true);
  }
}
```

Instantiate four buttons and create two panels for containing the four buttons. The buttons are added into the panels. Instead of just adding the panels into the frame, the content pane of the frame is first prepared with a layout managed by a layout manager.

A BorderLayout layout manager is used. This type of layout manager divides a container into five segments – North, South, East, West, and Center. panel1 is added into the North segment while panel2 is added into the South segment. Figure 2.11 shows the locations where the two panels are placed. Since Button1 and Button2 are in panel1, they appear in the North segment of the content pane.

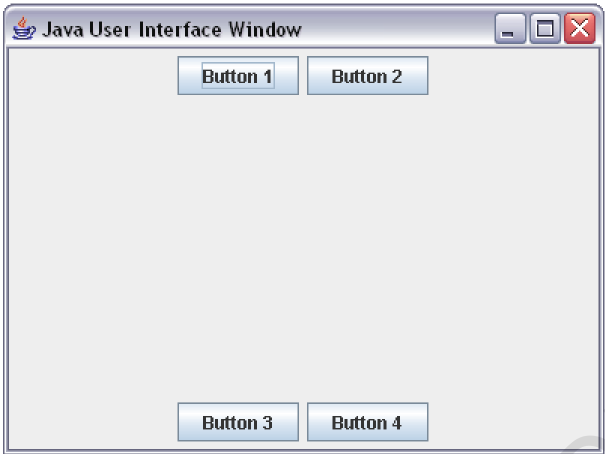


FIGURE 2.11: Displaying All Components in a Frame Using Layout Manager

2.3.2 FlowLayout Layout Manager

2.3.2.1 What is a FlowLayout Layout Manager?

A FlowLayout layout manager arranges components in an ordered manner from left to right and in the sequence the components are added into the container.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

Code 2.8: Demonstrating FlowLayout Layout Manager

```
import java.awt.Container;
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;

class LayoutManager1 extends JFrame {

  LayoutManager1() {
    // get content pane of frame
    Container contentPane = getContentPane();

    // set FlowLayout using default flow layout
    FlowLayout layout = new FlowLayout();
    contentPane.setLayout(layout);

    // create 16 buttons as components to add to frame
    for (int i=0; i<16; i++) {
      JButton button = new JButton("Button " + (i+1));
      contentPane.add(button);
    }
  }

  public static void main(String argv[]) {
    LayoutManager1 frame = new LayoutManager1();
    frame.setTitle("FlowLayout Manager Test Frame");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(400, 300);
    frame.setLocation(300, 100);
    frame.setVisible(true);
  }
}
```

In Code 2.8, a FlowLayout layout manager is instantiated. Recall that it is in the content pane that components added into a frame are stored. The responsibility of the layout manager is therefore to arrange the components in the content pane. The instantiated FlowLayout layout manager is associated with the contentPane container. To demonstrate how the FlowLayout layout manager arranges the components, we create sixteen buttons and add them into the contentPane container. When the frame is displayed, Figure 2.12 is produced.

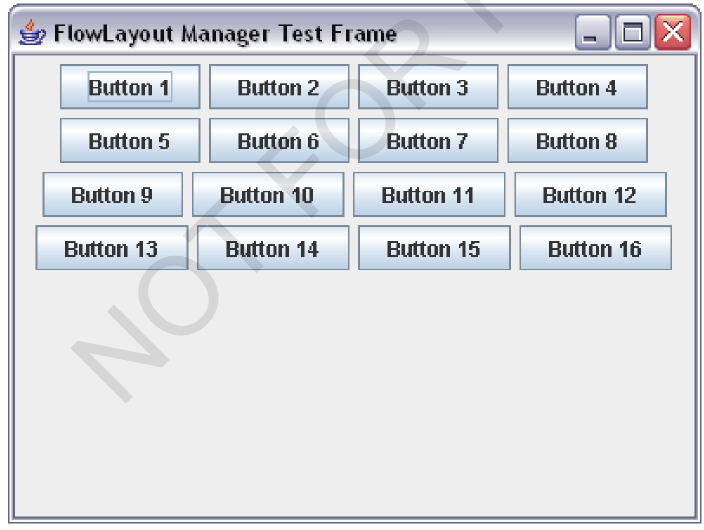


FIGURE 2.12: Demonstrating FlowLayout Layout Manager

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

2.3.2.2 How does the FlowLayout Layout Manager Arrange the Components?

The sixteen buttons are added one at a time starting from the left and moving to the right. When a row is filled, a new row is created. This process is repeated until all the buttons have been added.

2.3.2.3 More about the Code

Some interesting characteristics about Code 2.8 are discussed below:

1. Four classes from two packages are imported.
2. The demonstrating frame (LayoutManager1) is extended from JFrame. This approach is recommended since any graphical user interface is basically a frame and by means of inheritance, all properties of a frame are made available to the user class.
3. The demonstrating frame is instantiated from the main() method. The constructor of the frame contains code for getting the content pane, sets the layout manager for the content pane (via the setLayout() method) and adds the components into the container (the demonstrating frame).
4. Since the demonstrating frame is extended from JFrame, a number of methods not defined in LayoutManager1 are made available from its superclasses. setDefaultCloseOperation() is inherited from javax.swing.JFrame. setTitle() is inherited from java.awt.Frame. setSize(), setVisible() and setLocation() are inherited from java.awt.Component.
5. The display of the frame is done at the main() method.

2.3.2.4 What are the Constructors?

The FlowLayout class belongs to the java.awt package. It has three constructors as stated in Table 2.2.

TABLE 2.2: FlowLayout Constructors

No.	Constructor	Description
1	`FlowLayout()`	Creates a new FlowLayout object with a default center alignment and a default gap of 5 pixels horizontally and vertically.
2	`FlowLayout(int align)`	Creates a new FlowLayout object with the specified alignment and a default gap of 5 pixels horizontally and vertically. The align value can take any of the three constants: FlowLayout.CENTER, FlowLayout.LEFT, FlowLayout.RIGHT.
3	`FlowLayout(int align, int hGap, int vGap)`	Creates a new FlowLayout object with the specified alignment, and horizontal and vertical gaps between components.

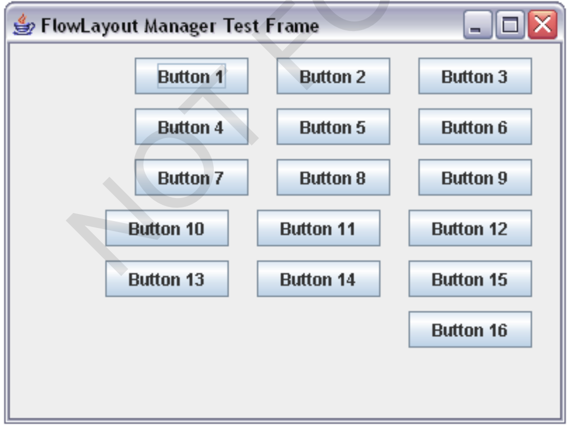


FIGURE 2.13: FlowLayout with Right Alignment

We have seen the use of the first constructor in Code 2.8. In Figure 2.13, we show how the third constructor of the FlowLayout class can be used:

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

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

Notice the buttons have been flushed to the right and a maximum of three buttons are available on each row. This is due to the spacing between components.

2.3.3 GridLayout Layout Manager

2.3.3.1 What is a GridLayout Layout Manager?

The GridLayout layout manager divides a container into equal-sized portions in a rectangular grid. The size of the grid is determined at construction time by specifying the number of rows and columns the rectangular grid should have. The components are inserted into each of the cell from left to right or right to left horizontally, depending on the setting of the container's ComponentOrientation property. Code 2.9 highlights the use of GridLayout layout manager.

Code 2.9: Demonstrating GridLayout Layout Manager

```
import java.awt.Container;
import java.awt.GridLayout;
import javax.swing.JButton;
import javax.swing.JFrame;

class LayoutManager2 extends JFrame {

  LayoutManager2() {
    // get content pane of frame
    Container contentPane = getContentPane();

    // set FlowLayout using default flow layout
    GridLayout layout = new GridLayout(4, 4);
    contentPane.setLayout(layout);

    // create 16 buttons as components to add to frame
    for (int i=0; i<16; i++) {
      JButton button = new JButton("Button " + (i+1));
      contentPane.add(button);
    }
  }

  public static void main(String argv[]) {
    LayoutManager2 frame = new LayoutManager2();
    frame.setTitle("GridLayout Manager Test Frame");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(400, 300);
    frame.setLocation(300, 100);
    frame.setVisible(true);
  }
}
```

In Code 2.9, the ComponentOrientation property of the frame, by default, has been set from Left to Right; and the size of the grid has been set to 4 rows by 4 columns.

2.3.3.2 More about the Code

Some interesting characteristics about Code 2.9 are discussed below:

1. The GridLayout class is imported.
2. The GridLayout layout manager organizes the container into a 4 by 4 rectangular grid.
3. The GridLayout layout manager, by default, inserts components one by one into the grid from left to right and row by row.

Figure 2.14 is the output of Code 2.9.

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

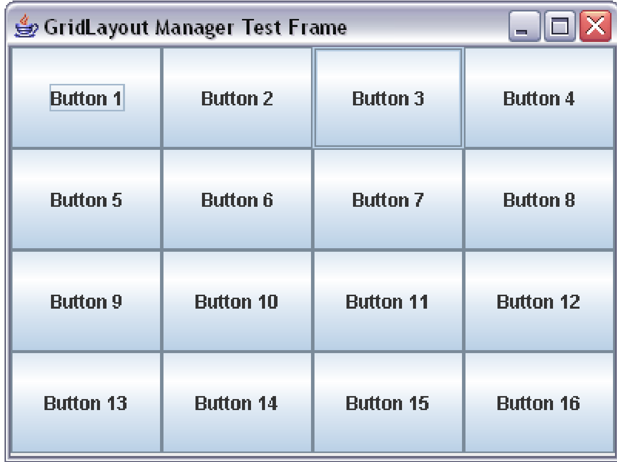


FIGURE 2.14: Demonstrating GridLayout Layout Manager

2.3.3.3 Rules on Number of Rows and Columns

You may specify whatever value for the number of rows and columns but the behavior of the GridLayout layout manager is subject to the following three rules:

1. ***If the number of rows and columns are non-zero e.g. (3, 4).*** The number of rows is fixed and the number of columns specified is ignored by the GridLayout layout manager. Instead, the layout manager does a calculation dynamically to determine the number of columns based on the number of components to be arranged. For example, given a grid specification of (3, 7), three rows will be defined for the grid but the number of columns is 6 and not 7, as shown in Figure 2.15. Similarly, for a grid specification of (5, 3), the layout is shown in Figure 2.16. Note that there is an empty row in the grid as the number of rows has been fixed at 5 and four buttons can fit into each row (i.e. 4 columns are determined).

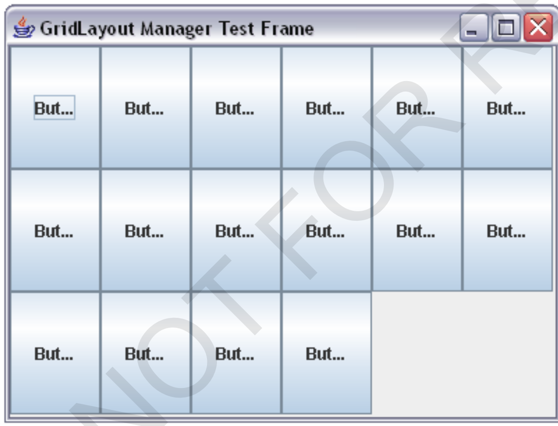


FIGURE 2.15: 3 by 7 Grid Size

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

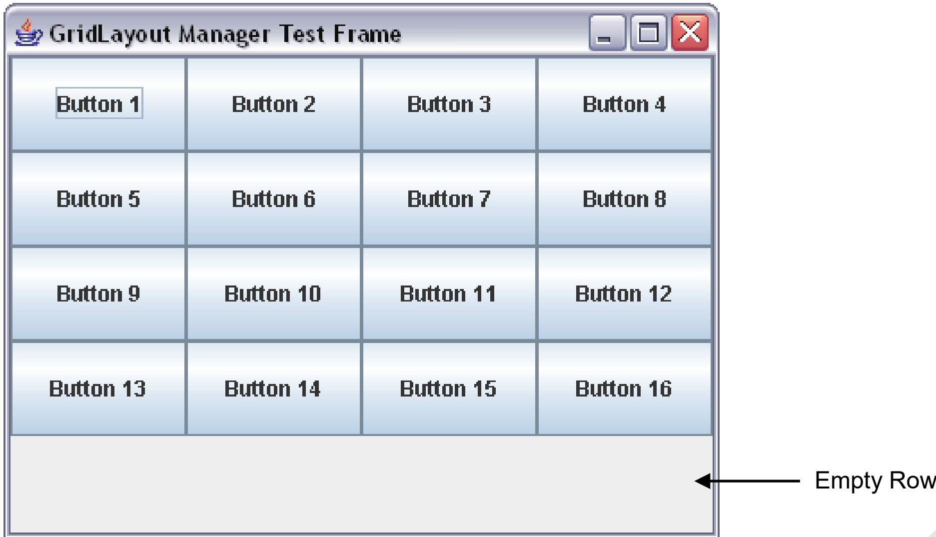


FIGURE 2.16: 5 by 3 Grid Size

2. ***If either the number of rows or the number of columns is zero e.g. (0, 3) or (4, 0).*** The non-zero dimension is fixed, while the zero dimension is dynamically calculated by the GridLayout Layout Manager.
3. Both the number of rows and the number of columns is zero e.g. (0, 0). This is not permissible and an exception is reported.

Table 2.3 provides a more comprehensive description of the various situations with examples.

TABLE 2.3: Rules on Number of Rows and Columns in GridLayout

S/No.	Number of Rows	Number of Columns	Number of Components* > Any +ve Non-zero?	Outcome	Examples
1.	0	0	Yes	Exception	
2.	+ve Non-zero	+ve Non-zero	Yes	Number of rows is fixed. Number of columns is dynamically determined by GridLayout Layout Manager.	#Components = 16 #Rows = 10 #Columns = 13 Grid size is (10, 2), first 8 rows filled, last 2 rows empty. *#Components = 16* *#Rows = 13* *#Columns = 10* *Grid size is (13, 2), first 8 rows filled, last 5 rows empty.*
3.	+ve Non-zero	+ve Non-zero	No	Number of rows is fixed. Number of columns is dynamically determined by GridLayout Layout Manager.	#Components = 16 #Rows = 18 #Columns = 13 Grid size is (18, 1), first 16 rows filled, last 2 rows empty. *#Components = 16* *#Rows = 9* *#Columns = 20* *Grid size is (9, 2), first 8 rows filled, last row empty.*
4.	–ve	–ve Non-	No	Exception	

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

	Non-zero	zero			
5.	0	+ve Non-zero	Yes/No	Number of columns is fixed. Number of rows is dynamically determined by GridLayout Layout Manager.	#Components = 16 #Rows = 0 #Columns = 20 Grid size is (1, 20), first 16 columns from left filled, last 4 columns empty. *#Components = 16* *#Rows = 0* *#Columns = 10* *Grid size is (2, 10), first row filled, 2nd row 6 columns from left filled, the rest empty.*
6.	+ve Non-zero	0	Yes/No	Number of rows is fixed. Number of columns is dynamically determined by GridLayout Layout Manager.	#Components = 16 #Rows = 18 #Columns = 0 Grid size is (18, 1), first 16 rows filled, last 2 rows empty. *#Components = 16* *#Rows = 6* *#Columns = 0* *Grid size is (6, 3), first 5 rows filled, last row 1st left column filled, the rest empty.*
7.	0	–ve Non-zero	No	Exception	
8.	–ve Non-zero	0	No	Exception	
9.	–ve Non-zero	+ve Non-zero	Yes/No	Number of columns is fixed. Number of rows is dynamically determined by GridLayout Layout Manager.	Same as 5.
10.	+ve Non-zero	–ve Non-zero	Yes/No	Number of rows is fixed. Number of columns is dynamically determined by GridLayout Layout Manager.	Same as 6.

*Number of Components must be +ve Non-zero

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

2.3.3.4 What are the Constructors?

The GridLayout class belongs to the java.awt package. It has three constructors as stated in Table 2.4.

TABLE 2.4: GridLayout Constructors

No.	Constructor	Description
1	`GridLayout()`	Creates a new GridLayout object with a default of one column per component in a single row.
2	`GridLayout (int rows, int cols)`	Creates a new GridLayout object with the specified number of rows and columns.
3	`GridLayout (int rows, int cols, hGap, int vGap)`	Creates a new GridLayout object with the specified number of rows, columns, horizontal gap and vertical gap.

The default component orientation for the container (in this case, the demonstrating frame) is Left to Right. However, this orientation can be changed to Right to Left by applying the component orientation constant RIGHT_TO_LEFT to the container, as shown in Code 2.10. The output is shown in Figure 2.17.

Code 2.10: Applying Component Orientation

```
import java.awt.Container;
import java.awt.GridLayout;
import java.awt.ComponentOrientation;
import javax.swing.JButton;
import javax.swing.JFrame;

class LayoutManager2b extends JFrame {

  LayoutManager2b() {
    // get content pane of frame
    Container contentPane = getContentPane();

    // set FlowLayout using default flow layout
    GridLayout layout = new GridLayout(5, 3);
    contentPane.setLayout(layout);

    // create 16 buttons as components to add to frame
    for (int i=0; i<16; i++) {
      JButton button = new JButton("Button " + (i+1));
      contentPane.add(button);
    }
  }

  public static void main(String argv[]) {
    LayoutManager2b frame = new LayoutManager2b();
    frame.applyComponentOrientation(ComponentOrientation.RIGHT_TO_LEFT);
    frame.setTitle("GridLayout Manager Test Frame");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(400, 300);
    frame.setLocation(300, 100);
    frame.setVisible(true);
  }
}
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

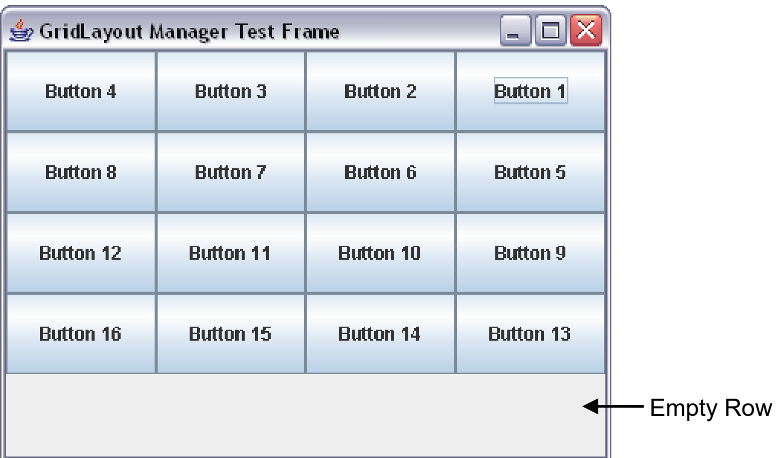


FIGURE 2.17: Right to Left Component Orientation

2.3.3.5 Resizing

All components in a container are sized equally by the GridLayout layout manager. Resizing the container (or frame in this case) does not change the number of rows and columns. The gaps between the components do not change too.

2.3.4 BorderLayout Layout Manager

2.3.4.1 What is a BorderLayout Layout Manager?

A BorderLayout layout manager divides a container into 5 segments – North, South, East, West, and Center. An example of BorderLayout is given in Figure 2.18. In this example, five buttons are added into the same container discussed earlier.

2.3.4.2 More about the Code

Figure 2.18 is produced from Code 2.11.

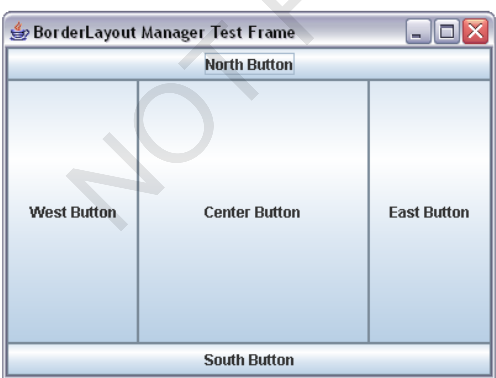


FIGURE 2.18: Demonstrating BorderLayout Layout Manager

Some interesting characteristics about Code 2.11 are discussed below:

1. A BorderLayout object is created and set as the layout manager for the container.
2. Create five JButton objects and add them into the container.

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

3. Components are added via the method add(Component, index); index is any of the following constants: BorderLayout.NORTH, BorderLayout.SOUTH, BorderLayout.EAST, BorderLayout.WEST, and BorderLayout.CENTER.
4. The layout manager treats the positioning index as BorderLayout.CENTER, if the index constant in the add(Component, index) method is omitted.
5. Adding two components into the same location will result in the display of the last component (as we have discussed in previous examples). For example,
   container.add(button1);
   container.add(button2);
   displays only button2.
6. We could alternatively use anonymous Jbutton objects to replace the earlier lines to instantiate3 the various buttons since the objects are not directly referenced in any part of the program.

Code 2.11: Demonstrating BorderLayout Layout Manager

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

class LayoutManager3 extends JFrame {

  LayoutManager3() {
    // get content pane of frame
    Container contentPane = getContentPane();

    // set FlowLayout using default flow layout
    BorderLayout layout = new BorderLayout();
    contentPane.setLayout(layout);

    // create 5 buttons as components to add to frame
    JButton northButton  = new JButton("North Button");
    JButton southButton  = new JButton("South Button");
    JButton eastButton   = new JButton("East Button");
    JButton westButton   = new JButton("West Button");
    JButton centerButton = new JButton("Center Button");
    contentPane.add(northButton, BorderLayout.NORTH);
    contentPane.add(southButton, BorderLayout.SOUTH);
    contentPane.add(eastButton, BorderLayout.EAST);
    contentPane.add(westButton, BorderLayout.WEST);
    contentPane.add(centerButton, BorderLayout.CENTER);

    /* a shorter version using anonymous JButton objects
    contentPane.add(new JButton("North Button"), BorderLayout.NORTH);
    contentPane.add(new JButton("South Button"), BorderLayout.SOUTH);
    contentPane.add(new JButton("East Button"), BorderLayout.EAST);
    contentPane.add(new JButton("West Button"), BorderLayout.WEST);
    contentPane.add(new JButton("Center Button"), BorderLayout.CENTER);
    */
  }

  public static void main(String argv[]) {
    LayoutManager3 frame = new LayoutManager3();
    frame.setTitle("BorderLayout Manager Test Frame");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(400, 300);
    frame.setLocation(300, 100);
    frame.setVisible(true);
  }
}
```

2.3.4.3 What are the Constructors?

The BorderLayout class belongs to the java.awt package. It has two constructors as stated in Table 2.5.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

TABLE 2.5: BorderLayout Constructors

No.	Constructor	Description
1	`BorderLayout()`	Creates a new BorderLayout object without any horizontal or vertical gaps.
2	`BorderLayout (int hGap, int vGap)`	Creates a new BorderLayout object with the specified horizontal and vertical gaps between components.

We have seen the use of the first constructor in Code 2.15. The following statement shows the use of the second constructor of BorderLayout:

BorderLayout layout = new BorderLayout(30, 7);

Figure 2.19 is the output from using this constructor. Notice the presence of horizontal and vertical gaps between the buttons.

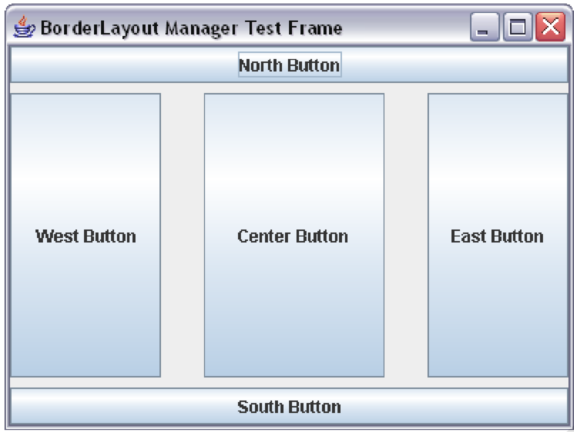


FIGURE 2.19: BorderLayout with Horizontal and Vertical Gaps

NOT FOR REDISTRIBUTION

2.3.4.4 Laying out Components

The BorderLayout layout manager positions components based on their preferred sizes and location (North, South, East, West, or Center). It attempts to fill out the entire space in the container. The North and South components can stretch horizontally and the East and West components can stretch vertically. The Center component can stretch both horizontally and vertically.

In Code 2.12, the West and South components have been removed. The result is a frame with the Center component stretched horizontally and vertically, and the East component stretched vertically (see Figure 2.20).

Code 2.12: Laying out Components

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

class LayoutManager3a extends JFrame {

  LayoutManager3a() {
    // get content pane of frame
    Container contentPane = getContentPane();

    // set FlowLayout using default flow layout
    BorderLayout layout = new BorderLayout();
    contentPane.setLayout(layout);

    // a shorter version using anonymous JButton objects
```

Copyright © 2020, Danny Poo.

```
    contentPane.add(new JButton("North Button"), BorderLayout.NORTH);
    // contentPane.add(new JButton("South Button"), BorderLayout.SOUTH);
    contentPane.add(new JButton("East Button"), BorderLayout.EAST);
    // contentPane.add(new JButton("West Button"), BorderLayout.WEST);
    contentPane.add(new JButton("Center Button"), BorderLayout.CENTER);
  }

  public static void main(String argv[]) {
    LayoutManager3a frame = new LayoutManager3a();
    frame.setTitle("BorderLayout Manager Test Frame");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(400, 300);
    frame.setLocation(300, 100);
    frame.setVisible(true);
  }
}
```

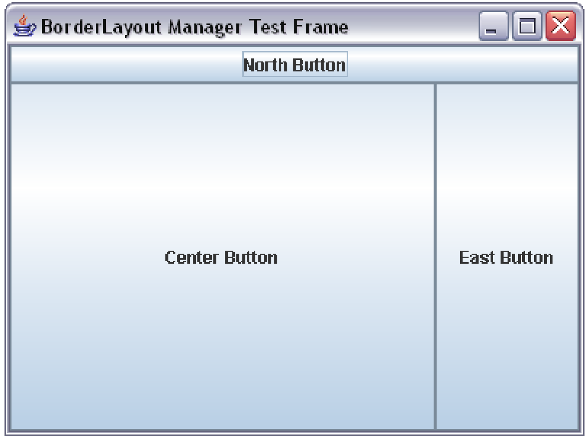


FIGURE 2.20: Laying out Components

2.3.5 Considerations on Using Layout Managers

Take note of the followings when using layout managers:

1. There can *only be one* layout manager for a container at any one time.
2. To change a layout manager, use the setLayout() method. Subsequently, use the validate() method of the new layout manager to ensure the new layout is effected in the container.
3. If you change any property of a layout manager, use doLayout() method to ensure the updated properties are effective in the new layout of the container.

2.4 Panels

At this stage you should be familiar with how to create a window, how to add components and how to position components in a window. If not, you should return to the previous sections to review these concepts.

In this section, you will learn how to create an application with a decent graphical user interface using the Java Swing components.

Let us create a *calculator* application. Figure 2.21 shows a calculator frame. This frame has a text field that displays the result of a calculation. It also has twenty-five buttons for entering values. How do we create this frame?

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

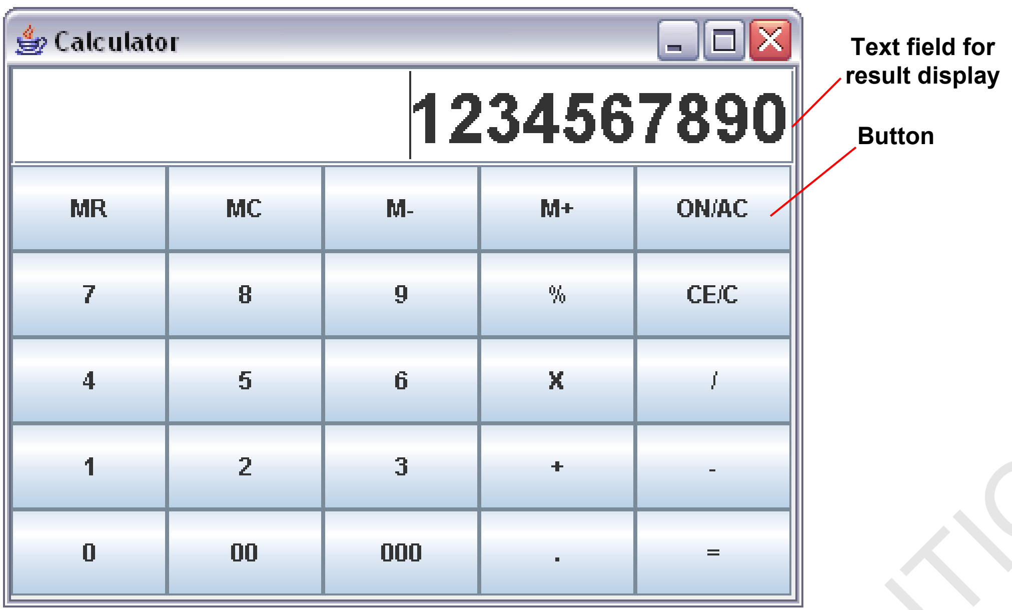


FIGURE 2.21: Calculator Frame

2.4.1 Problems in Displaying Components in Frames

One of the challenges in developing the Calculator frame is in knowing how to display the components (a text-field and twelve-five buttons) in the Calculator frame:

1. If we do not use a layout manager and simply add the components into the frame, we will end up with displaying only the last added component in the frame i.e. displaying only the "=" button.
2. If we use the GridLayout layout manager, we will notice that there is a problem with the result display – the latter does not occupy the entire stretch of row 1.
3. If we use the FlowLayout layout manager, we will end up with an output as shown in Figure 2.22.
4. We would not use the BorderLayout layout manager as it would not be suitable for the format of layout we require.

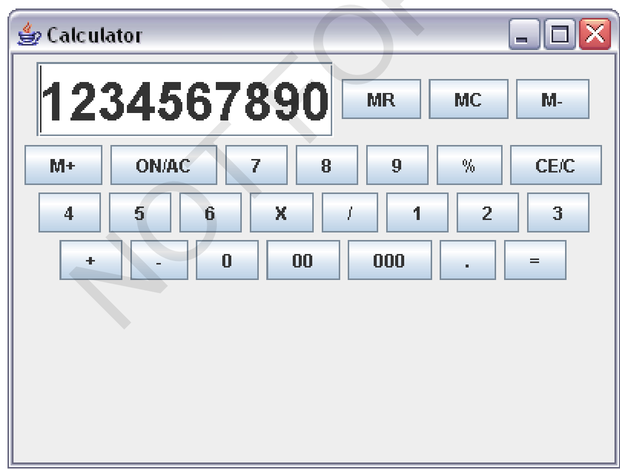


FIGURE 2.22: Displaying Calculator using FlowLayout Layout Manager

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

2.4.2 Positioning Components with Panels

We can solve the above problems with the use of panels to group components. A panel is implemented in the Java Swing as a JPanel object. The class hierarchy for JPanel is shown in Figure 2.23. It is clear from this class hierarchy that JPanel is a component as well as a container[5]. As a container, a JPanel has the capability of containing other components and, as a component, a JPanel can be contained in containers such as frames.

```
java.lang.Object
└──java.awt.Component
    └──java.awt.Container
        └──javax.swing.JComponent
            └──javax.swing.JPanel
```

FIGURE 2.23: Class Hierarchy of JPanel

2.4.3 Panels as Containers

Let us analyze the structure we need for representing our Calculator frame. We will use a TextField to represent the display. We will add the twenty-five buttons into a panel. With this setup, we only have two components: a TextField and a Panel. This structure is shown in Figure 2.24 which is produced by Code 2.13.

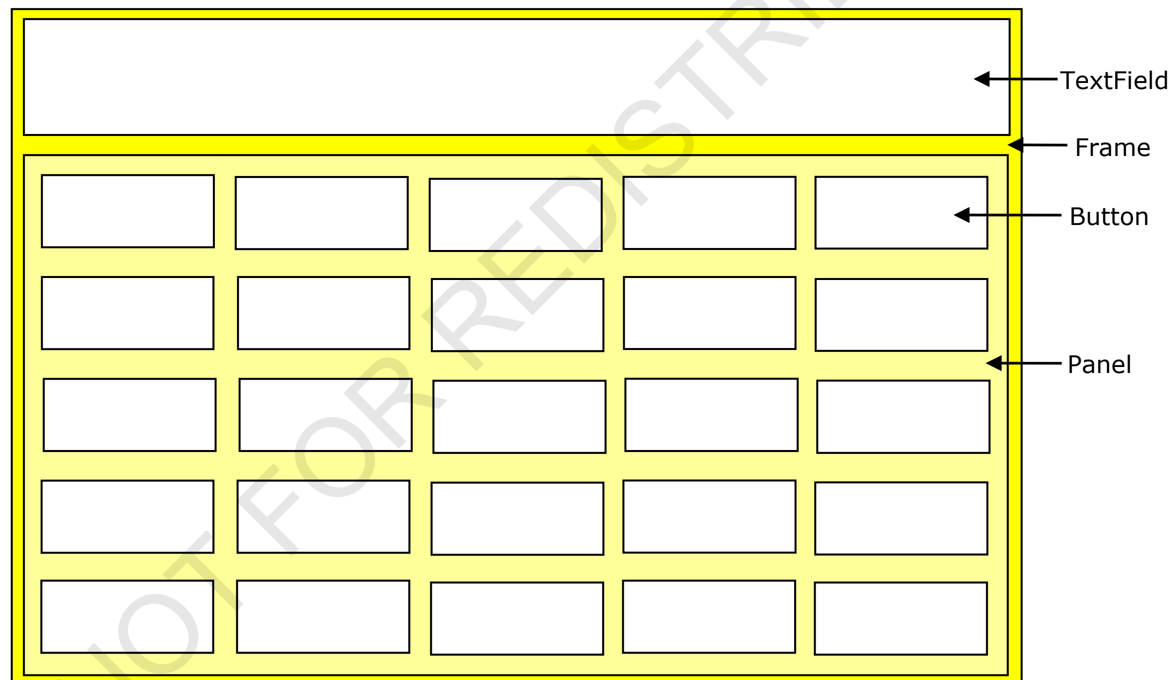


FIGURE 2.24: Calculator Frame Structure

[5] Because JPanel is a subclass of Component and Container.

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

Code 2.13: Calculator

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

class Panel2 extends JFrame {

  Panel2() {
    // get content pane of frame
    Container contentPane = getContentPane();

    // set BorderLayout
    BorderLayout layout = new BorderLayout();
    contentPane.setLayout(layout);

    // creates a panel
    JPanel panelA = new JPanel();

    // creates a textField with a Helvetica font
    Font font = new Font("Helvetica", Font.BOLD, 35);
    JTextField textField = new JTextField("1234567890");
    textField.setHorizontalAlignment(JTextField.RIGHT);
    textField.setFont(font);

    // add buttons into panelA
    panelA.setLayout(new GridLayout(5,5));
    panelA.add(new JButton("MR"));
    panelA.add(new JButton("MC"));
    panelA.add(new JButton("M-"));
    panelA.add(new JButton("M+"));
    panelA.add(new JButton("ON/AC"));

    for (int i=7; i<=9; i++) {
      panelA.add(new JButton("" + i));
    }
    panelA.add(new JButton("%"));
    panelA.add(new JButton("CE/C"));

    for (int i=4; i<=6; i++) {
      panelA.add(new JButton("" + i));
    }
    panelA.add(new JButton("X"));
    panelA.add(new JButton("/"));

    for (int i=1; i<=3; i++) {
      panelA.add(new JButton("" + i));
    }
    panelA.add(new JButton("+"));
    panelA.add(new JButton("-"));

    panelA.add(new JButton("0"));
    panelA.add(new JButton("00"));
    panelA.add(new JButton("000"));
    panelA.add(new JButton("."));
    panelA.add(new JButton("="));

    // add textfield and panel to contentPane
    contentPane.add(textField, BorderLayout.NORTH);
    contentPane.add(panelA, BorderLayout.CENTER);
  }

  public static void main(String argv[]) {
    Panel2 frame = new Panel2();
    frame.setTitle("Calculator");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(400, 300);
    frame.setLocation(300, 100);
    frame.setVisible(true);
  }
}
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

2.4.3.1 More about the Code

Some interesting characteristics about Code 2.13 are discussed below:

1. A panel is created using the javax.swing.JPanel class.
2. Create a JTextField object that can hold texts for display. To set the font on a JTextField object, we use setFont() method in JTextField. The setFont() method takes in a Font object as its parameter. Instantiates a Font object with “Helvetica” (Helvetica), “Bold” (Font.BOLD), and “35 points” (35) as its font, style, and size respectively. The texts are also aligned RIGHT.
3. A JPanel object is instantiated and set with a GridLayout layout manager[6].
4. Add all the twenty-five buttons into the panel. We have solicited the help of the for loop to create some of the numeric buttons.
5. Finally, the textField and panelA are added into the contentPane and positioned using a BorderLayout layout manager. Note that the textField object is a standalone object – it is not added into a panel. If we have included the textField object into a panel before adding it into the contentPane, the textField display area will be restricted. Our adopted approach ensures that the textField is stretched to the width of the frame.

In summary, we have used two different layout managers: panelA uses a GridLayout layout manager while the contentPane uses a BorderLayout layout manager. You may have wondered why a GridLayout layout manager with a grid size of 2 by 1 is not used to organize the contentPane; instead, we used a BorderLayout layout manager. If we had used GridLayout layout manager, we would have gotten Figure 2.25 – obviously, this is not what is required.

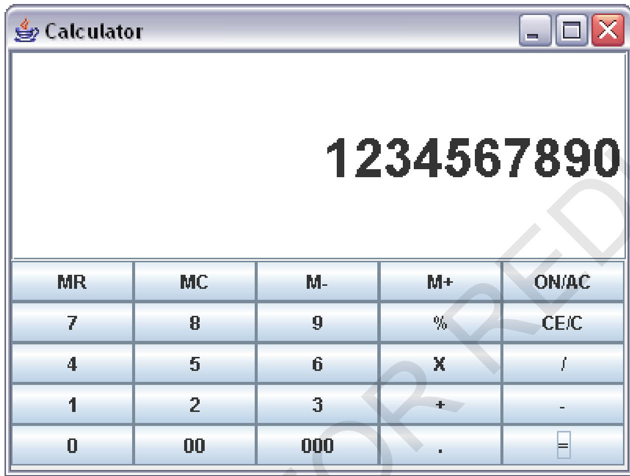


FIGURE 2.25: Using GridLayout instead of BorderLayout

2.4.3.2 What are the Constructors?

The JPanel class belongs to the javax.swing package. It has four constructors as stated in Table 2.6.

TABLE 2.6: JPanel Constructors

No.	Constructor	Description
1	`JPanel()`	Creates a new JPanel object with a double buffer[7] and a default FlowLayout manager.
2	`JPanel(boolean isDoubleBuffered)`	Creates a new JPanel object with a default FlowLayout manager and the specified buffering strategy.
3	`JPanel(LayoutManager layout)`	Creates a new buffered JPanel object with a specified layout manager.

[6] JPanel comes with a default FlowLayout layout manager.
[7] Double buffering uses additional memory to achieve fast and flicker-free updates.

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

4	`JPanel(LayoutManager layout, boolean isDoubleBuffered)`	Creates a new JPanel object with the specified layout manager and buffering strategy.

2.5 Five Basic Steps for Creating a Window

From the above examples, we can generalize five basic steps for creating a window (or frame):

1. Instantiate a frame and set its attributes e.g. its title.
2. Get the content pane of the frame.
3. Set a layout for the content pane.
4. Add components into the frame or remove components from the frame or add components into a container that is subsequently added into the frame.
5. Make the frame visible.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

CHAPTER 3: USER INTERACTIONS AND EVENT HANDLING

In the previous chapter, we discussed how to create windows using frames and other components. The discussion had focused on the look-and-feel of windows but did not deal with user interactions and event handling. How do we program a user interface that is interactive and how do we deal with events generated by the GUI components? We will show you how in this chapter.

3.1 The Calculator

The Calculator that we introduced in the last chapter has a number of buttons but none of them could respond to user clicks.

A button is an event-generating component. It fires events when clicked. Unless the events are caught, the Calculator will not be able to respond to the clicks. The next step in making our Calculator useful is to add event handling into the design of the Calculator.

3.1.1 A Revised Calculator

Let us simplify the design of our Calculator so that our discussion can be more focused on event handling. Our revised Calculator looks like Figure 3.1.

FIGURE 3.1: The Revised Calculator

The revised Calculator has a text-field for displaying any output in the calculation process. It also has sixteen buttons (10 Digit buttons, an Entry Clear button (C), a Multiplication button (X), a Division button (/), an Addition button (+), a Subtraction button (-), and a Compute button (=)). The final result of a calculation is produced when the Compute button is pressed.

3.1.2 Code for the Revised Calculator

We can produce Figure 3.1 with Code 3.1. The Calculator frame has been divided into two parts: Display Part and Buttons Part.

1. The Display Part is made up of a JTextField object while the Buttons Part is defined by a JPanel object (panelA).
2. These two parts are arranged in the Content Pane of the frame using a BorderLayout layout manager.
3. The display is set with an "Arial" font while the buttons are set with a "Verdana" font.
4. Buttons are organized in the JPanel object using a 4 by 4 GridLayout layout manager.

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

5. All buttons are systematically added into the panel before the text field and panel is added into the Content Pane of the frame.

Code 3.1: The Revised Calculator

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

class Calculator1 extends JFrame {

  Calculator1() {

    JButton button = new JButton("");

    // get content pane of frame
    Container contentPane = getContentPane();

    // set BorderLayout
    BorderLayout layout = new BorderLayout();
    contentPane.setLayout(layout);

    // creates a panel
    JPanel panelA = new JPanel();

    // creates a textField with a Helvetica font
    Font displayFont = new Font("Arial", Font.BOLD, 45);
    JTextField display = new JTextField("1234567890");
    display.setHorizontalAlignment(JTextField.RIGHT);
    display.setFont(displayFont);

    // creates a standard font
    Font buttonFont = new Font("Verdana", Font.PLAIN, 20);

    // add buttons into panelA
    panelA.setLayout(new GridLayout(4,4));

    for (int i=7; i<=9; i++) {
      button  = new JButton("" + i);
      button.setFont(buttonFont);
      panelA.add(button);
    }

    button  = new JButton("X");
    button.setFont(buttonFont);
    panelA.add(button);

    for (int i=4; i<=6; i++) {
      button  = new JButton("" + i);
      button.setFont(buttonFont);
      panelA.add(button);
    }

    button  = new JButton("/");
    button.setFont(buttonFont);
    panelA.add(button);

    for (int i=1; i<=3; i++) {
      button  = new JButton("" + i);
      button.setFont(buttonFont);
      panelA.add(button);
    }

    button  = new JButton("+");
    button.setFont(buttonFont);
    panelA.add(button);

    button  = new JButton("C");
    button.setFont(buttonFont);
    panelA.add(button);

    button  = new JButton("0");
    button.setFont(buttonFont);
```

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

```
    panelA.add(button);

    button  = new JButton("=");
    button.setFont(buttonFont);
    panelA.add(button);

    button  = new JButton("-");
    button.setFont(buttonFont);
    panelA.add(button);

    // add textfield and panel to contentPane
    contentPane.add(display, BorderLayout.NORTH);
    contentPane.add(panelA, BorderLayout.CENTER);
  }


  public static void main(String argv[]) {
    Calculator1 frame = new Calculator1();
    frame.setTitle("Calculator");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(400, 300);
    frame.setLocation(300, 100);
    frame.setVisible(true);
  }
}
```

3.2 The Java Event Model

When a user clicks on a button, an event is generated. An event signifies to the Calculator that something has happened. How does the Calculator handle the event? To understand this, we need to examine the Java Event Model.

3.2.1 Events

An *event* is a signal from the external environment that something has happened. Events are generated by user actions in the form of mouse movement, keystrokes, mouse button clicks, or by a timer in the operating system. The event-receiving program (such as the Calculator above) can choose to respond to the event or simply ignores it.

An event is an object of an event class. There are a number of event classes in the Java Class Library: MouseEvent, KeyEvent, ContainerEvent, FocusEvent, PaintEvent, WindowEvent, ActionEvent, AdjustmentEvent, ItemEvent, ListSelectionEvent and TextEvent.

All event classes inherit from the root class EventObject in the java.util package. Figure 3.2 shows the hierarchical relationships of these event classes.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

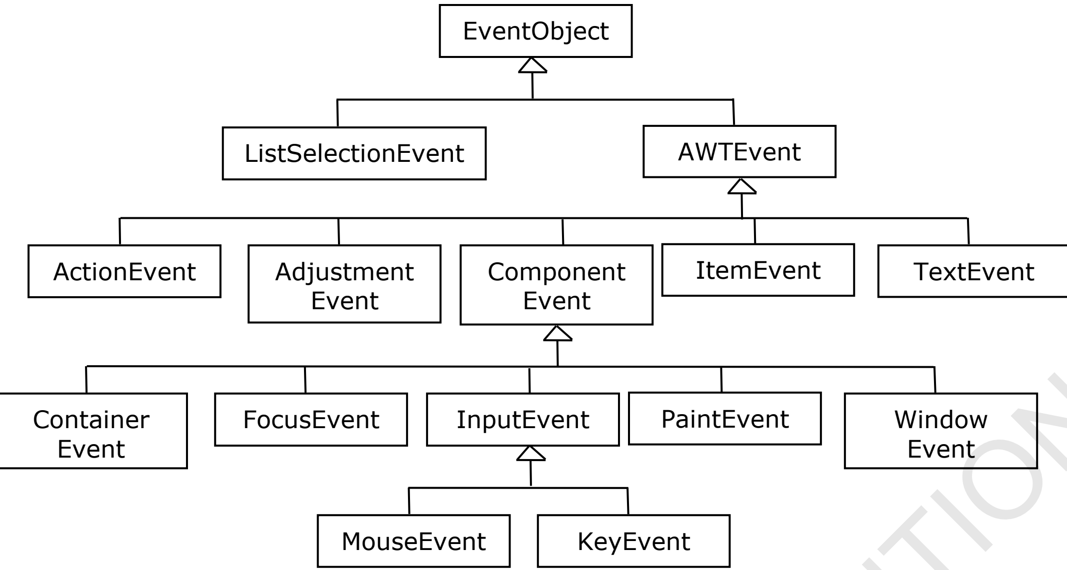


FIGURE 3.2: Class Relationship of Event Classes

3.2.2 Source Objects, Listeners and Handlers

The ***source object*** of an event is the object that generated the event. For example, when a button is clicked, an event is generated by the button; the button is known as the source object of the event.

On the other hand, there are objects that are interested in responding to the generated events. These objects are known as ***listeners*** because they listen for events to happen. For an object to be a listener, it has to register itself as a listener with the source object.

A source object can have many listeners but each listener must be separately registered with the source object. A source object can also be a listener to its own events. In this case, the object doubles its role as source and listener of events.

When an event happens, listeners registered with a source object will be *notified* by the source object. Notification is carried out by invoking an event-handling method known as the ***handler*** on the listener object.

3.2.3 Responding to Events

Java event handling makes use of a *delegation-based model* as shown in Figure 3.3. When an external user action happens (e.g. a button click), an event is generated by the source object (the clicked button). The source object checks out the list of registered listeners and notifies them by invoking the event-handling method on the listener objects.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

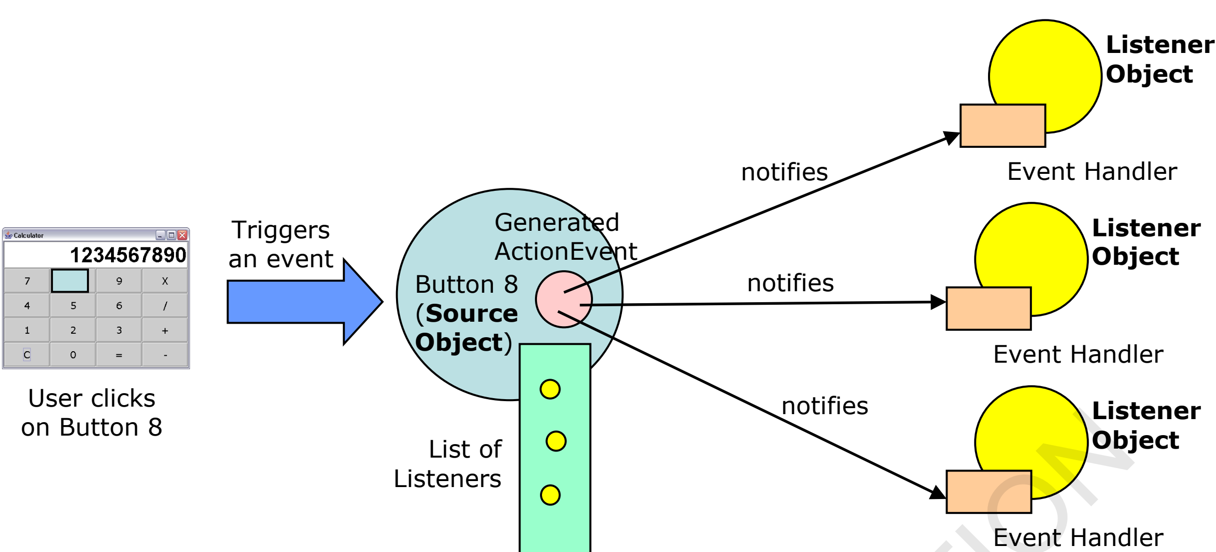


FIGURE 3.3: Delegation-based Model of Event Handling

To demonstrate the event model in the Calculator, we will modify Code 3.1 to take into consideration events and listeners. The revised code is shown in Code 3.2. Our revised Calculator is now able to respond to user clicks on the buttons. For example, when a user clicks on the following numbers on the Calculator keypad "1, 2, 3, 4, 5, 6, 7, 8, 9, 0", the following outputs are observed at the command prompt window:

```
Button 1 clicked
Button 2 clicked
Button 3 clicked
Button 4 clicked
Button 5 clicked
Button 6 clicked
Button 7 clicked
Button 8 clicked
Button 9 clicked
Button 0 clicked
```

All buttons of the Calculator are source objects, capable of generating events. Creates a listener object bnListener (of the Listener class). The addActionListener() method in the javax.swing.AbstractButton class registers bnListener as the listener object. When any of the buttons is clicked, bnListener is notified since it is the only object registered with the buttons.

Code 3.2: Calculator with Event Listeners

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

class Calculator2 extends JFrame {

  Calculator2() {

    // Create a button object
    JButton button = new JButton("");

    // Create a button listener object
    Listener bnListener = new Listener();

    // get content pane of frame
    Container contentPane = getContentPane();

    // set BorderLayout
    BorderLayout layout = new BorderLayout();
    contentPane.setLayout(layout);

    // creates a panel
```

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

```
JPanel panelA = new JPanel();

// creates a textField with a Helvetica font
Font displayFont = new Font("Arial", Font.BOLD, 45);
JTextField display = new JTextField("1234567890");
display.setHorizontalAlignment(JTextField.RIGHT);
display.setFont(displayFont);

// creates a standard font
Font buttonFont = new Font("Verdana", Font.PLAIN, 20);

// add buttons into panelA
panelA.setLayout(new GridLayout(4,4));

for (int i=7; i<=9; i++) {
  button  = new JButton("" + i);
  button.setFont(buttonFont);
  button.addActionListener(bnListener);
  panelA.add(button);
}

button  = new JButton("X");
button.setFont(buttonFont);
button.addActionListener(bnListener);
panelA.add(button);

for (int i=4; i<=6; i++) {
  button  = new JButton("" + i);
  button.setFont(buttonFont);
  button.addActionListener(bnListener);
  panelA.add(button);
}

button  = new JButton("/");
button.setFont(buttonFont);
button.addActionListener(bnListener);
panelA.add(button);

for (int i=1; i<=3; i++) {
  button  = new JButton("" + i);
  button.setFont(buttonFont);
  button.addActionListener(bnListener);
  panelA.add(button);
}

button  = new JButton("+");
button.setFont(buttonFont);
button.addActionListener(bnListener);
panelA.add(button);

button  = new JButton("C");
button.setFont(buttonFont);
button.addActionListener(bnListener);
panelA.add(button);

button  = new JButton("0");
button.setFont(buttonFont);
button.addActionListener(bnListener);
panelA.add(button);

button  = new JButton("=");
button.setFont(buttonFont);
button.addActionListener(bnListener);
panelA.add(button);

button  = new JButton("-");
button.setFont(buttonFont);
button.addActionListener(bnListener);
panelA.add(button);

// add textfield and panel to contentPane
contentPane.add(display, BorderLayout.NORTH);
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
    contentPane.add(panelA, BorderLayout.CENTER);
  }


  public static void main(String argv[]) {
    Calculator2 frame = new Calculator2();
    frame.setTitle("Calculator");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(400, 300);
    frame.setLocation(300, 100);
    frame.setVisible(true);
  }
}

class Listener implements ActionListener {

  /** Event Handler when button is clicked */
  public void actionPerformed(ActionEvent e) {

    char c = e.getActionCommand().charAt(0);
    if (c >= '0' && c <= '9') {
      System.out.println("Button " + c + " clicked");
    }
  }
}
```

In order for a listener object to respond to an event, the listener must implement an event handler method. The event handler method for a button is the actionPerformed() method. Defines the Listener class. This class implements the ActionListener interface (in the java.awt.event package). Accordingly, all classes that implement the ActionListener interface must implement the actionPerformed() method. It is clear from the code that the outputs:

```
Button 1 clicked
Button 2 clicked
Button 3 clicked
Button 4 clicked
Button 5 clicked
Button 6 clicked
Button 7 clicked
Button 8 clicked
Button 9 clicked
Button 0 clicked
```

are produced from the actionPerformed() method when the buttons are clicked. Within the actionPerformed() method, e.getActionCommand() returns the action command of the button (i.e. the text of the button). Using charAt(0) method, the first character of the text is examined to determine what value to print.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

3.2.4 Event Handling Methods

There is a set of event handling methods (or handlers) defined for each listener interface. Table 3.1 lists the set of handlers for each event type.

TABLE 3.1: Event Handlers

S/ No.	External User Action	Source Object	Event Generated	Event Handling Methods that must be implemented by Listener Object	Listener Interface
1.	Click a button	JButton	ActionEvent	actionPerformed(ActionEvent e)	ActionListener
2.	Change text	JTextComponent	TextEvent	textValueChanged(TextEvent e)	TextListener
3.	Click return on a text field	JTextField	ActionEvent	Same as 1	Same as 1
4.	Select a new item	JComboBox	ItemEvent, ActionEvent	itemStateChanged(ItemEvent e) Same as 1	ItemListener Same as 1
5.	Select item	JList	ListSelectionEvent		
6.	Click a check box	JCheckBox	ItemEvent, ActionEvent	Same as 4	Same as 4
7.	Click a radio button	JRadioButton	ItemEvent, ActionEvent	Same as 4	Same as 4
8.	Select a menu item	JMenuItem	ActionEvent	Same as 1	Same as 1
9.	Move the scroll bar	JScrollBar	AdjustmentEvent	adjustmentValueChanged(AdjustmentEvent e)	AdjustmentListener
10.	Window opened, closed, iconified, deiconified, or closing	Window	WindowEvent	windowClosing(WindowEvent e) windowOpened(WindowEvent e) windowIconified(WindowEvent e) windowDeiconified(WindowEvent e) windowClosed(WindowEvent e) windowActivated(WindowEvent e) windowDeactivated(WindowEvent e)	WindowListener
11.	Component added or removed from the container	Container	ContainerEvent	componentAdded(ContainerEvent e) componentRemoved(ContainerEvent e)	ContainerListener
12.	Component moved, resized, hidden, or shown	Component	ComponentEvent	componentMoved(ContainerEvent e) componentHidden(ContainerEvent e) componentResized(ContainerEvent e) componentShown(ContainerEvent e)	ComponentListener
13.	Component gained or lost focus	Component	FocusEvent	focusGained(FocusEvent e) focusLost(FocusEvent e)	FocusListener
14.	Key	Component	KeyEvent	keyPressed(KeyEvent	KeyListener

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

	released or pressed			e) keyReleased(KeyEvent e) keyTyped(KeyEvent e)	
15.	Mouse pressed, released, clicked, entered, or exited	Component	MouseEvent	mousePressed(MouseEvent e) mouseReleased(MouseEvent e) mouseEntered(MouseEvent e) mouseExited(MouseEvent e) mouseClicked(MouseEvent e)	MouseListener
16.	Mouse moved or dragged	Component	MouseEvent	mouseDragged(MouseEvent e) mouseMoved(MouseEvent e)	MouseMotionListener

3.2.5 Frames as Listeners

The user class Calculator2 in Code 3.2 is a frame. Like the Listener class, Calculator2 can also be an action listener if it implements the ActionListener interface. In Code 3.3, the user class Calculator3 demonstrates how to make a frame as an ActionListener.

To behave as an action listener, Calculator3 has to implement the actionPerformed() method.

To be able to listen for events generated by the buttons, Calculator3 has to add itself to the buttons as ActionListener object using the this reference.

Code 3.3: The Calculator Frame implementing ActionListener

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

class Calculator3 extends JFrame implements ActionListener{


  Calculator3() {

    // Create a button object
    JButton button = new JButton("");

    // get content pane of frame
    Container contentPane = getContentPane();

    // set BorderLayout
    BorderLayout layout = new BorderLayout();
    contentPane.setLayout(layout);

    // creates a panel
    JPanel panelA = new JPanel();

    // creates a textField with a Helvetica font
    Font displayFont = new Font("Arial", Font.BOLD, 45);
    JTextField display = new JTextField("1234567890");
    display.setHorizontalAlignment(JTextField.RIGHT);
    display.setFont(displayFont);

    // creates a standard font
    Font buttonFont = new Font("Verdana", Font.PLAIN, 20);

    // add buttons into panelA
    panelA.setLayout(new GridLayout(4,4));

    for (int i=7; i<=9; i++) {
      button  = new JButton("" + i);
```

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

```
      button.setFont(buttonFont);
      button.addActionListener(this);
      panelA.add(button);
    }

    button  = new JButton("X");
    button.setFont(buttonFont);
    button.addActionListener(this);
    panelA.add(button);

    for (int i=4; i<=6; i++) {
      button  = new JButton("" + i);
      button.setFont(buttonFont);
      button.addActionListener(this);
      panelA.add(button);
    }

    button  = new JButton("/");
    button.setFont(buttonFont);
    button.addActionListener(this);
    panelA.add(button);

    for (int i=1; i<=3; i++) {
      button  = new JButton("" + i);
      button.setFont(buttonFont);
      button.addActionListener(this);
      panelA.add(button);
    }

    button  = new JButton("+");
    button.setFont(buttonFont);
    button.addActionListener(this);
    panelA.add(button);

    button  = new JButton("C");
    button.setFont(buttonFont);
    button.addActionListener(this);
    panelA.add(button);

    button  = new JButton("0");
    button.setFont(buttonFont);
    button.addActionListener(this);
    panelA.add(button);

    button  = new JButton("=");
    button.setFont(buttonFont);
    button.addActionListener(this);
    panelA.add(button);

    button  = new JButton("-");
    button.setFont(buttonFont);
    button.addActionListener(this);
    panelA.add(button);

    // add textfield and panel to contentPane
    contentPane.add(display, BorderLayout.NORTH);
    contentPane.add(panelA, BorderLayout.CENTER);
  }

  /** Event Handler when button is clicked */
  public void actionPerformed(ActionEvent e) {

    char c = e.getActionCommand().charAt(0);
    if (c >= '0' && c <= '9') {
      System.out.println("Button " + c + " clicked");
    }
  }


  public static void main(String argv[]) {
    Calculator3 frame = new Calculator3();
    frame.setTitle("Calculator");
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setLocation(300, 100);
    frame.setSize(400, 300);
    frame.setVisible(true);
  }
}
```

3.3 Completing the Calculator

So far, our Calculator has a decent look-and-feel. It is also able to respond to events. What is lacking in our Calculator is an engine to do the arithmetical calculations for the calculator.

The calculator we have designed is an integer calculator that does not return decimal values (e.g. 3 divide by 6 returns 0 rather than 0.5, and 9 divide by 2 returns 4 rather than 4.5).

3.3.1 Calculator Engine

Code 3.4 illustrates a calculator engine. It provides an abstraction of functionalities required for the computation of numbers and operations entered by the user. Some salient characteristics of the code follow:

1. Three variables are declared as registers for storing values in the computation process: entry is the number entered by user, stored stores the intermediate entry and value is the computed value.
2. binaryOperation() is a generic operation for handling the four types of operation (add, subtract, multiply and divide). This operation takes in two operands. When the first entry is entered, it is stored away while waiting for the second entry. The binaryOperation() computes the value when both entries are available. The entry variable is subsequently initialized. The Boolean variable firstEntry is used to detect if more than one entry has been entered. stored contains the computed value.
3. Add, subtract, multiply and divide operation make use of the binaryOperation() method to compute the value.
4. compute() adds, subtracts, multiplies or divides depending on the operation requested.
5. computeFinal() is similar to compute() except that it is a final compute for the series of operations entered.
6. clear() initializes entry, value, and stored.
7. digit() takes in a digit one at a time and turn them into a number for processing.
8. displayValue() returns the computed value.
9. displayEntry() returns the last entry.
10. To demonstrate how the calculator engine works, the following sequence of numbers and operations are passed to the engine via main(): 100 / 10 – 13 + 32 * 3 = 87.
11. Executing CalculatorEngine produces 87 on the command prompt.
12. As mentioned earlier, the calculator returns 90 for 100 / 9 – 13 + 32 * 3 instead of 90.3333 since the calculator only return integer values.

Code 3.4: Calculator Engine

```
class CalculatorEngine {

  // declare three variables as calculator registers
  private int entry     = 0;
  private int stored    = 0;
  private int value     = 0;

  // declare variables for processing
  private char    operation  = ' ';
  private boolean firstEntry = true;


  CalculatorEngine() { clear(); }

  void binaryOperation(char op)  {
    if (!firstEntry) { // not the first entry
      compute();
```

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

```
    operation = op;
    stored    = value;
    entry     = 0;
  }
  else {
    stored     = entry;
    entry      = 0;
    operation  = op;
    firstEntry = false;
    value      = 0;
  }
}

void add()      { binaryOperation('+'); }

void subtract() { binaryOperation('-'); }

void multiply() { binaryOperation('*'); }

void divide()   { binaryOperation('/'); }

void compute() {
  if (operation == '+')
    value = stored + entry;
  else if (operation == '-')
    value = stored - entry;
  else if (operation == '*')
    value = stored * entry;
  else if (operation == '/')
    value = stored / entry;
  stored = 0;
  entry  = 0;
}

void computeFinal() {
  if (operation == '+')
    value = stored + entry;
  else if (operation == '-')
    value = stored - entry;
  else if (operation == '*')
    value = stored * entry;
  else if (operation == '/')
    value = stored / entry;
  firstEntry = true;
}

void clear() {
  entry  = 0;
  value  = 0;
  stored = 0;
}

void digit(int x)  {
  entry = entry *10 + x;
}

int displayValue() {
  return(value);
}

int displayEntry() {
  return(entry);
}

public static void main(String arg[]) {
  CalculatorEngine c = new CalculatorEngine();
  c.digit(1);
  c.digit(0);
  c.digit(0);
  c.divide();
  c.digit(1);
  c.digit(0);
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
    c.subtract();
    c.digit(1);
    c.digit(3);
    c.add();
    c.digit(3);
    c.digit(2);
    c.multiply();
    c.digit(3);
    c.computeFinal();
    System.out.println(c.displayValue());
  }
}
```

3.3.2 A Fully Working Calculator

Let us connect the engine to the calculator. Code 3.5 represents the modified calculator. The major change is in the actionPerformed() method – the event handler for the Calculator. Any click on the buttons invokes this method.

Code 3.5: Calculator with Engine Calls in Event Handler

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

class Calculator extends JFrame implements ActionListener{

  // Create a calculator engine to compute values
  CalculatorEngine engine = new CalculatorEngine();
  JTextField display;

  Calculator() {

    // Create a button object
    JButton button = new JButton("");

    // get content pane of frame
    Container contentPane = getContentPane();

    // set BorderLayout
    BorderLayout layout = new BorderLayout();
    contentPane.setLayout(layout);

    // creates a panel
    JPanel panelA = new JPanel();

    // creates a textField with a Helvetica font
    Font displayFont = new Font("Arial", Font.BOLD, 45);
    display = new JTextField("0");
    display.setHorizontalAlignment(JTextField.RIGHT);
    display.setFont(displayFont);

    // creates a standard font
    Font buttonFont = new Font("Verdana", Font.PLAIN, 20);

    // add buttons into panelA
    panelA.setLayout(new GridLayout(4,4));

    for (int i=7; i<=9; i++) {
      button  = new JButton("" + i);
      button.setFont(buttonFont);
      button.addActionListener(this);
      panelA.add(button);
    }

    button  = new JButton("X");
    button.setFont(buttonFont);
    button.addActionListener(this);
    panelA.add(button);
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
    for (int i=4; i<=6; i++) {
      button  = new JButton("" + i);
      button.setFont(buttonFont);
      button.addActionListener(this);
      panelA.add(button);
    }

    button  = new JButton("/");
    button.setFont(buttonFont);
    button.addActionListener(this);
    panelA.add(button);

    for (int i=1; i<=3; i++) {
      button  = new JButton("" + i);
      button.setFont(buttonFont);
      button.addActionListener(this);
      panelA.add(button);
    }

    button  = new JButton("+");
    button.setFont(buttonFont);
    button.addActionListener(this);
    panelA.add(button);

    button  = new JButton("C");
    button.setFont(buttonFont);
    button.addActionListener(this);
    panelA.add(button);

    button  = new JButton("0");
    button.setFont(buttonFont);
    button.addActionListener(this);
    panelA.add(button);

    button  = new JButton("=");
    button.setFont(buttonFont);
    button.addActionListener(this);
    panelA.add(button);

    button  = new JButton("-");
    button.setFont(buttonFont);
    button.addActionListener(this);
    panelA.add(button);

    // add textfield and panel to contentPane
    contentPane.add(display, BorderLayout.NORTH);
    contentPane.add(panelA, BorderLayout.CENTER);
  }

  /** Event Handler when button is clicked */
  public void actionPerformed(ActionEvent e) {

    char c = e.getActionCommand().charAt(0);
    if (c == '+') engine.add();
    else
      if (c == '-') engine.subtract();
      else
        if (c == 'X') engine.multiply();
        else
          if (c == '/') engine.divide();
          else
            if (c >= '0' && c <= '9') {
              engine.digit(c - '0');
              display.setText(new Integer(
                              engine.displayEntry()).toString());
            }
            else
              if (c == '=') {
                engine.computeFinal();
                display.setText(new Integer(
                                engine.displayValue()).toString());
                engine.clear();
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
            }
            else
              if (c == 'C') {
                engine.clear();
                display.setText("0");
              }
  }

  public static void main(String argv[]) {
    Calculator frame = new Calculator();
    frame.setTitle("Calculator");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(400, 300);
    frame.setLocation(300, 100);
    frame.setVisible(true);
  }
}
```

A series of if…else statement checks on the requested action command. The calculator responds to action command accordingly, with the engine supporting the calculation in the backend. Figure 3.4 shows a snapshot of a result in the use of the calculator.

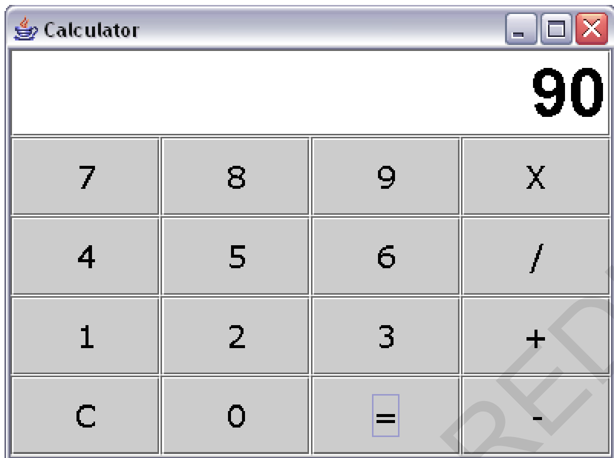


FIGURE 3.4: Calculator with Engine Calls in Event Handler

NOT FOR REDISTRIBUTION

3.4 Window Listeners

Besides buttons, there are other objects that generate events. For example, a minimizer, maximizer or close icon, typically found at the top right-hand corner of a window, triggers an event when clicked. The Calculator frame can also respond to such events. All we need to do is to add listener objects to the minimizer, maximizer and close icons. Since these events are related to windows, we add *window listeners* to the Calculator frame.

3.4.1 Listening to Events

Code 3.6 illustrates event handling associated with windows. Some salient characteristics of the code follow:

1. Implement ActionListener and WindowListener. The Calculator frame is now able to listen to events from the buttons and windows.
2. Adds the frame itself as a window listener (i.e. the frame is listening to events happening to itself via the minimizer, maximizer and close symbols). Note that for the frame to be a window listener, the frame must *implement* all the methods defined in the java.awt.event.WindowListener interface: windowActivated(), windowClosed(), windowClosing(), windowDeactivated(), windowDeiconified(), windowIconified(), and windowOpened(). We have placed a

Copyright © 2020, Danny Poo.

System.out.println() statement in each of these methods to print the name of the method on the command prompt.

Execute Code 3.6 and observe the behavior of the Calculator in the command prompt. Table 3.2 summarizes what happens when the Calculator is used.

TABLE 3.2: Actions on Calculator Frame and Outputs at Command Prompt

Action	Output at DOS Prompt
When the application started	windowActivated windowOpened
When the minimizer is clicked	windowIconified windowDeactivated
When the maximizer is clicked	windowDeiconified windowActivated
When the close symbol is clicked	windowClosing

The event handler for the close icon is windowClosing() and not windowClosed() method as expected. windowClosing() method causes the window to close and the application to exit. This statement has the same effect as frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE) which we have deliberately commented to demonstrate the effect of System.exit(0).

Code 3.6: Adding Window Listener to the Calculator

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

class Calculator4 extends JFrame
  implements ActionListener, WindowListener {

  // Create a calculator engine to compute values
  CalculatorEngine engine =  new CalculatorEngine();
  JTextField display;

  Calculator4() {

    // Create a button object
    JButton button = new JButton("");

    // get content pane of frame
    Container contentPane = getContentPane();

    // set BorderLayout
    BorderLayout layout = new BorderLayout();
    contentPane.setLayout(layout);

    // creates a panel
    JPanel panelA = new JPanel();

    // creates a textField with a Helvetica font
    Font displayFont = new Font("Arial", Font.BOLD, 45);
    display = new JTextField("0");
    display.setHorizontalAlignment(JTextField.RIGHT);
    display.setFont(displayFont);

    // creates a standard font
    Font buttonFont = new Font("Verdana", Font.PLAIN, 20);

    // add buttons into panelA
    panelA.setLayout(new GridLayout(4,4));

    for (int i=7; i<=9; i++) {
      button  = new JButton("" + i);
      button.setFont(buttonFont);
      button.addActionListener(this);
      panelA.add(button);
    }
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
    button  = new JButton("X");
    button.setFont(buttonFont);
    button.addActionListener(this);
    panelA.add(button);

    for (int i=4; i<=6; i++) {
      button  = new JButton("" + i);
      button.setFont(buttonFont);
      button.addActionListener(this);
      panelA.add(button);
    }

    button  = new JButton("/");
    button.setFont(buttonFont);
    button.addActionListener(this);
    panelA.add(button);

    for (int i=1; i<=3; i++) {
      button  = new JButton("" + i);
      button.setFont(buttonFont);
      button.addActionListener(this);
      panelA.add(button);
    }

    button  = new JButton("+");
    button.setFont(buttonFont);
    button.addActionListener(this);
    panelA.add(button);

    button  = new JButton("C");
    button.setFont(buttonFont);
    button.addActionListener(this);
    panelA.add(button);

    button  = new JButton("0");
    button.setFont(buttonFont);
    button.addActionListener(this);
    panelA.add(button);

    button  = new JButton("=");
    button.setFont(buttonFont);
    button.addActionListener(this);
    panelA.add(button);

    button  = new JButton("-");
    button.setFont(buttonFont);
    button.addActionListener(this);
    panelA.add(button);

    // add textfield and panel to contentPane
    contentPane.add(display, BorderLayout.NORTH);
    contentPane.add(panelA, BorderLayout.CENTER);
    addWindowListener(this);
  }

  /** Event Handler when button is clicked */
  public void actionPerformed(ActionEvent e) {

    char c = e.getActionCommand().charAt(0);
    if (c == '+') engine.add();
    else
      if (c == '-') engine.subtract();
      else
        if (c == 'X') engine.multiply();
        else
          if (c == '/') engine.divide();
          else
            if (c >= '0' && c <= '9') {
              engine.digit(c - '0');
              display.setText(new Integer(
                              engine.displayEntry()).toString());
            }
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
          else
            if (c == '=') {
              engine.computeFinal();
              display.setText(new Integer(
                              engine.displayValue()).toString());
              engine.clear();
            }
            else
              if (c == 'C') {
                engine.clear();
                display.setText("0");
              }
  }

  public void windowActivated(WindowEvent e) {
    System.out.println("windowActivated");
  }

  public void windowClosed(WindowEvent e) {
    System.out.println("windowClosed");
  }

  public void windowClosing(WindowEvent e) {
    System.out.println("windowClosing");
    System.exit(0);
  }

  public void windowDeactivated(WindowEvent e) {
    System.out.println("windowDeactivated");
  }

  public void windowDeiconified(WindowEvent e) {
    System.out.println("windowDeiconified");
  }

  public void windowIconified(WindowEvent e) {
    System.out.println("windowIconified");
  }

  public void windowOpened(WindowEvent e) {
    System.out.println("windowOpened");
  }

  public static void main(String argv[]) {
    Calculator4 frame = new Calculator4();
    frame.setTitle("Calculator");
    //frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(400, 300);
    frame.setLocation(300, 100);
    frame.setVisible(true);
  }
}
```

NOT FOR REDISTRIBUTION

3.4.2 windowClosed() vs windowClosing()

What is the difference between windowClosed() and windowClosing()? windowClosed() is invoked when a window has been closed as a result of calling dispose() on the window. windowClosing() is invoked when the user *attempts to close* the window from the window's system menu. As an example, we will add the statement this.dispose() in the windowIconified() method:

```
  public void windowIconified(WindowEvent e) {
    System.out.println("windowIconified");
    this.dispose();
  }
```

When the window is minimized, windowIconified() is invoked and the dispose() method is activated. The dispose() method closes the window. At the command prompt, we observe:

Copyright © 2020, Danny Poo.

```
windowIconified
windowDeactivated
windowClosed
```

This suggests that the window is iconified and deactivated; the windowClosed() method is subsequently invoked when this.dispose() executes, and the application exits.

All methods in the WindowListener interface *must be implemented* even if your program does not use these methods. Code 3.7 is the complete code for demonstrating windowClosed() and windowClosing().

Code 3.7: windowClosed() vs windowClosing()

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

class Calculator5 extends JFrame
  implements ActionListener, WindowListener {

  // Create a calculator engine to compute values
  CalculatorEngine engine =  new CalculatorEngine();
  JTextField display;

  Calculator5() {

    // Create a button object
    JButton button = new JButton("");

    // get content pane of frame
    Container contentPane = getContentPane();

    // set BorderLayout
    BorderLayout layout = new BorderLayout();
    contentPane.setLayout(layout);

    // creates a panel
    JPanel panelA = new JPanel();

    // creates a textField with a Helvetica font
    Font displayFont = new Font("Arial", Font.BOLD, 45);
    display = new JTextField("0");
    display.setHorizontalAlignment(JTextField.RIGHT);
    display.setFont(displayFont);

    // creates a standard font
    Font buttonFont = new Font("Verdana", Font.PLAIN, 20);

    // add buttons into panelA
    panelA.setLayout(new GridLayout(4,4));

    for (int i=7; i<=9; i++) {
      button  = new JButton("" + i);
      button.setFont(buttonFont);
      button.addActionListener(this);
      panelA.add(button);
    }

    button  = new JButton("X");
    button.setFont(buttonFont);
    button.addActionListener(this);
    panelA.add(button);

    for (int i=4; i<=6; i++) {
      button  = new JButton("" + i);
      button.setFont(buttonFont);
      button.addActionListener(this);
      panelA.add(button);
    }

    button  = new JButton("/");
    button.setFont(buttonFont);
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
    button.addActionListener(this);
    panelA.add(button);

    for (int i=1; i<=3; i++) {
      button  = new JButton("" + i);
      button.setFont(buttonFont);
      button.addActionListener(this);
      panelA.add(button);
    }

    button  = new JButton("+");
    button.setFont(buttonFont);
    button.addActionListener(this);
    panelA.add(button);

    button  = new JButton("C");
    button.setFont(buttonFont);
    button.addActionListener(this);
    panelA.add(button);

    button  = new JButton("0");
    button.setFont(buttonFont);
    button.addActionListener(this);
    panelA.add(button);

    button  = new JButton("=");
    button.setFont(buttonFont);
    button.addActionListener(this);
    panelA.add(button);

    button  = new JButton("-");
    button.setFont(buttonFont);
    button.addActionListener(this);
    panelA.add(button);

    // add textfield and panel to contentPane
    contentPane.add(display, BorderLayout.NORTH);
    contentPane.add(panelA, BorderLayout.CENTER);
    addWindowListener(this);
  }

  /** Event Handler when button is clicked */
  public void actionPerformed(ActionEvent e) {

    char c = e.getActionCommand().charAt(0);
    if (c == '+') engine.add();
    else
      if (c == '-') engine.subtract();
      else
        if (c == 'X') engine.multiply();
        else
          if (c == '/') engine.divide();
          else
            if (c >= '0' && c <= '9') {
              engine.digit(c - '0');
              display.setText(new Integer(
                              engine.displayEntry()).toString());
            }
            else
              if (c == '=') {
                engine.computeFinal();
                display.setText(new Integer(
                                engine.displayValue()).toString());
                engine.clear();
              }
              else
                if (c == 'C') {
                  engine.clear();
                  display.setText("0");
                }
  }
```

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

```
  public void windowActivated(WindowEvent e) {
    System.out.println("windowActivated");
  }

  public void windowClosed(WindowEvent e) {
    System.out.println("windowClosed");
  }

  public void windowClosing(WindowEvent e) {
    System.out.println("windowClosing");
    System.exit(0);
  }

  public void windowDeactivated(WindowEvent e) {
    System.out.println("windowDeactivated");
  }

  public void windowDeiconified(WindowEvent e) {
    System.out.println("windowDeiconified");
  }

  public void windowIconified(WindowEvent e) {
    System.out.println("windowIconified");
    this.dispose();
  }

  public void windowOpened(WindowEvent e) {
    System.out.println("windowOpened");
  }

  public static void main(String argv[]) {
    Calculator5 frame = new Calculator5();
    frame.setTitle("Calculator");
    //frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(400, 300);
    frame.setLocation(300, 100);
    frame.setVisible(true);
  }
}
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

CHAPTER 4: LABELS, BUTTONS AND COMBO BOXES

We have been discussing *containers* in the last few chapters. It is time for us to shift our attention to *components*. There are three types of component that are commonly used in graphical user interface design and we will discuss them in this chapter; they include:

1. Labels – display areas for short texts, images or both
2. Buttons – components that trigger action events when clicked
3. ComboBoxes – users can choose an item from a list of choices

4.1 Labels

A *label* is a display area for a short text string, image or both. Labels are often used to label components such as text fields. Four labels are shown in Figure 4.1 – two text labels and two gif labels.

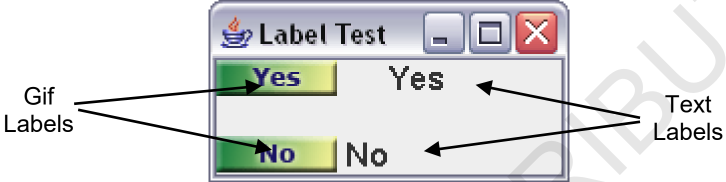


FIGURE 4.1: Labels

4.1.1 JLabel

Labels are implemented by the JLabel class in the Java Swing. Figure 4.2 is the class hierarchy for JLabel.

```
java.lang.Object
└──java.awt.Component
    └──java.awt.Container
        └──javax.swing.JComponent
            └── javax.swing.JLabel
```

FIGURE 4.2: Class Hierarchy of JLabel

Note that JLabel is a subclass of JComponent, Container and Component. This suggests that all fields and methods defined in JComponent, Container and Component are applicable to JLabel.

4.1.1.1 What are the Constructors?

JLabel has six constructor methods (see Table 4.1).

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

TABLE 4.1: JLabel Constructors

No.	Constructor	Description
1	`JLabel()`	Creates a new JLabel object with no set text or image.
2	`JLabel(Icon image)`	Creates a new JLabel object with the specified image.
3	`JLabel(Icon image, int horizontalAlignment)`	Creates a new JLabel object with the specified image and horizontal alignment.
4	`JLabel(String text)`	Creates a new JLabel object with the specified text.
5	`JLabel(String text, Icon image, int horizontalAlignment)`	Creates a new JLabel object with the specified text, image and horizontal alignment.
6	`JLabel(String text, int horizontalAlignment)`	Creates a new JLabel object with the specified text and horizontal alignment.

4.1.1.2 How are Events Handled?

A JLabel object generates events that Component and JComponent objects are capable of e.g. mouse events. In order for a JLabel object to be responsive to mouse events, mouse listeners must be added to JLabel objects. There are six methods that must be implemented by the listeners and they include: mouseClicked(), mouseEntered(), mouseExited(), mousePressed(), and mouseReleased().

4.1.1.3 Show Me an Application of JLabel

Let us look at how Figure 4.1 can be produced (see Code 4.1):

1. The third constructor in Table 4.1 is used to create a label (yesGif) with an image; the image is aligned horizontally to the left. SwingConstants.LEFT, a constant from the javax.swing.SwingConstants interface, sets the alignment to the left. This interface contains all the constants required for Swing components. All Swing components implement SwingConstants. Therefore, the constants in SwingConstants can be directly referenced by Swing components e.g. we may replace SwingConstants.LEFT with JLabel.LEFT.
2. Other labels are similarly constructed using the various types of constructors in Table 4.1.
3. A GridLayout is set on the contentPane to arrange five labels. The image "No" and the text "No" have been deliberately grouped together as one label. The image "Yes" and the text "Yes", however, are two separate labels. ***This explains why the texts on the display are not in line with one another, although they have been aligned to the left horizontally.*** The empty space between the images is due to the two empty labels used in the code.

Code 4.1: Label Test

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

class Labels extends JFrame implements MouseListener{

  // Create labels
  JLabel yesGif  = new JLabel(new ImageIcon("../images/yes_button.gif"),
                              SwingConstants.LEFT);
  JLabel yesText = new JLabel("Yes", SwingConstants.LEFT);
  JLabel noGif   = new JLabel("No", new ImageIcon("../images/no_button.gif"),
                              SwingConstants.RIGHT);
  JLabel empty1 = new JLabel("");
  JLabel empty2 = new JLabel("");

  Labels() {

    // get content pane of frame
    Container contentPane = getContentPane();
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
    // set GridLayout
    GridLayout layout = new GridLayout(3,1);
    contentPane.setLayout(layout);

    // creates a font
    Font font = new Font("Arial", Font.BOLD, 16);

    // set font
    yesText.setFont(font);
    noGif.setFont(font);

    // add listener
    yesText.addMouseListener(this);
    yesGif.addMouseListener(this);
    noGif.addMouseListener(this);

    // add labels to contentPane
    contentPane.add(yesGif);
    contentPane.add(yesText);
    contentPane.add(empty1); // to create empty space
    contentPane.add(empty2); // to create empty space
    contentPane.add(noGif);
  }

  // Event Handlers
  public void mouseClicked(MouseEvent e) {
    System.out.println("mouseClicked");
  }

  public void mouseEntered(MouseEvent e) {
    System.out.println("mouseEntered");
  }

  public void mouseExited(MouseEvent e) {
    System.out.println("mouseExited");
  }

  public void mousePressed(MouseEvent e) {
    System.out.println("mousePressed");
  }

  public void mouseReleased(MouseEvent e) {
    System.out.println("mouseReleased");
  }

  public static void main(String argv[]) {

    // Create frame
    Labels frame = new Labels();
    frame.setTitle("Label Test");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setLocation(300, 100);
    frame.setSize(400, 300);

    //Display frame
    frame.pack();
    frame.setVisible(true);
  }

}
```

4.1.1.4 How to Implement Mouse Listeners?

The “Label Test” frame is tasked to listen to all mouse events on yesText, yesGif and noGif labels. Event handlers for mouse events are included: mouseClicked(), mouseEntered(), mouseExited(), mousePressed(), and mouseReleased(). Move the mouse to the labels and you will notice, on the command prompt, which mouse events are activated and what are generated in respond to the events:

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

```
mouseEntered
mouseExited
mouseEntered
mouseExited
mouseEntered
mouseExited
mouseEntered
mouseExited
mouseEntered
mouseExited
mouseEntered
mouseExited
mouseEntered
mouseExited
```

The size of the display is much smaller than those shown in Chapter 2 and 3 even though the size of the frame has been set to be the same at (400, 300). Why? The answer lies in the statement **frame.pack()**. pack() is a method inherited from the Window class in java.awt package. This method causes the frame to be sized to fit the preferred size and layouts of its components.

4.2 Buttons

Buttons have been introduced in Chapter 2 and you possibly have used them in web pages – the ubiquitous "Submit" button. In Java, a button is a component that triggers an action event when clicked. There are altogether three types of button in the Java Swing: JButton, JCheckBox, and JRadioButton.

4.2.1 JButton

The standard button is implemented in Java as JButton. Figure 4.3 is the class hierarchy for JButton.

```
java.lang.Object
└──java.awt.Component
    └──java.awt.Container
        └──javax.swing.JComponent
            └── javax.swing.AbstractButton
                └── javax.swing.JButton
```

FIGURE 4.3: Class Hierarchy of JButton

A JButton class is a direct subclass of AbstractButton which defines the common behavior for buttons and menu items. JButton is also a subclass of JComponent, Container and Component. All fields and methods defined in JComponent, Container and Component are therefore applicable to JButton. Technically, a JButton object can behave like a container e.g. setting layout on it to contain other components. However, for practical reason, this is not encouraged.

4.2.1.1 What are the Constructors?

JButton objects can be created using any of the five constructors in Table 4.2.

TABLE 4.2: JButton Constructors

No.	Constructor	Description
1	`JButton()`	**Creates a new JButton object with no set text or icon. To set text, use setText("text") method inherited from AbstractButton class.**
2	`JButton(Action a)`	Creates a new JButton object where properties are taken from the Action supplied.
3	`JButton(Icon icon)`	Creates a new JButton object with an icon. An icon is a fixed-size picture stored in an image file.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

4	`JButton(String text)`	Creates a new JButton object with an initial text.
5	`JButton(String text, Icon icon)`	Creates a new JButton object with an initial text and an icon.

4.2.1.2 How are Events Handled?

A JButton object generates action events. For a JButton object to respond to action events, the container in which the button is a component of must implement the actionPerformed() method in the ActionListener interface. That is, the container is made an ActionListener.

4.2.1.3 Show Me an Application of JButton

Figure 4.4 shows four buttons on a frame. Some salient characteristics of the code follow:

1. When the "Yes" buttons (image and text buttons) are clicked, a "Yes Button Clicked" message is displayed on the command prompt. Similarly, when the "No" buttons are clicked, you should get a "No Button Clicked" displayed on the prompt.
2. The top two buttons have set texts on the buttons. Notice "Y" and "N" are underlined. "Y" and "N" are shortcut keys to the "Yes" and "No" button respectively. To activate shortcut keys (also known as mnemonics), press ALT+Y (i.e. press the "ALT" key and the "Y" key on the keyboard together) for "Yes" button and ALT+N for "No" button.
3. The bottom two buttons use icons to represent the buttons.

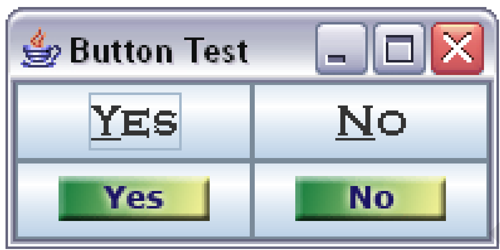


FIGURE 4.4: Buttons with Set Texts and Icons

Code 4.1: Button Test

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

class Buttons extends JFrame implements ActionListener{

  // Create button objects
  JButton yesGifButton  = new JButton("");
  JButton yesTextButton = new JButton("Yes");
  JButton noGifButton   = new JButton("");
  JButton noTextButton  = new JButton("No");

  Buttons() {
    // set mnemonics for text buttons
    yesTextButton.setMnemonic('Y');
    noTextButton.setMnemonic('N');

    // set image icon for buttons
    yesGifButton.setIcon(new ImageIcon("../images/yes_button.gif"));
    noGifButton.setIcon(new ImageIcon("../images/no_button.gif"));

    // get content pane of frame
    Container contentPane = getContentPane();

    // set GridLayout
    GridLayout layout = new GridLayout(2,2);
    contentPane.setLayout(layout);
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
    // creates a button font
    Font buttonFont = new Font("Copperplate Gothic Bold", Font.PLAIN, 18);

    // set button font
    yesTextButton.setFont(buttonFont);
    noTextButton.setFont(buttonFont);

    // add listener
    yesTextButton.addActionListener(this);
    noTextButton.addActionListener(this);
    yesGifButton.addActionListener(this);
    noGifButton.addActionListener(this);

    // add buttons to contentPane
    contentPane.add(yesTextButton);
    contentPane.add(noTextButton);
    contentPane.add(yesGifButton);
    contentPane.add(noGifButton);
  }

  /** Event Handler when button is clicked */
  public void actionPerformed(ActionEvent e) {

    if ((e.getSource() == yesTextButton) || (e.getSource() == yesGifButton))
      System.out.println("Yes Button Clicked");
    else
      System.out.println("No Button Clicked");
  }


  public static void main(String argv[]) {

    // Create frame
    Buttons frame = new Buttons();
    frame.setTitle("Button Test");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setLocation(300, 100);

    //Display frame
    frame.pack();
    frame.setVisible(true);
  }

}
```

NOT FOR REDISTRIBUTION

Figure 4.4 is produced from Code 4.2. Let us examine the code:

1. Make use of the first constructor (see Table 4.2) to create two buttons with icons set later.
2. Make use of the fourth constructor to set texts for two new "Yes" and "No" buttons.
3. These two buttons are also set with mnemonics. The mnemonic used must be one of the letters in the button; otherwise, users will not be able to distinguish which letter to use to trigger the button.
4. A new image icon is first created and then set. The image is taken from an image file ("yes_button.gif"). The same applies to the "No" button icon.
5. A GridLayout layout manager is used to set the layout for the frame. No panel is used here. The frame has been tasked to respond to the events generated by the four buttons.
6. The buttons are subsequently added into the contentPane of the frame.
7. The actionPerformed() method is implemented to respond to the events. Depending on the source of the event, the application displays "Yes Button Clicked" or "No Button Clicked" on the command prompt.

4.2.2 JCheckBox

A *checkbox* is a component that can be selected or deselected. Much like a switch, a checkbox can be turned on or off. The state of a checkbox is visible to the user. Figure 4.5 shows an example of checkboxes.

Copyright © 2020, Danny Poo.

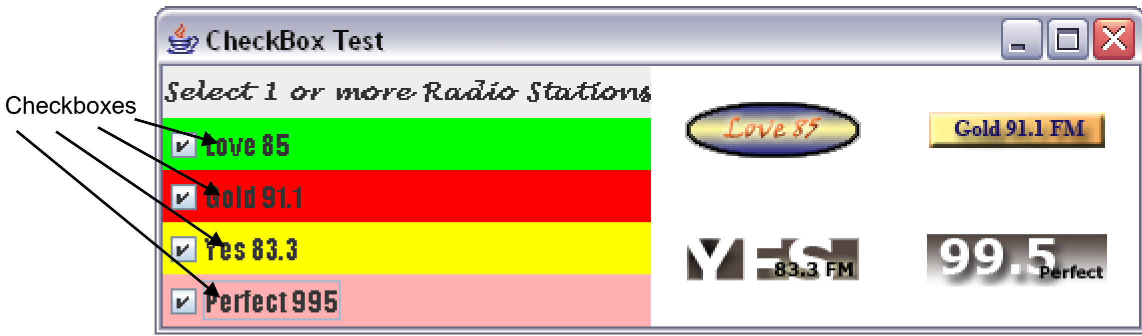


FIGURE 4.5: Selecting Radio Stations using CheckBoxes

The Java Swing version of a checkbox is JCheckBox. It is manifested as a square. Checking the box with a tick indicates that the option is selected and a blank indicates a deselection of the option. The class hierarchy for JCheckBox is shown in Figure 4.6. The JCheckBox class is a direct subclass of JToggleButton which is an implementation of a two-state button. A JCheckBox is therefore a two-state button.

```
java.lang.Object
└── java.awt.Component
     └── java.awt.Container
          └── javax.swing.JComponent
               └── javax.swing.AbstractButton
                    └── javax.swing.JToggleButton
                         └── javax.swing.JCheckBox
```

FIGURE 4.6: Class Hierarchy of JcheckBox

4.2.2.1 What are the Constructors?

JCheckBox objects can be created using any of the eight constructors in Table 4.3.

TABLE 4.3: JCheckBox Constructors

No.	Constructor	Description
1	`JCheckBox()`	Creates a new, initially unselected JCheckBox object with no set text or icon. To set text, use setText(“text”) method inherited from AbstractButton class.
2	`JCheckBox(Action a)`	Creates a new JCheckBox object where properties are taken from the Action supplied.
3	`JCheckBox(Icon icon)`	Creates a new, initially unselected JCheckBox object with an icon.
4	`JCheckBox(Icon icon, boolean selected)`	Creates a new JCheckBox object with an icon and specifies whether or not it is initially selected.
5	`JCheckBox(String text)`	Creates a new, initially unselected JCheckBox object with text.
6	`JCheckBox(String text, boolean selected)`	Creates a new JCheckBox object with text and specifies whether or not it is initially selected.
7	`JCheckBox(String text, Icon icon)`	Creates a new, initially unselected JCheckBox object with specified text and icon.
8	`JCheckBox(String text, Icon icon, boolean selected)`	Creates a new JCheckBox object with text and icon, and specifies whether or not it is initially selected.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

4.2.2.2 How are Events Handled?

A JCheckBox object generates two types of event: ActionEvent and ItemEvent. For a container to respond to these events, it must implements the actionPerformed() method of ActionListener interface and itemStateChanged() method of ItemListener interface.

4.2.2.3 Show Me an Application of JCheckBox

Earlier in Figure 4.5 we show an application that allows users to select one or more radio stations. We can use checkboxes to facilitate the selection. In Figure 4.5, each radio station is named by a label and its icon displayed on the right side of the frame when the radio station is selected.

Code 4.3: CheckBox Test
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class CheckBoxes extends JFrame implements ActionListener, ItemListener{

  // Create checkbox objects
  JCheckBox love85       = new JCheckBox("Love 85");
  JCheckBox gold911      = new JCheckBox("Gold 91.1", true);
  JCheckBox yes833       = new JCheckBox("Yes 83.3");
  JCheckBox perfect995   = new JCheckBox("Perfect 995");

  // create image labels
  JLabel love85Image     = new JLabel(new ImageIcon("../images/love85.gif"));
  JLabel gold911Image    = new JLabel(new ImageIcon("../images/gold911.gif"));
  JLabel yes833Image     = new JLabel(new ImageIcon("../images/yes833.gif"));
  JLabel perfect995Image = new JLabel(new ImageIcon("../images/perfect995.gif"));

  // Create label
  JLabel    label  = new JLabel("Select 1 or more Radio Stations");


  CheckBoxes() {

    // create 2 panels
    JPanel leftPanel  = new JPanel();
    JPanel rightPanel = new JPanel();
    rightPanel.setBackground(Color.white);

    // get content pane of frame
    Container contentPane = getContentPane();

    // set GridLayout
    GridLayout layout = new GridLayout(1,2);
    contentPane.setLayout(layout);

    // creates a label font
    Font labelFont = new Font("Lucida handwriting", Font.BOLD, 12);

    // creates a checkbox font
    Font checkBoxFont = new Font("Impact", Font.PLAIN, 14);

    // set font and background colors
    label.setFont(labelFont);
    love85.setFont(checkBoxFont);
    gold911.setFont(checkBoxFont);
    yes833.setFont(checkBoxFont);
    perfect995.setFont(checkBoxFont);
    label.setBackground(Color.white);
    love85.setBackground(Color.green);
    gold911.setBackground(Color.red);
    yes833.setBackground(Color.yellow);
    perfect995.setBackground(Color.pink);

    // set visibility
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
    love85Image.setVisible(false);
    gold911Image.setVisible(true);
    yes833Image.setVisible(false);
    perfect995Image.setVisible(false);

    // add listener
    love85.addActionListener(this);
    love85.addItemListener(this);
    gold911.addActionListener(this);
    gold911.addItemListener(this);
    yes833.addActionListener(this);
    yes833.addItemListener(this);
    perfect995.addActionListener(this);
    perfect995.addItemListener(this);

    // set panel layout
    leftPanel.setLayout(new GridLayout(5,1));
    rightPanel.setLayout(new GridLayout(2,2));

    // add into panels
    leftPanel.add(label);
    leftPanel.add(love85);
    leftPanel.add(gold911);
    leftPanel.add(yes833);
    leftPanel.add(perfect995);
    rightPanel.add(love85Image);
    rightPanel.add(gold911Image);
    rightPanel.add(yes833Image);
    rightPanel.add(perfect995Image);

    // add panels to contentPane
    contentPane.add(leftPanel);
    contentPane.add(rightPanel);
  }

  /** Event Handler when checkbox is clicked */
  public void actionPerformed(ActionEvent e) {

    if ((e.getSource() == love85))
      System.out.println("Love 85 Clicked");
    else
      if ((e.getSource() == gold911))
        System.out.println("Gold 91.1 Clicked");
      else
        if ((e.getSource() == yes833))
          System.out.println("Yes 83.3 Clicked");
        else
          System.out.println("Perfect 99.5 Clicked");
  }

  /** Event Handler when checkbox is selected */
  public void itemStateChanged(ItemEvent e) {

    if (e.getSource() == love85)
      if (love85.isSelected()) {
        System.out.println("Love 85 Selected");
        love85Image.setVisible(true);
      }
      else {
        System.out.println("Love 85 deSelected");
        love85Image.setVisible(false);
      }
    else
      if (e.getSource() == gold911)
        if (gold911.isSelected()) {
          System.out.println("Gold 91.1 Selected");
          gold911Image.setVisible(true);
        }
        else {
          System.out.println("Gold 91.1 deSelected");
          gold911Image.setVisible(false);
        }
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
      else
        if (e.getSource() == yes833)
          if (yes833.isSelected()) {
            System.out.println("Yes 83.3 Selected");
            yes833Image.setVisible(true);
          }
          else {
            System.out.println("Yes 83.3 deSelected");
            yes833Image.setVisible(false);
          }
        else
          if (perfect995.isSelected()) {
            System.out.println("Perfect 99.5 Selected");
            perfect995Image.setVisible(true);
          }
          else {
            System.out.println("Perfect 99.5 deSelected");
            perfect995Image.setVisible(false);
          }
  }


  public static void main(String argv[]) {

    // Create frame
    CheckBoxes frame = new CheckBoxes();
    frame.setTitle("CheckBox Test");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setLocation(300, 100);

    //Display frame
    frame.pack();
    frame.setVisible(true);
  }
}
```

Code 4.3 implements Figure 4.5. Besides showing how Figure 4.5 can be displayed, we also demonstrate how to generate and handle events in checkboxes. Let us examine the code:

1. JCheckBoxes generate two types of events: Action Events and Item Events.
2. The CheckBoxes application implements ActionListener and ItemListener interface so that it can respond to the generated events.
3. Four checkboxes are created. All checkboxes are with initial text when they are constructed using the fifth constructor.
4. Checkbox gold911 is set as selected using the sixth constructor.
5. Four image-based JLabel objects are created.
6. A text-based JLabel object is created.

All components created are arranged using two panels – one on the left and the other on the right. Note the use of setBackground() method to set the right panel in white.

To add colors to the checkboxes, set font and background colors onto the labels and checkboxes. Need to ensure that only gold911 is visible since that is the only pre-selected radio station at initialization time.

The Checkboxes frame is added onto the checkboxes as their action listeners and item listeners.

Action events are handled via the actionPerformed() method and item events are handled via the itemStateChanged() method.

Observe how the events are captured and displayed in the command prompt when the application is executed. For example, the following output is produced in the command prompt when yes833 radio station is selected and deselected:

```
Yes 83.3 Selected
Yes 83.3 Clicked
Yes 83.3 deSelected
Yes 83.3 Clicked
```

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

"Yes 83.3 Selected" and "Yes 83.3 deSelected" are generated by the itemStateChanged() method while "Yes 83.3 Clicked" is generated by actionPerformed() method. Note that on the right side of the frame, the yes833 icon appears and then disappears. The appearance and disappearance of an icon is effected by the setVisible() method which is inherited from the Component class by all components.

4.2.3 JRadioButton

Radio buttons are similar to checkboxes in that they present choices from a list of items. However, unlike checkboxes, a user can only choose one item from a group of radio buttons. Radio buttons are also known as *option buttons*. Figure 4.7 shows an example of radio buttons in action.

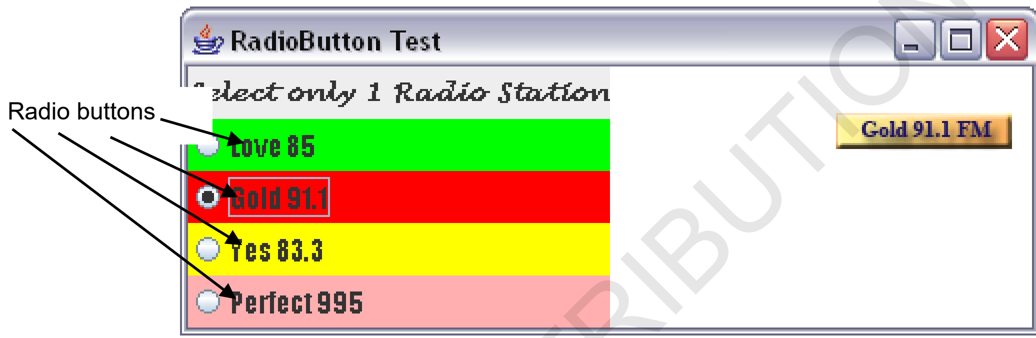


FIGURE 4.7: Selecting Only 1 Radio Stations using Radio Buttons

A radio button is manifested as a circle. The circle is filled when the radio button is selected and empty when it is not selected. Like a checkbox, the state of a radio button is visible to the user. The Java Swing version of a radio button is JRadioButton. The class hierarchy for JRadioButton is shown in Figure 4.8. The JRadioButton is a direct subclass of JToggleButton which is an implementation of a two-state button. A JRadioButton is therefore a two-state button.

```
java.lang.Object
└──java.awt.Component
    └──java.awt.Container
        └──javax.swing.JComponent
            └── javax.swing.AbstractButton
                └── javax.swing.JToggleButton
                    └── javax.swing.JRadioButton
```

FIGURE 4.8: Class Hierarchy of JRadioButton

4.2.3.1 What are the Constructors?

JRadioButton objects can be created using any of the eight constructors in Table 4.4.

TABLE 4.4: JRadioButton Constructors

No.	Constructor	Description
1	`JRadioButton()`	Creates a new, initially unselected JRadioButton object with no set text or icon. To set text, use setText("text") method inherited from AbstractButton class.
2	`JRadioButton(Action a)`	Creates a new JRadioButton object where properties are

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

		taken from the Action supplied.
3	`JRadioButton(Icon icon)`	Creates a new, initially unselected JRadioButton object with an icon but no text.
4	`JRadioButton(Icon icon, boolean selected)`	Creates a new JRadioButton object with the specified icon and specifies whether or not it is initially selected, but no text.
5	`JRadioButton(String text)`	Creates a new, initially unselected JRadioButton object with the specified text.
6	`JRadioButton(String text, boolean selected)`	Creates a new JRadioButton object with text and specifies whether or not it is initially selected.
7	`JRadioButton(String text, Icon icon)`	Creates a new, initially unselected JRadioButton object with specified text and icon.
8	`JRadioButton(String text, Icon icon, boolean selected)`	Creates a new JRadioButton object with text and icon, and specifies whether or not it is initially selected.

4.2.3.2 How are Events Handled?

A JRadioButton object generates two types of event: ActionEvent and ItemEvent. The actionPerformed() method of ActionListener interface and itemStateChanged() method of ItemListener interface must be implemented.

4.2.3.3 Show Me an Application of JRadioButton

Let us assume that only one radio station can be selected at any one time in the Radio Station selection application introduced earlier. In Figure 4.7, we have selected “Yes 83.3” radio station. To select another radio station will deselect “Yes 83.3”. For example, selecting “Love 85”, deselects “Yes 83.3”. No two or more stations can be selected using radio buttons at any one time.

Code 4.4: Radio Button Test

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

class RadioButtons extends JFrame implements ActionListener, ItemListener{

  // Create radio button objects
  JRadioButton love85    = new JRadioButton("Love 85");
  JRadioButton gold911   = new JRadioButton("Gold 91.1", true);
  JRadioButton yes833    = new JRadioButton("Yes 83.3");
  JRadioButton perfect995= new JRadioButton("Perfect 995");


  // create image labels
  JLabel love85Image     = new JLabel(new ImageIcon("../images/love85.gif"));
  JLabel gold911Image    = new JLabel(new ImageIcon("../images/gold911.gif"));
  JLabel yes833Image     = new JLabel(new ImageIcon("../images/yes833.gif"));
  JLabel perfect995Image = new JLabel(new ImageIcon("../images/perfect995.gif"));

  // Create label
  JLabel    label  = new JLabel("Select only 1 Radio Station");

  // Create a radio button group
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
ButtonGroup rbuttonGroup = new ButtonGroup();

RadioButtons() {

  // Group radio buttons
  rbuttonGroup.add(love85);
  rbuttonGroup.add(gold911);
  rbuttonGroup.add(yes833);
  rbuttonGroup.add(perfect995);

  // create 2 panels
  JPanel leftPanel  = new JPanel();
  JPanel rightPanel = new JPanel();
  rightPanel.setBackground(Color.white);

  // get content pane of frame
  Container contentPane = getContentPane();

  // set GridLayout
  GridLayout layout = new GridLayout(1,2);
  contentPane.setLayout(layout);

  // creates a label font
  Font labelFont = new Font("Lucida handwriting", Font.BOLD, 12);

  // creates a radio button font
  Font radioButtonFont = new Font("Impact", Font.PLAIN, 14);

  // set font and background colors
  label.setFont(labelFont);
  love85.setFont(radioButtonFont);
  gold911.setFont(radioButtonFont);
  yes833.setFont(radioButtonFont);
  perfect995.setFont(radioButtonFont);
  label.setBackground(Color.white);
  love85.setBackground(Color.green);
  gold911.setBackground(Color.red);
  yes833.setBackground(Color.yellow);
  perfect995.setBackground(Color.pink);

  // set visibility
  love85Image.setVisible(false);
  gold911Image.setVisible(true);
  yes833Image.setVisible(false);
  perfect995Image.setVisible(false);

  // add listener
  love85.addActionListener(this);
  love85.addItemListener(this);
  gold911.addActionListener(this);
  gold911.addItemListener(this);
  yes833.addActionListener(this);
  yes833.addItemListener(this);
  perfect995.addActionListener(this);
  perfect995.addItemListener(this);

  // set panel layout
  leftPanel.setLayout(new GridLayout(5,1));
  rightPanel.setLayout(new GridLayout(2,2));

  // add into panels
  leftPanel.add(label);
  leftPanel.add(love85);
  leftPanel.add(gold911);
  leftPanel.add(yes833);
  leftPanel.add(perfect995);
  rightPanel.add(love85Image);
  rightPanel.add(gold911Image);
  rightPanel.add(yes833Image);
  rightPanel.add(perfect995Image);

  // add panels to contentPane
```

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

```
    contentPane.add(leftPanel);
    contentPane.add(rightPanel);
  }

  /** Event Handler when radio button is clicked */
  public void actionPerformed(ActionEvent e) {

    if ((e.getSource() == love85))
      System.out.println("Love 97.2 Clicked");
    else
      if ((e.getSource() == gold911))
        System.out.println("Gold 90.5 Clicked");
      else
        if ((e.getSource() == yes833))
          System.out.println("Yes 93.3 Clicked");
        else
          System.out.println("Perfect 10 Clicked");
  }

  /** Event Handler when radio button is selected */
  public void itemStateChanged(ItemEvent e) {

    if (e.getSource() == love85)
      if (love85.isSelected()) {
        System.out.println("Love 97.2 Selected");
        love85Image.setVisible(true);
      }
      else {
        System.out.println("Love 97.2 deSelected");
        love85Image.setVisible(false);
      }
    else
      if (e.getSource() == gold911)
        if (gold911.isSelected()) {
          System.out.println("Gold 90.5 Selected");
          gold911Image.setVisible(true);
        }
        else {
          System.out.println("Gold 90.5 deSelected");
          gold911Image.setVisible(false);
        }
      else
        if (e.getSource() == yes833)
          if (yes833.isSelected()) {
            System.out.println("Yes 93.3 Selected");
            yes833Image.setVisible(true);
          }
          else {
            System.out.println("Yes 93.3 deSelected");
            yes833Image.setVisible(false);
          }
        else
          if (perfect995.isSelected()) {
            System.out.println("Perfect 10 Selected");
            perfect995Image.setVisible(true);
          }
          else {
            System.out.println("Yes 93.3 deSelected");
            perfect995Image.setVisible(false);
          }
  }


  public static void main(String argv[]) {

    // Create frame
    RadioButtons frame = new RadioButtons();
    frame.setTitle("RadioButton Test");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setLocation(300, 100);

    //Display frame
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
    frame.pack();
    frame.setVisible(true);
  }
}
```

Code 4.4 implements Figure 4.7 and is based on the CheckBoxes solution of Code 4.3. Some parts of the code have been masked for simplicity. Let us examine the code:

1. Four radio buttons are created. To achieve the functionality of the application, the four radio buttons are grouped within a ButtonGroup.
2. The ButtonGroup class is part of the javax.swing package. It is used to create a multiple-exclusion scope for a set of buttons. Creating a set of buttons with the same ButtonGroup object means that turning on one of these buttons turns off all other buttons in the group automatically.
3. The rest of the code is the same as Code 4.3.

Execute Code 4.4 and observe the behavior of the radio buttons. From the outputs in the command prompt, notice how the events generated by the buttons are handled.

4.2.4 Buttons Application

In this section, we will illustrate how JLabel, JCheckBox and JRadioButton can be used in an application. A display panel of a hi-fi system is shown in Figure 4.9.

On the east side of the frame is a set of radio buttons for the selection of the music setting: Disco, Hall, Stadium, Pop, Rock and Classic. We use radio buttons because only one music setting is allowed at any one time.

On the south side of the frame is a set of checkboxes for turning on/off available services:

1. Auto Off (if checked, to automatically turn off the hi-fi system after 1 hour),
2. Daily Timer (if checked, to automatically switch on the hi-fi system at a set time),
3. Record Timer (if checked, to automatically start the recording of the radio programme at a set time).

Users may select one or more of these services. The center portion of the frame shows the results of the settings selected by the user.

FIGURE 4.9: Buttons Application – HiFiSystem Display

Let us examine the code for this application (see Code 4.5):

1. Create the three types of components required for the application: JLabel, JCheckBox and JRadioButton.
2. Create a ButtonGroup object to associate the radio buttons into one group.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

3. Add the various music-setting radio buttons into the button group. This ensures a single music setting is selected at any one time.
4. Three panels are created; these panels are set in a BorderLayout on the contentPane.
5. The HiFiSystem frame is set as the listener for changes in the radio buttons and checkboxes. Note that only the ItemListener is implemented, even though radio buttons and checkboxes are able to respond to ActionListener too. Implementing item listening ensures all events are acted upon in one method: itemStateChanged().
6. The labels on the display are initialized via setAllMusicLabelsOff() in the itemStateChanged() method each time an event happens. The series of if statements in itemStateChanged() method sets the appropriate on/off message in the labels for the selected components.

Code 4.5: Buttons Application – HiFiSystem Display

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

class HiFiSystem extends JFrame implements ItemListener{

  // Create radio button objects
  JRadioButton disco   = new JRadioButton("Disco", true);
  JRadioButton hall    = new JRadioButton("Hall");
  JRadioButton stadium = new JRadioButton("Stadium");
  JRadioButton rock    = new JRadioButton("Rock");
  JRadioButton pop     = new JRadioButton("Pop");
  JRadioButton classic = new JRadioButton("Classic");

  // create checkboxes
  JCheckBox autoOff    = new JCheckBox("Auto Off");
  JCheckBox dailyTimer = new JCheckBox("Daily Timer");
  JCheckBox recTimer   = new JCheckBox("Record Timer");

  // Create label
  JLabel labelA   = new JLabel("DISCO Music");
  JLabel labelB   = new JLabel("Auto Power Off OFF");
  JLabel labelC   = new JLabel("Daily Timer OFF");
  JLabel labelD   = new JLabel("Record Timer OFF");

  // Create a radio button group
  ButtonGroup rbuttonGroup = new ButtonGroup();

  HiFiSystem() {

    // Group radio buttons
    rbuttonGroup.add(disco);
    rbuttonGroup.add(hall);
    rbuttonGroup.add(stadium);
    rbuttonGroup.add(rock);
    rbuttonGroup.add(pop);
    rbuttonGroup.add(classic);

    // create 3 panels
    JPanel panelA  = new JPanel();
    JPanel panelB  = new JPanel();
    JPanel panelC  = new JPanel();

    // get content pane of frame
    Container contentPane = getContentPane();

    // set BorderLayout
    BorderLayout layout = new BorderLayout();
    contentPane.setLayout(layout);

    // creates a label font
    Font labelFont = new Font("Arial", Font.BOLD, 16);

    // creates a radio button font
    Font radioButtonFont = new Font("Impact", Font.PLAIN, 14);

    // set font
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
    labelA.setFont(labelFont);
    labelB.setFont(labelFont);
    labelC.setFont(labelFont);
    labelD.setFont(labelFont);
    panelA.setBackground(Color.pink);

    // add listener
    disco.addItemListener(this);
    hall.addItemListener(this);
    stadium.addItemListener(this);
    rock.addItemListener(this);
    pop.addItemListener(this);
    classic.addItemListener(this);
    autoOff.addItemListener(this);
    dailyTimer.addItemListener(this);
    recTimer.addItemListener(this);

    // set panel layout
    panelA.setLayout(new GridLayout(4,1));
    panelB.setLayout(new FlowLayout());
    panelC.setLayout(new GridLayout(6,1));

    // add into panels
    panelA.add(labelA);
    panelA.add(labelB);
    panelA.add(labelC);
    panelA.add(labelD);
    panelB.add(autoOff);
    panelB.add(dailyTimer);
    panelB.add(recTimer);
    panelC.add(disco);
    panelC.add(hall);
    panelC.add(stadium);
    panelC.add(rock);
    panelC.add(pop);
    panelC.add(classic);

    // add panels to contentPane
    contentPane.add(panelA, BorderLayout.CENTER);
    contentPane.add(panelB, BorderLayout.SOUTH);
    contentPane.add(panelC, BorderLayout.EAST);
  }


  private void setAllMusicLabelsOff() {
    labelA.setText("DISCO Music OFF");
    labelA.setText("HALL Music OFF");
    labelA.setText("STADIUM Music OFF");
    labelA.setText("ROCK Music OFF");
    labelA.setText("POP Music OFF");
    labelA.setText("CLASSIC Music OFF");
  }


  /** Event Handler when radio button is selected */
  public void itemStateChanged(ItemEvent e) {

    setAllMusicLabelsOff();

    if (disco.isSelected()) labelA.setText("DISCO Music ON");
    if (hall.isSelected()) labelA.setText("HALL Music ON");
    if (stadium.isSelected()) labelA.setText("STADIUM Music ON");
    if (rock.isSelected()) labelA.setText("ROCK Music ON");
    if (pop.isSelected()) labelA.setText("POP Music ON");
    if (classic.isSelected()) labelA.setText("CLASSIC Music ON");

    if (autoOff.isSelected())
      labelB.setText("Auto Power Off ON");
    else labelB.setText("Auto Power Off OFF");

    if (dailyTimer.isSelected())
      labelC.setText("Daily Timer ON");
```

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

```
    else labelC.setText("Daily Timer OFF");

    if (recTimer.isSelected())
      labelD.setText("Record Timer ON");
    else labelD.setText("Record Timer OFF");
  }


  public static void main(String argv[]) {

    // Create frame
    HiFiSystem frame = new HiFiSystem();
    frame.setTitle("HiFiSystem Display");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setLocation(300, 100);

    //Display frame
    frame.pack();
    frame.setVisible(true);
  }
}
```

4.3 Combo Boxes

A *combo box* is a mix of a button (a clickable component) and a drop-down list. The latter is a list of items from which a user can select. The selected item becomes the displayed item of the drop-down list. Figure 4.10 shows an example of a combo box with its drop-down list is extended.

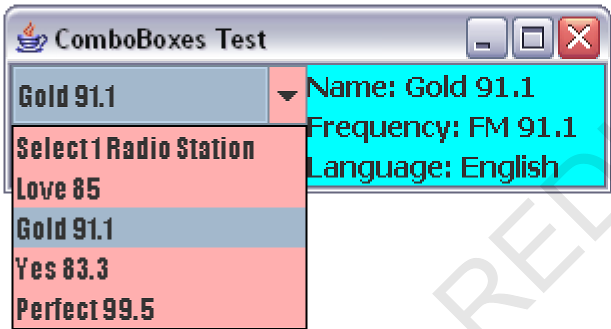


FIGURE 4.10: A Combo Box with its Drop-Down List Extended

If a combo box is defined to be editable, then the combo box is both an editable field and a drop-down list. A user may then change the value of the items in the drop-down list. Combo boxes are useful components for restricting the set of choices a user can choose thereby easing the input validation process.

The Java Swing version of a combo box is JComboBox. The JComboBox is a direct subclass of JComponent as shown in Figure 4.11, the class hierarchy for JComboBox.

```
java.lang.Object
└──java.awt.Component
    └──java.awt.Container
        └──javax.swing.JComponent
            └── javax.swing.JComboBox
```

FIGURE 4.11: Class Hierarchy of JComboBox

4.3.1 What are the Constructors?

JComboBox objects can be created using any of the four constructors in Table 4.5.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

TABLE 4.5: JComboBox Constructors

No.	Constructor	Description
1	`JComboBox()`	Creates a new JComboBox object with a default data model.
2	`JComboBox(ComboBoxModel aModel)`	Creates a new JComboBox object that takes its items from an existing ComboBoxModel.
3	`JComboBox(Object[] items)`	Creates a new JComboBox object that contains the elements in the specified array.
4	`JComboBox(Vector items)`	Creates a new JComboBox object that contains the elements in the specified vector.

4.3.2 How are Events Handled?

A JComboBox object generates two types of event: ActionEvent and ItemEvent. In order for a listener to respond to these events, the listener must implement actionPerformed() method of ActionListener interface and itemStateChanged() method of ItemListener interface.

Two ItemEvents are generated each time a drop-down list is scrolled up or down. One event is for deselecting a currently selected item and another event for selecting a new item. An ActionEvent is generated after an ItemEvent.

As you may have observed, ItemEvent is generated by a number of component types. To detect if a combo box is generating an ItemEvent, we use instanceof and getSource() method (of ItemEvent) to process the itemStateChanged() method as follows:

```
public void itemStateChanged(ItemEvent e) {
  if (e.getSource() instanceof JComboBox)
    ...
}
```

To know which item in a drop-down list has been selected, we use getItem() method to identify the string chosen:

```
public void itemStateChanged(ItemEvent e) {
  if (e.getSource() instanceof JComboBox)
    String s = (String) e.getItem();
}
```

There are other properties and methods that are often associated with event handling involving JComboBox (see Table 4.6).

TABLE 4.6: Useful JComboBox Methods

Method	Description
`int` **`getSelectedIndex`**`()`	Returns an int value indicating the index of the selected item in the combo box.
`Object` **`getSelectedItem`**`()`	Returns the selected item whose type is Object.
`addItem(Object item)`	Adds the specified item of any object type into the combo box.
`Object` **`getItemAt`**`(int index)`	Returns the item at the specified index in the combo box.
`void` **`removeItem`**`(Object item)`	Removes the specified item from the item list.
`void` **`removeAllItems`**`()`	Removes all the items from the item list.
`void` **`removeItemAt`**`(int index)`	Removes an item at the specified index from the combo box.

4.3.3 Show Me an Application of JComboBox

Let us extend our previous application on radio buttons in Section 4.2.3 and use a combo box to select the radio stations instead. Figure 4.12 shows the selection of a radio station (Perfect 99.5) using a combo box. A textual description of the selected station is given on the right-hand side of the frame. An image icon representing the station is displayed below the combo box.

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

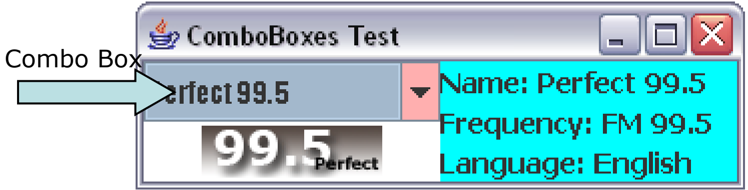


FIGURE 4.12: CombBoxes Test

Code 4.6 is an implementation of Figure 4.12. Let us examine the code:

1. A String array radioStations is defined to contain the names of the stations. This array is subsequently added into a combo box using the third JComboBox constructor (see Table 4.5).
2. Three other String arrays (names, frequencies, and languages) representing the descriptions of radio stations are created. The purpose of these arrays is for setting the labels of radio stations when they are selected.
3. The frame is set in the usual way using panels.
4. To respond to the item selected, the itemStateChanged() method of the ItemListener interface is implemented.
5. The ActionListener is not implemented as there is no requirement to do so in this application.
6. The getSelectedIndex() method is used to identify the index of the selected item. Using this index, the corresponding information of the selected radio station is assigned to the three labels in the right panel of the frame. Similarly, the corresponding image icon is displayed using imageLabel.

Code 4.6: ComboBoxes Test

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

class ComboBoxes extends JFrame implements ItemListener{

  // Create radio stations
  private String[] radioStations = {"Select 1 Radio Station",
                                   "Love 85",
                                   "Gold 91.1",
                                   "Yes 83.3",
                                   "Perfect 99.5"
                                  };

  // Create radio stations' descriptions
  private String[] names = {"Name: Love 85", "Name: Love 85",
                           "Name: Gold 91.1",
                           "Name: Yes 83.3", "Name: Perfect 99.5"
                          };
  private String[] frequencies = {"Frequency: FM 85",
                                 "Frequency: FM 85",
                                 "Frequency: FM 91.1",
                                 "Frequency: FM 83.3",
                                 "Frequency: FM 99.5"
                                };
  private String[] languages = {"Language: Mandarin",
                               "Language: Mandarin",
                               "Language: English",
                               "Language: Mandarin",
                               "Language: English"
                              };

  // Create combo box
  JComboBox comboBox = new JComboBox(radioStations);

  // create images
  ImageIcon love85Image   = new ImageIcon("../images/love85.gif");
  ImageIcon gold911Image   = new ImageIcon("../images/gold911.gif");
  ImageIcon yes833Image    = new ImageIcon("../images/yes833.gif");
  ImageIcon perfect995Image = new ImageIcon("../images/perfect995.gif");
```

Copyright © 2020, Danny Poo.

```
// Create image array
private ImageIcon[] icons = {love85Image, love85Image, gold911Image,
                             yes833Image, perfect995Image};

// Create label
JLabel imageLabel = new JLabel(love85Image);
JLabel label1     = new JLabel(names[0]);
JLabel label2     = new JLabel(frequencies[0]);
JLabel label3     = new JLabel(languages[0]);


ComboBoxes() {

  // create 2 panels
  JPanel leftPanel  = new JPanel();
  JPanel rightPanel = new JPanel();
  rightPanel.setBackground(Color.white);

  // get content pane of frame
  Container contentPane = getContentPane();

  // set GridLayout
  GridLayout layout = new GridLayout(1,2);
  contentPane.setLayout(layout);

  // creates a label font
  Font labelFont = new Font("Tahoma", Font.BOLD, 14);

  // creates a comboBox font
  Font comboBoxFont = new Font("Impact", Font.PLAIN, 14);

  // set font and background colors
  label1.setFont(labelFont);
  label2.setFont(labelFont);
  label3.setFont(labelFont);
  comboBox.setFont(comboBoxFont);
  comboBox.setBackground(Color.pink);
  leftPanel.setBackground(Color.white);
  rightPanel.setBackground(Color.cyan);

  // set visibility
  imageLabel.setVisible(true);

  // add listener
  comboBox.addItemListener(this);

  // set panel layout
  leftPanel.setLayout(new GridLayout(2,1));
  rightPanel.setLayout(new GridLayout(3,1));

  // add into panels
  leftPanel.add(comboBox);
  leftPanel.add(imageLabel);
  rightPanel.add(label1);
  rightPanel.add(label2);
  rightPanel.add(label3);

  // add panels to contentPane
  contentPane.add(leftPanel);
  contentPane.add(rightPanel);
}

/** Event Handler when comboBox is selected */
public void itemStateChanged(ItemEvent e) {

  int index = comboBox.getSelectedIndex();
  label1.setText(names[index]);
  label2.setText(frequencies[index]);
  label3.setText(languages[index]);
  imageLabel.setIcon(icons[index]);
}
```

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

```
  public static void main(String argv[]) {

    // Create frame
    ComboBoxes frame = new ComboBoxes();
    frame.setTitle("ComboBoxes Test");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setLocation(300, 100);

    //Display frame
    frame.pack();
    frame.setVisible(true);
  }
}
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

CHAPTER 5: TEXT COMPONENTS

Any graphical user interface involves the use of texts. In Java user interface design, *text fields* and *text areas* are two text components that are commonly used to input and display texts. In this chapter, we will discuss the various types of text components supported in the Java Swing; in particular,

1. Text fields,
2. Text areas,
3. Formatted text fields,
4. Password fields,
5. Editor pane, and
6. Text pane.

5.1 Text Fields

A *text field* is a text component for entering or displaying a line of text. It is commonly used for information gathering e.g. in the filling in of a form. Figure 5.1 shows three labels, three text fields and a button in a frame.

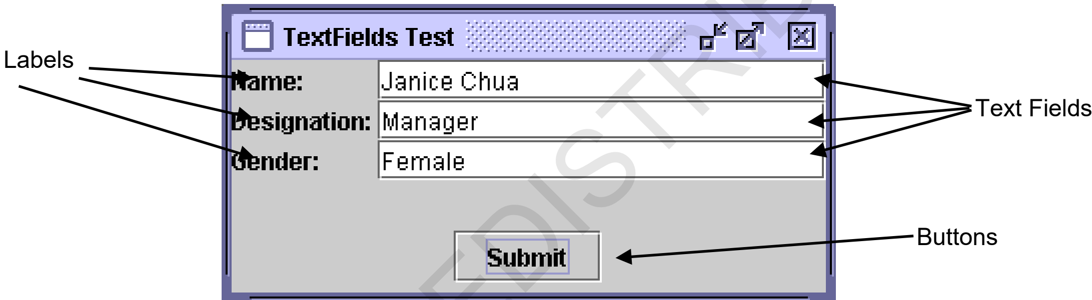


FIGURE 5.1: TextFields Test

5.1.1 JTextField – For Entering Texts

The Java Swing version of a text field is JTextField. Figure 5.2 is the class hierarchy for JTextField, a direct subclass of JTextComponent. The latter is the base class for Java Swing text components.

```
java.lang.Object
└── java.awt.Component
    └── java.awt.Container
        └── javax.swing.JComponent
            └── javax.swing.text.JTextComponent
                └── javax.swing.JTextField
```

FIGURE 5.2: Class Hierarchy of JTextField

5.1.1.1 What are the Constructors?

JTextField has five constructor methods (see Table 5.1).

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

TABLE 5.1: JTextField Constructors

No.	Constructor	Description
1	`JTextField()`	Creates a new JTextField object with no set text.
2	`JTextField(Document doc, String text, int columns)`	Creates a new JTextField object that uses the specified text storage model and the given number of columns.
3	`JTextField(int columns)`	Creates a new JTextField object with the specified number of columns.
4	`JTextField(String text)`	Creates a new JTextField object initialized with the specified text.
5	`JTextField(String text, int columns)`	Creates a new JTextField object initialized with the specified text and number of columns.

NOT FOR REDISTRIBUTION

5.1.1.2 How are Events Handled?

A JTextField object generates ActionEvent objects. In order for the application to respond to ActionEvent, the actionPerformed() method of ActionListener interface must be implemented.

5.1.1.3 What are the Useful Methods?

Table 5.2 highlights some useful methods of JTextField.

TABLE 5.2: Useful Methods

Method	Use
`void` **`setText`**`(String s)` *`inherited from JTextComponent`*	Set text field with the specified string.
`String` **`getText`**`()`	Get text from JTextField object.
`void` **`setEditable`**`(boolean b)` *`inherited from JTextComponent`*	Set the specified boolean to indicate whether or not this JTextField is editable.
`boolean` **`isEditable`**`()` *`inherited from JTextComponent`*	Returns the boolean indicating whether this JTextField is editable or not.
`void` **`setColumns`**`(int i)`	Set the number of columns displayed by this JTextField.
`int` **`getColumns`**`()`	Get the number of columns in this JTextField.
`void` **`setHorizontalAlignment`**`(int i)`	Set the horizontal alignment of this JTextField. Valid int value include: JTextField.LEFT, JTextField.CENTER, JTextField.RIGHT, JTextField.LEADING, JTextField.TRAILING
`int` **`getHorizontalAlignment`**`()`	Get the horizontal alignment of this JTextField.
`void` **`addActionListener`** `(ActionListener al)`	Add the specified action listener to receive action events from this JTextField.
`void` **`removeActionListener`** `(ActionListener al)`	Remove the specified action listener so that the container does not receive action events from this JTextField.
`void` **`selectAll`**`()` *`inherited from JTextComponent`*	Select all characters in this JTextField.

5.1.1.4 Show Me an Application of JTextField

We will implement the frame shown in Figure 5.1 in this section. In this test application, users are required to enter their name, designation and gender in three text fields. When the "Submit" button is clicked, the application will echo the entries in the command prompt.

Copyright © 2020, Danny Poo.

Code 5.1: TextFields Test

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

public class TextFields extends JFrame implements ActionListener {
  private JTextField name   = new JTextField(20);
  private JTextField desgn  = new JTextField(10);
  private JTextField gender = new JTextField(6);
  private JLabel label1 = new JLabel("Name: ");
  private JLabel label2 = new JLabel("Designation: ");
  private JLabel label3 = new JLabel("Gender: ");
  private JButton submitButton = new JButton("Submit");
  private JPanel panel1 = new JPanel();
  private JPanel panel2 = new JPanel();
  private JPanel panel3 = new JPanel();
  private JPanel panel  = new JPanel();

  public TextFields() {
    panel1.setLayout(new GridLayout(4,1));
    panel2.setLayout(new GridLayout(4,1));
    panel3.setLayout(new FlowLayout());
    panel.setLayout(new BorderLayout());

    //Add Components to this panel.
    panel1.add(label1);
    panel1.add(label2);
    panel1.add(label3);
    panel2.add(name);
    panel2.add(desgn);
    panel2.add(gender);
    panel3.add(submitButton);
    panel.add(panel1, BorderLayout.CENTER);
    panel.add(panel2, BorderLayout.EAST);
    panel.add(panel3, BorderLayout.SOUTH);

    // add listeners
    submitButton.addActionListener(this);
  }

  public void actionPerformed(ActionEvent e) {
    System.out.println("name        = " + name.getText());
    System.out.println("designation = " + desgn.getText());
    System.out.println("gender      = " + gender.getText());
  }

  private static void createFrameAndShow() {
    //Make sure we have nice window decorations.
    JFrame.setDefaultLookAndFeelDecorated(true);

    //Create and set up the window.
    TextFields frame = new TextFields();
    frame.setTitle("TextFields Test");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    //Create and set up the content pane.
    JComponent contentPane = (JComponent)frame.getContentPane();
    contentPane.setOpaque(true); //content panes must be opaque
    frame.setContentPane(contentPane);
    contentPane.add(frame.panel);

    //Display the window.
    frame.pack();
    frame.setVisible(true);
  }

  public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        createFrameAndShow();
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
        }
      });
    }
}
```

Code 5.1 implements the above requirement. Let us examine the code:

1. Declare and create three JTextField objects, three JLabel objects, one JButton object, and four JPanel objects. The width of all text fields in the display (see Figure 5.1) is similar and it follows the width of the JTextField object, name. The specified number of columns for each text field, as defined, merely dictates the size of the text field for display purpose but they do not restrict the number of characters the text field can be assigned with.
2. The various components are first added into their respective panels before all the panels (panel1, panel2, and panel3) are added into a holding panel (panel).
3. The frame listens to only one button (the submitButton) for events. This explains why nothing happens when texts are entered into the text fields. When the "Submit" button is clicked, the actionPerformed() event handler is triggered, following the event listening model as explained in Chapter 3. The output in the command prompt is produced by this method.
4. You may have observed that the look-and-feel of the GUI display is different from the examples in the previous chapters. A new method createFrameAndShow() defines the statements for creating the frame. Sets the default look-and-feel to ensure that a nice window decoration is produced.
5. Another difference is in main(): The javax.swing.SwingUtilities class contains a collection of utility methods for Swing. The invokeLater() method of SwingUtilities causes doRun.run() to be executed asynchronously on the AWT event dispatching thread. This causes createFrameAndShow() to be executed.

5.1.2 JPasswordField – For Entering Passwords

There are times when we want to enter some texts but we do not want the texts to be echoed as they are. This situation happens when a password or PIN (Personal Identification Number) is entered.

Java provides a specialized text field that echoes a preset character in place of the texts entered. Such text fields are known as *password fields*. The Java Swing version of a password field is JPasswordField. Figure 5.3 is the class hierarchy for JPasswordField.

```
java.lang.Object
└──java.awt.Component
    └──java.awt.Container
        └──javax.swing.JComponent
            └── javax.swing.text.JTextComponent
                └── javax.swing.JTextField
                    └── javax.swing.JPasswordField
```

FIGURE 5.3: Class Hierarchy of JPasswordField

5.1.2.1 What are the Constructors?

JPasswordField has five constructor methods (see Table 5.3).

TABLE 5.3: JPasswordField Constructors

No.	Constructor	Description
1	`JPasswordField()`	Creates a new JPasswordField object with a default document, null starting text string, and 0 column width.
2	`JPasswordField(Document doc, String text, int columns)`	Creates a new JPasswordField object that uses the specified text storage model and the given number of columns.
3	`JPasswordField(int columns)`	Creates a new, empty JPasswordField object with the specified number of columns.

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

4	`JPasswordField(String text)`	Creates a new JPasswordField object initialized with the specified text.
5	`JPasswordField(String text, int columns)`	Creates a new JPasswordField object initialized with the specified text and number of columns.

5.1.2.2 How are Events Handled?

A JPasswordField object generates ActionEvent objects. In order for the application to respond to ActionEvent, the actionPerformed() method of ActionListener interface must be implemented.

5.1.2.3 What are the Useful Methods?

Table 5.4 highlights some useful methods of JPasswordField.

TABLE 5.4: Useful Methods

Method	Use
`char` **`getEchoChar`**`()`	Get the character used for echoing
`char[]` **`getPassword`**`()`	Get the text contained in this JPasswordField object.
`protected String` **`paramString`**`()`	Returns a string representation of this JPasswordField object.
`void` **`setEchoChar`**`(char c)`	Set the echo character for this JPasswordField object.

5.1.2.4 Show Me an Application of JPasswordField

We will extend our previous "TextFields Test" application to include a password field. Figure 5.4 includes an additional field called "Password".

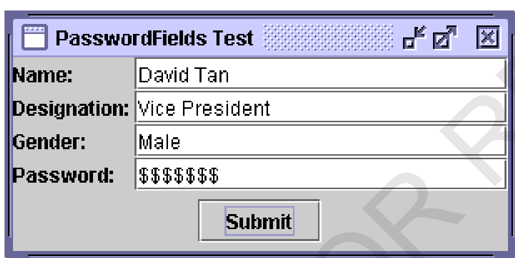


FIGURE 5.4: PasswordFields Test

As you enter the password, the field echoes the "$" character instead of the entered character. As for the display in the command prompt, the password has been encrypted when "secrets" is entered as the password:

```
name        = David Tan
designation = Vice President
gender      = Male
password    = [C@7109c4
```

Code 5.2 produces Figure 5.4. Let us examine the code:

1. We create a password field using the third constructor method of Table 5.3.
2. A JLabel object to display the "Password:" label is created.
3. The "$" character is set as the echo character using the setEchoChar() method.
4. Finally, the encrypted value of the password is achieved using the getPassword() method.

Code 5.2: PasswordFields Test

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

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
public class PasswordFields extends JFrame implements ActionListener {
  private JTextField name   = new JTextField(20);
  private JTextField desgn  = new JTextField(10);
  private JTextField gender = new JTextField(6);
  private JPasswordField pw = new JPasswordField(8);
  private JLabel label1 = new JLabel("Name: ");
  private JLabel label2 = new JLabel("Designation: ");
  private JLabel label3 = new JLabel("Gender: ");
  private JLabel label4 = new JLabel("Password: ");
  private JButton submitButton = new JButton("Submit");
  private JPanel panel1 = new JPanel();
  private JPanel panel2 = new JPanel();
  private JPanel panel3 = new JPanel();
  private JPanel panel  = new JPanel();

  public PasswordFields() {

    // set layout
    panel1.setLayout(new GridLayout(4,1));
    panel2.setLayout(new GridLayout(4,1));
    panel3.setLayout(new FlowLayout());
    panel.setLayout(new BorderLayout());

    // set password echo char
    pw.setEchoChar('$');

    //Add Components to panel
    panel1.add(label1);
    panel1.add(label2);
    panel1.add(label3);
    panel1.add(label4);
    panel2.add(name);
    panel2.add(desgn);
    panel2.add(gender);
    panel2.add(pw);
    panel3.add(submitButton);
    panel.add(panel1, BorderLayout.CENTER);
    panel.add(panel2, BorderLayout.EAST);
    panel.add(panel3, BorderLayout.SOUTH);

    // add listeners
    submitButton.addActionListener(this);
  }

  public void actionPerformed(ActionEvent e) {
    System.out.println("name        = " + name.getText());
    System.out.println("designation = " + desgn.getText());
    System.out.println("gender      = " + gender.getText());
    System.out.println("password    = " + pw.getPassword());
  }

  private static void createFrameAndShow() {
    //Make sure we have nice window decorations.
    JFrame.setDefaultLookAndFeelDecorated(true);

    //Create and set up the window.
    PasswordFields frame = new PasswordFields();
    frame.setTitle("PasswordFields Test");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    //Create and set up the content pane.
    JComponent contentPane = (JComponent)frame.getContentPane();
    contentPane.setOpaque(true); //content panes must be opaque
    frame.setContentPane(contentPane);
    contentPane.add(frame.panel);

    //Display the window.
    frame.pack();
    frame.setLocation(300, 100);
    frame.setVisible(true);
  }
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
  public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        createFrameAndShow();
      }
    });
  }
}
```

5.1.3 JFormattedTextField – For Entering Texts with Validation

In our previous examples, the validity of the texts entered was not considered. A user may enter "David %$#@$#" as his name and the application will accept it as it is, without any verification. To ensure correct values are entered, all texts entered by users must be validated.

Although it is possible for developers to write their own input validation code, Java Swing, from J2SE(Java 2 Standard Edition) Release 1.4 onward, provides a *formatted text field component* for input validation.

The Java Swing version of a formatted text field component is JFormattedTextField. Figure 5.5 is the class hierarchy for JFormattedTextField.

```
java.lang.Object
└──java.awt.Component
    └──java.awt.Container
        └──javax.swing.JComponent
            └── javax.swing.text.JTextComponent
                └── javax.swing.JTextField
                    └── javax.swing.JFormattedTextField
```

FIGURE 5.5: Class Hierarchy of JFormattedTextField

JFormattedTextField is an extension of JTextField. It adds support for formatting values. Basically, JFormattedTextField adds a *formatter* and a *value* object to the features inherited from JTextField. The value object represents the text field that is being edited and the formatter object is the intermediary between the value and the string that is displayed in the GUI.

value is of type Object and it varies depending on what is being edited. Standard formatters that Java Swing provides in JFormattedTextField include editing support for:

- Numbers: The value object could be an instance of Integer or Double class.
- Dates: The value object could be an instance of Date class.
- String objects via a simple mask.

To display on a graphical user interface, value must be converted into a String and JFormattedTextField does it by using the valueToString() method. A similar stringToValue() method is used to convert the String representation into a value object. valueToString() and stringToValue() are methods of javax.swing.JFormattedTextField.AbstractFormatter, a nested class of JFormattedTextField.

NOT FOR REDISTRIBUTION

5.1.3.1 What are the Constructors?

JFormattedTextField has six constructor methods (see Table 5.5).

TABLE 5.5: JFormattedTextField Constructors

No.	Constructor	Description
1	`JFormattedTextField()`	Creates a new JFormattedTextField object with no AbstractFormatterFactory.
2	`JFormattedTextField(Format format)`	Creates a new JFormattedTextField object with a specified format.

Copyright © 2020, Danny Poo.

3	JFormattedTextField(JFormattedTextField.AbstractFormatter formatter)	Creates a new, empty JFormattedTextField with the specified AbstractFormatter.
4	JFormattedTextField(JFormattedTextField.AbstractFormatterFactory factory)	Creates a new JFormattedTextField object with the specified AbstractFormatterFactory.
5	JFormattedTextField(JFormattedTextField.AbstractFormatterFactory factory, Object value)	Creates a new JFormattedTextField object with the specified AbstractFormatterFactory and initial value.
6	JFormattedTextField(Object value)	Creates a new JFormattedTextField object with the specified value.

5.1.3.2 How are Events Handled?

A JFormattedTextField object generates ActionEvent and PropertyChangeEvent objects. In order for the application to respond to ActionEvent and PropertyChangeEvent, the actionPerformed() method of ActionListener interface and propertyChange() method of PropertyChangeListener interface must be implemented.

5.1.3.3 What are the Useful Methods?

Table 5.6 highlights some useful methods of JFormattedTextField.

TABLE 5.6: Useful Methods

Method	Use
void **setValue**(Object value)	Set the value that will be formatted by an AbstractFormatter obtained from the current AbstractFormatterFactory.
Object **getValue**()	Get the last valid value of this JFormattedTextField object.
abstract Object **stringToValue**(String text) of JFormattedTextField.AbstractFormatter	Parses text returning an arbitrary Object.
abstract String **valueToString**(Object value) of JFormattedTextField.AbstractFormatter	Returns the string value to display for value.

5.1.3.4 Show Me an Application of JFormattedTextField

The example that we will use to illustrate JFormattedTextField is similar to our previous "TextFields Test" application. The user is required to enter six fields: name, address1, address2, postal code, date of birth, and contribution (a currency amount). Figure 5.6 shows the output of this application.

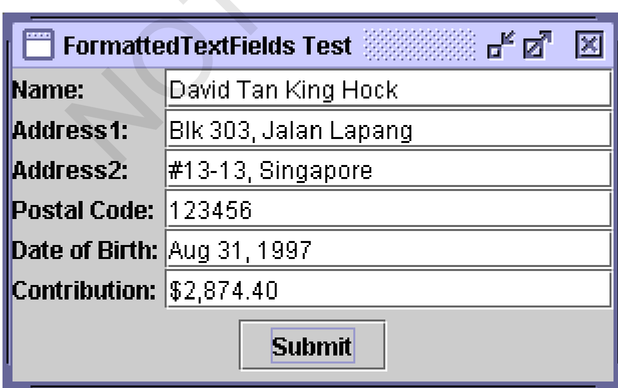


FIGURE 5.6: FormattedTextFields Test

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

Test this application by executing FormattedTextFields.class. Note that you can only enter up to 25 characters for name, address1, and address2. This is the constraint that has been included as part of the formatting policy. As the formatter that we use is fairly limited, it is still possible for the users to enter names such as "David %$#@$#". We will discuss how to resolve this problem a little while later.

Postal code has been limited to only six digits and you will not be able to enter anything beyond that. Date of birth has been restricted to be month, day, year format. Finally, contribution follows a currency format with values up to two decimal places.

To demonstrate PropertyChangeEvent, the following lines are printed when the propertyChange() event handler is invoked (i.e. when any of the text fields is changed).

```
IN PROPERTYCHANGE
name         = David Tan King Hock

IN PROPERTYCHANGE
address1     = Blk 303, Jalan Lapang

IN PROPERTYCHANGE
address2     = #13-13, Singapore

IN PROPERTYCHANGE
postalCode   = 123456

IN PROPERTYCHANGE
dateOfBirth  = Sun Aug 31 00:00:00 SGT 1997

IN PROPERTYCHANGE
contribution = 2874.4
```

ActionEvents are triggered by the "SubmitButton". The actionPerformed() method is invoked when this event is generated:

```
IN ACTIONPERFORMED
name         = David Tan King Hock
address1     = Blk 303, Jalan Lapang
address2     = #13-13, Singapore
postalCode   = 123456
dateOfBirth  = Sun Aug 31 00:00:00 SGT 1997
contribution = 2874.4
```

Code 5.3: FormattedTextFields Test

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.text.*;
import java.util.*;
import java.text.*;
import java.beans.*;

public class FormattedTextFields
  extends JFrame
  implements ActionListener, PropertyChangeListener {

  // define fields
  private String name         = new String("");
  private String address1     = new String("");
  private String address2     = new String("");
  private String postalCode   = new String("");
  private Date dateOfBirth    = new Date();
  private double contribution;

  // define formatted text fields
  private JFormattedTextField nameFTfield;
  private JFormattedTextField address1FTfield;
  private JFormattedTextField address2FTfield;
  private JFormattedTextField postalCodeFTfield;
  private JFormattedTextField dateOfBirthFTfield;
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
  private JFormattedTextField contributionFTfield;

  // define formats for formatting and parsing numbers
  private MaskFormatter nameFormat;
  private MaskFormatter address1Format;
  private MaskFormatter address2Format;
  private MaskFormatter postalCodeFormat;
  private DateFormat    dateOfBirthFormat;
  private NumberFormat  contributionFormat;

  // create labels to identify fields
  private JLabel nameLabel         = new JLabel("Name: ");
  private JLabel address1Label     = new JLabel("Address1: ");
  private JLabel address2Label     = new JLabel("Address2: ");
  private JLabel postalCodeLabel   = new JLabel("Postal Code: ");
  private JLabel dateOfBirthLabel  = new JLabel("Date of Birth: ");
  private JLabel contributionLabel = new JLabel("Contribution: ");

  // create submit button
  private JButton submitButton = new JButton("Submit");

  // create panels
  private JPanel panel1 = new JPanel();
  private JPanel panel2 = new JPanel();
  private JPanel panel3 = new JPanel();
  private JPanel panel  = new JPanel();


  public FormattedTextFields() {

    // setup formats for formattedTextFields
    setUpFormats();

    // create formatted text fields and set them up
    createAndFormatFields();

    // set panels
    setPanels();

    // set listeners
    setListeners();
  }

  private void setPanels() {
    // set layout
    panel1.setLayout(new GridLayout(6,1));
    panel2.setLayout(new GridLayout(6,1));
    panel3.setLayout(new FlowLayout());
    panel.setLayout(new BorderLayout());

    // add components
    panel1.add(nameLabel);
    panel1.add(address1Label);
    panel1.add(address2Label);
    panel1.add(postalCodeLabel);
    panel1.add(dateOfBirthLabel);
    panel1.add(contributionLabel);
    panel2.add(nameFTfield);
    panel2.add(address1FTfield);
    panel2.add(address2FTfield);
    panel2.add(postalCodeFTfield);
    panel2.add(dateOfBirthFTfield);
    panel2.add(contributionFTfield);
    panel3.add(submitButton);
    panel.add(panel1, BorderLayout.CENTER);
    panel.add(panel2, BorderLayout.EAST);
    panel.add(panel3, BorderLayout.SOUTH);
  }

  private void setUpFormats() {
    nameFormat = createFormatter("*************************");
    address1Format = createFormatter("*************************");
```

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

```
    address2Format = createFormatter("*************************");
    postalCodeFormat = createFormatter("######");
    dateOfBirthFormat = DateFormat.getDateInstance();
    contributionFormat = NumberFormat.getCurrencyInstance();
    contributionFormat.setMinimumFractionDigits(2);
  }

  public void createAndFormatFields() {
    //Create the text fields and set them up.
    nameFTfield = new JFormattedTextField(nameFormat);
    nameFTfield.setValue(name);
    nameFTfield.setColumns(20);

    address1FTfield = new JFormattedTextField(address1Format);
    address1FTfield.setValue(address1);
    address1FTfield.setColumns(20);

    address2FTfield = new JFormattedTextField(address2Format);
    address2FTfield.setValue(address2);
    address2FTfield.setColumns(20);

    postalCodeFTfield = new JFormattedTextField(postalCodeFormat);
    postalCodeFTfield.setValue(postalCode);
    postalCodeFTfield.setColumns(10);

    dateOfBirthFTfield = new JFormattedTextField(dateOfBirthFormat);
    dateOfBirthFTfield.setValue(dateOfBirth);
    dateOfBirthFTfield.setColumns(10);

    contributionFTfield = new JFormattedTextField(contributionFormat);
    contributionFTfield.setValue(new Double(contribution));
    contributionFTfield.setColumns(10);
  }

  private void setListeners() {
    nameFTfield.addPropertyChangeListener("value", this);
    address1FTfield.addPropertyChangeListener("value", this);
    address2FTfield.addPropertyChangeListener("value", this);
    postalCodeFTfield.addPropertyChangeListener("value", this);
    dateOfBirthFTfield.addPropertyChangeListener("value", this);
    contributionFTfield.addPropertyChangeListener("value", this);
    submitButton.addActionListener(this);
  }

  protected MaskFormatter createFormatter(String s) {
    MaskFormatter formatter = null;
    try {
        formatter = new MaskFormatter(s);
    } catch (java.text.ParseException exc) {
        System.err.println("formatter is bad: " + exc.getMessage());
        System.exit(-1);
    }
    return formatter;
  }

  // event handlers
  public void actionPerformed(ActionEvent e) {
    System.out.println("\nIN ACTIONPERFORMED");

    System.out.println("name         = " + nameFTfield.getValue());
    System.out.println("address1     = " + address1FTfield.getValue());
    System.out.println("address2     = " + address2FTfield.getValue());
    System.out.println("postalCode   = " + postalCodeFTfield.getValue());
    System.out.println("dateOfBirth  = " + dateOfBirthFTfield.getValue());
    System.out.println("contribution = " + contributionFTfield.getValue());
  }

  public void propertyChange(PropertyChangeEvent e) {
    System.out.println("\nIN PROPERTYCHANGE");

    Object source = e.getSource();
    if (source == nameFTfield) {
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
      System.out.println("name         = " + nameFTfield.getValue());
    }
    else
      if (source == address1FTfield) {
        System.out.println("address1     = " + address1FTfield.getValue());
      }
      else
        if (source == address2FTfield) {
          System.out.println("address2     = " + address2FTfield.getValue());
        }
        else
          if (source == postalCodeFTfield) {
            System.out.println("postalCode   = " + postalCodeFTfield.getValue());
          }
          else
            if (source == dateOfBirthFTfield) {
              System.out.println("dateOfBirth  = " + dateOfBirthFTfield.getValue());
            }
            else
              if (source == contributionFTfield) {
                System.out.println("contribution = " + contributionFTfield.getValue());
              }
  }

  private static void createFrameAndShow() {
    //Make sure we have nice window decorations.
    JFrame.setDefaultLookAndFeelDecorated(true);

    //Create and set up the window.
    FormattedTextFields frame = new FormattedTextFields();
    frame.setTitle("FormattedTextFields Test");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    //Create and set up the content pane.
    JComponent contentPane = (JComponent)frame.getContentPane();
    contentPane.setOpaque(true); //content panes must be opaque
    frame.setContentPane(contentPane);
    contentPane.add(frame.panel);

    //Display the window.
    frame.pack();
    frame.setLocation(300, 100);
    frame.setVisible(true);
  }

  public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        createFrameAndShow();
      }
    });
  }
}
```

Code 5.3 produces Figure 5.6. Let us examine the code:

1. Additional import statements are needed:

import java.text.*	for DateFormat and NumberFormat
import javax.swing.text.*	for MaskFormatter
import java.util.*	for Date
import java.beans.*	for PropertyChangeEvent and PropertyChangeListener

2. Define the values required.
3. Define the formatted text fields for the application. Three types of format are used:
   MaskFormatter for name, address1, address2, and postalCode;
   DateFormat for dateOfBirth; and
   NumberFormat for contribution.

A MaskFormatter implements a formatter that specifies exactly which characters are legal in each position of the formatted text field. Table 5.7 shows the characters that can be used as the formatting mask.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

TABLE 5.7: Masking Characters

Formatting Mask Character	Use
`#`	Any valid number (Character.isDigit)
`' (A single quote)`	Escape character, used to escape any of the special formatting characters.
`U`	Any character (Character.isLetter). All lowercase letters are mapped to uppercase.
`L`	Any character (Character.isLetter). All uppercase letters are mapped to lowercase.
`A`	Any character or number (Character.isLetter or Character.isDigit).
`?`	Any character (Character.isLetter).
`*`	Anything.
`H`	Any hexadecimal character (0-9, a-f or A-F).

The formatting mask provided here may not necessarily be sufficient e.g. "David %$#@$#" is still possible for the name text field. There are two approaches to resolve this problem:

- Break name into its components (first name, middle name and family name).
- Use AbstractFormatter (a nested class of JFormattedTextField) and define a regular expression for editing the text field. We will not discuss this approach as the use of *regular expression* is beyond the scope of this book.

A DateFormat class provides date formatting facilities while a NumberFormat class provides the interface for formatting and parsing numbers. The various formatters for the six text fields are defined. The rest of the declaration is obvious by now.

The first method called is setUpFormats(). This method creates a format for each of the formatted text field. We have used MaskFormatter for name, address1, address2 and postalCode. Note the use of "*" and "#" mask character. You may have noticed how we have used "######" to restrict the size of the postal code.

Use getDateInstance() method of DateFormat class to get a date formatter and use getCurrencyInstance() of NumberFormat class to get a number formatter for currency values. You may use getIntegerInstance(), getNumberInstance(), or getPercentInstance() method to get integer number format, general number format or percentage format respectively.

We use setMinimumFractionDigits() to set the number of decimal places for contribution. There are other similar methods like: setMaximumFractionDigits(), setMinimumIntegerDigits(), and setMaximumIntegerDigits() in NumberFormat.

createAndFormatFields() is called to create the six text fields with their respective formatter obtained from setUpFormats().

setPanels() method in positions the various components to achieve the look-and-feel of Figure 5.6.

Finally, we set the frame as the listener for PropertyChangeEvent and ActionEvent.

5.2 Text Areas

Some of the outputs we have produced from our examples have been printed in the command prompt. Suppose we want to print the same information using a graphical user interface, how do we do it?

An approach is to use text fields but we would need a number of them since text fields can only display one line at a time. A better approach would be to use a *text area* where multiple lines of texts can be displayed. Figure 5.7 shows an example of the use of text area to display lines of texts.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

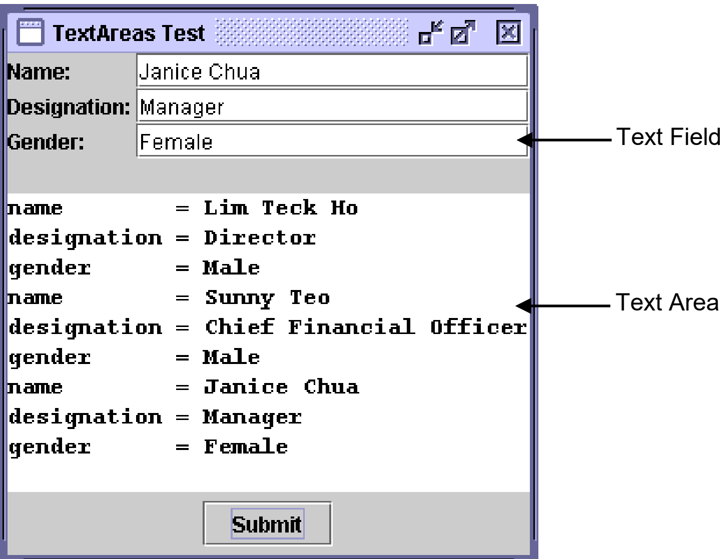


FIGURE 5.7: TextAreas Test

5.2.1 JTextArea – For Entering and Displaying Lines of Texts

A JTextArea is a text component for entering and displaying multi-lines of *plain text* in a single area. Users may enter unformatted text of any length or display unformatted help information in text areas.

The Java Swing version of a text area is JTextArea. Figure 5.8 is the class hierarchy for JTextArea.

```
java.lang.Object
└── java.awt.Component
     └── java.awt.Container
          └── javax.swing.JComponent
               └── javax.swing.text.JTextComponent
                    └── javax.swing.JTextArea
```

FIGURE 5.8: Class Hierarchy of JTextArea

5.2.1.1 What are the Constructors?

JTextArea has six constructor methods (see Table 5.8).

TABLE 5.8: JTextArea Constructors

No.	Constructor	Description
1	`JTextArea()`	Creates a new JTextArea object with no set text.
2	`JTextArea(Document doc)`	Creates a new JTextArea object with the given document model, and defaults for all of the other arguments.
3	`JTextArea(Document doc, String text, int rows, int columns)`	Creates a new JTextArea object with the specified number of rows and columns, and the given model.
4	`JTextArea(int rows, int columns)`	Creates a new JTextArea object with the specified number of rows and columns, but with no set text.
5	`JTextArea(String text)`	Creates a new JTextArea object initialized with the specified text.

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

6	`JTextArea(String text, int rows, int columns)`	Creates a new JTextArea object initialized with the specified text and number of rows and columns.

5.2.1.2 How are Events Handled?

Although java.awt.TextArea could be monitored for TextEvents by adding a TextListener, the same does not apply to javax.swing.JTextArea. Since JTextArea is a JTextComponent component, changes to JTextArea can be broadcasted from the model via a DocumentEvent to DocumentListeners.

DocumentEvent and DocumentListeners are modeled based on the Model-View-Controller concept. Basically, the concept requires the separation of data (known as the *model*) from their *view*. The objective of the Model-View-Controller concept is to reduce the maintenance effort associated with change since views may change while the data remain unchanged.

5.2.1.3 What are the Useful Methods?

Table 5.9 highlights some useful methods of JTextArea.

TABLE 5.9: Useful Methods

Method	Use
`void` **`append`**`(String s)`	Append the given string to the end of the text area.
`int` **`getColumns`**`()`	Returns the number of columns in the JTextArea object.
`int` **`getLineCount`**`()`	Determines the number of lines contained in the JTextArea object.
`boolean` **`getLineWrap`**`()`	Gets the line-wrapping policy. If true, the line in the JTextArea object is automatically wrapped; if false, it is not automatically wrapped.
`int` **`getRows`**`()`	Returns the number of rows in the JTextArea object.
`int` **`getTabSize`**`()`	Get the number of characters used to expand tabs.
`boolean` **`getWrapStyleWord`**`()`	Get the style of wrapping used if the JTextArea object is wrapping lines.
`void` **`insert`**`(String s, int position)`	Insert the specified text at the specified position.
`void` **`replaceRange`**`(String s, int start, int end)`	Replace text from the specified start to end position with the new text specified.
`void` **`setColumns`**`(int columns)`	Set the number of columns for this JTextArea object.
`void` **`setFont`**`(font f)`	Set the current font for this JTextArea object.
`void` **`setLineWrap`**`()`	Set the line-wrapping policy for this JTextArea object.
`void` **`setRows`**`(int rows)`	Set the number of rows for this JTextArea object.
`void` **`setTabSize`**`(int size)`	Set the number of characters to expand tabs to.
`void` **`setWrapStyleWord`**`(boolean word)`	Set the style of wrapping used if this JTextArea object is wrapping lines.

5.2.1.4 Show Me an Application of JTextArea

Figure 5.7 is an extension of our previous example on text fields (see Section 5.1). The main difference is the addition of a text area to display the texts entered via the text fields.

Code 5.4: TextAreas Test
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class TextAreas extends JFrame implements ActionListener {
  private JTextField name   = new JTextField(20);
  private JTextField desgn  = new JTextField(10);
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
private JTextField gender = new JTextField(6);
private JLabel label1 = new JLabel("Name: ");
private JLabel label2 = new JLabel("Designation: ");
private JLabel label3 = new JLabel("Gender: ");
private JButton submitButton = new JButton("Submit");
private JPanel panel1 = new JPanel();
private JPanel panel2 = new JPanel();
private JPanel panel3 = new JPanel();
private JPanel panel4 = new JPanel();
private JPanel panel  = new JPanel();
private JTextArea ta  = new JTextArea(10,20);

public TextAreas() {

  // set layout
  panel1.setLayout(new GridLayout(4,1));
  panel2.setLayout(new GridLayout(4,1));
  panel3.setLayout(new BorderLayout());
  panel.setLayout(new BorderLayout());

  // settings for textarea
  Font font = new Font("Courier New", Font.BOLD, 14);
  ta.setFont(font);

  //Add Components to this panel.
  panel1.add(label1);
  panel1.add(label2);
  panel1.add(label3);
  panel2.add(name);
  panel2.add(desgn);
  panel2.add(gender);
  panel3.add(ta, BorderLayout.NORTH);
  panel4.add(submitButton);
  panel3.add(panel4, BorderLayout.SOUTH);
  panel.add(panel1, BorderLayout.CENTER);
  panel.add(panel2, BorderLayout.EAST);
  panel.add(panel3, BorderLayout.SOUTH);

  // add listeners
  submitButton.addActionListener(this);
}

public void actionPerformed(ActionEvent e) {
  ta.append("name        = " + name.getText() + "\n");
  ta.append("designation = " + desgn.getText() + "\n");
  ta.append("gender      = " + gender.getText() + "\n");
}

private static void createFrameAndShow() {
  //Make sure we have nice window decorations.
  JFrame.setDefaultLookAndFeelDecorated(true);

  //Create and set up the window.
  TextAreas frame = new TextAreas();
  frame.setTitle("TextAreas Test");
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

  //Create and set up the content pane.
  JComponent contentPane = (JComponent)frame.getContentPane();
  contentPane.setOpaque(true); //content panes must be opaque
  frame.setContentPane(contentPane);
  contentPane.add(frame.panel);

  //Display the window.
  frame.pack();
  frame.setLocation(300, 100);
  frame.setVisible(true);
}

public static void main(String[] args) {
  javax.swing.SwingUtilities.invokeLater(new Runnable() {
    public void run() {
```

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

```
            createFrameAndShow();
        }
    });
  }
}
```

Code 5.4 implements Figure 5.7. Let us examine Code 5.4:

1. A new panel, panel4, is created to help organize the text area and button.
2. A JTextArea object, ta, is created. It is set with 10 rows and 20 columns of text space.
3. Sets the font for the text area. All of the text in the text area shares the same font.
4. The submitButton is added into panel4 before the entire panel4, together with the text area, is added into panel3.
5. A BorderLayout is used for panel3.
6. All the three panels are finally added into a single panel.
7. The entered texts in the text fields are appended into the text area when the submitButton is clicked.
8. The frame only listens to ActionEvent generated by the submitButton and responds to it via the event handler method, actionPerformed().

You may have observed in testing Code 5.4 that the output is not very satisfactory, particularly when you attempt to add beyond three sets of information into the text area. Figure 5.9 is what you are likely to observe.

We get this rather ugly output because the text area we have defined does not handle *scrolling*. To rectify this problem, we have to place the JTextArea object inside a JScrollPane object so that scrolling can be facilitated. We will discuss JScrollPane in detail in Chapter 6. For now, we show how to achieve a *scrollable text area*.

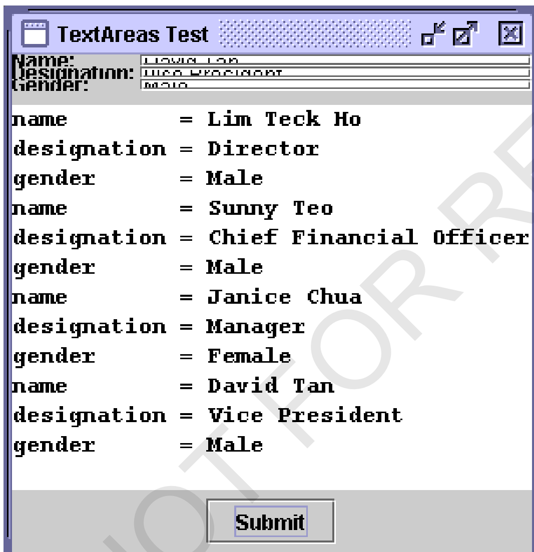


FIGURE 5.9: TextAreas without Scrolling Capability

5.2.2 Scrollable Text Areas

Figure 5.10 illustrates a scrollable text area produced from Code 5.5. Note that Code 5.5 is very similar to Code 5.4.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

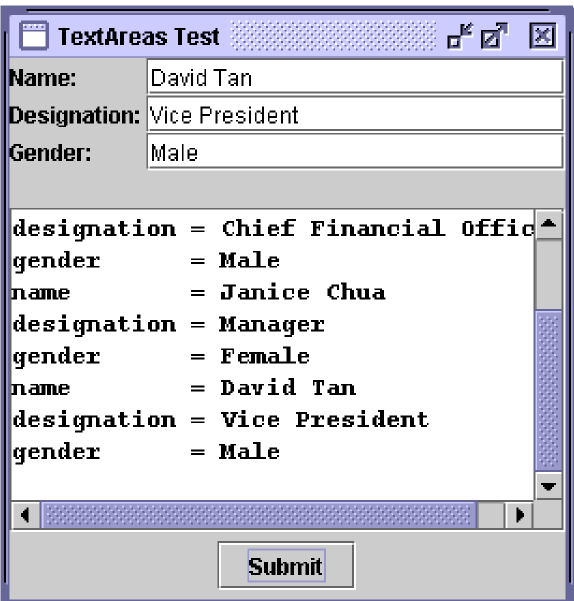


FIGURE 5.10: TextAreas with Scrolling Capability

A JScrollPane object, with the JTextArea object ta placed inside it, is created. Instead of adding ta into panel3, we add the scroll pane, sp into panel3. The rest of the code remains the same.

Code 5.5: TextAreas with Scrolling Capability

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

public class TextAreas1 extends JFrame implements ActionListener {
  private JTextField name   = new JTextField(20);
  private JTextField desgn  = new JTextField(10);
  private JTextField gender = new JTextField(6);
  private JLabel label1 = new JLabel("Name: ");
  private JLabel label2 = new JLabel("Designation: ");
  private JLabel label3 = new JLabel("Gender: ");
  private JButton submitButton = new JButton("Submit");
  private JPanel panel1 = new JPanel();
  private JPanel panel2 = new JPanel();
  private JPanel panel3 = new JPanel();
  private JPanel panel4 = new JPanel();
  private JPanel panel  = new JPanel();
  private JTextArea ta  = new JTextArea(10,20);

  public TextAreas1() {

    // set layout
    panel1.setLayout(new GridLayout(4,1));
    panel2.setLayout(new GridLayout(4,1));
    panel3.setLayout(new BorderLayout());
    panel.setLayout(new BorderLayout());

    // settings for textarea
    Font font = new Font("Courier New", Font.BOLD, 14);
    ta.setFont(font);

    // create a scroll pane to hold textarea
    JScrollPane sp = new JScrollPane(ta);

    //Add Components to this panel.
    panel1.add(label1);
    panel1.add(label2);
    panel1.add(label3);
    panel2.add(name);
```

NOT FOR DISTRIBUTION

Copyright © 2020, Danny Poo.

```
    panel2.add(desgn);
    panel2.add(gender);
    //panel3.add(ta, BorderLayout.NORTH);
    panel3.add(sp, BorderLayout.NORTH);
    panel4.add(submitButton);
    panel3.add(panel4, BorderLayout.SOUTH);
    panel.add(panel1, BorderLayout.CENTER);
    panel.add(panel2, BorderLayout.EAST);
    panel.add(panel3, BorderLayout.SOUTH);

    // add listeners
    submitButton.addActionListener(this);
  }

  public void actionPerformed(ActionEvent e) {
    ta.append("name        = " + name.getText() + "\n");
    ta.append("designation = " + desgn.getText() + "\n");
    ta.append("gender      = " + gender.getText() + "\n");
  }

  private static void createFrameAndShow() {
    //Make sure we have nice window decorations.
    JFrame.setDefaultLookAndFeelDecorated(true);

    //Create and set up the window.
    TextAreas1 frame = new TextAreas1();
    frame.setTitle("TextAreas Test");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    //Create and set up the content pane.
    JComponent contentPane = (JComponent)frame.getContentPane();
    contentPane.setOpaque(true); //content panes must be opaque
    frame.setContentPane(contentPane);
    contentPane.add(frame.panel);

    //Display the window.
    frame.pack();
    frame.setLocation(300, 100);
    frame.setVisible(true);
  }

  public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        createFrameAndShow();
      }
    });
  }
}
```

NOT FOR REDISTRIBUTION

5.3 Styled Text Areas

So far, our examples have been based on plain texts. JTextField, JFormattedTextField, JPasswordField, and JTextArea allow users to enter and control texts entered but they do not provide users with any control over the *style* of the texts. Neither can users undo or redo what they have done. In Java Swing, control in style is provided for in styled text components that include JEditorPane and JTextPane.

5.3.1 JTextComponent – Superclass of All Text Components

Figure 5.11 shows the relationships among JTextComponent, JTextField, JFormattedTextField, JPasswordField, JTextArea, JEditorPane and JTextPane.

Copyright © 2020, Danny Poo.

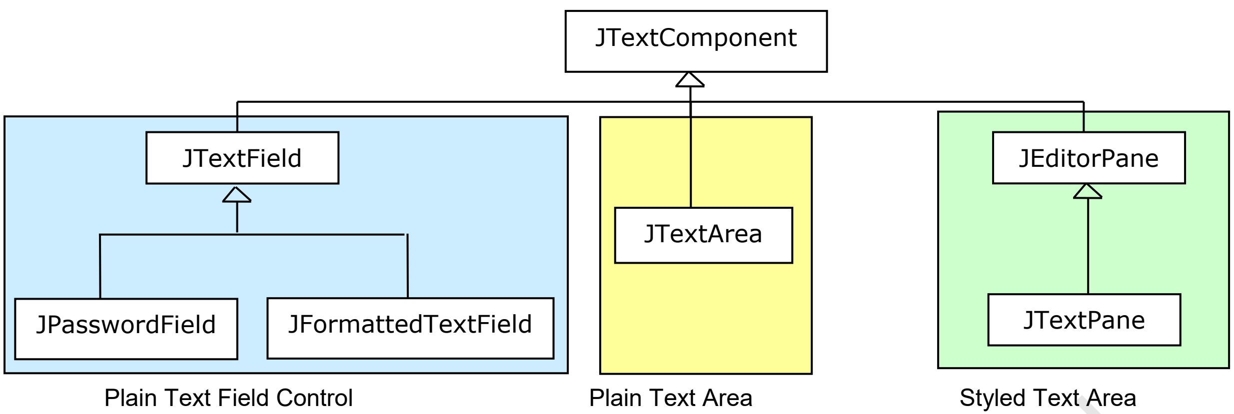


FIGURE 5.11: JTextComponent and its Subclasses

Table 5.10 puts in perspective the characteristics of these components.

TABLE 5.10: Characteristics of Text Components

Component	Characteristics
`JTextField, JFormattedTextField, and JPasswordField`	• Cater to plain text field control. • These components can display and edit one line of text. • Action events are generated by these components. • Use them when a limited amount of information is required from the user.
`JTextArea`	• Cater to plain text area control. • Can display and edit multiple lines of text. • Text font in a JTextArea can be set but once set all texts in the text area share the same font. • Use JTextArea when multiple lines of unformatted text needs to be display e.g. when displaying help information.
`JEditorPane and JTextPane`	• Cater to styled text area control. • A styled text component can display and edit text using more than one font. • Embedded image and embedded components are possible. • Styled text components are more powerful than plain text components and offer more facilities for customization to satisfy any high-end needs.
`JTextComponent`	• Superclass of all text components. • Provides customizable features for all its subclasses, including: ◊ A *model*, known as a *document*, to manage the component's content. ◊ A *view* which is in charge of displaying the component on screen. ◊ A *controller*, known as an *editor kit* that can read and write text. The controller implements editing capabilities with actions. • Provides supports for infinite undo and redo. • Provides pluggable caret and support for caret change listeners and navigation filters.

We will return to discussing JEditorPane and JTextPane in Chapter 6 when we have fully understood the concept of **Panes**.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

CHAPTER 6: VIEWING CONTENTS THROUGH PANES

The concept of pane was first introduced in Chapter 2 when we mentioned Content Pane. As you may recall, a content pane serves as a container of displayable components on a frame. Only components placed in the content pane are visible on the frame. Fundamentally, a pane provides a *view* to displayable contents.

In this chapter, we will discuss the following types of pane supported in the Java Swing:

1. *Scroll Pane* – enables scrolling for viewing other parts of contents
2. *Split Pane* – enables the splitting of a pane into multiple sections
3. *Tabbed Pane* – a pane for containing a group of components
4. *Internal Frame* – a frame within a frame
5. *Desktop Pane* – a container for creating virtual desktop
6. *Layered Pane* – a container for positioning components with a depth dimension

There are two other types of pane supported in the Java Swing: *Editor Panes* and *Option Panes*. Editor Panes are used for editing styled texts and will be covered in Chapter 7. Option Panes are used in standard dialog boxes[8]. They will be covered in Chapter 8.

6.1 Scroll Pane

Figure 6.1 is an image of the cover design for a book entitled “Learn to Program Enterprise JavaBeans”.

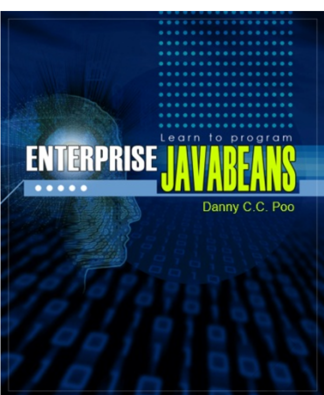


FIGURE 6.1: “Learn To Program Enterprise JavaBeans” Book Cover Design

Suppose we want to display this image on a content pane whose size is smaller than the image. It would not be possible to display the entire image on the content pane. To view the image, we would require a scrolling facility.

Figure 6.2 shows a scrollable content pane. The image can be scrolled up and down or left and right. A scrollable pane is known as a *scroll pane* in Java.

[8] A dialog box is used to inform users of some information or to prompt users to enter some inputs.

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

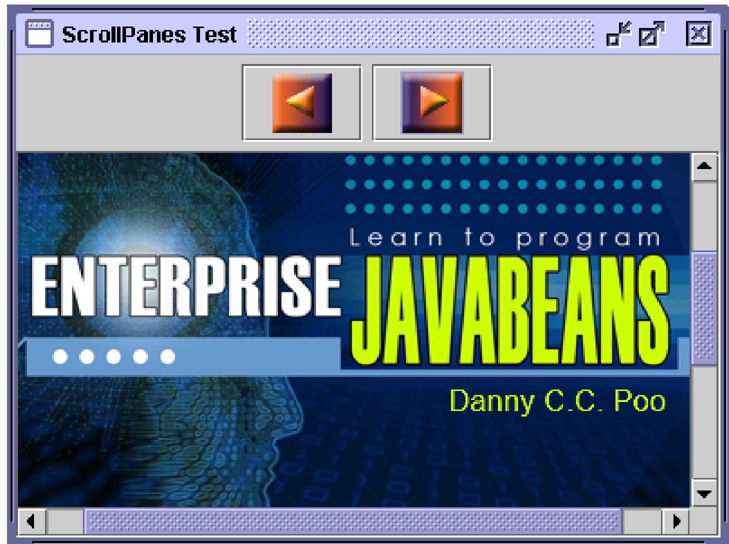


FIGURE 6.2: ScrollPanes Test

6.1.1 JScrollPane – A Scrolling Pane

The Java Swing version of a scrolling pane is JScrollPane. Its class hierarchy is given in Figure 6.3. JScrollPane is a direct subclass of JComponent.

```
java.lang.Object
└── java.awt.Component
    └── java.awt.Container
        └── javax.swing.JComponent
            └── javax.swing.JScrollPane
```

FIGURE 6.3: Class Hierarchy of JScrollPane

A JScrollPane is made up of the following components:

1. A viewport
2. A vertical scrollbar
3. A horizontal scrollbar
4. A row header
5. A column header and
6. Up to 4 corner components.

These components are shown in Figure 6.4.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

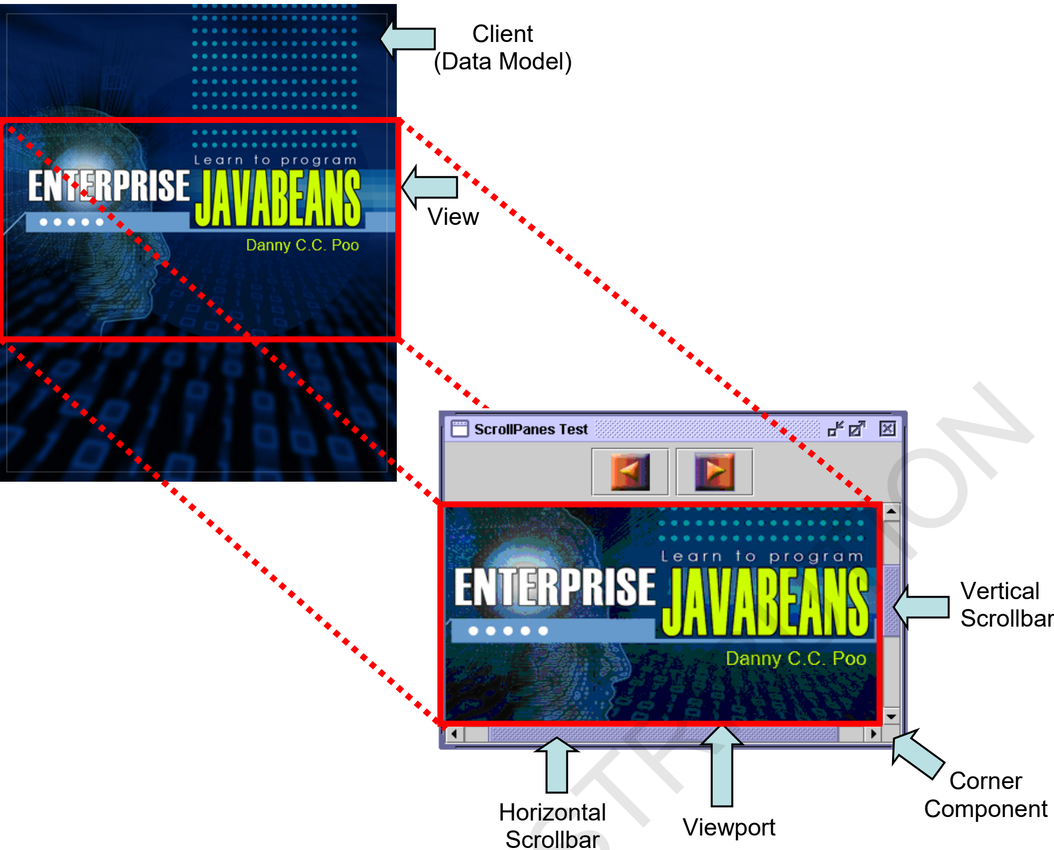


FIGURE 6.4: Components of JScrollPane

The data model (e.g. the book cover design image, in this case) with which a scroll pane is associated with is known as the *client*. The *vertical* and *horizontal scrollbars* appear *only if* the client is larger than the scroll pane. The *viewport* is the window by which a user views the client. The section of the client displayed on a viewport is known as the *view*. There can be up to 4 corner components in a JScrollPane – one on each corner of the viewport. By default, the corner component is empty. A scroll pane may also have a row and column header. Components can be added as part of the row or column headers.

6.1.1.1 What are the Constructors?

JScrollPane has four constructor methods (see Table 6.1).

TABLE 6.1: JScrollPane Constructors

No.	Constructor	Description
1	`JScrollPane()`	Creates a new and empty JScrollPane object, horizontal and vertical scrollbars will appear if needed.
2	`JScrollPane(Component view)`	Creates a new JScrollPane object that displays the contents of the specified component, where both horizontal and vertical scrollbars appear whenever the component's contents are larger than the view.
3	`JScrollPane(Component view, int vsbPolicy, int hsbPolicy)`	Creates a new JScrollPane object that displays the the specified component in a viewport whose view position can be controlled with a pair of scrollbars.
4	`JScrollPane(int vsbPolicy, int hsbPolicy)`	Creates an empty (no view) JScrollPane object

Copyright © 2020, Danny Poo.

		with the specified scrollbar policies.

6.1.1.2 How are Events Handled?

A JScrollPane object does not generate any specific event object of its own. However, as a subclass of java.awt.Component, java.awt.Container, and javax.swing.JComponent, we may add listeners to listen to events that are typical of these superclasses on JScrollPane.

6.1.1.3 What are the Useful Methods?

Table 6.2 highlights some useful methods of JScrollPane.

TABLE 6.2: Useful Methods

Method	Use
Component **getCorner**(String key)	Returns the component at the specified corner.
JViewport **getColumnHeader**()	Returns the column header.
JViewport **getRowHeader**()	Returns the row header.
JViewport **getViewport**()	Returns the current JViewport.
void **setColumnHeader**(JViewport columnHeader)	Removes the old columnHeader, if it exists.
void **setRowHeader**(JViewport rowHeader)	Removes the old rowHeader, if it exists.
void **setLayout**(LayoutManager layout)	Sets the layout manager for this JScrollPane.

NOT FOR REDISTRIBUTION

6.1.1.4 Show Me an Application of JScrollPane

The application we will discuss here is shown in Figure 6.2. This application allows you to view five images using two buttons "previous" and "next". An image is first displayed in a scrolling pane. You can view the other images by clicking on the "previous" or "next" button on the top of the pane. Code 6.1 implements this application and produces Figure 6.2 as its output.

Code 6.1: ScrollPanes Test

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

public class ScrollPanes extends JFrame implements ActionListener {

  // labels
  private JLabel imageLabel;

  // paths
  private String previousPath = "./images/previous.gif";
  private String nextPath     = "./images/next.gif";
  private String imagePath    = "./images/";

  // buttons
  private JButton previousBtn = new JButton(getImageIcon(previousPath));
  private JButton nextBtn     = new JButton(getImageIcon(nextPath));

  // panels
  private JPanel panel        = new JPanel();
  private JPanel panel1       = new JPanel();

  // images
  ImageIcon image;

  // array to hold images
  ImageIcon[] imageArray      = new ImageIcon[4];
  int         index           = 0;
```

Copyright © 2020, Danny Poo.

```
public ScrollPanes() {

  // initialize array
  initImageArray();

  // set layout
  panel1.setLayout(new FlowLayout());
  panel.setLayout(new BorderLayout());

  // create a scroll pane to display images
  image          = imageArray[index];
  imageLabel     = new JLabel(image);
  JScrollPane sp = new JScrollPane(imageLabel);

  // add Components to panel
  panel1.add(previousBtn);
  panel1.add(nextBtn);
  panel.add(panel1, BorderLayout.NORTH);
  panel.add(sp, BorderLayout.CENTER);

  // add listeners
  previousBtn.addActionListener(this);
  nextBtn.addActionListener(this);
}

public void actionPerformed(ActionEvent e) {

  if ((e.getSource() == previousBtn))
    setPrevious();
  else
    setNext();
  imageLabel.setIcon(imageArray[index]);
}

private void initImageArray() {
  String path = new String(imagePath + "pic");
  for (int i=0; i<imageArray.length; i++) {
    imageArray[i] = new ImageIcon(path + i + ".jpg");
  }
}

private void setPrevious() {
  if (index == 0) {
    index = imageArray.length - 1;
  }
  else
    index--;
}

private void setNext() {
  if (index == (imageArray.length-1)) {
    index = 0;
  }
  else
    index++;
}

protected static ImageIcon getImageIcon(String path) {
  // create an ImageIcon object if path is valid
  // else returns null
  java.net.URL imageUrl = ScrollPanes.class.getResource(path);
  if (imageUrl != null) {
    return new ImageIcon(imageUrl);
  }
  else {
   System.err.println("No image found at: " + path);
   return null;
  }
}

private static void createFrameAndShow() {
  //Make sure we have nice window decorations.
```

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

```
    JFrame.setDefaultLookAndFeelDecorated(true);

    //Create and set up the window.
    ScrollPanes frame = new ScrollPanes();
    frame.setTitle("ScrollPanes Test");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    //Create and set up the content pane.
    JComponent contentPane = (JComponent)frame.getContentPane();
    contentPane.setOpaque(true); //content panes must be opaque
    frame.setContentPane(contentPane);
    contentPane.add(frame.panel);

    //Display the window.
    frame.pack();
    frame.setLocation(300, 100);
    frame.setSize(400, 300);
    frame.setVisible(true);
  }

  public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        createFrameAndShow();
      }
    });
  }
}
```

Let us examine Code 6.1:

1. Two buttons are created using the image icons defined in the given paths. The two buttons are added into panel1 before the latter is included into an enclosing panel, panel.
2. There are altogether five images and they are organized within an array, imageArray. The array is initialized with the images.
3. A JLabel object, imageLabel, holds the current image to be displayed on the frame and is initialized as a client of a JScrollPane.
4. Add the components into their respective panels before listeners are added to the buttons. Note that sp, a JScrollPane, is added directly into panel. The solution will not work if sp had been added into a JPanel object before it is added into panel.
5. Any click on the buttons triggers the execution of the actionPerformed() method. If the current image is the last image in the array (index = 4), a click on the next button displays the image indexed 0. Similarly, if the current image is the first image in the array, clicking the previous button displays the last image in the array (index = 4).

6.1.2 Adding Headers and Corners

Column and row headers can be added into a JScrollPane. They provide a means for labeling the sides of a scroll pane.

There are altogether four corners in a scroll pane. A corner component is visible only if there is an intersection of two sides and if the corner contains visible components. Figure 6.5 shows a column header and two corner components in the “ScrollPanes Test” frame.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

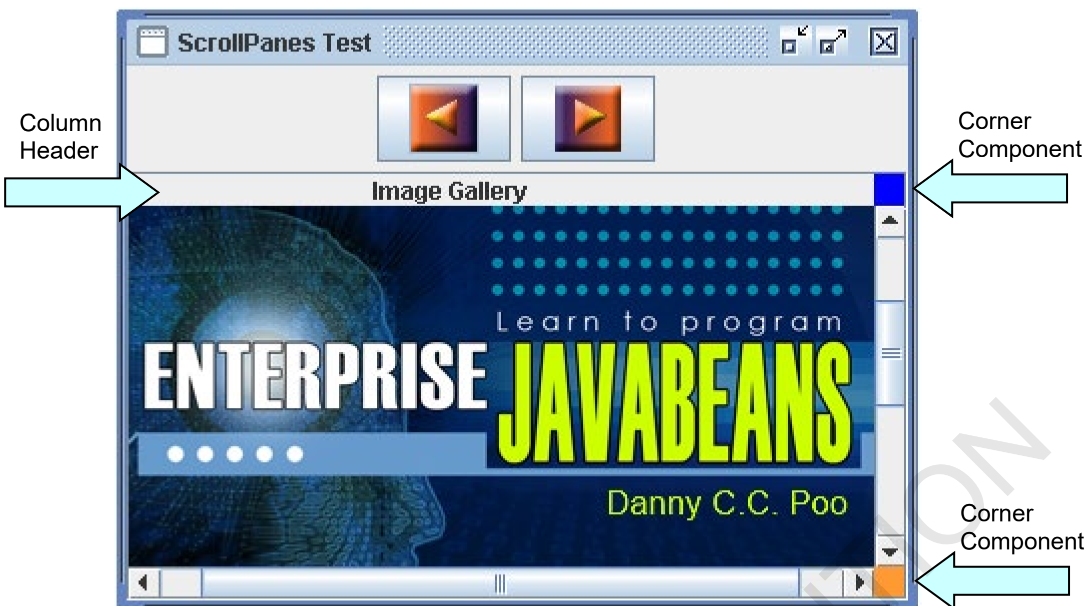


FIGURE 6.5: Components of a JScrollPane

NOT FOR REDISTRIBUTION

Code 6.2: Adding Column Header and Corner Components

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

public class ScrollPanes1 extends JFrame implements ActionListener {

  // labels
  private JLabel imageLabel;
  private JLabel columnHeaderLabel = new JLabel("Image Gallery");
  private JLabel upperLeftLabel;
  private JLabel upperRightLabel;
  private JLabel lowerLeftLabel;
  private JLabel lowerRightLabel;

  // paths
  private String previousPath   = "./images/previous.gif";
  private String nextPath       = "./images/next.gif";
  private String imagePath      = "./images/";
  private String upperLeftPath  = "./images/corner0.gif";
  private String upperRightPath = "./images/corner1.gif";
  private String lowerLeftPath  = "./images/corner2.gif";
  private String lowerRightPath = "./images/corner3.gif";

  // buttons
  private JButton previousBtn = new JButton(getImageIcon(previousPath));
  private JButton nextBtn     = new JButton(getImageIcon(nextPath));

  // panels
  private JPanel panel        = new JPanel();
  private JPanel panel1       = new JPanel();

  // images
  ImageIcon image;
  ImageIcon upperLeft         = getImageIcon(upperLeftPath);
  ImageIcon upperRight        = getImageIcon(upperRightPath);
  ImageIcon lowerLeft         = getImageIcon(lowerLeftPath);
  ImageIcon lowerRight        = getImageIcon(lowerRightPath);

  // array to hold images
  ImageIcon[] imageArray      = new ImageIcon[4];
  int         index           = 0;
```

Copyright © 2020, Danny Poo.

```
public ScrollPanes1() {

  // initialize array
  initImageArray();

  // set layout
  panel1.setLayout(new FlowLayout());
  panel.setLayout(new BorderLayout());

  // create a scroll pane to display images
  image          = imageArray[index];
  imageLabel     = new JLabel(image);
  JScrollPane sp = new JScrollPane(imageLabel);

  // set corners of scroll pane
  upperLeftLabel  = new JLabel(upperLeft);
  upperRightLabel = new JLabel(upperRight);
  lowerLeftLabel  = new JLabel(lowerLeft);
  lowerRightLabel = new JLabel(lowerRight);
  sp.setCorner(JScrollPane.UPPER_LEFT_CORNER, upperLeftLabel);
  sp.setCorner(JScrollPane.UPPER_RIGHT_CORNER, upperRightLabel);
  sp.setCorner(JScrollPane.LOWER_LEFT_CORNER, lowerLeftLabel);
  sp.setCorner(JScrollPane.LOWER_RIGHT_CORNER, lowerRightLabel);

  // set column header view
  columnHeaderLabel.setHorizontalAlignment(SwingConstants.CENTER);
  sp.setColumnHeaderView(columnHeaderLabel);

  // add Components to panel
  panel1.add(previousBtn);
  panel1.add(nextBtn);
  panel.add(panel1, BorderLayout.NORTH);
  panel.add(sp, BorderLayout.CENTER);

  // add listeners
  previousBtn.addActionListener(this);
  nextBtn.addActionListener(this);
}

public void actionPerformed(ActionEvent e) {

  if ((e.getSource() == previousBtn))
    setPrevious();
  else
    setNext();
  imageLabel.setIcon(imageArray[index]);
}

private void initImageArray() {
  String path = new String(imagePath + "pic");
  for (int i=0; i<imageArray.length; i++) {
    imageArray[i] = new ImageIcon(path + i + ".jpg");
  }
}

private void setPrevious() {
  if (index == 0) {
    index = imageArray.length - 1;
  }
  else
    index--;
}

private void setNext() {
  if (index == (imageArray.length-1)) {
    index = 0;
  }
  else
    index++;
}

protected static ImageIcon getImageIcon(String path) {
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
    // create an ImageIcon object if path is valid
    // else returns null
    java.net.URL imageUrl = ScrollPanes.class.getResource(path);
    if (imageUrl != null) {
      return new ImageIcon(imageUrl);
    }
    else {
     System.err.println("No image found at: " + path);
     return null;
    }
  }

  private static void createFrameAndShow() {
    //Make sure we have nice window decorations.
    JFrame.setDefaultLookAndFeelDecorated(true);

    //Create and set up the window.
    ScrollPanes1 frame = new ScrollPanes1();
    frame.setTitle("ScrollPanes Test");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    //Create and set up the content pane.
    JComponent contentPane = (JComponent)frame.getContentPane();
    contentPane.setOpaque(true); //content panes must be opaque
    frame.setContentPane(contentPane);
    contentPane.add(frame.panel);

    //Display the window.
    frame.pack();
    frame.setLocation(300, 100);
    frame.setSize(400, 300);
    frame.setVisible(true);
  }

  public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        createFrameAndShow();
      }
    });
  }
}
```

Code 6.2 illustrates how to add a column header and four corner components. Let us examine the code:

1. A JLabel component is defined to contain the string for the column header.
2. Four labels, one for each corner, are defined.
3. The four corner components are coloured square images whose paths are defined.
4. We get the image icons for the corner components given the paths.
5. We set the corner components into the scroll pane, sp, and set the column header using setColumnHeaderView() method into sp. Note that setColumnHeaderView() takes in a Component as its argument; this suggests that a column header (or row header) can be a text area or any object that is a subclass of Component.
6. We have earlier set the horizontal alignment for the column header to CENTER.
7. Although four corner components were set, only two corner components are visible. Why? ***A corner component is visible only if there is an intersection of two sides.*** Since a row header is not defined, there is no intersection in the upper left and lower left corners, hence no corner component appears at these locations.

6.2 Split Pane

Take a close look at Figure 6.6. There are two scrollable text areas on a single pane. How do we produce this?

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

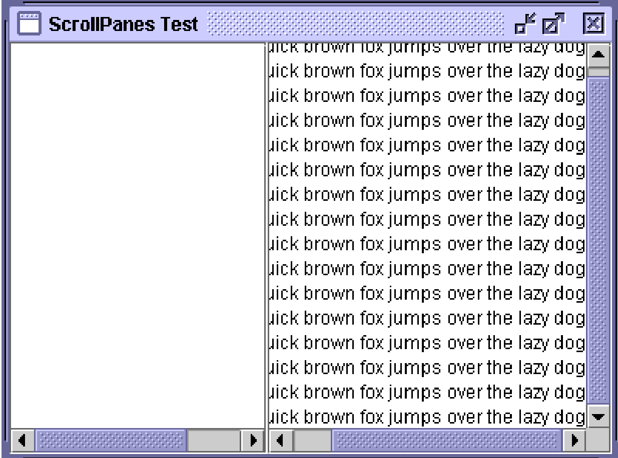


FIGURE 6.6: Two Scrollable Components on a Pane

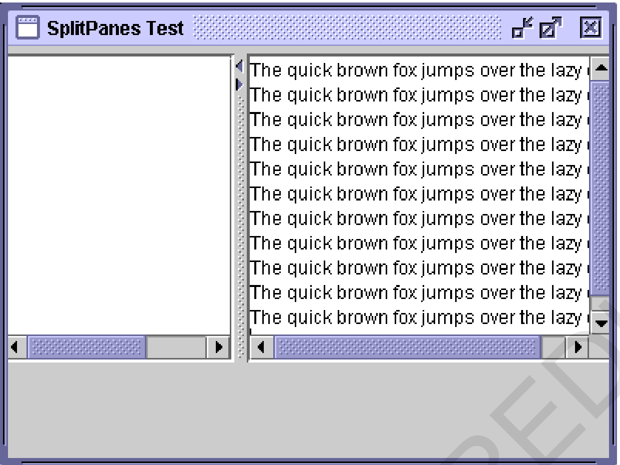


FIGURE 6.7: Two Scrollable Components on a Split Pane

One solution is to define two JScrollPanes, each with its own JTextArea component, and place them side by side using a layout manager (such as BorderLayout) in a panel. There is a restriction with this solution – the size of the two scroll panes cannot be adjusted once they are set. Another solution is to use a Split Pane (see Figure 6.7). Instead of adding the two scroll panes into a panel, we add them into a *split pane*. The latter is then added into a panel.

With a split pane, the size of the scroll panes can be adjusted dynamically during run-time (see Figure 6.8).

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

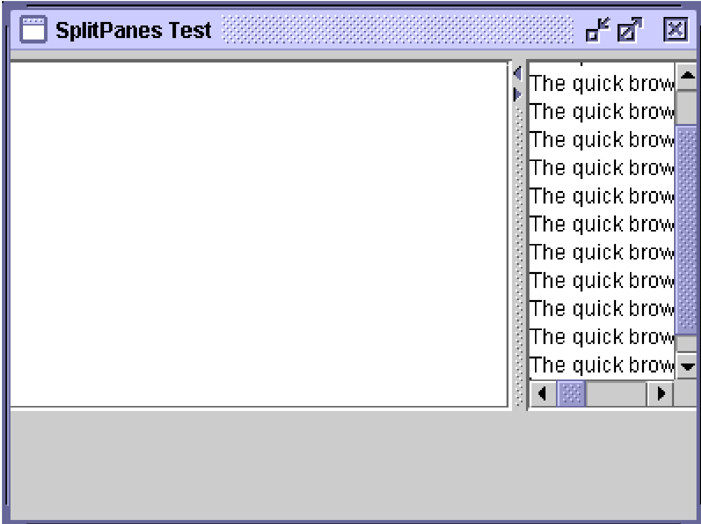


FIGURE 6.8: Adjusting Scrollable Components on a Split Pane

6.2.1 JSplitPane – Splitting a Pane into Multiple Panes

The Java Swing version of a split pane is JSplitPane. Figure 6.9 is the class hierarchy for JSplitPane.

```
java.lang.Object
└── java.awt.Component
    └── java.awt.Container
        └── javax.swing.JComponent
            └── javax.swing.JSplitPane
```

FIGURE 6.9: Class Hierarchy of JSplitPane

6.2.1.1 What are the Constructors?

JSplitPane has five constructor methods (see Table 6.3).

TABLE 6.3: JSplitPane Constructors

No.	Constructor	Description
1	`JSplitPane()`	Creates a new JSplitPane object, configured to arrange the child components side by side horizontally with no continuous layout, using two buttons for the components.
2	`JSplitPane(int newOrientation)`	Creates a new JSplitPane object, configured with the specified orientation and no continuous layout.
3	`JSplitPane(int newOrientation, boolean newContinuousLayout)`	Creates a new JSplitPane object, configured with the specified orientation and redrawing style.
4	`JSplitPane(int newOrientation, boolean newContinuousLayout, Component newLeftComponent, Component newRightComponent)`	Creates a new JSplitPane object, configured with the specified orientation, redrawing style and with the specified components.
5	`JSplitPane(int newOrientation, Component newLeftComponent, Component newRightComponent)`	Creates a new JSplitPane object, configured with the specified orientation and with the specified components that do not do continuous redrawing.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

6.2.1.2 How are Events Handled?

A JSplitPane object does not generate any specific event object of its own. However, as a subclass of java.awt.Component, java.awt.Container, and javax.swing.JComponent, we may add listeners to listen to events that are typical of these classes on JSplitPane.

6.2.1.3 What are the Useful Methods?

Table 6.4 highlights some useful methods of JSplitPane.

TABLE 6.4: Useful Methods

Method	Use
`void setOneTouchExpandable(boolean newValue)`	Sets the value of oneTouchExpandable property, which must be true for the JSplitPane to provide a UI widget on the divider to quickly expand/collapse the divider.
`void setDividerLocation(int location)`	Sets the location of the divider.
`void setPreferredSize(Dimension preferredSize)` *inherited from JComponent*	Sets the preferred size of this component. If preferredSize is null, the UI will be asked for the preferred size.

6.2.1.4 Show Me an Application of JSplitPane

We will modify the application first introduced in Section 6.1.1 to illustrate the use of JSplitPane. Figure 6.10 shows the new display.

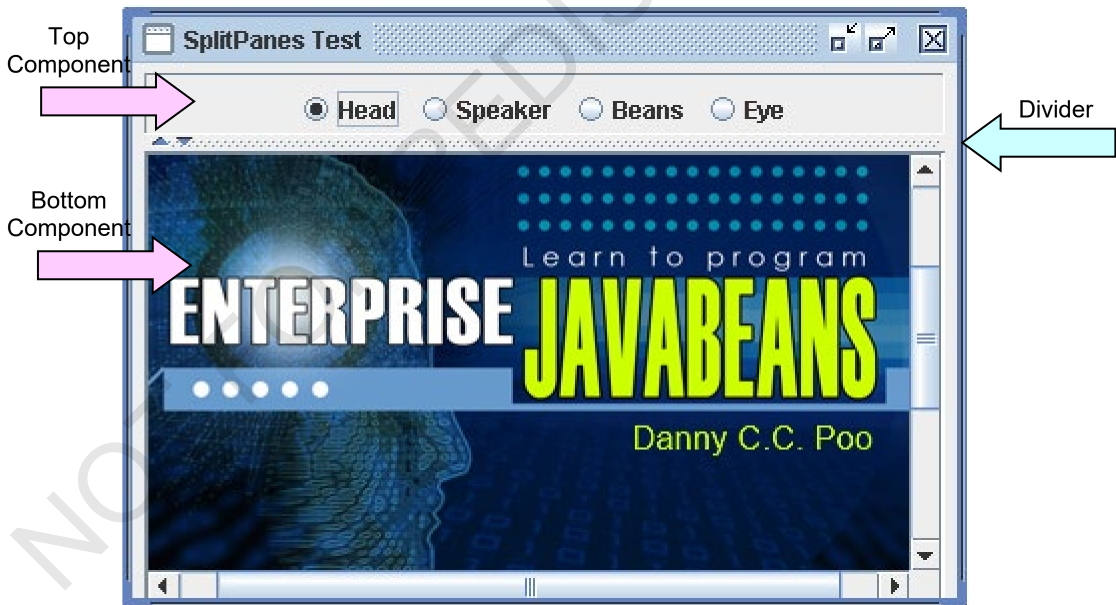


FIGURE 6.10: SplitPanes Test

The display is distinctively a JSplitPane by the presence of a *divider* separating the top component from the bottom component. Four radio-buttons are used to select the images in the top component.

With a JSplitPane, the size of the top and bottom components can be adjusted dynamically by the user (compare Figure 6.10 with Figure 6.11).

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

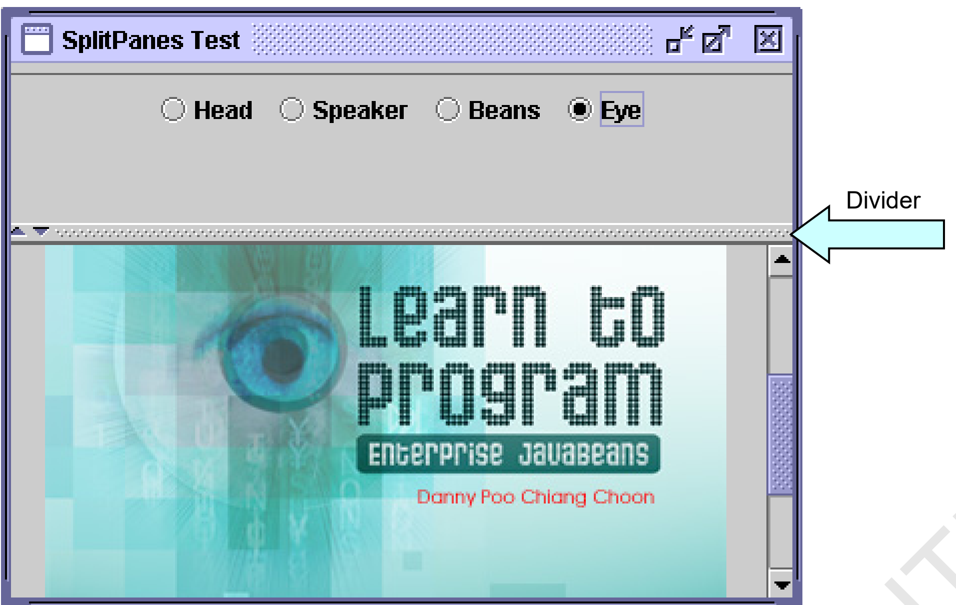


FIGURE 6.11: Adjusting the Size of the Two Components via Divider

Code 6.3: SplitPanes Test

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

public class SplitPanes extends JFrame
                        implements ItemListener {

  // labels
  private JLabel imageLabel;

  // paths
  private String imagePath     = "./images/";

  // radio buttons
  private JRadioButton head     = new JRadioButton("Head", true);
  private JRadioButton speaker = new JRadioButton("Speaker");
  private JRadioButton beans    = new JRadioButton("Beans");
  private JRadioButton eye      = new JRadioButton("Eye");

  // create a radio button group
  ButtonGroup rbuttonGroup = new ButtonGroup();

  // panels
  private JPanel panel         = new JPanel();
  private JPanel panel1        = new JPanel();

  // images
  ImageIcon image;

  // array to hold images
  ImageIcon[] imageArray       = new ImageIcon[4];
  int         index            = 0;

  // scroll panes
  JScrollPane sp = new JScrollPane(imageLabel);

  // split panes
  JSplitPane splitPane;


  public SplitPanes() {

    // initialize array
```

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

```
    initImageArray();

    // group radio buttons
    rbuttonGroup.add(head);
    rbuttonGroup.add(speaker);
    rbuttonGroup.add(beans);
    rbuttonGroup.add(eye);

    // set layout
    panel1.setLayout(new FlowLayout());

    // create a scroll pane to display images
    image          = imageArray[index];
    imageLabel     = new JLabel(image);
    JScrollPane sp = new JScrollPane(imageLabel);

    // add Components to panel1
    panel1.add(head);
    panel1.add(speaker);
    panel1.add(beans);
    panel1.add(eye);

    // create a split pane with the two components in it.
    splitPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT,
                               panel1, sp);
    splitPane.setOneTouchExpandable(true);
    splitPane.setDividerLocation(30);

    // provide minimum sizes for components in the split pane
    Dimension minimumSize = new Dimension(400, 30);
    panel1.setMinimumSize(minimumSize);
    sp.setMinimumSize(minimumSize);

    // provide a preferred size for the split pane.
    splitPane.setPreferredSize(new Dimension(400,266));

    // add split pane into panel
    panel.add(splitPane);

    // add listeners
    head.addItemListener(this);
    speaker.addItemListener(this);
    beans.addItemListener(this);
    eye.addItemListener(this);
  }

  /** Event Handler when radio button is selected */
  public void itemStateChanged(ItemEvent e) {
    if (e.getSource() == head) {
        System.out.println("Head Selected");
        imageLabel.setIcon(imageArray[0]);
    }
    else
      if (e.getSource() == speaker) {
          System.out.println("Speaker Selected");
          imageLabel.setIcon(imageArray[1]);
      }
      else
        if (e.getSource() == beans) {
            System.out.println("Beans Selected");
            imageLabel.setIcon(imageArray[2]);
        }
        else
          if (e.getSource() == eye) {
            System.out.println("Eye Selected");
            imageLabel.setIcon(imageArray[3]);
          }
  }

  private void initImageArray() {
    String path = new String(imagePath + "pic");
    for (int i=0; i<imageArray.length; i++) {
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
      imageArray[i] = new ImageIcon(path + i + ".jpg");
    }
  }

  protected static ImageIcon getImageIcon(String path) {
    // create an ImageIcon object if path is valid
    // else returns null
    java.net.URL imageUrl = ScrollPanes.class.getResource(path);
    if (imageUrl != null) {
      return new ImageIcon(imageUrl);
    }
    else {
     System.err.println("No image found at: " + path);
     return null;
    }
  }

  private static void createFrameAndShow() {
    //Make sure we have nice window decorations.
    JFrame.setDefaultLookAndFeelDecorated(true);

    //Create and set up the window.
    SplitPanes frame = new SplitPanes();
    frame.setTitle("SplitPanes Test");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    //Create and set up the content pane.
    JComponent contentPane = (JComponent)frame.getContentPane();
    contentPane.setOpaque(true); //content panes must be opaque
    frame.setContentPane(contentPane);
    contentPane.add(frame.panel);

    //Display the window.
    frame.pack();
    frame.setLocation(300, 100);
    frame.setSize(400, 300);
    frame.setVisible(true);
  }

  public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        createFrameAndShow();
      }
    });
  }
}
```

Code 6.3 produces Figure 6.10. Let us examine the code:

1. Defines a SplitPanes class. This class implements the ItemListener interface.
2. Radio buttons are created.
3. Creates a ButtonGroup to associate the four radio buttons as one group.
4. A JSplitPane is defined and an object of this class is created using the fifth constructor method.
5. Set the necessary parameters to ensure the split pane behave properly during an interaction with the user.
6. The split pane, splitPane, is added into a panel to ensure it is viewable.
7. Add the frame as the item listener for the four buttons. Implementation of the ItemListener interface requires the implementation of the itemStateChanged() method. In this method, we display a line indicating the name of the button selected. From the output on the command prompt as shown below,

```
Head Selected
Speaker Selected
Speaker Selected
Beans Selected
Beans Selected
Eye Selected
Eye Selected
Head Selected
```

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

```
Head Selected
Beans Selected
Beans Selected
Speaker Selected
```

It is clear that two events are generated for each click on a button. One event is generated for the deselection of the previously selected button while the other event is generated for the selection of the current button.

6.2.2 Horizontal and Vertical Split

Figure 6.10 is split vertically, but a split pane can also be split horizontally as in Figure 6.12.

FIGURE 6.12: Horizontal Split

Code 6.4, with much of the code masked for simplicity, demonstrates how a horizontal split can be achieved. In addition to changing the layout manager from FlowLayout to GridLayout, we have also changed the orientation from JSplitPane.VERTICAL_SPLIT to JSplitPane.HORIZONTAL_SPLIT. Two other parameters have also been changed to maintain a pleasant look-and-feel; these include minimumSize and preferredSize.

Code 6.4: Horizontal Split

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

public class SplitPanes2 extends JFrame
                        implements ItemListener {

  // labels
  private JLabel imageLabel;

  // paths
  private String imagePath    = "./images/";

  // radio buttons
  private JRadioButton eagle     = new JRadioButton("Eagle", true);
  private JRadioButton redGlobe  = new JRadioButton("Red Globe");
  private JRadioButton orchid    = new JRadioButton("Orchid");
  private JRadioButton blueGlobe = new JRadioButton("Blue Globe");
  private JRadioButton digital   = new JRadioButton("Digital");

  // create a radio button group
  ButtonGroup rbuttonGroup = new ButtonGroup();
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
// panels
private JPanel panel        = new JPanel();
private JPanel panel1       = new JPanel();

// images
ImageIcon image;

// array to hold images
ImageIcon[] imageArray      = new ImageIcon[5];
int         index           = 0;

// scroll panes
JScrollPane sp = new JScrollPane(imageLabel);

// split panes
JSplitPane splitPane;


public SplitPanes2() {

  // initialize array
  initImageArray();

  // group radio buttons
  rbuttonGroup.add(eagle);
  rbuttonGroup.add(redGlobe);
  rbuttonGroup.add(orchid);
  rbuttonGroup.add(blueGlobe);
  rbuttonGroup.add(digital);

  // set layout
  panel1.setLayout(new GridLayout(5,1));

  // create a scroll pane to display images
  image          = imageArray[index];
  imageLabel     = new JLabel(image);
  JScrollPane sp = new JScrollPane(imageLabel);

  // add Components to panel1
  panel1.add(eagle);
  panel1.add(redGlobe);
  panel1.add(orchid);
  panel1.add(blueGlobe);
  panel1.add(digital);

  // create a split pane with the two components in it.
  splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
                             panel1, sp);
  splitPane.setOneTouchExpandable(true);
  splitPane.setDividerLocation(100);

  // provide minimum sizes for the two components in the split pane
  Dimension minimumSize = new Dimension(30,400);
  panel1.setMinimumSize(minimumSize);
  sp.setMinimumSize(minimumSize);

  // provide a preferred size for the split pane.
  splitPane.setPreferredSize(new Dimension(388,260));

  // add split pane into panel
  panel.add(splitPane);

  // add listeners
  eagle.addItemListener(this);
  redGlobe.addItemListener(this);
  orchid.addItemListener(this);
  blueGlobe.addItemListener(this);
  digital.addItemListener(this);
}

/** Event Handler when radio button is selected */
public void itemStateChanged(ItemEvent e) {
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
    if (e.getSource() == eagle) {
        System.out.println("Eagle Selected");
        imageLabel.setIcon(imageArray[0]);
    }
    else
      if (e.getSource() == redGlobe) {
          System.out.println("redGlobe Selected");
          imageLabel.setIcon(imageArray[1]);
      }
      else
        if (e.getSource() == orchid) {
            System.out.println("orchid Selected");
            imageLabel.setIcon(imageArray[2]);
        }
        else
          if (e.getSource() == blueGlobe) {
            System.out.println("blueGlobe Selected");
            imageLabel.setIcon(imageArray[3]);
          }
          else
            if (e.getSource() == digital) {
              System.out.println("digital Selected");
              imageLabel.setIcon(imageArray[4]);
            }
  }

  private void initImageArray() {
    String path = new String(imagePath + "javaPic");
    for (int i=0; i<imageArray.length; i++) {
      imageArray[i] = new ImageIcon(path + i + ".jpg");
    }
  }

  protected static ImageIcon getImageIcon(String path) {
    // create an ImageIcon object if path is valid
    // else returns null
    java.net.URL imageUrl = ScrollPanes.class.getResource(path);
    if (imageUrl != null) {
      return new ImageIcon(imageUrl);
    }
    else {
     System.err.println("No image found at: " + path);
     return null;
    }
  }

  private static void createFrameAndShow() {
    //Make sure we have nice window decorations.
    JFrame.setDefaultLookAndFeelDecorated(true);

    //Create and set up the window.
    SplitPanes2 frame = new SplitPanes2();
    frame.setTitle("SplitPanes Test");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    //Create and set up the content pane.
    JComponent contentPane = (JComponent)frame.getContentPane();
    contentPane.setOpaque(true); //content panes must be opaque
    frame.setContentPane(contentPane);
    contentPane.add(frame.panel);

    //Display the window.
    frame.pack();
    frame.setLocation(300, 100);
    frame.setSize(400, 300);
    frame.setVisible(true);
  }

  public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        createFrameAndShow();
```

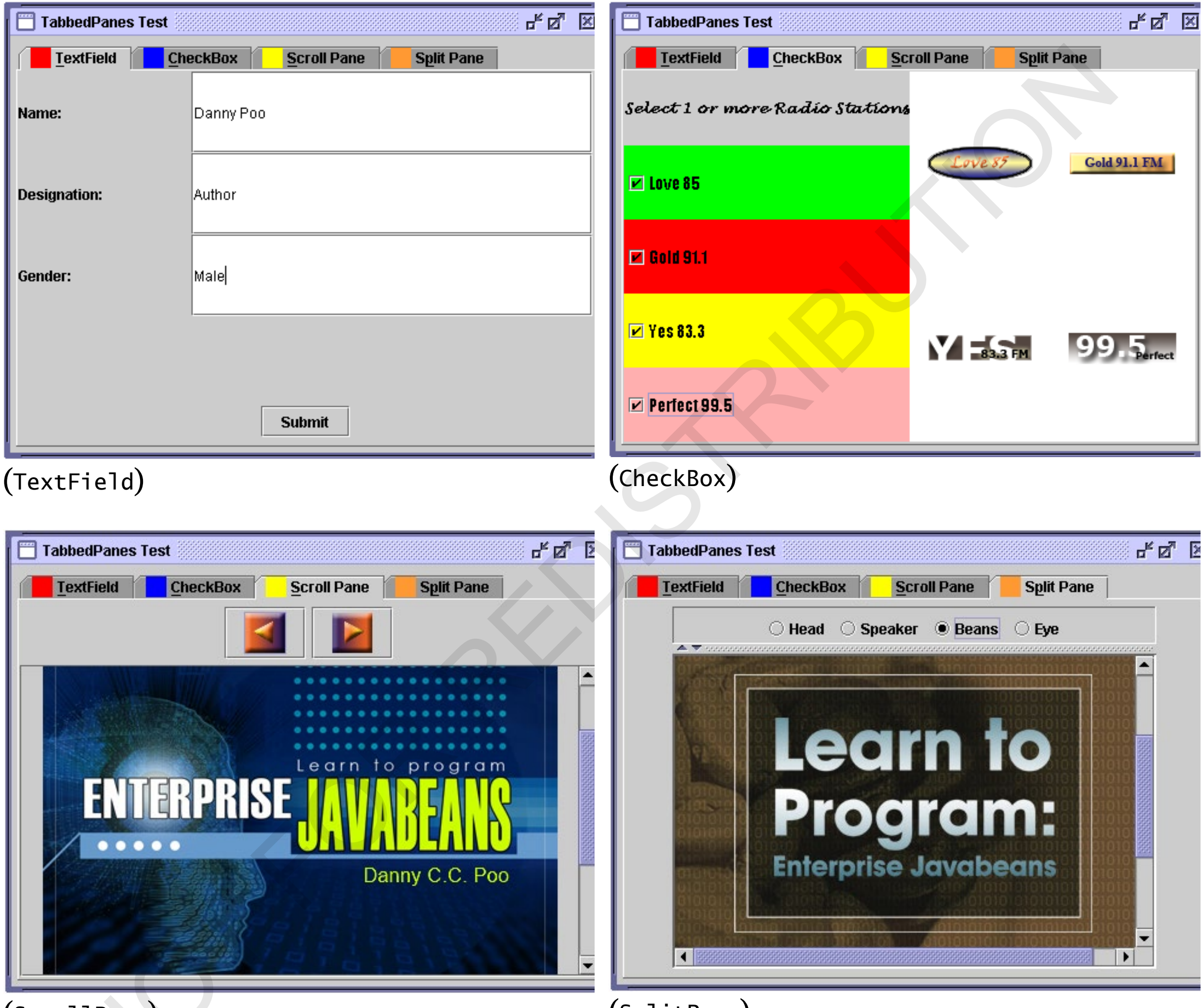
NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
            }
        });
    }
}
```

6.3 Tabbed Pane

So far, our example applications have been designed to occupy the entire frame. There are situations when applications become sub-applications of an application. How do we display sub-applications within a frame?



NOT FOR REDISTRIBUTION

FIGURE 6.13: Four Components in a Tabbed Pane

To illustrate, consider Figure 6.13. There are technically four applications within a single frame. Each of the applications is represented by a coloured tab at the top of the frame. We switch applications by selecting the coloured tab. The application introduced here include: text field, checkbox, scroll pane, and split pane. The switching functionality is provided by a *tabbed pane*.

A *tabbed pane* is a pane that can contain a group of components (usually panels). Users may switch among these components by clicking on a tab representing the desired component.

In Figure 6.13, a click on the CheckBox tab produces Figure 6.13(CheckBox). Clicking on Scroll Pane tab, we get Figure 6.13(ScrollPane) and clicking on Split Pane tab, we get Figure 6.13(SplitPane).

Copyright © 2020, Danny Poo.

6.3.1 JTabbedPane – Sub-Applications in a Pane

The Java Swing version of a tabbed pane is JTabbedPane. Figure 6.14 is the class hierarchy for JTabbedPane.

```
java.lang.Object
└── java.awt.Component
    └── java.awt.Container
        └── javax.swing.JComponent
            └── javax.swing.JTabbedPane
```

FIGURE 6.14: Class Hierarchy of JTabbedPane

6.3.1.1 What are the Constructors?

JTabbedPane has three constructor methods (see Table 6.5).

TABLE 6.5: JTabbedPane Constructors

No.	Constructor	Description
1	`JTabbedPane()`	Creates a new empty JTabbedPane object, with a default tab placement of JTabbedPane.TOP.
2	`JTabbedPane(int tabPlacement)`	Creates a new empty JTabbedPane object, with the specified tab placement of either JTabbedPane.TOP, JTabbedPane.BOTTOM, JTabbedPane.LEFT, or JTabbedPane.RIGHT.
3	`JTabbedPane(int tabPlacement, int tabLayoutPolicy)`	Creates a new empty JTabbedPane object, with the specified tab placement and tab layout policy.

6.3.1.2 How are Events Handled?

A JTabbedPane object does not generate any specific event object of its own. However, as a subclass of java.awt.Component, java.awt.Container, and javax.swing.JComponent, we may add listeners to listen to events that are typical of these classes on JTabbedPane.

6.3.1.3 What are the Useful Methods?

Table 6.6 highlights some useful methods of JTabbedPane. Tab components are added into a JTabbedPane object via addTab() and insertTab() methods.

TABLE 6.6: Useful Methods

Method	Use
void **addTab**(String title, Component component)	Adds a component specified by title with no icon.
void **addTab**(String title, Icon icon, Component component)	Adds a component specified by title and/or icon, either of which can be null.
void **addTab**(String title, Icon icon, Component component, String tip)	Adds a component specified by title, tip and/or icon, either of which can be null.
void **setTabLayoutPolicy**(int layout)	Set the policy the tabbed pane uses in layout out the tabs when all the tabs do not fit within a single run. Possible values are WRAP_TAB_LAYOUT (default and preferred) and SCROLL_TAB_LAYOUT.
void **getTabLayoutPolicy**()	Get the policy the tabbed pane uses in layout out the tabs when all the tabs do not fit within a single run.
void **setTabPlacement**(int pos)	Set where the tabs appear, relative to the content. Possible values: TOP, BOTTOM, LEFT, and

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

	RIGHT.
void **getTabPlacement**()	Get the tab's placement setting relative to the content.
void **insertTab**(String title, Icon icon, Component component, String tip, int index)	Inserts a component, at index, specified by title, tip and/or icon, either of which can be null.
void **remove**(Component c)	Remove the tab corresponding to the specified component.
void **removeTabAt**(int index)	Remove the tab corresponding to the specified index.
void **removeAll**()	Remove all tabs.
int **indexOfComponent**(Component c)	Return the index of the tab that has the specified component.
int **indexOfTab**(String title)	Return the index of the tab that has the specified title.
int **indexOfTab**(Icon c)	Return the index of the tab that has the specified icon.
void **setSelectedIndex**(int i)	Select the tab that has the specified index. Selecting a tab has the effect of displaying its associated component.
void **setSelectedComponent** (Component c)	Select the tab that has the specified component. Selecting a tab has the effect of displaying its associated component.
int **getSelectedIndex**()	Get the index for the selected tab.
Component **getSelectedComponent**()	Return the component for the selected tab.
void **setComponent**(int index, Component c)	Set which component is associated with the tab at the specified index. The first tab is at index 0.
Component **getComponentAt**(int index)	Get the component associated with the tab at specified index.
void **setTitleAt**(int index, String title)	Set the title of the tab at the specified index.
String **getTitleAt**(int index)	Get the title of the tab at the specified index.
void **setIconAt**(int index, Icon icon)	Set the icon displayed by the tab at the specified index.
Icon **getIconAt**(int index)	Get the icon displayed by the tab at the specified index.
void **setDisabledIconAt**(int index, Icon icon)	Set the icon displayed by the tab at the specified index.
Icon **getDisabledIconAt**(int index)	Get the icon displayed by the tab at the specified index.
void **setBackgroundAt**(int index, Color c)	Set the background colour used by the tab at the specified index. By default, a tab uses the tabbed pane's background colours.
Color **getBackgroundAt**(int index)	Get the background colour used by the tab at the specified index.
void **setForegroundAt**(int index, Color c)	Set the foreground colour used by the tab at the specified index. By default, a tab uses the tabbed pane's foreground colours.
Color **getForegroundAt**(int index)	Get the foreground colour used by the tab at the specified index.
void **setEnabledAt**(int index, boolean b)	Set the enabled state of the tab at the specified index.
boolean **isEnabledAt**(int index)	Get the enabled state of the tab at the specified index.
void **setMnemonicAt**(int index, int mnemonic)	Set the keyboard mnemonic for accessing the tab at the specified index.
void **getMnemonicAt**(int index)	Get the keyboard mnemonic for accessing the tab at the specified index.
void **setDisplayedMnemonicIndexAt**(int index, int mnemonic)	Set the character for representing the mnemonic. This is useful when the mnemonic character appears multiple times in the tab's title and you do not want the first occurrence to be underlined.
void **getDisplayedMnemonicIndexAt**(int index, int mnemonic)	Get the character for representing the mnemonic.
void **setToolTipTextAt**(int index, String tip)	Set the text displayed on tool tips for the tab at the

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

	specified index.
void **getToolTipTextAt** (int index)	Get the text displayed on tool tips for the tab at the specified index.

6.3.1.4 Show Me an Application of JTabbedPane

The application of JTabbedPane is based on the one shown in Figure 6.13. In this application, four sub-applications have been added into a JTabbedPane. The sub-application behaves in the same way we have discussed earlier. Each sub-application is treated as a component in the JTabbedPane.

Code 6.5 implements the application. Let us examine the code:

1. A JTabbedPane is defined and created.
2. Four panel components (TextFieldPanel, CheckBoxPanel, ScrollPanePanel, SplitPanePanel) are created. These panel components were defined as frames in previous chapters. We have converted them into JPanels in this example. JTabbedPane essentially contain panels.
3. The four panels are added into the JTabbedPane using the addTab() method. For example, the first parameter identifies the tab as "TextField". The red square image icon is identified by icon0 and panel1 refers to the TextField panel. Finally, "Text Field Text" is the text displayed when the mouse is over the tab.
4. We use the setMnemonicAt() method to assign the letter 'T' on the keyboard to trigger the selection of the tab. To activate mnemonic on TextField pane, press ALT+T (press 'ALT'and 'T' together on the keyboard).
5. The tabbed pane is finally added into a displaying panel.
6. You should be familiar with the rest of the statements and therefore would not require further explanation of the code.

Code 6.5: TabbedPanes Test

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

public class TabbedPanes extends JFrame {

  // paths
  private String imagePath    = "./images/";

  // panels
  private JPanel panel        = new JPanel();

  // tabbed panes
  JTabbedPane tabbedPane;


  public TabbedPanes() {

    // create tabbed pane
    tabbedPane = new JTabbedPane();

    // set image icons
    ImageIcon icon0 = getImageIcon(imagePath + "corner0.gif");
    ImageIcon icon1 = getImageIcon(imagePath + "corner1.gif");
    ImageIcon icon2 = getImageIcon(imagePath + "corner2.gif");
    ImageIcon icon3 = getImageIcon(imagePath + "corner3.gif");

    // create panels
    JPanel panel1 = new TextFieldPanel();
    JPanel panel2 = new CheckBoxPanel();
    JPanel panel3 = new ScrollPanePanel();
    JPanel panel4 = new SplitPanePanel();

    // add panels to tabbed pane
    tabbedPane.addTab("TextField", icon0, panel1,
                      "Text Field Test");
    tabbedPane.setMnemonicAt(0, 'T');
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
    tabbedPane.addTab("CheckBox", icon1, panel2,
                      "Check Box Test");
    tabbedPane.setMnemonicAt(1, 'C');

    tabbedPane.addTab("Scroll Pane", icon2, panel3,
                      "Scroll Pane Test");
    tabbedPane.setMnemonicAt(2, 'S');

    tabbedPane.addTab("Split Pane", icon3, panel4,
                      "Split Pane Test");
    tabbedPane.setMnemonicAt(3, 'P');

    // add tabbed pane to displaying panel
    panel.add(tabbedPane);

  }

  protected static ImageIcon getImageIcon(String path) {
    // create an ImageIcon object if path is valid
    // else returns null
    java.net.URL imageUrl = TabbedPanes.class.getResource(path);
    if (imageUrl != null) {
      return new ImageIcon(imageUrl);
    }
    else {
     System.err.println("No image found at: " + path);
     return null;
    }
  }

  private static void createFrameAndShow() {
    //Make sure we have nice window decorations.
    JFrame.setDefaultLookAndFeelDecorated(true);

    //Create and set up the window.
    TabbedPanes frame = new TabbedPanes();
    frame.setTitle("TabbedPanes Test");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    //Create and set up the content pane.
    JComponent contentPane = (JComponent)frame.getContentPane();
    contentPane.setOpaque(true); //content panes must be opaque
    frame.setContentPane(contentPane);
    contentPane.add(frame.panel);

    //Display the window.
    frame.pack();
    frame.setLocation(300, 100);
    //frame.setSize(400, 300);
    frame.setVisible(true);
  }

  public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        createFrameAndShow();
      }
    });
  }
}
```

Code for the four panels are given in Code 6.6 (TextFieldPanel), Code 6.7(CheckBoxPanel), Code 6.8(ScrollPanePanel), and Code 6.9(SplitPanePanel). We have discussed these components before, so we would not elaborate them any further here.

Execute and test the application. As you select and click on the application, notice the outputs generated on the command prompt, for example:

```
Head Selected
```

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

```
Beans Selected
Beans Selected
Speaker Selected
Speaker Selected
Eye Selected
Eye Selected
Head Selected
Head Selected
name        = Danny Poo
designation = Author
gender      = Male
Love 85 Selected
Love 85 Clicked
Yes 83.3 Selected
Yes 83.3 Clicked
Perfect 99.5 Selected
Perfect 99.5 Clicked
Beans Selected
Beans Selected
Head Selected
```

Code 6.6: TextFieldPanel Class

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

public class TextFieldPanel extends JPanel implements ActionListener {
  private JTextField name   = new JTextField(30);
  private JTextField desgn  = new JTextField(10);
  private JTextField gender = new JTextField(6);
  private JLabel label1 = new JLabel("Name: ");
  private JLabel label2 = new JLabel("Designation: ");
  private JLabel label3 = new JLabel("Gender: ");
  private JButton submitButton = new JButton("Submit");
  private JPanel panel1 = new JPanel();
  private JPanel panel2 = new JPanel();
  private JPanel panel3 = new JPanel();
  private JPanel panel  = new JPanel();

  public TextFieldPanel() {

    // set panel layout
    panel1.setLayout(new GridLayout(4,1));
    panel2.setLayout(new GridLayout(4,1));
    panel3.setLayout(new FlowLayout());
    this.setLayout(new BorderLayout());

    //add components to this panel
    panel1.add(label1);
    panel1.add(label2);
    panel1.add(label3);
    panel2.add(name);
    panel2.add(desgn);
    panel2.add(gender);
    panel3.add(submitButton);
    this.add(panel1, BorderLayout.CENTER);
    this.add(panel2, BorderLayout.EAST);
    this.add(panel3, BorderLayout.SOUTH);

    // add listeners
    submitButton.addActionListener(this);
  }

  public void actionPerformed(ActionEvent e) {
    System.out.println("name        = " + name.getText());
    System.out.println("designation = " + desgn.getText());
    System.out.println("gender      = " + gender.getText());
  }
}
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

Code 6.7: CheckBoxPanel Class

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

class CheckBoxPanel extends JPanel implements ActionListener, ItemListener{

  // create checkbox objects
  JCheckBox love85   = new JCheckBox("Love 85");
  JCheckBox gold911   = new JCheckBox("Gold 91.1", true);
  JCheckBox yes833    = new JCheckBox("Yes 83.3");
  JCheckBox perfect995 = new JCheckBox("Perfect 99.5");

  // create image labels
  JLabel love85Image   = new JLabel(new ImageIcon("./images/love85.gif"));
  JLabel gold911Image   = new JLabel(new ImageIcon("./images/gold911.gif"));
  JLabel yes833Image    = new JLabel(new ImageIcon("./images/yes833.gif"));
  JLabel perfect995Image = new JLabel(new ImageIcon("./images/perfect995.gif"));

  // Create label
  JLabel    label  = new JLabel("Select 1 or more Radio Stations");


  CheckBoxPanel() {

    // set GridLayout
    GridLayout layout = new GridLayout(1,2);
    this.setLayout(layout);

    // create 2 panels
    JPanel leftPanel  = new JPanel();
    JPanel rightPanel = new JPanel();
    rightPanel.setBackground(Color.white);

    // creates a label font
    Font labelFont = new Font("Lucida handwriting", Font.BOLD, 12);

    // creates a checkbox font
    Font checkBoxFont = new Font("Impact", Font.PLAIN, 14);

    // set font and background colors
    label.setFont(labelFont);
    love85.setFont(checkBoxFont);
    gold911.setFont(checkBoxFont);
    yes833.setFont(checkBoxFont);
    perfect995.setFont(checkBoxFont);
    label.setBackground(Color.white);
    love85.setBackground(Color.green);
    gold911.setBackground(Color.red);
    yes833.setBackground(Color.yellow);
    perfect995.setBackground(Color.pink);

    // set visibility
    love85Image.setVisible(false);
    gold911Image.setVisible(true);
    yes833Image.setVisible(false);
    perfect995Image.setVisible(false);

    // add listener
    love85.addActionListener(this);
    love85.addItemListener(this);
    gold911.addActionListener(this);
    gold911.addItemListener(this);
    yes833.addActionListener(this);
    yes833.addItemListener(this);
    perfect995.addActionListener(this);
    perfect995.addItemListener(this);

    // set panel layout
    leftPanel.setLayout(new GridLayout(5,1));
    rightPanel.setLayout(new GridLayout(2,2));
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
    // add into panels
    leftPanel.add(label);
    leftPanel.add(love85);
    leftPanel.add(gold911);
    leftPanel.add(yes833);
    leftPanel.add(perfect995);
    rightPanel.add(love85Image);
    rightPanel.add(gold911Image);
    rightPanel.add(yes833Image);
    rightPanel.add(perfect995Image);

    // add panels to this panel
    this.add(leftPanel);
    this.add(rightPanel);
  }

  /** Event Handler when checkbox is clicked */
  public void actionPerformed(ActionEvent e) {

    if ((e.getSource() == love85))
      System.out.println("Love 85 Clicked");
    else
      if ((e.getSource() == gold911))
        System.out.println("Gold 91.1 Clicked");
      else
        if ((e.getSource() == yes833))
          System.out.println("Yes 83.3 Clicked");
        else
          System.out.println("Perfect 99.5 Clicked");
  }

  /** Event Handler when checkbox is selected */
  public void itemStateChanged(ItemEvent e) {

    if (e.getSource() == love85)
      if (love85.isSelected()) {
        System.out.println("Love 85 Selected");
        love85Image.setVisible(true);
      }
      else {
        System.out.println("Love 85 deSelected");
        love85Image.setVisible(false);
      }
    else
      if (e.getSource() == gold911)
        if (gold911.isSelected()) {
          System.out.println("Gold 91.1 Selected");
          gold911Image.setVisible(true);
        }
        else {
          System.out.println("Gold 91.1 deSelected");
          gold911Image.setVisible(false);
        }
      else
        if (e.getSource() == yes833)
          if (yes833.isSelected()) {
            System.out.println("Yes 83.3 Selected");
            yes833Image.setVisible(true);
          }
          else {
            System.out.println("Yes 83.3 deSelected");
            yes833Image.setVisible(false);
          }
        else
          if (perfect995.isSelected()) {
            System.out.println("Perfect 99.5 Selected");
            perfect995Image.setVisible(true);
          }
          else {
            System.out.println("Perfect 99.5 deSelected");
            perfect995Image.setVisible(false);
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
        }
    }
}
```

Code 6.8: ScrollPanePanel Class

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

public class ScrollPanePanel extends JPanel implements ActionListener {

  // labels
  private JLabel imageLabel;

  // paths
  private String previousPath = "./images/previous.gif";
  private String nextPath     = "./images/next.gif";
  private String imagePath    = "./images/";

  // buttons
  private JButton previousBtn = new JButton(getImageIcon(previousPath));
  private JButton nextBtn     = new JButton(getImageIcon(nextPath));

  // panels
  private JPanel panel        = new JPanel();
  private JPanel panel1       = new JPanel();

  // images
  ImageIcon image;

  // array to hold images
  ImageIcon[] imageArray      = new ImageIcon[5];
  int         index           = 0;


  public ScrollPanePanel() {

    // initialize array
    initImageArray();

    // set layout
    panel1.setLayout(new FlowLayout());
    this.setLayout(new BorderLayout());
    this.setPreferredSize(new Dimension(400, 300));

    // create a scroll pane to display images
    image          = imageArray[index];
    imageLabel     = new JLabel(image);
    JScrollPane sp = new JScrollPane(imageLabel);

    // add Components to panel
    panel1.add(previousBtn);
    panel1.add(nextBtn);
    this.add(panel1, BorderLayout.NORTH);
    this.add(sp, BorderLayout.CENTER);

    // add listeners
    previousBtn.addActionListener(this);
    nextBtn.addActionListener(this);
  }

  public void actionPerformed(ActionEvent e) {

    if ((e.getSource() == previousBtn))
      setPrevious();
    else
      setNext();
    imageLabel.setIcon(imageArray[index]);
  }

  private void initImageArray() {
```

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

```
    String path = new String(imagePath + "javaPic");
    for (int i=0; i<imageArray.length; i++) {
      imageArray[i] = new ImageIcon(path + i + ".jpg");
    }
  }

  private void setPrevious() {
    if (index == 0) {
      index = imageArray.length - 1;
    }
    else
      index--;
  }

  private void setNext() {
    if (index == (imageArray.length-1)) {
      index = 0;
    }
    else
      index++;
  }

  protected static ImageIcon getImageIcon(String path) {
    // create an ImageIcon object if path is valid
    // else returns null
    java.net.URL imageUrl = ScrollPanes.class.getResource(path);
    if (imageUrl != null) {
      return new ImageIcon(imageUrl);
    }
    else {
     System.err.println("No image found at: " + path);
     return null;
    }
  }
}
```

Code 6.9: SplitPanePanel Class

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

public class SplitPanePanel extends JPanel
                        implements ItemListener {

  // labels
  private JLabel imageLabel;

  // paths
  private String imagePath    = "./images/";

  // radio buttons
  private JRadioButton eagle     = new JRadioButton("Eagle", true);
  private JRadioButton redGlobe  = new JRadioButton("Red Globe");
  private JRadioButton orchid    = new JRadioButton("Orchid");
  private JRadioButton blueGlobe = new JRadioButton("Blue Globe");
  private JRadioButton digital   = new JRadioButton("Digital");

  // create a radio button group
  ButtonGroup rbuttonGroup = new ButtonGroup();

  // panels
  private JPanel panel        = new JPanel();
  private JPanel panel1       = new JPanel();

  // images
  ImageIcon image;

  // array to hold images
  ImageIcon[] imageArray      = new ImageIcon[5];
  int         index           = 0;
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
// scroll panes
JScrollPane sp = new JScrollPane(imageLabel);

// split panes
JSplitPane splitPane;


public SplitPanePanel() {

  // initialize array
  initImageArray();

  // group radio buttons
  rbuttonGroup.add(eagle);
  rbuttonGroup.add(redGlobe);
  rbuttonGroup.add(orchid);
  rbuttonGroup.add(blueGlobe);
  rbuttonGroup.add(digital);

  // set layout
  panel1.setLayout(new FlowLayout());

  // create a scroll pane to display images
  image          = imageArray[index];
  imageLabel     = new JLabel(image);
  JScrollPane sp = new JScrollPane(imageLabel);

  // add Components to panel1
  panel1.add(eagle);
  panel1.add(redGlobe);
  panel1.add(orchid);
  panel1.add(blueGlobe);
  panel1.add(digital);

  // create a split pane with the two components in it.
  splitPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT,
                             panel1, sp);
  splitPane.setOneTouchExpandable(true);
  splitPane.setDividerLocation(30);

  // provide minimum sizes for the two components in the split pane
  Dimension minimumSize = new Dimension(400, 30);
  panel1.setMinimumSize(minimumSize);
  sp.setMinimumSize(minimumSize);

  // provide a preferred size for the split pane.
  splitPane.setPreferredSize(new Dimension(400,300));

  // add split pane into panel
  this.add(splitPane);

  // add listeners
  eagle.addItemListener(this);
  redGlobe.addItemListener(this);
  orchid.addItemListener(this);
  blueGlobe.addItemListener(this);
  digital.addItemListener(this);
}

/** Event Handler when radio button is selected */
public void itemStateChanged(ItemEvent e) {
  if (e.getSource() == eagle) {
      System.out.println("Eagle Selected");
      imageLabel.setIcon(imageArray[0]);
  }
  else
    if (e.getSource() == redGlobe) {
        System.out.println("redGlobe Selected");
        imageLabel.setIcon(imageArray[1]);
    }
    else
      if (e.getSource() == orchid) {
```

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

```
            System.out.println("orchid Selected");
            imageLabel.setIcon(imageArray[2]);
        }
        else
          if (e.getSource() == blueGlobe) {
            System.out.println("blueGlobe Selected");
            imageLabel.setIcon(imageArray[3]);
          }
          else
            if (e.getSource() == digital) {
              System.out.println("digital Selected");
              imageLabel.setIcon(imageArray[4]);
            }
  }

  private void initImageArray() {
    String path = new String(imagePath + "javaPic");
    for (int i=0; i<imageArray.length; i++) {
      imageArray[i] = new ImageIcon(path + i + ".jpg");
    }
  }

  protected static ImageIcon getImageIcon(String path) {
    // create an ImageIcon object if path is valid
    // else returns null
    java.net.URL imageUrl = ScrollPanes.class.getResource(path);
    if (imageUrl != null) {
      return new ImageIcon(imageUrl);
    }
    else {
     System.err.println("No image found at: " + path);
     return null;
    }
  }
}
```

6.4 Internal Frame, Layered Pane and Desktop Pane

The four sub-applications (Check Box, Text Field, Scroll Pane, and Split Pane) that we discussed in the previous section can also be organized as frames within a frame (see Figure 6.15). The individual application within the window behaves independently of each other.

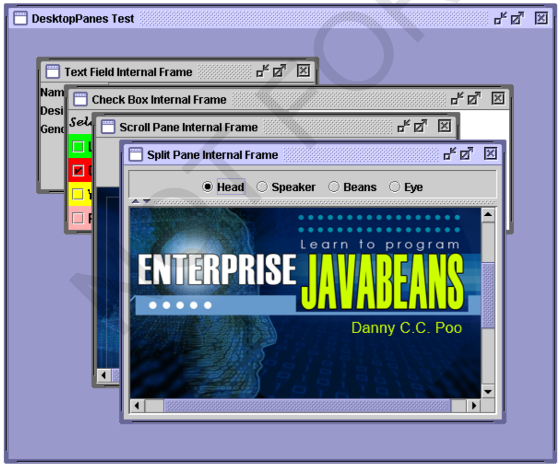


FIGURE 6.15: Frames within a Window

To produce Figure 6.15, we need to understand three components in the Java Swing: JInternalFrame, JLayeredPane and DesktopPane.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

6.4.1 JInternalFrame – A Frame Within a Frame

A JInternalFrame is a lightweight component that is similar to a JFrame. It provides many of the features a native JFrame supports including dragging, closing, iconifying, resizing, title display, and support for a menu bar. With JInternalFrames, we can add internal windows within a JFrame-like window.

Figure 6.16 shows the class hierarchy of JInternalFrame. Contrast this with the class hierarchy of JFrame in Figure 6.17. A JFrame is a java.awt.Window while JInternalFrame is a javax.swing.JComponent.

```
java.lang.Object
└──java.awt.Component
    └──java.awt.Container
        └──javax.swing.JComponent
             └── javax.swing.JInternalFrame
```

FIGURE 6.16: Class Hierarchy of JInternalFrame

```
java.lang.Object
└──java.awt.Component
    └──java.awt.Container
        └──java.awt.Window
             └── java.awt.Frame
                  └── javax.swing.JFrame
```

FIGURE 6.17: Class Hierarchy of JFrame

As a lightweight component, JInternalFrame is platform independent and it adds some features that native JFrame does not provide. For example, you have more control over the state and capabilities of JInternalFrame than JFrame.

There are other unique characteristics of JInternalFrame:

1. A JInternalFrame can be programmatically iconified or maximized. A developer can specify the icon that goes into a JInternalFrame's title bar and whether the JInternalFrame has the window decorations to support resizing, iconifying, closing, and maximizing.
2. Unlike JFrame, a JInternalFrame is not a top-level container. This means that a JInternalFrame cannot be the root of a containment hierarchy. A JInternalFrame has to be added into a container, such as a JDesktopPane. The latter serves as a content pane for JInternalFrame objects.
3. A JInternalFrame is by default invisible. To make it visible, invoke the method setVisible(true) or show().

6.4.1.1 What are the Constructors?

JInternalFrame has six constructor methods (see Table 6.7).

TABLE 6.7: JInternalFrame Constructors

No.	Constructor	Description
1	`JInternalFrame()`	Creates a new non-resizable, non-closable, non-maximizable, non-iconifiable JInternalFrame object with no title.
2	`JInternalFrame(String title)`	Creates a new non-resizable, non-closable, non-maximizable, non-iconifiable JInternalFrame object with the specified title.
3	`JInternalFrame(String title, boolean resizable)`	Creates a new non-closable, non-maximizable, non-iconifiable JInternalFrame object with the specified title and resizable setting.
4	`JInternalFrame(String title, boolean resizable, boolean closable)`	Creates a new non-maximizable, non-iconifiable JInternalFrame object with the specified title, resizable and closable setting.
5	`JInternalFrame(String title, boolean resizable, boolean closable, boolean`	Creates a new non-iconifiable JInternalFrame object with the specified title, resizable, closable and maximizable setting.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

	maximizable)	
6	JInternalFrame(String title, boolean resizable, boolean closable, boolean maximizable, boolean iconifiable)	Creates a new JInternalFrame object with the specified title, resizable, closable, maximizable and iconifiable setting.

6.4.1.2 How are Events Handled?

A JInternalFrame object does not generate window events. Any user actions on a window cause a frame to fire window events which in turn causes a JInternalFrame to fire frame events.

6.4.1.3 What are the Useful Methods?

Table 6.8 highlights some useful methods of JInternalFrame.

TABLE 6.8: Useful Methods

Method	Use
void **setContentPane**(Container c)	Set the internal frame's content pane.
Container **getContentPane**()	Get the internal frame's content pane.
void **setJMenuBar**(JMenuBar menuBar)	Sets the internal frame's menu bar.
JMenuBar **getJMenuBar**()	Get the internal frame's menu bar.
void **setJLayeredPane**(JLayeredPane layeredPane)	Sets the internal frame's layered pane.
JLayeredPane **getJLayeredPane**()	Get the internal frame's layered pane.
void **setVisible**(boolean visible) *inherited from Component*	Make the internal frame visible (if true) or invisible (if false).
void **pack**()	Size the internal frame so that its components are at their preferred sizes.
void **setLocation**(Point p) *inherited from Component*	Set the position of the internal frame. The top-left corner of the new location is specified by point p.
void **setLocation**(int x, int y) *inherited from Component*	Set the position of the internal frame. The top-left corner of the new location is specified by the x and y parameters in the coordinate space of this component's parent.
void **setBounds**(Rectangle rect) *inherited from Component*	Explicitly set the size and location of the internal frame.
void **setBounds**(int x, int y, int width, int height) *inherited from Component*	Explicitly set the size and location of the internal frame. The top-left corner is specified by x and y, and the new size is specified by width and height.
void **setDefaultCloseOperation** (int operation)	Set what the internal frame does when the user attempts to close the internal frame. Value of operation includes DISPOSE_ON_CLOSE (default), DO_NOTHING_ON_CLOSE and HIDE_ON_CLOSE and HIDE_ON_CLOSE.
int **getDefaultCloseOperation** ()	Get what the internal frame does when the user attempts to close the internal frame.
void **addInternalFrameListener**(InternalFrameListener listener)	Add an internal frame listener.
void **removeInternalFrameListener**(InternalFrameListener listener)	Remove an internal frame listener.
void **moveToFront**()	Move the internal frame to the front of its layer if the internal frame's parent is a layered pane.
void **moveToBack**()	Move the internal frame to the back of its layer if the internal frame's parent is a layered pane.
void **setClosed**(boolean closed)	Set whether the internal frame is currently closed.
boolean **isClosed**()	Get whether the internal frame is currently closed.
void **setIcon**(boolean b)	Iconify(if true) or deiconify (if false) the internal frame.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

`boolean isIcon()`	Determine if internal frame is currently iconified.
`void setMaximum(boolean b)`	Maximizes and restores the internal frame.
`boolean isMaximum()`	Determine if internal frame is maximized.
`void setSelected(boolean selected)`	Selects or deselects the internal frame if it is showing.
`boolean isSelected()`	Determine if internal frame is selected.
`void setFrameIcon(Icon icon)`	Set the icon displayed in the title bar of the internal frame (usually in top-left corner).
`Icon getFrameIcon()`	Get the icon displayed in the title bar of the internal frame.
`void setClosable(boolean b)`	Set whether the user can close the internal frame.
`boolean isClosable()`	Determine if internal frame can be closed by user.
`void setIconifiable(boolean b)`	Set whether the internal frame can be iconified.
`boolean isIconifiable()`	Determine if internal frame can be iconified.
`void setMaximizable(boolean b)`	Set whether the internal frame can be maximizable.
`boolean isMaximizable()`	Determine if internal frame can be maximized.
`void setResizable(boolean b)`	Set whether the internal frame can be resized.
`boolean isResizable()`	Determine if internal frame can be resized.
`void setTitle(String title)`	Set the window title.
`String getTitle()`	Get the window title.

6.4.1.4 Show Me an Application of JInternalFrame

We will discuss the application of JInternalFrame jointly with JLayeredPane and JDesktopPane.

6.4.2 JLayeredPane – Positioning Components with Depth

A JLayeredPane is a Java Swing container for positioning components with a depth dimension. The depth dimension is specified as an integer value. The greater the integer value, the greater the depth.

Components may overlap and if they do, components at a higher depth are drawn on top of components at a lower depth. There is a set of pre-determined constant integer value for the depth (see Table 6.9).

TABLE 6.9: JLayeredPane Depth

Depth	Description
DEFAULT_LAYER	The bottom-most and standard layer where most components go.
PALETTE_LAYER	One layer above the default layer. Useful for floating toolbars and palette.
MODAL_LAYER	The layer used for modal dialogs. The component will appear on top of any toolbars, palettes, or standard components in container.
POPUP_LAYER	This layer displays above dialogs. Popup windows associated with combo boxes, tooltips, and other help text will appear above the component, palette, or dialog that generated them.
DRAG_LAYER	When dragging a component, reassigning it to the drag layer ensures that it is positioned over every other component in the container. When finished dragging, it can be reassigned to its normal layer.

Figure 6.18 shows the various depths of the layers.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

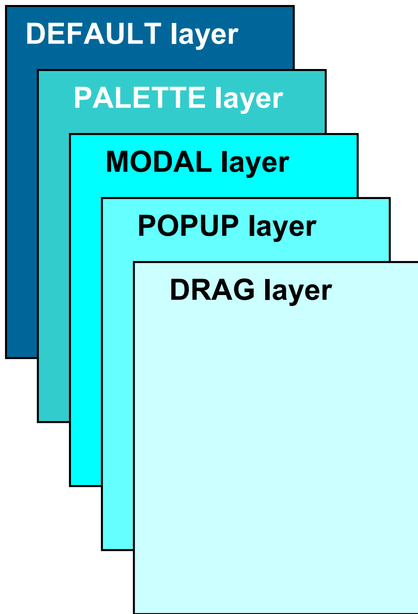


FIGURE 6.18: Layers

Figure 6.19 shows the class hierarchy of JLayeredPane.

```
java.lang.Object
└──java.awt.Component
    └──java.awt.Container
        └──javax.swing.JComponent
            └── javax.swing.JLayeredPane
```

FIGURE 6.19: Class Hierarchy of JLayeredPane

6.4.2.1 What are the Constructors?

JLayeredPane has only one constructor method (see Table 6.10).

TABLE 6.10: JLayeredPane Constructors

No.	Constructor	Description
1	`JLayeredPane()`	Creates a new JLayeredPane object.

6.4.2.2 How are Events Handled?

A JLayeredPane object responds to events typical of JComponent, Container and Component.

6.4.2.3 What are the Useful Methods?

Table 6.11 highlights some useful methods of JLayeredPane.

TABLE 6.11: Useful Methods

Method	Use
`void moveToBack(Component c)`	Moves the component to the bottom of the components in its current layer (position - 1).
`void moveToFront(Component c)`	Moves the component to the top of the components in its current layer (position 0).

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

`void setLayer(Component c, int layer)`	Sets the layer attribute on the specified component, making it the bottom-most component in that layer.
`void setLayer(Component c, int layer, int position)`	Sets the layer attribute for the specified component and also sets its position within that layer.
`void setPosition(Component c, int position)`	Moves the component to position within its current layer, where 0 is the top-most position within the layer and -1 is the bottom-most position.

6.4.2.4 Show Me an Application of JLayeredPane

We will discuss the application of JLayeredPane jointly with JInternalFrame and JDesktopPane.

6.4.3 JDesktopPane – Virtual Desktop

A JDesktopPane is a container for creating a virtual desktop. Figure 6.20 shows the class hierarchy of JDesktopPane. Note that JDesktopPane is a direct subclass of JLayeredPane. Objects added into a JDesktopPane can therefore be layered.

```
java.lang.Object
└──java.awt.Component
    └──java.awt.Container
        └──javax.swing.JComponent
            └── javax.swing.JLayeredPane
                └── javax.swing.JDesktopPane
```

FIGURE 6.20: Class Hierarchy of JDesktopPane

6.4.3.1 What are the Constructors?

JDesktopPane has only one constructor method (see Table 6.12).

TABLE 6.12: JDesktopPane Constructors

No.	Constructor	Description
1	`JDesktopPane ()`	Creates a new JDesktopPane object.

6.4.3.2 How are Events Handled?

A JDesktopPane object responds to events typical of JComponent, Container and Component.

6.4.3.3 What are the Useful Methods?

Table 6.13 highlights some useful methods of JDesktopPane.

TABLE 6.13: Useful Methods

Method	Use
`void moveToBack(Component c)`	Moves the component to the bottom of the components in its current layer (position - 1).
`void moveToFront(Component c)`	Moves the component to the top of the components in its current layer (position 0).
`void setLayer(Component c, int layer)`	Sets the layer attribute on the specified component, making it the bottom-most component in that layer.
`void setLayer(Component c, int layer, int position)`	Sets the layer attribute for the specified component and also sets its position within that layer.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

void **setPosition**(Component c, int position)	Moves the component to position within its current layer, where 0 is the top-most position within the layer and -1 is the bottom-most position.

6.4.3.4 Show Me an Application of JDesktopPane

We will make use of JDesktopPane, JLayeredPane, and JInternalFrame to implement the application shown in Figure 6.15 (reproduced here as Figure 6.21). JDesktopPane is defined as a container of multiple frames. There are four frames appearing within a single window. Each frame is an application with its own event handlers.

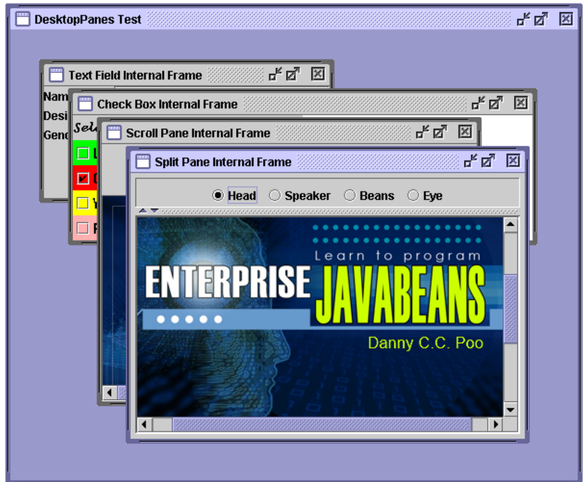


FIGURE 6.21: Frames within a Window

Code 6.10: DesktopPanes Test

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.JInternalFrame;
import javax.swing.JDesktopPane;
import javax.swing.JLayeredPane;
import javax.swing.JFrame;


public class DesktopPanes extends JFrame {

  static final int xOffset    = 30;
  static final int yOffset    = 30;
  static       int frameCount = 0;
  JDesktopPane desktop;

  public DesktopPanes() {

    // set up
    desktop = new JDesktopPane();
    createInternalFrames(); // creates internal frames
    setContentPane(desktop);
    desktop.setOpaque(true); //set content pane to opaque

    //Make dragging a little faster but perhaps uglier.
    desktop.setDragMode(JDesktopPane.OUTLINE_DRAG_MODE);
  }

  // creates internal frames
  protected void createInternalFrames() {

    // creates Text Field internal frame
    TextFieldInternalFrame textFieldIntFrame =
            new TextFieldInternalFrame("Text Field Internal Frame");
```

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

```
    textFieldIntFrame.setVisible(true);
    frameCount++;
    textFieldIntFrame.setLocation(
           xOffset*frameCount, yOffset*frameCount);
    desktop.add(textFieldIntFrame, JLayeredPane.PALETTE_LAYER);

    // creates Check Box internal frame
    CheckBoxInternalFrame checkBoxIntFrame =
           new CheckBoxInternalFrame("Check Box Internal Frame");
    checkBoxIntFrame.setVisible(true);
    frameCount++;
    checkBoxIntFrame.setLocation(
           xOffset*frameCount, yOffset*frameCount);
    desktop.add(checkBoxIntFrame, JLayeredPane.MODAL_LAYER);

    // creates Scroll Pane internal frame
    ScrollPaneInternalFrame scrollPaneIntFrame =
           new ScrollPaneInternalFrame("Scroll Pane Internal Frame");
    scrollPaneIntFrame.setVisible(true);
    frameCount++;
    scrollPaneIntFrame.setLocation(
           xOffset*frameCount, yOffset*frameCount);
    desktop.add(scrollPaneIntFrame, JLayeredPane.POPUP_LAYER);

    // creates Split Pane internal frame
    SplitPaneInternalFrame splitPaneIntFrame =
           new SplitPaneInternalFrame("Split Pane Internal Frame");
    splitPaneIntFrame.setVisible(true);
    frameCount++;
    splitPaneIntFrame.setLocation(
           xOffset*frameCount, yOffset*frameCount);
    desktop.add(splitPaneIntFrame, JLayeredPane.DRAG_LAYER);
  }

  private static void createFrameAndShow() {
    // make sure we have nice window decorations.
    JFrame.setDefaultLookAndFeelDecorated(true);

    // create and set up the window
    DesktopPanes frame = new DesktopPanes();
    frame.setTitle("DesktopPanes Test");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    // display the window
    frame.setSize(600,500);
    frame.setVisible(true);
    frame.setLocation(300, 100);

  }

  public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        createFrameAndShow();
      }
    });
  }
}
```

Figure 6.21 is produced from Code 6.10. Let us examine the code:

1. A frame is usually a top-level container and therefore cannot be treated as a component in a container. Since our application requires frames to be included within a DesktopPane, a container, the four frames have to be defined as JInternalFrames instead.
2. Code 6.10 begins by declaring DesktopPanes as a subclass of JFrame.
3. A variable frameCount keeps track of the number of internal frames instantiated.
4. We create a JDesktopPane.
5. We call createInternalFrames() method to create the four internal frames. The createInternalFrames() method is divided into four blocks: create a TextField internal frame, create a CheckBox internal frame, create a ScrollPane internal frame, and create a SplitPane

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

internal frame. Internal frames are created by instantiating the respective JInternalFrame class (TextFieldInternalFrame for TextField internal frame, CheckBoxInternalFrame for CheckBox internal frame, ScrollPaneInternalFrame for ScrollPane internal frame, and SplitPaneInternalFrame for SplitPane internal frame). These internal frame classes are given below (Code 6.11 to Code 6.14). Note that the code for the internal frames given here is similar to that of individual frames we discussed earlier.

6. The creation of an internal frame follows a set pattern of: instantiating an internal frame class, set the frame to be visible, set the location of the frame, and adding the internal frame object into the desktop pane.
7. All internal frames by default are set to be invisible, this explains why the need to call the setVisible() method.
8. In order for the desktop to be displayable on the frame, we set desktop as the content pane for the frame.
9. We set desktop to opaque to ensure that every pixel within the bounds of desktop is painted. Otherwise, not all the pixels may be painted, allowing the underlying pixels to show through.
10. We add the internal frames into desktop. The internal frames are positioned in accordance with the depth specified (see Section 6.4.2). The SplitPane internal frame appears on top of the rest because it has been set on JLayeredPane.DRAG_LAYER.

Code 6.11: TextFieldInternalFrame Class

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

public class TextFieldInternalFrame extends JInternalFrame
                                    implements ActionListener {

  private JTextField name       = new JTextField(20);
  private JTextField desgn      = new JTextField(10);
  private JTextField gender     = new JTextField(6);
  private JLabel     label1     = new JLabel("Name: ");
  private JLabel     label2     = new JLabel("Designation: ");
  private JLabel     label3     = new JLabel("Gender: ");
  private JButton submitButton = new JButton("Submit");
  private JPanel     panel1     = new JPanel();
  private JPanel     panel2     = new JPanel();
  private JPanel     panel3     = new JPanel();
  private JPanel     panel      = new JPanel();


  public TextFieldInternalFrame(String title) {
    super(title,
          true, //resizable
          true, //closable
          true, //maximizable
          true);//iconifiable

    // create gui
    panel1.setLayout(new GridLayout(4,1));
    panel2.setLayout(new GridLayout(4,1));
    panel3.setLayout(new FlowLayout());
    panel.setLayout(new BorderLayout());

    // add components to panel
    panel1.add(label1);
    panel1.add(label2);
    panel1.add(label3);
    panel2.add(name);
    panel2.add(desgn);
    panel2.add(gender);
    panel3.add(submitButton);
    panel.add(panel1, BorderLayout.CENTER);
    panel.add(panel2, BorderLayout.EAST);
    panel.add(panel3, BorderLayout.SOUTH);

    // add panel into content pane
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
    this.getContentPane().add(panel);

    // add listeners
    submitButton.addActionListener(this);

    // set size of internal frame
    pack();
  }

  public void actionPerformed(ActionEvent e) {
    System.out.println("name        = " + name.getText());
    System.out.println("designation = " + desgn.getText());
    System.out.println("gender      = " + gender.getText());
  }
}
```

Code 6.12: CheckBoxInternalFrame Class

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

public class CheckBoxInternalFrame extends JInternalFrame
                  implements ActionListener, ItemListener {

  // Create checkbox objects
  JCheckBox love85       = new JCheckBox("Love 85");
  JCheckBox gold911      = new JCheckBox("Gold 91.1", true);
  JCheckBox yes833       = new JCheckBox("Yes 83.3");
  JCheckBox perfect995   = new JCheckBox("Perfect 99.5");

  // create image labels
  JLabel love85Image     = new JLabel(new ImageIcon("../images/love85.gif"));
  JLabel gold911Image    = new JLabel(new ImageIcon("../images/gold911.gif"));
  JLabel yes833Image     = new JLabel(new ImageIcon("../images/yes833.gif"));
  JLabel perfect995Image = new JLabel(new ImageIcon("../images/perfect995.gif"));

  // Create label
  JLabel    label  = new JLabel("Select 1 or more Radio Stations");


  public CheckBoxInternalFrame(String title) {
    super(title,
          true, //resizable
          true, //closable
          true, //maximizable
          true);//iconifiable

    // create gui, create panels
    JPanel leftPanel  = new JPanel();
    JPanel rightPanel = new JPanel();
    rightPanel.setBackground(Color.white);

    // get content pane of frame
    Container contentPane = getContentPane();

    // set GridLayout
    GridLayout layout = new GridLayout(1,2);
    contentPane.setLayout(layout);

    // creates a label font
    Font labelFont = new Font("Lucida handwriting", Font.BOLD, 12);

    // creates a checkbox font
    Font checkBoxFont = new Font("Impact", Font.PLAIN, 14);

    // set font and background colors
    label.setFont(labelFont);
    love85.setFont(checkBoxFont);
    gold911.setFont(checkBoxFont);
    yes833.setFont(checkBoxFont);
```

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

```
    perfect995.setFont(checkBoxFont);
    label.setBackground(Color.white);
    love85.setBackground(Color.green);
    gold911.setBackground(Color.red);
    yes833.setBackground(Color.yellow);
    perfect995.setBackground(Color.pink);

    // set visibility
    love85Image.setVisible(false);
    gold911Image.setVisible(true);
    yes833Image.setVisible(false);
    perfect995Image.setVisible(false);

    // add listener
    love85.addActionListener(this);
    love85.addItemListener(this);
    gold911.addActionListener(this);
    gold911.addItemListener(this);
    yes833.addActionListener(this);
    yes833.addItemListener(this);
    perfect995.addActionListener(this);
    perfect995.addItemListener(this);

    // set panel layout
    leftPanel.setLayout(new GridLayout(5,1));
    rightPanel.setLayout(new GridLayout(2,2));

    // add into panels
    leftPanel.add(label);
    leftPanel.add(love85);
    leftPanel.add(gold911);
    leftPanel.add(yes833);
    leftPanel.add(perfect995);
    rightPanel.add(love85Image);
    rightPanel.add(gold911Image);
    rightPanel.add(yes833Image);
    rightPanel.add(perfect995Image);

    // add panels to contentPane
    contentPane.add(leftPanel);
    contentPane.add(rightPanel);

    // set size of internal frame
    pack();
  }

  /** Event Handler when checkbox is clicked */
  public void actionPerformed(ActionEvent e) {

    if ((e.getSource() == love85))
      System.out.println("Love 85 Clicked");
    else
      if ((e.getSource() == gold911))
        System.out.println("Gold 91.1 Clicked");
      else
        if ((e.getSource() == yes833))
          System.out.println("Yes 83.3 Clicked");
        else
          System.out.println("Perfect 99.5 Clicked");
  }

  /** Event Handler when checkbox is selected */
  public void itemStateChanged(ItemEvent e) {

    if (e.getSource() == love85)
      if (love85.isSelected()) {
        System.out.println("Love 85 Selected");
        love85Image.setVisible(true);
      }
      else {
        System.out.println("Love 85 deSelected");
        love85Image.setVisible(false);
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
    }
  else
    if (e.getSource() == gold911)
      if (gold911.isSelected()) {
        System.out.println("Gold 91.1 Selected");
        gold911Image.setVisible(true);
      }
      else {
        System.out.println("Gold 91.1 deSelected");
        gold911Image.setVisible(false);
      }
    else
      if (e.getSource() == yes833)
        if (yes833.isSelected()) {
          System.out.println("Yes 83.3 Selected");
          yes833Image.setVisible(true);
        }
        else {
          System.out.println("Yes 83.3 deSelected");
          yes833Image.setVisible(false);
        }
      else
        if (perfect995.isSelected()) {
          System.out.println("Perfect 99.5 Selected");
          perfect995Image.setVisible(true);
        }
        else {
          System.out.println("Yes 83.3 deSelected");
          perfect995Image.setVisible(false);
        }
  }
}
```

Code 6.13: ScrollPaneInternalFrame Class

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

public class ScrollPaneInternalFrame extends JInternalFrame
                                    implements ActionListener {

  // labels
  private JLabel imageLabel;

  // paths
  private String previousPath = "./images/previous.gif";
  private String nextPath     = "./images/next.gif";
  private String imagePath    = "./images/";

  // buttons
  private JButton previousBtn = new JButton(getImageIcon(previousPath));
  private JButton nextBtn     = new JButton(getImageIcon(nextPath));

  // panels
  private JPanel panel        = new JPanel();
  private JPanel panel1       = new JPanel();

  // images
  ImageIcon image;

  // array to hold images
  ImageIcon[] imageArray      = new ImageIcon[5];
  int         index           = 0;


  public ScrollPaneInternalFrame(String title) {
    super(title,
          true, //resizable
          true, //closable
          true, //maximizable
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
        true);//iconifiable

    // create gui, initialize array
    initImageArray();

    // set layout
    panel1.setLayout(new FlowLayout());
    panel.setLayout(new BorderLayout());

    // create a scroll pane to display images
    image          = imageArray[index];
    imageLabel     = new JLabel(image);
    JScrollPane sp = new JScrollPane(imageLabel);

    // add Components to panel
    panel1.add(previousBtn);
    panel1.add(nextBtn);
    panel.add(panel1, BorderLayout.NORTH);
    panel.add(sp, BorderLayout.CENTER);

    // add listeners
    previousBtn.addActionListener(this);
    nextBtn.addActionListener(this);

    // add panel into content pane
    this.getContentPane().add(panel);

    // set size of internal frame
    setSize(400,300);
  }


  public void actionPerformed(ActionEvent e) {

    if ((e.getSource() == previousBtn))
      setPrevious();
    else
      setNext();
    imageLabel.setIcon(imageArray[index]);
  }

  private void initImageArray() {
    String path = new String(imagePath + "javaPic");
    for (int i=0; i<imageArray.length; i++) {
      imageArray[i] = new ImageIcon(path + i + ".jpg");
    }
  }

  private void setPrevious() {
    if (index == 0) {
      index = imageArray.length - 1;
    }
    else
      index--;
  }

  private void setNext() {
    if (index == (imageArray.length-1)) {
      index = 0;
    }
    else
      index++;
  }

  protected static ImageIcon getImageIcon(String path) {
    // create an ImageIcon object if path is valid
    // else returns null
    java.net.URL imageUrl = ScrollPanes.class.getResource(path);
    if (imageUrl != null) {
      return new ImageIcon(imageUrl);
    }
    else {
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
      System.err.println("No image found at: " + path);
      return null;
    }
  }
}
```

Code 6.14: SplitPaneInternalFrame Class

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

public class SplitPaneInternalFrame extends JInternalFrame
                                    implements ItemListener {
  // labels
  private JLabel imageLabel;

  // paths
  private String imagePath    = "./images/";

  // radio buttons
  private JRadioButton eagle     = new JRadioButton("Eagle", true);
  private JRadioButton redGlobe  = new JRadioButton("Red Globe");
  private JRadioButton orchid    = new JRadioButton("Orchid");
  private JRadioButton blueGlobe = new JRadioButton("Blue Globe");
  private JRadioButton digital   = new JRadioButton("Digital");

  // create a radio button group
  ButtonGroup rbuttonGroup = new ButtonGroup();

  // panels
  private JPanel panel        = new JPanel();
  private JPanel panel1       = new JPanel();

  // images
  ImageIcon image;

  // array to hold images
  ImageIcon[] imageArray      = new ImageIcon[5];
  int         index           = 0;

  // scroll panes
  JScrollPane sp = new JScrollPane(imageLabel);

  // split panes
  JSplitPane splitPane;


  public SplitPaneInternalFrame(String title) {
    super(title,
          true, //resizable
          true, //closable
          true, //maximizable
          true);//iconifiable

    // create gui, initialize array
    initImageArray();

    // group radio buttons
    rbuttonGroup.add(eagle);
    rbuttonGroup.add(redGlobe);
    rbuttonGroup.add(orchid);
    rbuttonGroup.add(blueGlobe);
    rbuttonGroup.add(digital);

    // set layout
    panel1.setLayout(new FlowLayout());

    // create a scroll pane to display images
    image          = imageArray[index];
    imageLabel     = new JLabel(image);
```

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

```
    JScrollPane sp = new JScrollPane(imageLabel);

    // add Components to panel1
    panel1.add(eagle);
    panel1.add(redGlobe);
    panel1.add(orchid);
    panel1.add(blueGlobe);
    panel1.add(digital);

    // create a split pane with the two components in it.
    splitPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT,
                               panel1, sp);
    splitPane.setOneTouchExpandable(true);
    splitPane.setDividerLocation(30);

    // provide minimum sizes for the two components in the split pane
    Dimension minimumSize = new Dimension(400, 30);
    panel1.setMinimumSize(minimumSize);
    sp.setMinimumSize(minimumSize);

    // provide a preferred size for the split pane.
    splitPane.setPreferredSize(new Dimension(400,266));

    // add split pane into panel
    panel.add(splitPane);

    // add listeners
    eagle.addItemListener(this);
    redGlobe.addItemListener(this);
    orchid.addItemListener(this);
    blueGlobe.addItemListener(this);
    digital.addItemListener(this);

    // add panel into content pane
    this.getContentPane().add(panel);

    // set size of internal frame
    pack();
  }


  /** Event Handler when radio button is selected */
  public void itemStateChanged(ItemEvent e) {
    if (e.getSource() == eagle) {
        System.out.println("Eagle Selected");
        imageLabel.setIcon(imageArray[0]);
    }
    else
      if (e.getSource() == redGlobe) {
          System.out.println("redGlobe Selected");
          imageLabel.setIcon(imageArray[1]);
      }
      else
        if (e.getSource() == orchid) {
            System.out.println("orchid Selected");
            imageLabel.setIcon(imageArray[2]);
        }
        else
          if (e.getSource() == blueGlobe) {
            System.out.println("blueGlobe Selected");
            imageLabel.setIcon(imageArray[3]);
          }
          else
            if (e.getSource() == digital) {
              System.out.println("digital Selected");
              imageLabel.setIcon(imageArray[4]);
            }
  }

  private void initImageArray() {
    String path = new String(imagePath + "javaPic");
    for (int i=0; i<imageArray.length; i++) {
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
      imageArray[i] = new ImageIcon(path + i + ".jpg");
    }
  }

  protected static ImageIcon getImageIcon(String path) {
    // create an ImageIcon object if path is valid
    // else returns null
    java.net.URL imageUrl = ScrollPanes.class.getResource(path);
    if (imageUrl != null) {
      return new ImageIcon(imageUrl);
    }
    else {
     System.err.println("No image found at: " + path);
     return null;
    }
  }
}
```

When the above code is executed, listeners attached to the sub-applications behave in the same manner as they had been implemented earlier. For example, the following is an output on the command prompt when components of these applications are activated:

```
Head Selected
Speaker Selected
Love 85 Selected
Love 85 Clicked
name        = Danny Poo
designation = Author
gender      = Male
Yes 83.3 Selected
Yes 83.3 Clicked
Speaker Selected
Speaker Selected
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

CHAPTER 7: EDITING STYLED TEXTS IN EDITOR PANES

We introduced JTextArea in Chapter 5 as a component for entering and displaying multi-lines of *plain text* in a single area. JTextArea is limited since only plain texts are acceptable and we cannot add/manipulate *styled texts* in JTextArea. To achieve styled texts, we use *editor panes*, the subject of discussion in this chapter.

7.1 Limitations of Text Areas

Figure 7.1 shows a split pane with two sub-panes. The left sub-pane is a plain text area set with a single font type. The right sub-pane contains an image and some styled texts of two font types, namely Georgia and Impact. The left sub-pane is implemented as a JTextArea component and the right sub-pane is an editor pane (implemented by a JEditorPane component).

As a JTextArea, the left sub-pane can only accept texts of Arial font, size 14. Attempts to copy texts of other fonts from the right sub-pane to the left sub-pane result in Arial font, size 14 texts being pasted into the left sub-pane. This shows that JTextArea supports only one font type, no styled texts or images are allowed.

FIGURE 7.1: Text Area vs Editor Pane

7.2 Editor Panes

Editor panes are implemented in the Java Swing by JEditorPane. In addition to JEditorPane, styled texts are also supported by JTextPane. Both JEditorPane and JTextPane are editor panes.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

7.2.1 JEditorPane – Handling Styled Texts

Figure 7.2 is the class hierarchy for JEditorPane.

```
java.lang.Object
└── java.awt.Component
    └── java.awt.Container
        └── javax.swing.JComponent
            └── javax.swing.text.JTextComponent
                └── javax.swing.JEditorPane
```

FIGURE 7.2: Class Hierarchy of JEditorPane

7.2.1.1 What are the Constructors?

JEditorPane has four constructor methods (see Table 7.1).

TABLE 7.1: JEditorPane Constructors

No.	Constructor	Description
1	`JEditorPane()`	Creates a new empty JEditorPane object.
2	`JEditorPane (String url)`	Creates a JEditorPane object based on a string containing the URL specification.
3	`JEditorPane (String type, String text)`	Creates a JEditorPane object initialized with the given text.
4	`JEditorPane(URL initialPage)`	Creates a JEditorPane object with contents from a specified URL.

7.2.1.2 How are Events Handled?

A JEditorPane object does not generate any specific event object of its own. However, as a subclass of java.awt.Component, java.awt.Container, and javax.swing.JComponent, we may add listeners to listen to events that are typical of these classes on JEditorPane.

7.2.1.3 What are the Useful Methods?

JEditorPane, being a subclass of JTextComponent, inherits a number of useful methods from JTextComponent. There are some methods from JEditorPane class that you may want to pay attention to. They are highlighted in Table 7.2.

TABLE 7.2: Useful Methods

Method	Use
URL **getPage**()	Get the URL for the editor pane's current page being displayed.
void **read**(InputStream in, Object desc)	Initializes the component from a Reader.
void **setPage**(String url)	Load this component with the text at the specified url.
void **setPage**(URL page)	Initializes this component with the content from the URL page. The content type will be determined from the URL and the appropriate registered EditorKit will be set.
void **setText**(String t)	Set text of this component to the string t.
EditorKit **getEditorKit**()	Get the currently installed editor kit for handling contents.
void **setEditorKit**(Editor kit)	Sets the currently installed kit for handling contents.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

7.2.1.4 Show Me an Application of JEditorPane

The application we will illustrate is based on Figure 7.1. The code for Figure 7.1 is given in Code 7.1. The HTML file displayed in the editor pane (the right sub-pane) is given in Code 7.2.

Code 7.1: Text Area vs Editor Pane Test

```
import java.awt.*;
import javax.swing.*;
import java.io.IOException;
import java.net.URL;

public class TextAreaEditorPane extends JFrame {

  // text areas
  private JTextArea textArea  = new JTextArea();

  // panels
  private JPanel panel        = new JPanel();

  // editor panes
  private JEditorPane editPane = new JEditorPane();


  public TextAreaEditorPane() {

    // set layout
    panel.setLayout(new BorderLayout());

    // create text area and set it in a scroll pane
    textArea.setText("This is a JTextArea - a component" +
                     " for displaying" +
                     " and entering multi-lines of plain" +
                     " text in a single area."  +
                     " All text within this text area" +
                     " follows a single" +
                     " font type and size. Any text added" +
                     " will be converted and" +
                     " displayed uniformly in the same" +
                     " font type and size. " +
                     " Copy some texts from the right" +
                     " pane and paste them" +
                     " into this text area and you will" +
                     " know what I mean." +
                     " Any text you paste is now in plain" +
                     " Arial font, size 14. " +
                     " Texts in a text area can be set to" +
                     " BOLD but it is not done here." +
                     " You cannot add images in a text area" +
                     " but you can do so in an editor pane."
                    );
    textArea.setFont(new Font("Arial", Font.PLAIN, 16));
    textArea.setLineWrap(true);
    textArea.setWrapStyleWord(true);

    JScrollPane textAreaScrollPane = new JScrollPane(textArea);
    textAreaScrollPane.setVerticalScrollBarPolicy(
                        JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
    textAreaScrollPane.setPreferredSize(new Dimension(200, 330));

    // create an editor pane and set it in a scroll pane
    editPane  = createEditorPane();
    JScrollPane editScrollPane = new JScrollPane(editPane);
    editScrollPane.setVerticalScrollBarPolicy(
                     JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
    editScrollPane.setPreferredSize(new Dimension(290, 200));

    // set all components
    JPanel rightPanel = new JPanel(new GridLayout(1,0));
    JPanel leftPanel  = new JPanel(new BorderLayout());
    rightPanel.add(editScrollPane);
```

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

```
    leftPanel.add(textAreaScrollPane, BorderLayout.PAGE_START);

    panel.add(leftPanel, BorderLayout.LINE_START);
    panel.add(rightPanel, BorderLayout.LINE_END);
  }


  private JEditorPane createEditorPane() {
    JEditorPane editorPane = new JEditorPane();
    editorPane.setEditable(true);
    URL url = TextAreaEditorPane.class.getResource("edPane.html");
    if (url != null) {
      try {
        editorPane.setPage(url);
      } catch (IOException e) {
          System.err.println("Bad URL: " + url);
      }
    }
    else {
      System.err.println("File not found: edPane.html");
    }
    return editorPane;
  }


  private static void createFrameAndShow() {
    //Make sure we have nice window decorations.
    JFrame.setDefaultLookAndFeelDecorated(true);

    //Create and set up the window.
    TextAreaEditorPane frame = new TextAreaEditorPane();
    frame.setTitle("TextArea vs EditorPane Test");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    //Create and set up the content pane.
    JComponent contentPane = (JComponent)frame.getContentPane();
    contentPane.setOpaque(true); //content panes must be opaque
    frame.setContentPane(contentPane);
    contentPane.add(frame.panel);

    //Display the window.
    frame.pack();
    frame.setLocation(300, 100);
    frame.setSize(500, 360);
    frame.setVisible(true);
  }

  public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        createFrameAndShow();
      }
    });
  }
}
```

Code 7.2: edpane.html

```
<html>
<body>
<img src="images/LTPJ.jpg" width="104" height="64"><br>
<font face="Georgia" size=4>
An <b><i>editor pane</i></b> uses specialized editor kits to read, write, display, and edit
text of
different formats. The Swing text package includes editor kits for plain text, HTML, and RTF.
You can also develop custom editor kits for other formats.
The Java Swing version is <b>JEditorPane</b> - a component for editing various kinds of
contents. <b>The types of contents supported include: <i>text/plain</i>, <i>text/html</i>,
and <i>text/rtf</i></b>:
</font>
<font face="Impact" size=4>
<ul>
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
<li>text/plain: default type assumed if the type given is not recognised. </li>
<li>text/html: The editor kit used is the class javax.swing.text.html.HTMLEditorKit which
provides HTML 3.2 support. </li>
<li>text/rtf: The editor kit used is the class javax.swing.text.rtf.RTFEditorKit which
provides a limited support of the Rich Text Format. </li>
</ul>
</font>
<p>
</body>
</html>
```

Let us examine Code 7.1:

1. A JTextArea object is created.
2. A JPanel object is created.
3. A JEditorPane object is created.
4. When the test application is created, a JPanel object, created earlier is initialized with a BorderLayout layout manager.
5. The setText() of the JTextArea class is called to set the text for the text area component. The font type for the text area has been set to Arial, 16 point size.
6. The text in the JTextArea object is made scrollable.
7. The createEditorPane() method is called; it initializes the JEditorPane object, editPane, with contents from an HTML file, edPane.html.
8. editPane is made scrollable by setting it with a scroll pane.
9. Two JPanel objects, rightPanel and leftPanel, are created with their respective layout manager.
10. The text area and editor pane are added into these panels before the panels themselves are added into the main panel for display in the frame.

Execute this application and observe the behavior of JTextArea. Attempt to copy (by selecting and copying) some texts from the right sub-pane (the JEditorPane) and paste them into the JTextArea (the left sub-pane). Notice the text is converted into an Arial font even though the text is originally of a different font type. As mentioned, a JTextArea is a plain text area with a uniform font.

7.2.2 Providing Help with a JEditorPane

One common application of JEditorPane is in the provision of *help* information. Let us now revisit our previous example on radio stations. Figure 7.3 shows a help screen that provides some information about the various radio stations. We modify Code 7.1 to produce Code 7.3. The latter shows how help utilities can be provided.

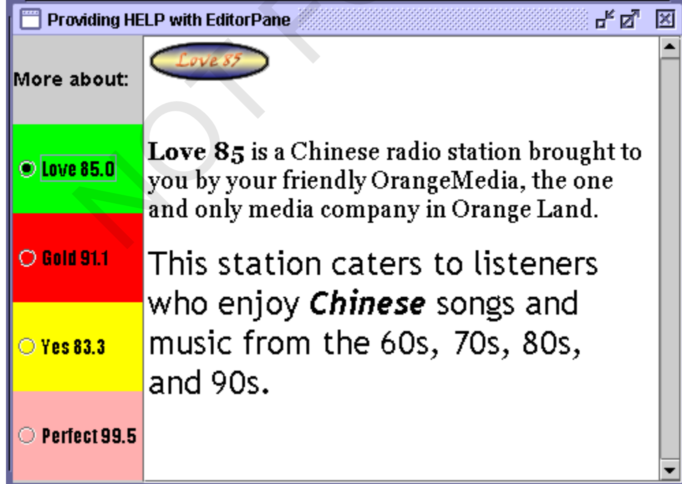


FIGURE 7.3: Providing Help with JEditorPane

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

Code 7.3: Providing Help with JEditorPane

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.IOException;
import java.net.URL;

public class EditorPane extends JFrame implements ItemListener{
  // panels
  private JPanel panel         = new JPanel();
  private JPanel rightPanel    = new JPanel(new GridLayout(1,0));
  private JPanel leftPanel     = new JPanel(new GridLayout(5,0));

  // editor panes
  private JEditorPane editPane = new JEditorPane();

  // create radio button objects
  JRadioButton aboutLove85     = new JRadioButton("Love 85.0", true);
  JRadioButton aboutGold911    = new JRadioButton("Gold 91.1");
  JRadioButton aboutYes833     = new JRadioButton("Yes 83.3");
  JRadioButton aboutPerfect995 = new JRadioButton("Perfect 99.5");

  // create a radio button group
  ButtonGroup rbuttonGroup = new ButtonGroup();

  // create label
  JLabel    label  = new JLabel("More about:");

  // create a scroll pane
  JScrollPane editScrollPane = new JScrollPane(editPane);


  public EditorPane() {

    // set layout
    panel.setLayout(new BorderLayout());

    // group radio buttons
    rbuttonGroup.add(aboutLove85);
    rbuttonGroup.add(aboutGold911);
    rbuttonGroup.add(aboutYes833);
    rbuttonGroup.add(aboutPerfect995);

    // creates font
    Font labelFont = new Font("Arial", Font.BOLD, 14);
    Font radioButtonFont = new Font("Impact", Font.PLAIN, 14);

    // set font and background colors
    label.setFont(labelFont);
    aboutLove85.setFont(radioButtonFont);
    aboutGold911.setFont(radioButtonFont);
    aboutYes833.setFont(radioButtonFont);
    aboutPerfect995.setFont(radioButtonFont);
    aboutLove85.setBackground(Color.green);
    aboutGold911.setBackground(Color.red);
    aboutYes833.setBackground(Color.yellow);
    aboutPerfect995.setBackground(Color.pink);

    // add listener
    aboutLove85.addItemListener(this);
    aboutGold911.addItemListener(this);
    aboutYes833.addItemListener(this);
    aboutPerfect995.addItemListener(this);

    // create an editor pane and set it in a scroll pane
    setEditorPane("love85.html");
    editScrollPane.setViewportView(editPane);
    editScrollPane.setVerticalScrollBarPolicy(
                 JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
    editScrollPane.setPreferredSize(new Dimension(395, 200));
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
    // set all components
    rightPanel.add(editScrollPane);
    leftPanel.add(label);
    leftPanel.add(aboutLove85);
    leftPanel.add(aboutGold911);
    leftPanel.add(aboutYes833);
    leftPanel.add(aboutPerfect995);

    panel.add(leftPanel, BorderLayout.LINE_START);
    panel.add(rightPanel, BorderLayout.LINE_END);
  }


  private void setEditorPane(String fileName) {
    editPane  = createEditorPane(fileName);
    editScrollPane.setViewportView(editPane);
  }


  private JEditorPane createEditorPane(String fileName) {
    JEditorPane editorPane = new JEditorPane();
    editorPane.setEditable(false);
    URL url = EditorPane.class.getResource(fileName);
    if (url != null) {
      try {
        editorPane.setPage(url);
      } catch (IOException e) {
          System.err.println("Bad URL: " + url);
      }
    }
    else {
      System.err.println("File not found: " + fileName);
    }
    return editorPane;
  }


  /** Event Handler when radio button is selected */
  public void itemStateChanged(ItemEvent e) {

    if (e.getSource() == aboutLove85)
      if (aboutLove85.isSelected()) {
        System.out.println("Love 85.0 Selected");
        setEditorPane("love85.html");
      }
      else { // do nothing
      }
    else
      if (e.getSource() == aboutGold911)
        if (aboutGold911.isSelected()) {
          System.out.println("Gold 91.1 Selected");
          setEditorPane("gold911.html");
        }
        else { // do nothing
        }
      else
        if (e.getSource() == aboutYes833)
          if (aboutYes833.isSelected()) {
            System.out.println("Yes 83.3 Selected");
            setEditorPane("yes833.html");
          }
          else { // do nothing
          }
        else
          if (aboutPerfect995.isSelected()) {
            System.out.println("Perfect 99.5 Selected");
            setEditorPane("perfect995.html");
          }
          else { // do nothing
          }
  }
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
  private static void createFrameAndShow() {
    //Make sure we have nice window decorations.
    JFrame.setDefaultLookAndFeelDecorated(true);

    //Create and set up the window.
    EditorPane frame = new EditorPane();
    frame.setTitle("Providing HELP with EditorPane");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    //Create and set up the content pane.
    JComponent contentPane = (JComponent)frame.getContentPane();
    contentPane.setOpaque(true); //content panes must be opaque
    frame.setContentPane(contentPane);
    contentPane.add(frame.panel);

    //Display the window.
    frame.pack();
    frame.setLocation(300, 100);
    frame.setSize(500, 360);
    frame.setVisible(true);
  }


  public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        createFrameAndShow();
      }
    });
  }
}
```

Let us examine Code 7.3:

1. A JEditorPane object, editPane, is created and initialized.
2. The editPane object is set with a call to setEditorPane() method. This method takes in a String parameter that denotes the HTML file to be displayed on the help screen. setViewportView() method of the JScrollPane class ensures that the correct editPane view is displayed each time it is updated in setEditorPane() method.
3. The editPane is set to be non-editable since help screen provides read-only information, the user is not permitted to make any update on the help screen.
4. Much of the code on radio buttons has been discussed before.
5. The frame implements the ItemListener interfaceso that it can respond to the events generated from the radio buttons.
6. The rest of the code had been discussed before and will not be elaborated here.

The HTML files describing the various radio stations are provided in the Appendix at the end of this book.

7.2.3 JTextPane – A More Specialized Editor Pane

A more specialized form of editor pane is a text pane. The Java Swing version of a text pane is JTextPane.

Figure 7.4 shows the class hierarchy for JTextPane. As is clear from the hierarchy, JTextPane is a subclass of JEditorPane. What is possible in JEditorPane is therefore possible in JTextPane: You may add texts, components as well as images into a JTextPane and manipulate them as in an editor pane. Technically, we can achieve what is displayed in Figure 7.3 using JTextPane.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
java.lang.Object
└──java.awt.Component
    └──java.awt.Container
        └──javax.swing.JComponent
            └── javax.swing.text.JTextComponent
                └── javax.swing.JEditorPane
                    └── javax.swing.JTextPane
```

FIGURE 7.4: Class Hierarchy of JTextPane

7.2.3.1 Why Use JTextPane?

Our discussion on JEditorPane so far has been on manipulating plain text, HTML files or RTF files. Formatting of texts has been achieved via the marked up tags of HTML or RTF. Therefore, to change the font of a component in a HTML document we have to use the   tag. Similarly, to italize a text, we have to use the <i></i> tag, etc. The limitation with this approach is obvious: we cannot change the style of texts at program run-time – an essential feature for text editing.

JTextPane supports text editing. **A JTextPane object is a text component with marked up attributes for changing the component's look and feel.** The attributes are represented as styles and added to a document associated with a JTextPane. A document with styles is known as a *styled document* (represented by the StyledDocument interface).

7.2.3.2 What are the Constructors?

JTextPane has two constructor methods (see Table 7.3).

TABLE 7.3: JTextPane Constructors

No.	Constructor	Description
1	`JTextPane()`	Creates a new empty JTextPane object.
2	`JTextPane (StyledDocument doc)`	Creates a new JTextPane object with a specified document model doc.

7.2.3.3 How are Events Handled?

A JTextPane object does not generate any specific event object of its own. However, as a subclass of java.awt.Component, java.awt.Container, and javax.swing.JComponent, we may add listeners to listen to events that are typical of these classes on JTextPane.

7.2.3.4 What are the Useful Methods?

JTextPane, being a subclass of JTextComponent, inherits a number of useful methods from JTextComponent. There are some methods from JTextPane class that you may want to pay attention to. They are highlighted in Table 7.4.

TABLE 7.4: Useful Methods

Method	Use
StyledDocument **getStyledDocument**()	Get the document model of the text pane.
void **setStyledDocument** (StyledDocument doc)	Set the text pane with document model doc.
StyledEditorKit **getStyledEditorKit**()	Get the currently installed editor kit for handling contents.
void **setEditorKit**(EditorKit kit)	Sets the currently installed editor kit for handling contents.

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

7.2.3.5 What are the Classes Associated to JTextPane?

To have a better understanding of JTextPane, we need to discuss the classes or interfaces closely associated with the behaviour of JTextPane.

A JTextPane component is associated with a styled document of the javax.swing.text.StyledDocument interface. **A styled document is a document with added styles.**
A *style* is a set of attributes associated with an element in a document. An *element* could be a character, string of characters, paragraph, or even an image. By associating the set of attributes to the elements, we change the look and feel of the elements. For example, we can change the font type, font size, text alignment, paragraph orientation, paragraph indentation, etc. of an element. A list of commonly used attributes is captured in the javax.swing.text.StyleConstants class.

Figure 7.5 shows the relationship of *styles*, *styled document* and *text pane*. Styles are added into a document known as a styled document which is then added into a text pane (JTextPane) for display.

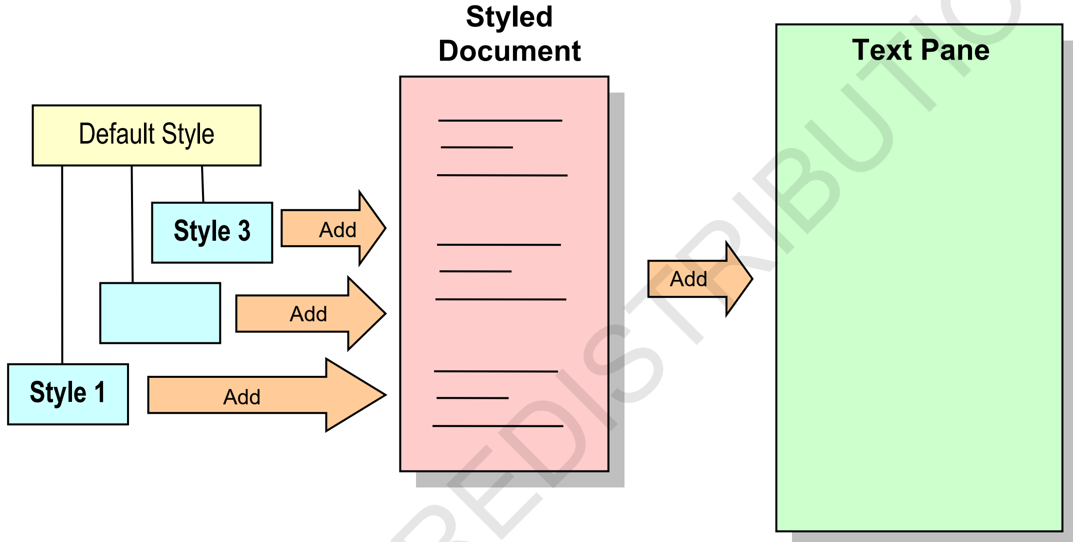


FIGURE 7.5: JTextPane Associated Classes

7.2.3.6 How to Process a JTextPane?

A JTextPane provides a *view* to a *model* (the styled document). Styled documents can be edited via an editor kit (the *controller* in the Model-View-Controller component design).

We begin the processing of JTextPane by getting the styled document of a JTextPane object. This is accomplished by the getStyledDocument() method of the JTextPane class:

```
JTextPane textPane = new JTextPane();
StyledDocument doc = textPane.getStyledDocument();
```

Next, we add styles to the document. To do this, we have to get the default style of the document via the javax.swing.text.StyleContext class first:

```
Style defaultStyle = StyleContext.getDefaultStyleContext().
                        getStyle(StyleContext.DEFAULT_STYLE);
```

Styles can then be added into the document (doc) via the addStyle() method of the StyledDocument interface; this interface is implemented by the javax.swing.text.DefaultStyledDocument class.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

The addStyle() method takes in two parameters (style name and parent style) and returns the style added. Styles added are organized into a logical style hierarchy. The default style may therefore serve as the root style of the hierarchy. Now you know why there is a parent style as a parameter in the addStyle() method.

The parent style associates styles in a hierarchical manner. For example, to add a new style, georgia, into the styled document doc, we write:

```
Style georgia = doc.addStyle("georgia", defaultStyle);
```

The parent style with which georgia style is connected to is defaultStyle.

The javax.swing.text.Style interface is a direct subinterface of javax.swing.text.MutableAttributeSet which in turn is a subinterface of javax.swing.text.AttributeSet. These interfaces are implemented by the javax.swing.text.SimpleAttributeSet class which is implemented by a hash table having a set of unique key-value pairs. The attributes of a style are therefore stored in a hash table managed by a MutableAttributeSet object.

The setting of the attributes is facilitated by the static methods of the StyleConstants class. For example, to set the font family of the georgia style to "Georgia", the setFontFamily() method is called:

```
StyleConstants.setFontFamily(georgia, "Georgia");
```

To set the size of the font to 18 points, the setFontSize() method is called:

```
StyleConstants.setFontSize(georgia, 18);
```

Different styles can be added into the same styled document. Let us now create a new style, boldGeorgia, and add it into the styled document:

```
Style s1 = doc.addStyle("boldGeorgia", georgia);
```

The parent style of boldGeorgia style has been set to georgia. Being a sibling of the georgia style, boldGeorgia style also takes on the attributes of the georgia style. In addition, boldGeorgia style may have additional attributes of its own, for example:

```
StyleConstants.setBold(s1, true);
```

A boldGeorgia style, therefore, sets an element with a bold Georgia font of 18 point size. Figure 7.6 shows the logical style hierarchy formed thus far.

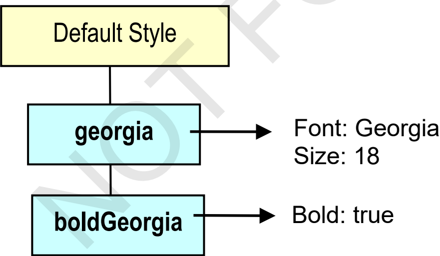


FIGURE 7.6: Logical Style Hierarchy

We are now ready to discuss a more complete application of JTextPane.

7.2.3.7 Show Me an Application of JTextPane

Figure 7.7 is an extract of the content in an editor pane from Figure 7.3. We will illustrate the use of JTextPane with this extract.

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

Love 85 is a Chinese radio station brought to you by your friendly OrangeMedia, the one and only media company in Orange Land.

This station caters to listeners who enjoy ***Chinese*** songs and music from the 60s, 70s, 80s, and 90s.

FIGURE 7.7: Styled Texts

Let us examine the characteristics of Figure 7.7:

1. There is an icon at the beginning of the frame, followed by two blank lines.
2. There are two paragraphs separated by a blank line.
3. Font of first paragraph is Georgia font, 18 points
4. “Love 85” is bold, Georgia font, 18 points
5. Font of second paragraph: Trebuchet MS font, 24 points
6. “Chinese” is italic, bold, Trebuchet MS font, 24 points

Code 7.4 produces Figure 7.7.

Code 7.4: TextPane Test

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

public class TextPane extends JFrame{
  // constants
  protected static final String newline = "\n";

  // panels
  private JPanel panel = new JPanel();

  // document initial texts
  String[] texts = {
    " " + newline,                           // icon
    " " + newline,                           // blank line, default
    " " + newline,                           // blank line, default
    "Love 85",                               // boldGeorgia
    " is a Chinese radio station brought" +  // georgia
    " to you by your friendly OrangeMedia," +
    " the one and only media company in" +
    " Orange Land." + newline,
    " " + newline,                           // blank line, default
    "This station caters to listeners" +     // trebuchetMS
    " who enjoy",
    " Chinese",                              // italicBoldTrebuchetMS
    " songs and music from the 60s, 70s," +  // trebuchetMS
    " 80s, and 90s."
  };

  // document initial text styles
  String[] styles = {
    "icon", "blank", "blank", "boldGeorgia", "georgia", "blank",
    "trebuchetMS", "italicBoldTrebuchetMS", "trebuchetMS"
  };
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
public TextPane() {
  // set layout
  panel.setLayout(new BorderLayout());

  // create a text pane
  JTextPane textPane = new JTextPane();

  // get the styled document associated with texPane
  StyledDocument doc = textPane.getStyledDocument();

  // add and set styles to styled document
  addStylesToStyledDocument(doc);
  setStyledDocument(doc);

  // set text pane in a scroll pane
  JScrollPane textPaneScrollPane = new JScrollPane(textPane);
  textPaneScrollPane.setVerticalScrollBarPolicy(
                JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
  textPaneScrollPane.setPreferredSize(new Dimension(250, 155));
  textPaneScrollPane.setMinimumSize(new Dimension(10, 10));

  // set components
  panel.add(textPaneScrollPane);
}

private void setStyledDocument(StyledDocument doc) {
  try {
    for (int i=0; i < texts.length; i++) {
      doc.insertString(doc.getLength(), texts[i],
                       doc.getStyle(styles[i]));
    }
  } catch (BadLocationException ble) {
      System.err.println("Error in inserting initial texts into text pane");
  }
}


protected void addStylesToStyledDocument(StyledDocument doc) {
  // get default style
  Style defaultStyle = StyleContext.getDefaultStyleContext().
                        getStyle(StyleContext.DEFAULT_STYLE);

  Style georgia = doc.addStyle("georgia", defaultStyle);
  StyleConstants.setFontFamily(georgia, "Georgia");
  StyleConstants.setFontSize(georgia, 18);

  Style s1 = doc.addStyle("boldGeorgia", georgia);
  StyleConstants.setBold(s1, true);

  Style trebuchetMS = doc.addStyle("trebuchetMS", defaultStyle);
  StyleConstants.setFontFamily(trebuchetMS, "Trebuchet MS");
  StyleConstants.setFontSize(trebuchetMS, 24);

  Style s2 = doc.addStyle("italicBoldTrebuchetMS", trebuchetMS);
  StyleConstants.setItalic(s2, true);
  StyleConstants.setBold(s2, true);

  Style s3 = doc.addStyle("icon", defaultStyle);
  ImageIcon love85Icon = getImageIcon("images/love85.gif");
  if (love85Icon != null) {
    StyleConstants.setIcon(s3, love85Icon);
  }

  Style s4 = doc.addStyle("blank", defaultStyle);
  StyleConstants.setFontSize(s4, 14);

}


protected static ImageIcon getImageIcon(String path) {
  // create an ImageIcon object if path is valid
  // else returns null
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
    java.net.URL imageUrl = ScrollPanes.class.getResource(path);
    if (imageUrl != null) {
      return new ImageIcon(imageUrl);
    }
    else {
     System.err.println("No image found at: " + path);
     return null;
    }
  }


  private static void createFrameAndShow() {
    //Make sure we have nice window decorations.
    JFrame.setDefaultLookAndFeelDecorated(true);

    //Create and set up the window.
    TextPane frame = new TextPane();
    frame.setTitle("TextPane Test");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    //Create and set up the content pane.
    JComponent contentPane = (JComponent)frame.getContentPane();
    contentPane.setOpaque(true); //content panes must be opaque
    frame.setContentPane(contentPane);
    contentPane.add(frame.panel);

    //Display the window.
    frame.pack();
    frame.setLocation(300, 100);
    frame.setSize(405, 315);
    frame.setVisible(true);
  }


  public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        createFrameAndShow();
      }
    });
  }
}
```

Let us examine Code 7.4:

1. We define two String arrays: texts and styles. These two arrays are used to set the styled document.
2. The processing begins with the creation of a JTextPane object, textPane.
3. We get the styled document associated with the textPane.
4. Styles are added to the styled document, doc.

Let us elaborate the addStylesToStyledDocument() method:

1. Gets the default style as discussed before.
2. A new style, georgia, is created and added into doc. The default style, defaultStyle, is the parent style of georgia style.
3. Two attributes of georgia style are set: font family and font size.
4. A new style, boldGeorgia, is created and added to doc. The parent style is set to georgia style. The bold attribute is set to true.
5. The third style, trebuchetMS, is created and added to doc. This time the parent style is set to defaultStyle. The font family and font size attribute of trebuchetMS style are set.
6. Create three more styles: italicBoldTrebuchetMS, icon, and blank, and them into doc.
7. Figure 7.8 shows the logical style hierarchy of this application.

With the styles created, the setStyledDocument() method initializes the styled document, doc, by setting the texts and styles into it. Let us elaborate this method:

1. The for loop iterates through the texts String array.
2. Each element of the array is inserted into the doc via the insertString() method.

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

3. As the iteration progresses, the element's corresponding style is applied to the element. For example, the boldGeorgia style (fourth element of styles) is applied to "Love 85" (fourth element of texts).
5. Set the textPane with a scroll pane. This provides scrolling facility for the textPane.
6. Finally, the scrollable textPane is added into the panel panel.

Execute Code 7.4 and you will get Figure7.9.

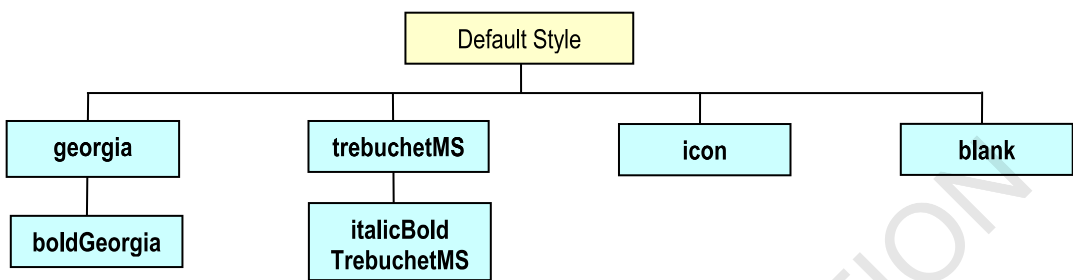


FIGURE 7.8: Style Hierarchy

FIGURE 7.9: TextPane

What we have shown above demonstrates how we can initialize a text pane. While this is sufficient for the application to display texts in different styles, it is admittedly very limited in utility.

For this text editing application to be useful, we should be able to change the style dynamically at run-time. How do we do this? We will discuss this when we introduce the concept of Menu in Chapter 9.

7.2.4 Differences between JEditorPane and JTextPane

JEditorPane and JTextPane are similar in that both are text components, but there are some differences that distinguish their use:

1. JTextPane is a subclass of JEditorPane and inherits the behavior of JEditorPane.
2. Texts can loaded into a JEditorPane or a JTextPane from a URL via the setPage() method. Texts can also be loaded from a URL via the constructors of a JEditorPane, but this is not possible with a JTextPane.
3. JEditorPane, by default, reads, writes, and edits plain, HTML and RTF texts. Although a JTextPane can do the same, there are limitations. The document model of a JTextPane must implement the StyledDocument interface. The styled document class for HTML and RTF texts is

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

implemented by HTMLDocument and RTFDocument respectively. The document model for plain texts is represented by the DefaultStyledDocument class.

4. A JTextPane provides support for named styles via its styled document and styled editor kit. A JEditorPane does not.
5. Images and components can be embedded in a JTextPane but this is not possible for JEditorPane. To add images and components into a JEditorPane, they have to be included in a HTML or RTF file.

7.2.5 Editor Kits

An *editor kit* provides the functionalities for reading and writing of documents into an editor pane or text pane. All text components use an EditorKit to provide themselves with a *view factory*, *document*, *caret* and *actions*. Even a text field or text area has an editor kit of its own but they are not visible to the users.

Editor panes and text panes have a getEditorKit() method to get the current editor kit and a setEditorKit() method to change it.

The Java Swing has three types of editor kit:

1. DefaultEditorKit. It reads and writes plain text, and provides a basic set of editing commands. The DefaultEditorKit is the superclass of the StyledEditorKit.
2. StyledEditorKit. It reads and writes styled text, and provides actions for styled texts. It is the default editor kit of JTextPane.
3. HTMLEditorKit. It reads, writes, and edits HTML. It is a subclass of StyledEditorKit.
4. RTFEditorKit. It reads, writes and edits RTF. It is a subclass of StyledEditorKit.

The above editor kits have been registered with the JEditorPane class. Depending on the format of the texts loaded into the JEditorPane, the pane checks the format of the file against its registered editor kits. If the registered kit is found to support the file format, the pane uses the kit to read, display, or edit the file. The pane effectively transforms itself into an editor kit for the file format.

Customized text format is supported in the Java Swing. You will need to register the customized editor kit via JEditorPane's registerEditorKitForContentType() method. This will associate the kit with the customized text format.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

CHAPTER 8: DIALOGS

The English language dictionary defines a *dialog* as a conversation between two or more people. In user interface design, a dialog is an interaction between a computer system and a human user. The interaction is accomplished with the display of a message in a dialog box by the computer system, and the human user responding to the message by selecting one of the available options in the dialog box.

Dialogs are manifested in various Java Swing classes:

1. Standard dialogs (JOptionPane)
2. Color choosing dialogs (JColorChooser)
3. File choosing dialogs (JFileChooser)
4. Progress monitoring dialogs (JProgressBar, ProgressMonitor, ProgressMonitorInputStream)
5. Document printing dialogs (PrinterJob)
6. Customized dialogs (JDialog)

We will only cover standard dialogs in this edition of the book.

8.1 Standard Dialogs

One way of displaying messages to users is to use labels (via JLabel, see Chapter 4). However, labels are limited as they are passive and lack provision for users to interact with the system.

There are situations when a more active approach to getting responses from users is required. We can use a dialog box to achieve this requirement.

The standard dialog box is produced via an *option pane* in Java. Figure 8.1 shows an example of an option pane implementing an *information-message dialog*. The user is expected to respond to the message by clicking the "OK" button.

FIGURE 8.1: An Information-Message Option Pane

Another example of an option pane is shown in Figure 8.2. Here, a question is asked and the user is expected to answer "Yes" or "No" by clicking on either of the options.

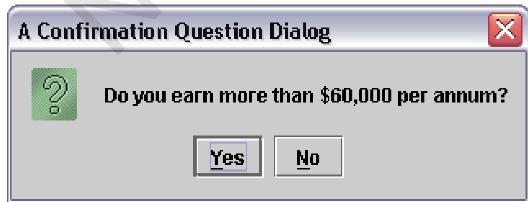


FIGURE 8.2: An Confirmation-Question Option Pane

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

8.1.1 JOptionPane – Providing Options

The Java Swing version of an option pane is JOptionPane. Figure 8.3 is the class hierarchy for JOptionPane.

```
java.lang.Object
└──java.awt.Component
    └──java.awt.Container
        └──javax.swing.JComponent
            └── javax.swing.JOptionPane
```

FIGURE 8.3: Class Hierarchy of JOptionPane

8.1.1.1 What are the Constructors?

JOptionPane has seven constructor methods (see Table 8.1).

TABLE 8.1: JOptionPane Constructors

No.	Constructor	Description
1	`JOptionPane()`	Creates a new JOptionPane object with a test message.
2	`JOptionPane (Object message)`	Creates a new JOptionPane object to display a message with the plain-message message type and the default options delivered by the user interface.
3	`JOptionPane(Object message, int messageType)`	Creates a new JOptionPane object to display a message with the specified message type and the default options delivered by the user interface.
4	`JOptionPane(Object message, int messageType, int optionType)`	Creates a new JOptionPane object to display a message with the specified message type and options.
5	`JOptionPane(Object message, int messageType, int optionType, Icon icon)`	Creates a new JOptionPane object to display a message with the specified message type, options, and icon.
6	`JOptionPane(Object message, int messageType, int optionType, Icon icon, Object[] options)`	Creates a new JOptionPane object to display a message with the specified message type, icon, and options.
7	`JOptionPane(Object message, int messageType, int optionType, Icon icon, Object[] options, Object initialValue)`	Creates a new JOptionPane object to display a message with the specified message type, icon, and options, with the initially-selected option specified.

8.1.1.2 How are Events Handled?

A JOptionPane object does not generate any specific event object of its own. However, as a subclass of java.awt.Component, java.awt.Container, and javax.swing.JComponent, we may add listeners to listen to events that are typical of these classes on JOptionPane.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

8.1.1.3 What are the Useful Methods?

A cursory browse of the Java APIs for JOptionPane suggests that JOptionPane has a large number of methods. However, most of the uses of JOptionPane involve calls to one of the static methods shown in Table 8.2.

TABLE 8.2: Useful Methods

Method	Use
`static int` **`showConfirmDialog`** `(Component parentComponent, Object message)`	Brings up a dialog asking a question and with options Yes, No, and Cancel. The accompanying title is Select an Option.
`static int` **`showConfirmDialog`** `(Component parentComponent, Object message, String title, int optionType)`	Brings up a dialog where the number of choices is determined by optionType.
`static int` **`showConfirmDialog`** `(Component parentComponent, Object message, String title, int optionType, int messageType)`	Brings up a dialog where the number of choices is determined by optionType and where the icon to display is determined by messageType.
`static int` **`showConfirmDialog`** `(Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon)`	Brings up a dialog with a specified icon, where the number of choices is determined by optionType.
`static String` **`showInputDialog`** `(Component parentComponent, Object message)`	Brings up a question-message dialog requesting input from the user via parentComponent.
`static String` **`showInputDialog`** `(Component parentComponent, Object message, Object initialSelectionValue)`	Brings up a question-message dialog requesting input from the user via parentComponent and with an initial selection value initialSelectionValue set.
`static String` **`showInputDialog`** `(Component parentComponent, Object message, String title, int messageType)`	Brings up a question-message dialog with a title title and message type messageType requesting input from the user via parentComponent.
`static Object` **`showInputDialog`** `(Component parentComponent, Object message, String title, int messageType, Icon icon, Object[] selectionValues, Object initialSelectionValue)`	Prompts the user for input in a blocking dialog where the initial selection, possible selections, and all other options can be specified.
`static String` **`showInputDialog`** `(Object message)`	Brings up a question-message dialog requesting input from the user.
`static String` **`showInputDialog`** `(Object message, Object initialSelectionValue)`	Brings up a question-message dialog requesting input from the user, with the input value initialized to initialSelectionValue.
`static void` **`showMessageDialog`** `(Component parentComponent, Object message)`	Brings up an information-message dialog titled "Message".
`static void` **`showMessageDialog`** `(Component parentComponent, Object message, String title, int messageType)`	Brings up a dialog that displays a message using a default icon determined by the messageType.
`static void` **`showMessageDialog`** `(Component parentComponent, Object message, String title, int messageType, Icon icon)`	Brings up a dialog that displays a message using a supplied icon icon for a message type messageType.
`static int` **`showOptionDialog`** `(Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon, Object[] options, Object initialValue)`	Brings up a dialog with a specified icon, where the initial choice is determined by initialValue and the number of choices is determined by optionType.

Methods in Table 8.2 can be summarized into four major method types as shown in Table 8.3.

TABLE 8.3: Method Types

Method Type	Description
`showConfirmDialog`	A dialog that asks a confirming question and expects user to answer by selecting one of yes/no/cancel option.
`showInputDialog`	A dialog that prompts for an input from user.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

showMessageDialog	A dialog that informs the user of something that has happened.
showOptionDialog	A dialog that is a unification of the above three types of dialogs.

8.1.1.4 Show Me an Application of JOptionPane

In this application, we show the use showMessageDialog() and showConfirmDialog() methods. All dialogs are tightly associated with a parent component. For this application, the parent component is shown in Figure 8.4 – the OptionPane class.

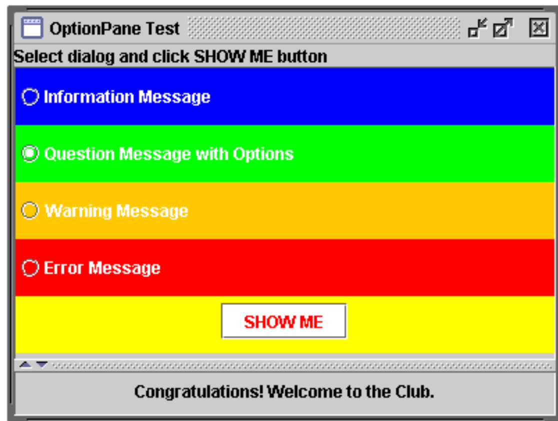


FIGURE 8.4: OptionPane Test

The Figure 8.4 frame shows four radio buttons. Each of these radio buttons selects a dialog box. The "Information Message" radio button selects an Information Message Dialog as shown in Figure 8.5.

NOT FOR REDISTRIBUTION

FIGURE 8.5: Information Message Dialog Box

A Warning Message Dialog is selected by clicking "Warning Message" and is shown in Figure 8.6.

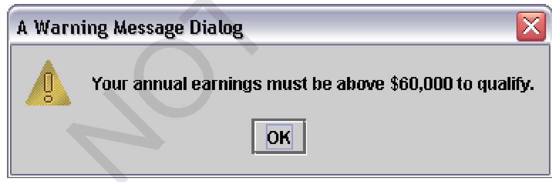


FIGURE 8.6: Warning Message Dialog Box

An Error Message Dialog is shown by selecting "Error Message" on the frame. This dialog box is shown in Figure 8.7.

Copyright © 2020, Danny Poo.

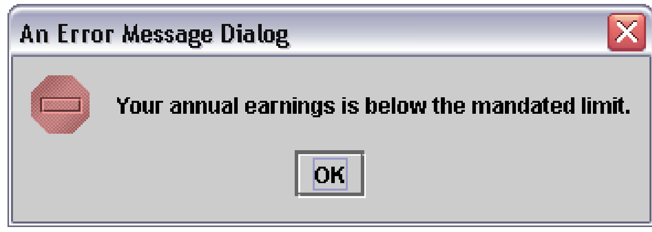


FIGURE 8.7: Error Message Dialog Box

The above three dialogs are also known generically as *Message Dialogs*. A *Message Dialog* displays a message accompanied by a button labeled "OK". A user responds to the message by clicking the "OK" button. The JOptionPane's method that produces a Message Dialog is JOptionPane.showMessageDialog().

Finally, we show an example of a Confirmation Question Dialog (or simply Confirmation Dialog) box in Figure 8.8 when the "Question Confirmation" radio button is selected. Note that this dialog box comes with options and the user has to select one of them. A *Confirmation Dialog* displays two option buttons: "Yes" and "No". It expects a user to confirm the message by selecting one of these two option buttons. The JOptionPane's method that produces an Confirmation Dialog is JOptionPane.showConfirmationDialog().

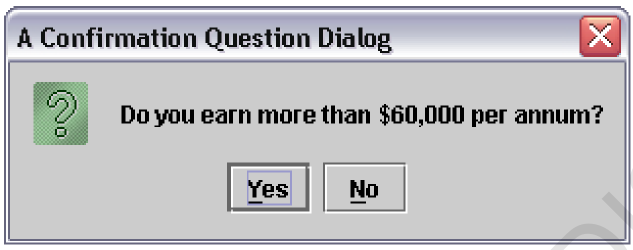


FIGURE 8.8: Confirmation Dialog Box

The code for producing the above application is given in Code 8.1.

Code 8.1: OptionPane Test

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

public class OptionPane extends JFrame implements ActionListener {

  // constants
  final static int numOfButtons   = 4;
  final static String informationAction = "information";
  final static String questionAction    = "question";
  final static String errorAction       = "error";
  final static String warningAction     = "warning";

  // message labels
  private JLabel instruction = new JLabel("Select dialog and click SHOW ME button");
  private JLabel message     = new JLabel();

  // buttons, radio buttons and radio button group
  private JRadioButton[] rButtons  = new JRadioButton[numOfButtons];
  private ButtonGroup rButtonGroup = new ButtonGroup();
  private JButton showMeButton     = new JButton("SHOW ME");

  // panels
  private JPanel panel        = new JPanel();
  private JPanel panel1       = new JPanel();
  private JPanel panel2       = new JPanel();
  private JPanel buttonPanel  = new JPanel();
```

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

```
// panes
JSplitPane splitPane;

public OptionPane() {

  // set layout
  panel.setLayout(new BorderLayout());
  panel1.setLayout(new GridLayout(5,1));

  // set buttons
  setRadioButtons();
  setShowMeButton();
  addRadioButtonsIntoPanel(); // add into panel1
  buttonPanel.add(showMeButton); // put button in a panel
  panel1.add(buttonPanel);
  panel2.add(message);
  createSplitPane();

  // add Components to panel
  panel.add(instruction, BorderLayout.NORTH);
  panel.add(splitPane, BorderLayout.CENTER);

  // add listeners
  showMeButton.addActionListener(this);
}


public void actionPerformed(ActionEvent e) {
  String action = rButtonGroup.getSelection().getActionCommand();

  // information dialog
  if (action.equals(informationAction)) {
    JOptionPane.showMessageDialog(this,
      "Enjoy free Golf Membership at the DP Country Club.",
      "An Information Message Dialog",
      JOptionPane.INFORMATION_MESSAGE
    );
    setMessage("Apply Now!");
  }
  else {// question dialog
    if (action.equals(questionAction)) {
      int j = JOptionPane.showConfirmDialog(
                    this, "Do you earn more than $60,000 per annum?",
                    "A Confirmation Question Dialog",
                    JOptionPane.YES_NO_OPTION);
      if (j == JOptionPane.YES_OPTION) {
        setMessage("Congratulations! Welcome to the Club.");
      }
      else {// No option
        setMessage("Sorry, you do not qualify.");

      }
    }
    else {// warning dialog
      if (action.equals(warningAction)) {
        JOptionPane.showMessageDialog(this,
          "Your annual earnings must be above $60,000 to qualify.",
          "A Warning Message Dialog",
          JOptionPane.WARNING_MESSAGE
        );
        setMessage("Try harder! Sorry.");
      }
      else {// error dialog
        JOptionPane.showMessageDialog(this,
          "Your annual earnings is below the mandated limit.",
          "An Error Message Dialog",
          JOptionPane.ERROR_MESSAGE
        );
        setMessage("Try again next time.");

      }
    }
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
    }
  }


  private void setRadioButtons() {

    rButtons[0] = new JRadioButton("Information Message");
    rButtons[0].setActionCommand(informationAction);
    rButtons[0].setForeground(Color.white);
    rButtons[0].setBackground(Color.blue);

    rButtons[1] = new JRadioButton("Question Message with Options");
    rButtons[1].setActionCommand(questionAction);
    rButtons[1].setForeground(Color.white);
    rButtons[1].setBackground(Color.green);

    rButtons[2] = new JRadioButton("Warning Message");
    rButtons[2].setActionCommand(warningAction);
    rButtons[2].setForeground(Color.white);
    rButtons[2].setBackground(Color.orange);

    rButtons[3] = new JRadioButton("Error Message");
    rButtons[3].setActionCommand(errorAction);
    rButtons[3].setForeground(Color.white);
    rButtons[3].setBackground(Color.red);

    for (int i = 0; i < numOfButtons; i++) {
      rButtonGroup.add(rButtons[i]);
    }
    rButtons[0].setSelected(true);
  }


  private void setShowMeButton() {
    showMeButton.setForeground(Color.red);
    showMeButton.setBackground(Color.white);
    buttonPanel.setBackground(Color.yellow);
  }


  private void addRadioButtonsIntoPanel() {
    for (int i = 0; i < numOfButtons; i++) {
      panel1.add(rButtons[i]);
    }
  }


  private void createSplitPane() {
    // create a split pane with the two components in it.
    splitPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT,
                               panel1, panel2);
    splitPane.setOneTouchExpandable(true);
    splitPane.setDividerLocation(210);

    // set minimum sizes for the two components in the split pane
    Dimension minimumSize = new Dimension(400, 30);
    panel1.setMinimumSize(minimumSize);
    panel2.setMinimumSize(minimumSize);

    // provide a preferred size for the split pane.
    splitPane.setPreferredSize(new Dimension(400,266));
  }


  private void setMessage(String text) {
    message.setText(text);
  }


  private static void createFrameAndShow() {
    //Make sure we have nice window decorations.
    JFrame.setDefaultLookAndFeelDecorated(true);
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
    //Create and set up the window.
    OptionPane frame = new OptionPane();
    frame.setTitle("OptionPane Test");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    //Create and set up the content pane.
    JComponent contentPane = (JComponent)frame.getContentPane();
    contentPane.setOpaque(true); //content panes must be opaque
    frame.setContentPane(contentPane);
    contentPane.add(frame.panel);

    //Display the window.
    frame.pack();
    frame.setLocation(300, 100);
    frame.setSize(400, 300);
    frame.setVisible(true);
  }

  public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        createFrameAndShow();
      }
    });
  }
}
```

Let us examine Code 8.1:

1. There are two parts in this application: the main frame for user to select the dialog and the selected message dialog box.
2. The main frame is structured into two sections: an instruction on the top and a split pane at the bottom of the frame.
3. The split pane is further divided into two sections: top section containing the four radio buttons and "Show Me" button, and bottom section containing a message.
4. Depending on which radio button is selected, a corresponding message dialog box is displayed.
5. We begin our discussion from the constructor method of OptionPane class since main() and createFrameAndShow() methods have been discussed before.
6. The setup of the buttons and split pane needs no further introduction as they have been discussed before in previous chapters.
7. The OptionPane frame is added to the "Show Me" button as an action listener. Any action on the "Show Me" triggers a call to the actionPerformed() method.
8. In the actionPerformed() method, we check the radio button group to determine which of the radio buttons has been selected by examining the value of action command via the getActionCommand() method.
9. Except for the question action, all other actions call the JOptionPane.showMessageDialog() method to generate the corresponding message dialog box. The question action opens a confirm dialog box with two options "Yes" and "No". JOptionPane.showConfirmDialog() returns an integer value (0 for YES, 1 for NO) that determines if the "Yes" or "No" option has been selected.
10. setMessage() displays a message on the message label of the split pane.

NOT FOR REDISTRIBUTION

8.1.2 More on JOptionPane

From the above example, we can see that the basic behavior of a JOptionPane is like a dialog. It displays a dialog box where a user can respond to it.

All dialogs created via JOptionPane are *modal* i.e. all user inputs to other available windows are blocked and the user has to respond to the dialog box. To get a non-modal dialog, we use the JDialog class to create one.

Technically, all dialogs are created via the JDialog class (a subclass of java.awt.Dialog). A JOptionPane is a container that can automatically create a JDialog object and adds itself to the content pane of the

Copyright © 2020, Danny Poo.

JDialog object. When a JOptionPane object is added to the content pane of the JDialog object, it becomes displayable. What this suggests is that a JDialog object behaves, like a frame, as a *top-level container*.

A dialog is usually created from a frame which behaves as the parent of the dialog. When the frame is iconified, deiconified, or destroyed, all the sibling dialogs are iconified, deiconified, or destroyed. Such behavior is provided automatically via the AWT (Advanced Window ToolKit).

8.1.3 Structure of an Option Pane Dialog Box

Let us examine the structure of an option pane dialog box. Figure 8.9 shows an annotated dialog.

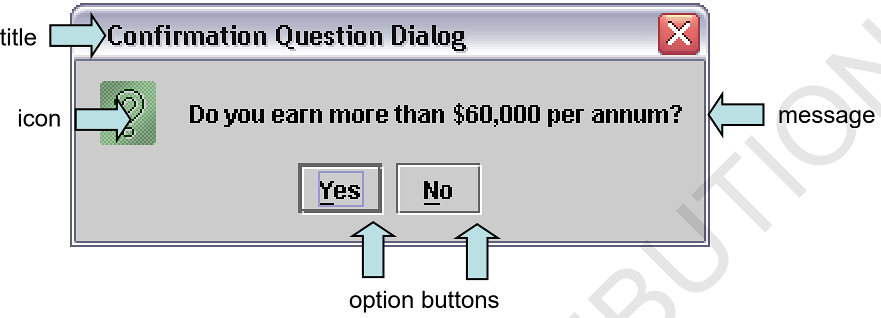


FIGURE 8.9: Confirmation Dialog Box

A typical option pane dialog box has the following elements: *title*, *message*, *option buttons* and *icon*. An additional element "*input value*" may be included as part of a dialog in cases where users are expected to input some values.

8.1.4 Forms of Option Pane Dialog

A JOptionPane can behave in one of four forms of dialog:

1. Message Dialog
2. Confirmation Dialog
3. Option Dialog
4. Input Dialog

We have earlier seen the first two forms of dialogs (Message and Confirmation Dialog) in Section 8.1.1.

An *Option Dialog* is very similar to a Confirmation Dialog. Instead of a single button, an Option Dialog displays more than one button. The button texts are not fixed and can be customized to suit the need. The JOptionPane's method that produces an Option Dialog is JOptionPane.showOptionDialog().

An *Input Dialog* displays a message requesting users to input some values via a text field or an uneditable combo box. The JOptionPane's method that produces an Input Dialog is JOptionPane.showInputDialog().

Note that all the above dialog types are modal i.e. all user inputs to other available windows are blocked and the user has to respond to the dialog box.

8.1.5 Parameters

The JOptionPane.showXxxDialog() method includes a number of parameters. It is through these parameters that variations of dialogs are available. Table 8.4 summarizes the parameters.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

TABLE 8.4: Parameters of JOptionPane.showXxxDialog() Method

Parameter	Description	Possible Values
`parentComponent`	Always the first argument. The parent component is the window from which the dialog originates. It must be a frame, a component inside a frame, or null. If frame is specified, the dialog will appear over the center of the frame. If component in a frame is specified, the dialog will appear over the center of the component. If null is specified, the look and feels manager will pick an appropriate position for the dialog.	a frame, a component in a frame or null
`message`	The element displayed in the *message* area of the dialog. If a string is specified, the dialog displays a label with the specified text. Alternatively, the message can be split into several lines using newline ("\n") characters inside the message string. For example, "Do you earn\n more than $60,000 per annum?"	String
`messageType`	It determines the icon to be displayed in the dialog. No icon will be displayed for PLAIN_MESSAGE type.	ERROR_MESSAGE, INFORMATION_MESSAGE, WARNING_MESSAGE, QUESTION_MESSAGE, PLAIN_MESSAGE
`optionType`	It specifies the set of buttons to appear at the *option buttons* area of the dialog.	DEFAULT_OPTION, YES_NO_OPTION, YES_NO_CANCLE_OPTION, OK_CANCEL_OPTION
`options`	An object array that specifies the string to be displayed by each button at the *option buttons* area of the dialog. It can also be used to specify the icon to be displayed by the buttons or non-button components to be added to the *option buttons* area.	
`icon`	It specifies the icon to be displayed in the *icon* area of the dialog.	
`title`	It specifies the title to be displayed in the *title* area of the dialog.	
`initialValue`	It specifies the default value to be selected (in an input dialog).	

8.1.6 Dialog, Frame and Internal Frame

In Chapter 6, we discussed the concept of internal frame as a frame within a frame. Internal frames are dependent on the main frame. When a main frame is iconified, deiconified, or closed, the internal frame is also iconified, deiconified, or closed.

A dialog can also be treated as an internal frame. To display a dialog as an internal frame, we use JOptionPane.showInternalXxxDialog() methods. For example, to display a message dialog as an internal frame, we use JOptionPane.showInternalMessageDialog() method instead of the JOptionPane.showMessageDialog() method.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

CHAPTER 9: MENUS

The example applications that we have shown so far do not have much user interactions. However, in any useful application, user interaction is essential. For example, an application may have multiple functions but users have to choose a particular function for the application to work on. The selection of the appropriate choice of function requires users to interact with the application. Choices of functions are usually displayed in the form of a *menu*, much like what you are presented with when you visit a restaurant and the waiter requires you to select from the menu the dishes you want to order.

Figure 9.1 shows an example of a menu entitled "File". This menu has a list of items "Open…", "Save as…", "Close", "Print" and "Exit".

The objective of this chapter is to teach you the concept of menu and to show you how to create menus in a Java software application. In particular, we will cover the following items:

1. Menu
2. Menu bar
3. Menu item
4. Submenu
5. Popup menu

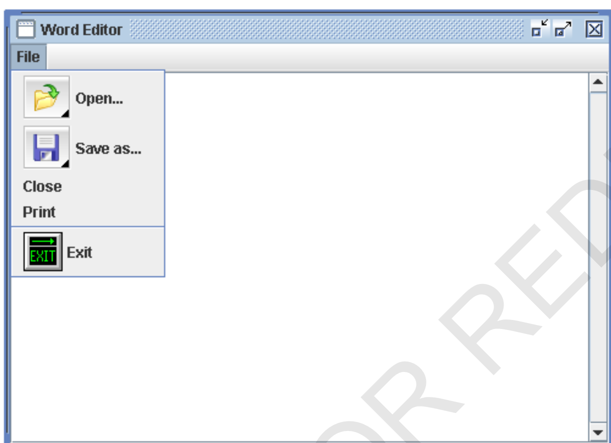


FIGURE 9.1: A Menu with a List of Menu Items in a Word Editor

NOT FOR REDISTRIBUTION

9.1 A Window with Menus

We will be developing a text editor, known here as the "Word Editor", incrementally throughout this chapter to illustrate the various concepts related to menus. The first version of our "Word Editor" has been presented in Figure 9.1.

Figure 9.2 is an annotated version of Figure 9.1. Let us take a closer look at the components:

1. *Menu item*. A menu item is an item placed within a menu. It has a text that describes what it does (e.g. open a file) and is implemented in the Java Swing by the JMenuItem class. A menu item can also be specified by a gif or image icon.
2. *Menu*. A menu is a button that is set on a list of menu items. It appears in a menu bar or as a popup menu. When a menu is selected at the menu bar, a popup window of menu items appears. Menu items within a menu can be divided using separators. A menu is implemented in the Java Swing by the JMenu class.
3. *Menu bar*. A menu bar is an object with which a menu is associated. A number of menus can be added into a menu bar. When a menu is selected at the menu bar, the menu becomes the top-level

Copyright © 2020, Danny Poo.

window for selecting menu items. A menu bar is implemented in the Java Swing by the JMenuBar class.

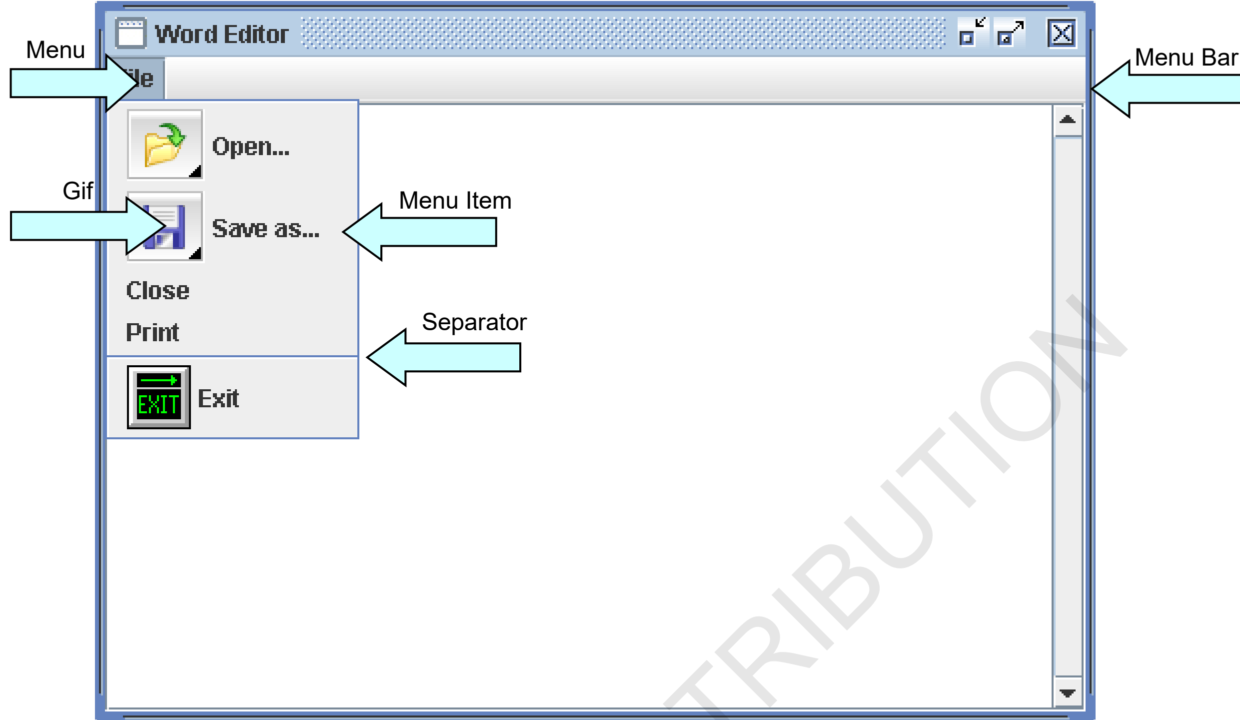


FIGURE 9.2: An Annotated "Word Editor"

9.1.1 JMenuItem

The Java Swing version of a menu item is JMenuItem. Figure 9.3 is the class hierarchy for JMenuItem. Note that a JMenuItem object is essentially a button (as it extends from javax.swing.AbstractButton) added into a list (the menu). This means that a JMenuItem object can respond to events, as a button would, when selected.

```
java.lang.Object
└── java.awt.Component
     └── java.awt.Container
          └── javax.swing.JComponent
               └── javax.swing.AbstractButton
                    └── javax.swing.JMenuItem
```

FIGURE 9.3: Class Hierarchy of JMenuItem

9.1.1.1 What are the Constructors?

JMenuItem has six constructor methods (see Table 9.1).

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

TABLE 9.1: JMenuItem Constructors

No.	Constructor	Description
1	`JMenuItem()`	Creates a new JMenuItem object with no set text or icon.
2	`JMenuItem(Action a)`	Creates a new JMenuItem object whose properties are taken from Action a.
3	`JMenuItem(Icon i)`	Creates a new JMenuItem object with the specified Icon i.
4	`JMenuItem(String text)`	Creates a new JMenuItem object with the specified String text.
5	`JMenuItem(String text, Icon i)`	Creates a new JMenuItem object with the specified text and icon.
6	`JMenuItem(String text, int mnemonic)`	Creates a new JMenuItem object with the specified text and keyboard mnemonic.

9.1.1.2 Show Me an Application of JMenuItem

We will show an application of JMenuItem after we have discussed JMenu and JMenuBar.

9.1.2 JMenu

The Java Swing version of a menu is JMenu. Figure 9.4 is the class hierarchy for JMenu. It is a direct subclass of JMenuItem; this suggests that a JMenu object is by itself a menu item and behaves like a button too. When the JMenu object (or button) is selected (or "pressed"), a popup menu[9] that displays a list of menu items appears.

```
java.lang.Object
└──java.awt.Component
     └──java.awt.Container
          └──javax.swing.JComponent
               └── javax.swing.AbstractButton
                    └── javax.swing.JMenuItem
                         └── javax.swing.JMenu
```

FIGURE 9.4: Class Hierarchy of JMenu

If the JMenu object is selected from the menu bar, the object is known as the top-level menu object. However, if it is selected from another menu item, then the popup menu appears as a "pull-right" menu or submenu.

9.1.2.1 What are the Constructors?

JMenu has four constructor methods (see Table 9.2).

TABLE 9.2: JMenu Constructors

No.	Constructor	Description
1	`JMenu ()`	Creates a new JMenu object.
2	`JMenu (Action a)`	Creates a new JMenu object whose properties are taken from Action a.
3	`JMenu (String text)`	Creates a new JMenu object with the specified String text.
4	`JMenu (String text, boolean b)`	Creates a new JMenu object with the specified String text and a boolean b indicating if the object is a tear-off menu or not.

[9] Implemented by JPopupMenu – to be discussed later in this chapter.

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

9.1.2.2 Show Me an Application of JMenu

We will show an application of JMenu after we have discussed JMenuBar.

9.1.3 JMenuBar

The Java Swing version of a menu bar is JMenuBar. Figure 9.5 is the class hierarchy for JMenuBar. It is a direct subclass of JComponent.

```
java.lang.Object
└──java.awt.Component
    └──java.awt.Container
        └──javax.swing.JComponent
            └── javax.swing.JMenuBar
```

FIGURE 9.5: Class Hierarchy of JMenuBar

9.1.3.1 What are the Constructors?

JMenuBar has one constructor method (see Table 9.3).

TABLE 9.3: JMenuBar Constructors

No.	Constructor	Description
1	`JMenuBar()`	Creates a new JMenuBar object.

9.1.3.2 Show Me an Application of JMenuBar

We will show an application of JMenuBar, JMenu and JMenuItem within the development of the "Word Editor" application in the next section.

9.2 Developing the Word Editor

The "Word Editor" is a text editing application. Version 1 of this application has five functions:

1. OPEN a file (for editing)
2. SAVEAS a file (write the file onto disk, replacing the original file)
3. CLOSE a file (and leaving the original file as it is)
4. PRINT a file (write the content of the file to screen)
5. EXIT (from the Word Editor application)

The graphical user interface of this application has earlier been shown in Figure 9.2.

9.2.1 WordEditor1 – Adding a Menu with Menu Items

The code for "Word Editor" Version 1 is shown in Code 9.1.

Code 9.1: Word Editor Version 1

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

public class WordEditor1 extends WordEditor {

  public WordEditor1() {

    // create and set menu
    setMenu();
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
    // create a panel and set layout
    panel.setLayout(new BorderLayout());

    // create text area and set it in a scroll pane
    textArea.setText("");
    textArea.setFont(new Font("Courier New",
                              Font.TRUETYPE_FONT, 12));
    textArea.setLineWrap(true);
    textArea.setWrapStyleWord(true);
    JScrollPane textAreaScrollPane = new JScrollPane(textArea);
    textAreaScrollPane.setVerticalScrollBarPolicy(
                    JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);

    // add text area into panel
    panel.add(textAreaScrollPane);
  }

  private void setMenu() {
    // add File Menu into menuBar
    menuBar.add(menu = new JMenu("File"));
    menuItem = new JMenuItem("Open...",
                         new ImageIcon("gif/open.gif"));
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menuItem = new JMenuItem("Save as...",
                         new ImageIcon("gif/save.gif"));
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menuItem = new JMenuItem("Close");
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menuItem = new JMenuItem("Print");
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menu.addSeparator();
    menuItem = new JMenuItem("Exit",
                         new ImageIcon("gif/exit.gif"));
    menuItem.addActionListener(this);
    menu.add(menuItem);

    // set menu bar and window listener
    setJMenuBar(menuBar);
    addWindowListener(this);
  }

  private static void createFrameAndShow() {
    //Make sure we have nice window decorations.
    JFrame.setDefaultLookAndFeelDecorated(true);

    //Create and set up the window.
    WordEditor1 frame = new WordEditor1();
    frame.setTitle("Word Editor");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    //Create and set up the content pane.
    JComponent contentPane = (JComponent)frame.getContentPane();
    contentPane.setOpaque(true); //content panes must be opaque
    frame.setContentPane(contentPane);
    contentPane.add(frame.panel);

    //Display the window.
    frame.pack();
    frame.setLocation(300, 100);
    frame.setSize(500, 360);
    frame.setVisible(true);
  }

  public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        createFrameAndShow();
```

NUS: FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
      }
    });
  }
}
```

Let us examine Code 9.1:

1. As usual, we create WordEditor1 as a frame so that it becomes displayable as a window.
2. The setMenu() method sets the menu and menu items onto the menu bar. It is called and detailed.
3. Creates a JMenu object and labeled it as "File". The JMenu object is subsequently added into menuBar, a JMenuBar object.
4. Creates a menu item specified with text "Open…" and a gif as an image icon for opening a file. The gif is stored separately in a "gif" folder. Adds the WordEditor1 object (which is the frame you see) as an action listener of this menu item. This menu item is then added to the JMenu object.
5. Continue with the addition of more menu items to the JMenu object. An action listener is added to each menu item that has to respond to user's clicks/selections. The action listener is WordEditor1 (the frame itself). As before, the menu items created are added to the JMenu object.
6. A separator (a horizontal line) to separate the "Exit" menu item from the rest of the menu items is added.
7. "…" have been deliberately added to "Open" and "Save as" menu items to indicate these menu items have accompanying dialogs for user to act on. Figure 9.6 shows the open file dialog after the "Open…" command is selected. Figure 9.7 shows the outcome of this action.
8. Sets the menu bar of the WordEditor1 frame to menuBar, a JMenuBar object.
9. Finally, we add the WordEditor1 frame as the window listener to receive window events (e.g. minimize, maximize, and close) on the frame. As a window listener, WordEditor1 has to implement methods of a java.awt.event.WindowListener interface; they include windowClosing(), windowActivated(), windowClosed(), windowDeactivated(), windowDeiconified(), windowIconified(), and windowOpened().
10. We create a JTextArea object to contain the texts of the file opened for editing. The JTextArea object is included in a JPanel object using a BorderLayout layout manager.

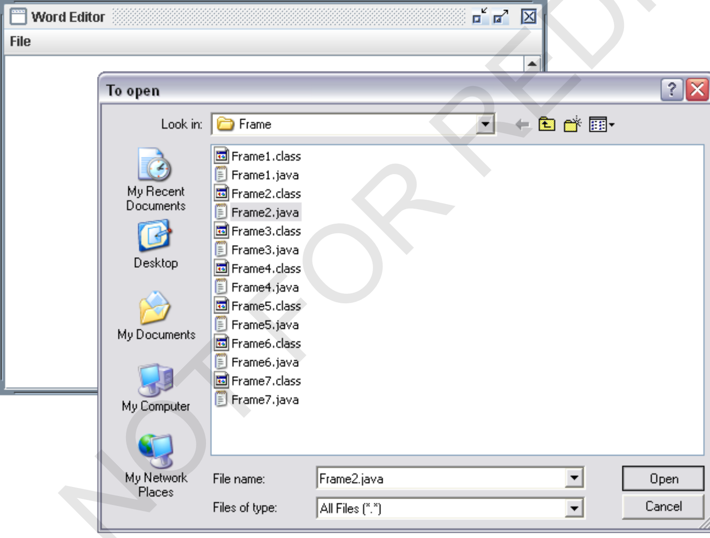


FIGURE 9.6: Open a File Command

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

Editing: Frame2.java

File

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

class Frame2 {

  public static void main(String argv[]) {
    JFrame frame = new JFrame("Java User Interface Window");
    frame.setSize(400, 300);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setLocation(300, 100);

    // add components
    JButton button = new JButton("OK");
    frame.getContentPane().add(button);

    // show frame
    frame.setVisible(true);
  }
}
```

FIGURE 9.7: Outcome of Open “Frame2.java” File

9.2.2 WordEditor Class

WordEditor1 extends WordEditor in Code 9.1. The code for the WordEditor class is given in Code 9.2.

Code 9.2: WordEditor Class

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.io.*;
import java.util.StringTokenizer;


public class WordEditor extends JFrame
             implements WindowListener, ActionListener,
                        ItemListener {

  // menu and its components
  protected JMenuBar  menuBar   = new JMenuBar();
  protected JMenu     menu;
  protected JMenuItem menuItem;

  // text areas
  protected JTextArea textArea  = new JTextArea();

  // panels
  protected JPanel panel        = new JPanel();


  public WordEditor() {}

  public void actionPerformed(ActionEvent ae) {
    String command = formClassName(ae.getActionCommand());
    try {
      Class c = Class.forName(command);
      ApplnWindow w = (ApplnWindow) c.newInstance();
      w.execute(this);
    } catch (InstantiationException i) {
        System.out.println("Instance not created");
    } catch (ClassNotFoundException c) {
        System.out.println("Class not found");
    } catch (IllegalAccessException i) {
        System.out.println("Illegal access");
    }
  }

  private String extract(String s) {
    int i = s.indexOf("text=");
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
    int j = s.indexOf("]", i);
    String command = s.substring(i+5, j);
    return command;
  }

  public String formClassName(String s) {
    String command = "";
    StringTokenizer t = new StringTokenizer(s, ". ");
    while (t.hasMoreTokens())
      command = command+t.nextToken();
    return command;
  }

  public void itemStateChanged(ItemEvent ie) {
    String command = new String();
    command = formClassName(extract(ie.getItem().toString()));
    if (command.equals("DocumentNOTeditable")) {
      if (ie.getStateChange() == ItemEvent.DESELECTED) {
        command = formClassName("DocumentEditable");
      }
    }
    try {
      Class c = Class.forName(command);
      ApplnWindow w = (ApplnWindow) c.newInstance();
      w.execute(this);
    } catch (InstantiationException i) {
        System.out.println("Instance not created");
    } catch (ClassNotFoundException c) {
        System.out.println("Class not found");
    } catch (IllegalAccessException i) {
        System.out.println("Illegal access");
    }
  }

  public void windowClosing(WindowEvent we) {
    System.exit(0); // exit from system
  }

  public void windowActivated(WindowEvent we)   { }
  public void windowClosed(WindowEvent we)      { }
  public void windowDeactivated(WindowEvent we) { }
  public void windowDeiconified(WindowEvent we) { }
  public void windowIconified(WindowEvent we)   { }
  public void windowOpened(WindowEvent we)      { }
}
```

WordEditor class has been designed as a superclass of all other versions of Word Editor (such as WordEditor1, WordEditor2, WordEditor3, etc.) that we will demonstrate in this chapter. Let us discuss Code 9.2:

1. WordEditor class extends from JFrame suggesting that all subclasses of WordEditor are frames and their instances are therefore displayable as windows.
2. Events on the frame are handled by having WordEditor implements WindowListener, ActionListener, and ItemListener. From Section 3.2.4, we note that a WindowListener listens to events on a window. An ActionListener listens to events on a button and an ItemListener listens to events on radio buttons, checkboxes, or comboboxes. WindowListener and ActionListener are applicable to WordEditor1. We will return to discussing ItemListener later in this chapter.
3. Declare the necessary components for constructing our menu system on the word editor.
4. The actionPerformed() method is the method we must implement for ActionListener interface. This method is invoked when a button is clicked. Since a menu item is basically a button, selecting a menu item triggers the execution of actionPerformed(). In this method, getActionCommand() returns the menu item text e.g. "Open…" for opening a file, or "Save as…" for saving an updated file, etc. The menu item text is used by formClassName() to create a string (known as command) for forming a class name e.g. "Open…" becomes Open, and "Save as…" becomes Saveas, and "Print" becomes Print, etc. Turns the command into an executable Java class e.g. command Open becomes Open.class, Saveas becomes Saveas.class, Exit becomes Exit.class, etc. Creates an instance of the class formed. Note that the class is cast into an ApplnWindow object, a superclass of all command classes. We will discuss the ApplnWindow

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

class in the next section. Finally, makes use of *polymorphism*[10] to invoke the method execute() of the class concerned. The polymorphic classes are declared as subclasses of ApplnWindow class. We will discuss these subclasses in the next section.

5. There are two other methods in this class that we will discuss later when we deal with radio buttons and checkboxes as menu items: extract() and itemStateChanged().
6. To implement the WindowListener interface, WordEditor must provide the implementation of seven methods as shown: windowClosing(), windowActivated(), windowClosed(), windowDeactivated(), windowDeiconified(), windowIconified(), and windowOpened(). Except for windowClosing(), all methods are left blank. You may add some System.out.println() statements in these methods to visualize the behavior of event handling in windows. We have added System.exit(0) to close the window and cause the WordEditor application to terminate execution.

9.2.3 Applying Polymorphism

The ApplnWindow class was introduced in Code 9.2. It is elaborated in Code 9.3.

Code 9.3: ApplnWindow Class

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


abstract class ApplnWindow extends JFrame {
  ApplnWindow() {}
  ApplnWindow(String s) {super(s);}
  abstract void execute(WordEditor parentFrame);
}

class Open extends ApplnWindow {
  Open() {}
  public void execute(WordEditor parentFrame) {
    FileDialog f = new FileDialog(
                       parentFrame, "To open", FileDialog.LOAD);
    f.pack();
    f.show();
    if (f.getFile() == null)
      return; // no file selected
    String name = f.getDirectory() + f.getFile(); // full pathname
    String fileName = f.getFile(); // only filename
    try {
      FileInputStream fs = new FileInputStream(name);
      BufferedReader  d = new BufferedReader(
                            new InputStreamReader(fs));
      StringBuffer b = new StringBuffer();
      String ln;
      while ((ln = d.readLine()) != null) {
        b.append(ln);
        b.append("\n");
      }
      fs.close();
      parentFrame.setTitle("Editing: " + fileName);
      parentFrame.textArea.setText(b.toString());
    } catch (Exception e) {
        parentFrame.textArea.setText("Error in reading stream");
    }
  }
}

class Saveas extends ApplnWindow {
  Saveas() {}
  public void execute(WordEditor parentFrame) {
    FileDialog f = new FileDialog(
                       parentFrame, "Save as", FileDialog.SAVE);
```

[10] Polymorphism is the ability of objects of different class definition to respond to the same message.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
    f.pack();
    f.show();
    if (f.getFile() == null)
      return; // no file selected
    String name = f.getDirectory()+f.getFile();
    if (name.endsWith(".*.*"))
      name = name.substring(0, name.length()-5);
    try {
      FileOutputStream s = new FileOutputStream(name);
      DataOutputStream d = new DataOutputStream(s);
      d.writeBytes(parentFrame.textArea.getText());
      d.close();
      s.close();
    } catch (Exception e) {
        System.err.println("Error in writing to "+name);
    }
  }
}

class Close extends ApplnWindow {
  Close() {}
  public void execute(WordEditor parentFrame) {
    parentFrame.textArea.setText("");
    parentFrame.setTitle("Word Editor");
  }
}

class Print extends ApplnWindow {
  Print() {}
  public void execute(WordEditor parentFrame) {
    System.out.println(parentFrame.textArea.getText());
  }
}

class Exit extends ApplnWindow {
  Exit() {}
  public void execute(WordEditor parentFrame) {
    System.exit(0);
  }
}

class Cut extends ApplnWindow {
  Cut() {}
  public void execute(WordEditor parentFrame) {
    parentFrame.textArea.cut();
  }
}

class Copy extends ApplnWindow {
  Copy() {}
  public void execute(WordEditor parentFrame) {
    parentFrame.textArea.copy();
  }
}

class Paste extends ApplnWindow {
  Paste() {}
  public void execute(WordEditor parentFrame) {
    parentFrame.textArea.paste();
  }
}

class Clearall extends ApplnWindow {
  Clearall() {}
  public void execute(WordEditor parentFrame) {
    if (parentFrame.textArea.isEditable()) {
      parentFrame.textArea.setText("");
    }
  }
}

class Selectall extends ApplnWindow {
```

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

```
  Selectall() {}
  public void execute(WordEditor parentFrame) {
    if (parentFrame.textArea.isEditable()) {
      parentFrame.textArea.selectAll();
    }
  }
}

class ToUpperCase extends ApplnWindow {
  ToUpperCase() {}
  public void execute(WordEditor parentFrame) {
    if (parentFrame.textArea.isEditable()) {
      String s = parentFrame.textArea.getSelectedText();
      if (s != null) {
        s = s.toUpperCase();
        parentFrame.textArea.replaceSelection(s);
      }
    }
  }
}

class ToLowerCase extends ApplnWindow {
  ToLowerCase() {}
  public void execute(WordEditor parentFrame) {
    if (parentFrame.textArea.isEditable()) {
      String s = parentFrame.textArea.getSelectedText();
      if (s != null) {
        s = s.toLowerCase();
        parentFrame.textArea.replaceSelection(s);
      }
    }
  }
}

class CourierNew16 extends ApplnWindow {
  CourierNew16() {}
  public void execute(WordEditor parentFrame) {
    if (parentFrame.textArea.isEditable()) {
      Font font = new Font("Courier New", Font.PLAIN, 16);
      parentFrame.textArea.setFont(font);
    }
  }
}

class LucidaSansTypewriter18 extends ApplnWindow {
  LucidaSansTypewriter18() {}
  public void execute(WordEditor parentFrame) {
    if (parentFrame.textArea.isEditable()) {
      Font font = new Font("Lucida Sans Typewriter", Font.PLAIN, 18);
      parentFrame.textArea.setFont(font);
    }
  }
}

class Verdana18 extends ApplnWindow {
  Verdana18() {}
  public void execute(WordEditor parentFrame) {
    if (parentFrame.textArea.isEditable()) {
      Font font = new Font("Verdana", Font.PLAIN, 18);
      parentFrame.textArea.setFont(font);
    }
  }
}

class DocumentEditable extends ApplnWindow {
  DocumentEditable() {}
  public void execute(WordEditor parentFrame) {
    parentFrame.textArea.setEditable(true);
  }
}

class DocumentNOTeditable extends ApplnWindow {
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
  DocumentNOTeditable() {}
  public void execute(WordEditor parentFrame) {
    parentFrame.textArea.setEditable(false);
  }
}

class Bluebackground extends ApplnWindow {
  Bluebackground() {}
  public void execute(WordEditor parentFrame) {
    parentFrame.textArea.setForeground(Color.WHITE);
    parentFrame.textArea.setBackground(Color.BLUE);
  }
}

class Whitebackground extends ApplnWindow {
  Whitebackground() {}
  public void execute(WordEditor parentFrame) {
    parentFrame.textArea.setForeground(Color.BLACK);
    parentFrame.textArea.setBackground(Color.WHITE);
  }
}
```

Let us examine Code 9.3:

1. ApplnWindow class extends JFrame. All the subclasses of ApplnWindow are therefore frames and their instances are displayable as windows.
2. The ApplnWindow class has been declared as an abstract class[11] to prevent the instantiation of objects from this class. Since the subclasses of ApplnWindow class are to provide the implementation of the execute() method, the latter has been declared as abstract too.
3. "Open", "Saveas", "Close", "Print", and "Exit" commands have been turned into a class with class name such as Open.class, Saveas.class, Close.class, Print.class, and Exit.class respectively. These classes have been conveniently placed within the same Java file, ApplnWindow.java. Note that these subclasses extend ApplnWindow.
4. Each of the subclasses implements its own execute() method and it is the subclass' execute() method that is executed when the object's execute() method is invoked (see Code 9.2). We will not go into the details of each of these execute() methods and will leave the analysis of these methods as an exercise to the reader.

In the design of the Word Editor, we have applied the concept of polymorphism[12] in the calling of the execute() method. In Code 9.2, all ApplnWindow objects are called with the same message – execute() – but each has its own implementation of the execute() method. As is clear from Code 9.3, the implementation of this method is different for each command.

The application of polymorphism has significantly increased the extendibility of the Word Editor. As will be clear later in our discussion, the design of the WordEditor can be easily extended with new commands without incurring much cost due to the expansion of functionalities in the Word Editor.

9.3 Menu Keyboard Shortcuts

Take a look at Figure 9.8 and you will notice mnemonics (or keyboard shortcuts) have been added to some of the commands we introduced in WordEditor1. Mnemonics are keyboard shortcuts representing commands. They are also known as accelerators in the Java Swing. For example, pressing ALT-O (press ALT and O keys together) is the same as selecting from the "File" menu "Open…" command. How do we implement mnemonics or menu keyboard shortcuts?

[11] We define a class as abstract when we do not intend to instantiate objects from the class or when the class contains one or more abstract methods.
[12] Polymorphism is the ability of objects of different class definition to respond to the same message.

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

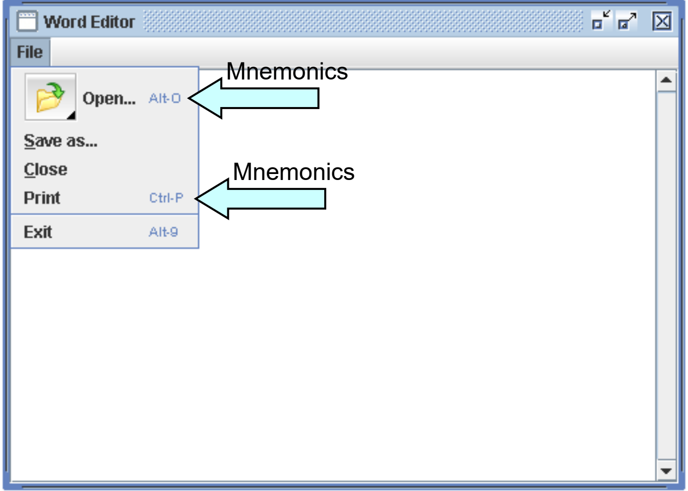


FIGURE 9.8: Demonstrating Mnemonics

9.3.1 WordEditor2 – Adding Menu Keyboard Shortcuts

Code 9.4 shows WordEditor2 (Word Editor Version 2) which is an extension of WordEditor1. We have masked out some of the code in createFrameAndShow() and main() method for simplicity.

Code 9.4: WordEditor2 Class

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

public class WordEditor2 extends WordEditor {


  public WordEditor2() {

    // create and set menu
    setMenu();

    // create a panel and set layout
    panel.setLayout(new BorderLayout());

    // create text area and set it in a scroll pane
    textArea.setText("");
    textArea.setFont(new Font("Courier New",
                              Font.TRUETYPE_FONT, 12));
    textArea.setLineWrap(true);
    textArea.setWrapStyleWord(true);
    JScrollPane textAreaScrollPane = new JScrollPane(textArea);
    textAreaScrollPane.setVerticalScrollBarPolicy(
                        JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);

    // add text area into panel
    panel.add(textAreaScrollPane);
  }

  private void setMenu() {
    // add File Menu into menuBar
    menuBar.add(menu = new JMenu("File"));
    menuItem = new JMenuItem("Open...",
                             new ImageIcon("gif/open.gif"));
    menuItem.setHorizontalTextPosition(JMenuItem.RIGHT);
    menuItem.setAccelerator(KeyStroke.getKeyStroke(
                KeyEvent.VK_O, ActionEvent.ALT_MASK));
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menuItem = new JMenuItem("Save as...", KeyEvent.VK_S);
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menuItem = new JMenuItem("Close");
    menuItem.setMnemonic(KeyEvent.VK_C);
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menuItem = new JMenuItem("Print");
    menuItem.setAccelerator(KeyStroke.getKeyStroke('P',
      Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()));
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menu.addSeparator();
    menuItem = new JMenuItem("Exit");
    menuItem.setAccelerator(KeyStroke.getKeyStroke(
            KeyEvent.VK_9, ActionEvent.ALT_MASK));
    menuItem.addActionListener(this);
    menu.add(menuItem);

    // set menu bar and window listener
    setJMenuBar(menuBar);
    addWindowListener(this);
  }

  private static void createFrameAndShow() {
    //Make sure we have nice window decorations.
    JFrame.setDefaultLookAndFeelDecorated(true);

    //Create and set up the window.
    WordEditor2 frame = new WordEditor2();
    frame.setTitle("Word Editor");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    //Create and set up the content pane.
    JComponent contentPane = (JComponent)frame.getContentPane();
    contentPane.setOpaque(true); //content panes must be opaque
    frame.setContentPane(contentPane);
    contentPane.add(frame.panel);

    //Display the window.
    frame.pack();
    frame.setLocation(300, 100);
    frame.setSize(500, 360);
    frame.setVisible(true);
  }

  public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        createFrameAndShow();
      }
    });
  }
}
```

Changes have been made to the setMenu() method in WordEditor2. We will focus our discussion on this method:

1. Creates the "Open" command as a JMenuItem object using the fifth constructor method in Table 9.1. An image icon is attached to the text of the command. Fix the text to the RIGHT of the image icon.
2. To set a mnemonic to a menu item, we use the setAccelerator() method. The parameter for this method sets ALT-O as the keyboard shortcut for activating the "Open" command. This has the same effect as selecting the "Open…" menu item from the "File" menu.
3. The parameter for setAccelerator() method refers to the static getKeyStroke() method of the javax.swing.KeyStroke class. This method returns a shared KeyStroke object, given a numeric key code (KeyEvent.VK_O) and a set of modifiers (ActionEvent.ALT_MASK). The returned KeyStroke object corresponds to the key pressed. The numeric key code KeyEvent.VK_O (Note: For a full list of numeric key codes or virtual keys, refer to java.awt.event.KeyEvent API documentation) corresponds to the key "O" on the keyboard while ActionEvent.ALT_MASK

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

(Note: For a full list of modifiers, refer to java.awt.event.InputEvent API documentation) modifier corresponds to the "ALT" key.

4. Creates the "Save as" command as a JMenuItem object using the sixth constructor method in Table 9.1. This constructor method sets the "ALT-S" as the mnemonic for saving a file. However, setting a mnemonic with this approach requires you to select the "File" menu to display the popup menu before you can press "ALT-S" to activate a file dialog box to name a file for saving the contents into. In the previous method using the setAccelerator() method, there is no requirement for you to select the "File" menu to activate the "Open" command. Pressing "ALT-O" *accelerates* the display of the file dialog box for opening a file. *The setAccelerator() method therefore allows you to set the key combination for invoking the menu item's action listeners without navigating the menu hierarchy.*
5. Creates the "Close" command as a JMenuItem object using the fourth constructor method in Table 9.1. Sets "ALT-C" as the key combination for closing a file.
6. Creates the "Print" command as a JMenuItem object using the fourth constructor method in Table 9.1.
7. Calls the setAccelerator() method to set "CTRL-P" as the key combination instead of "ALT-P". The method getMenuShortcutKeyMask() gets the modifier key as the appropriate accelerator key for menu keyboard shortcuts. A modifier key can be CTRL_MASK (for CONTROL key), SHIFT_MASK (for SHIFT key), ALT_MASK (for ALT key), or META_MASK (for META key). The default modifier key is the CTRL_MASK. This explains why "CTRL-P" is used instead. See the java.awt.event.InputEvent API documentation for a full list of possible modifier keys.
8. Creates the "Exit" command as a JMenuItem object using the fourth constructor method in Table 9.1. Set the accelerator key to "ALT-9" using the ALT_MASK. This example shows that you can combine "ALT" key with a digit (0 to 9) too. You can also combine Function Keys (F1 to F12) with ALT key to form a menu keyboard shortcut. See the java.awt.event.KeyEvent API documentation for a full list of possible virtual keys.

9.3.2 setAccelerator() and setMnemonic() for Shortcuts

The above example application, WordEditor2, demonstrates the use of setAccelerator() and setMnemonic() methods. They are used to set menu keyboard shortcuts without having to navigate the menu hierarchy. Readers are advised to check out the java.awt.event.InputEvent and java.awt.event.KeyEvent API documentation for a full list of possible key combinations.

Note that only setAccelerator() method allows you to activate the menu keyboard shortcuts without selecting the menu.

9.4 Adding More Menus

Our Word Editor currently has a set of functions for manipulating files but it lacks editing facilities for copying, pasting or clearing texts in a file. To provide such facilities, we will add a new menu for users to select.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

9.4.1 WordEditor3 – Adding More Menus and Menu Items

Figure 9.9 shows the addition of a new menu with "Cut", "Copy", "Paste", "Clear all", and "Select all" commands. This section discusses the extension of Word Edition Version 2 (WordEditor2) to include editing facilities and examine how they can be accommodated within the design of the Word Editor.

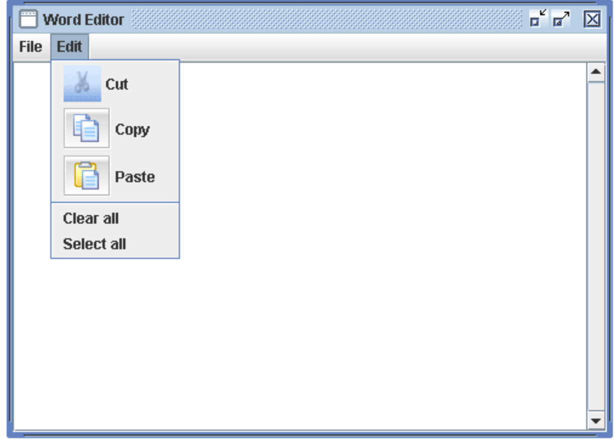


FIGURE 9.9: Adding More Menus

Code 9.5 is the code for WordEditor3, Version 3 of Word Editor. As before, we have masked out createFrameAndShow() and main() methods.

Code 9.5: WordEditor3 Class

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

public class WordEditor3 extends WordEditor {


  public WordEditor3() {

    // create and set menu
    setMenu();

    // create a panel and set layout
    panel.setLayout(new BorderLayout());

    // create text area and set it in a scroll pane
    textArea.setText("");
    textArea.setFont(new Font("Courier New",
                          Font.TRUETYPE_FONT, 12));
    textArea.setLineWrap(true);
    textArea.setWrapStyleWord(true);
    JScrollPane textAreaScrollPane = new JScrollPane(textArea);
    textAreaScrollPane.setVerticalScrollBarPolicy(
                    JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);

    // add text area into panel
    panel.add(textAreaScrollPane);
  }

  private void setMenu() {
    // add File Menu into menuBar
    menuBar.add(menu = new JMenu("File"));
    menuItem = new JMenuItem("Open...",
                         new ImageIcon("gif/open.gif"));
    menuItem.setHorizontalTextPosition(JMenuItem.RIGHT);
    menuItem.setAccelerator(KeyStroke.getKeyStroke(
                KeyEvent.VK_O, ActionEvent.ALT_MASK));
    menuItem.addActionListener(this);
```

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

```
    menu.add(menuItem);
    menuItem = new JMenuItem("Save as...",
                             new ImageIcon("gif/save.gif"));
    menuItem.setMnemonic(KeyEvent.VK_S);
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menuItem = new JMenuItem("Close");
    menuItem.setMnemonic(KeyEvent.VK_C);
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menuItem = new JMenuItem("Print");
    menuItem.setAccelerator(KeyStroke.getKeyStroke('P',
      Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()));
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menu.addSeparator();
    menuItem = new JMenuItem("Exit",
                             new ImageIcon("gif/exit.gif"));
    menuItem.setAccelerator(KeyStroke.getKeyStroke(
                KeyEvent.VK_9, ActionEvent.ALT_MASK));
    menuItem.addActionListener(this);
    menu.add(menuItem);

    // add Edit Menu into menuBar
    menuBar.add(menu = new JMenu("Edit"));
    menuItem = new JMenuItem("Cut",
                             new ImageIcon("gif/cut.gif"));
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menuItem = new JMenuItem("Copy",
                             new ImageIcon("gif/copy.gif"));
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menuItem = new JMenuItem("Paste",
                             new ImageIcon("gif/paste.gif"));
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menu.addSeparator();
    menuItem = new JMenuItem("Clear all");
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menuItem = new JMenuItem("Select all");
    menuItem.addActionListener(this);
    menu.add(menuItem);

    // set menu bar and window listener
    setJMenuBar(menuBar);
    addWindowListener(this);
  }

  private static void createFrameAndShow() {
    //Make sure we have nice window decorations.
    JFrame.setDefaultLookAndFeelDecorated(true);

    //Create and set up the window.
    WordEditor3 frame = new WordEditor3();
    frame.setTitle("Word Editor");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    //Create and set up the content pane.
    JComponent contentPane = (JComponent)frame.getContentPane();
    contentPane.setOpaque(true); //content panes must be opaque
    frame.setContentPane(contentPane);
    contentPane.add(frame.panel);

    //Display the window.
    frame.pack();
    frame.setLocation(300, 100);
    frame.setSize(500, 360);
    frame.setVisible(true);
  }
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
  public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        createFrameAndShow();
      }
    });
  }
}
```

Once again, our discussion of Code 9.5 will be directed at setMenu() method:

1. To add a new menu, we create a new JMenu object initialized with text "Edit" and add it to menuBar, a JMenuBar object.
2. As before, we create a JMenuItem object for each of the commands in the "Edit" menu. We add an action listener to each of the menu items created. The addition of action listeners will ensure that WordEditor3 is able to respond to user selection of the commands. The menu items are subsequently added to the menu.

As mentioned in Section 9.2.3, the commands are polymorphic. We create five classes, one for each of the commands "Cut", "Copy", "Paste", "Clear all", and "Select all". The classes implement the responses to the commands. The five classes are conveniently included in the ApplnWindow.java file as shown in Code 9.6; they include Cut.class, Copy.class, Paste.class, Clearall.class, and Selectall.class. Note that each of these classes has its own execute() method.

No changes are made to the other classes. The use of polymorphic classes has greatly enhanced the addition of new commands.

Code 9.6: ApplnWindow Class Continued

```
class Cut extends ApplnWindow {
  Cut() {}
  public void execute(WordEditor parentFrame) {
    parentFrame.textArea.cut();
  }
}

class Copy extends ApplnWindow {
  Copy() {}
  public void execute(WordEditor parentFrame) {
    parentFrame.textArea.copy();
  }
}

class Paste extends ApplnWindow {
  Paste() {}
  public void execute(WordEditor parentFrame) {
    parentFrame.textArea.paste();
  }
}

class Clearall extends ApplnWindow {
  Clearall() {}
  public void execute(WordEditor parentFrame) {
    if (parentFrame.textArea.isEditable()) {
      parentFrame.textArea.setText("");
    }
  }
}

class Selectall extends ApplnWindow {
  Selectall() {}
  public void execute(WordEditor parentFrame) {
    if (parentFrame.textArea.isEditable()) {
      parentFrame.textArea.selectAll();
    }
  }
}
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

9.4.2 Adding Components into Containers

Components can only be added to at most one container. If a component added to a container is later added to a second container, the component would be removed from the first container before it is added to the second container. A menu item is a component and would behave in the same way if it is added to more than one menu (a container).

Let us illustrate this situation with an example (see Code 9.7). The code creates three JMenuItem objects (menuItem1, menuItem2, and menuItem3) and two JMenu objects (menuA and menuB).

Code 9.7: Adding Components into Containers

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


public class AddingMenuItems extends JFrame
             implements WindowListener, ActionListener {

  // menu and its components
  JMenuBar  menuBar   = new JMenuBar();
  JMenu     menuA, menuB;
  JMenuItem menuItem1, menuItem2, menuItem3;

  // panels
  JPanel panel        = new JPanel();

  public AddingMenuItems() {
    // create and set menu
    setMenu();
  }

  private void setMenu() {
    // add menuA into menuBar
    menuBar.add(menuA = new JMenu("Menu A"));

    // add menuItem1 into menuA
    menuItem1 = new JMenuItem("Menu Item 1");
    menuItem1.addActionListener(this);
    menuA.add(menuItem1);

    // add menuItem2 into menuA
    menuItem2 = new JMenuItem("Menu Item 2");
    menuItem2.addActionListener(this);
    menuA.add(menuItem2);

    // add menuB into menuBar
    menuBar.add(menuB = new JMenu("Menu B"));

    // add menuItem3 into menuB
    menuItem3 = new JMenuItem("Menu Item 3");
    menuItem3.addActionListener(this);
    menuB.add(menuItem3);

    // add menuItem1 into menuB
    // menuItem1 is removed from menuA
    menuB.add(menuItem1);

    // add a new menuItem2 into menuB
    menuItem2 = new JMenuItem("New Menu Item 2");
    menuItem2.addActionListener(this);
    menuB.add(menuItem2);

    // set menu bar and window listener
    setJMenuBar(menuBar);
    addWindowListener(this);
  }

  public void actionPerformed(ActionEvent ae) {
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
    System.out.println(ae.getActionCommand());
  }

  public void windowClosing(WindowEvent we) {
    System.exit(0); // exit from system
  }

  public void windowActivated(WindowEvent we)   { }
  public void windowClosed(WindowEvent we)      { }
  public void windowDeactivated(WindowEvent we) { }
  public void windowDeiconified(WindowEvent we) { }
  public void windowIconified(WindowEvent we)   { }
  public void windowOpened(WindowEvent we)      { }

  private static void createFrameAndShow() {
    //Make sure we have nice window decorations.
    JFrame.setDefaultLookAndFeelDecorated(true);

    //Create and set up the window.
    AddingMenuItems frame = new AddingMenuItems();
    frame.setTitle("Word Editor");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    //Create and set up the content pane.
    JComponent contentPane = (JComponent)frame.getContentPane();
    contentPane.setOpaque(true); //content panes must be opaque
    frame.setContentPane(contentPane);
    contentPane.add(frame.panel);

    //Display the window.
    frame.pack();
    frame.setLocation(300, 100);
    frame.setSize(500, 360);
    frame.setVisible(true);
  }

  public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        createFrameAndShow();
      }
    });
  }
}
```

Let us discuss Code 9.7:

1. menuItem1 is added into menuA container.
2. menuItem2 is added into menuA container.
3. menuItem3 is added into menuB container.
4. menuItem1 is later added into menuB. As expected, menuItem1 is removed from menuA before it is added into menuB. The final outcome is menuA has menuItem2 and menuB has menuItem3 and menuItem1 (see Figure 9.10).

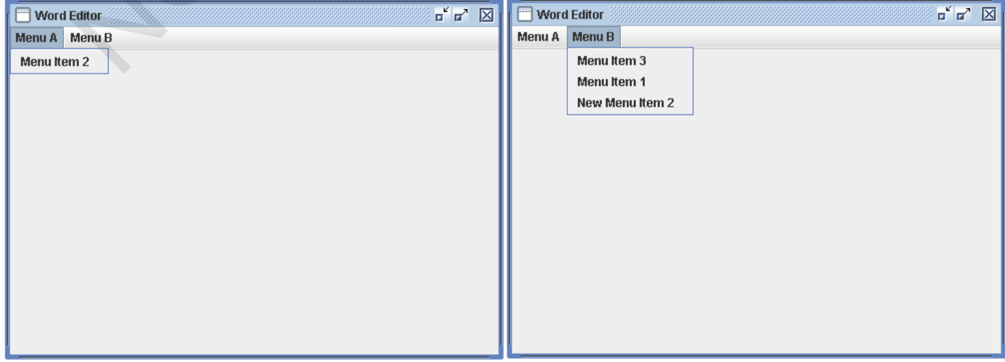


FIGURE 9.10: Adding Components into Containers

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

However, if we have created a new menuItem2 object as and add it into menuB, the previous menuItem2 created is not removed from menuA. Instead, the new menuItem2 component appears in menuB as shown by the text "New Menu Item 2" in menuB of Figure 9.10.

Selecting the various menu items in the two menus produces the following output at the command prompt:

```
Menu Item 1
New Menu Item 2
Menu Item 2
Menu Item 3
Menu Item 1
Menu Item 2
New Menu Item 2
```

As the output shows, the new menuItem2 has its own action listener attached to it.

9.5 Submenus

A *submenu* is a menu within a menu. To demonstrate the use of submenus, we will extend our Word Editor example to include facility for changing the font and casing of texts.

Figure 9.11 shows the extension of the Word Editor with an additional "Change" menu that includes submenus "Word Casing" and "Font". How do we implement submenus?

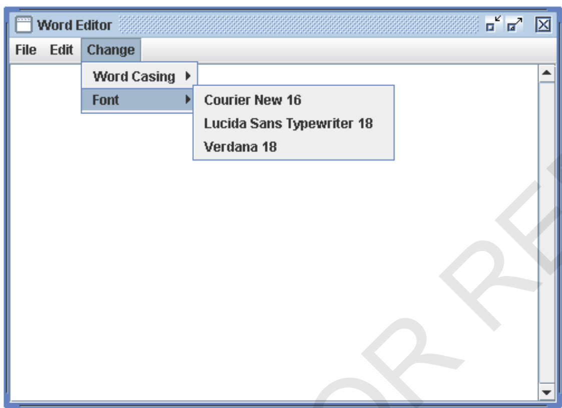


FIGURE 9.11: Submenus

NOT FOR REDISTRIBUTION

9.5.1 WordEditor4 – Adding Submenus

Submenus are treated as JMenu objects i.e. submenus are themselves menus with their own set of menu items. A submenu is formed by creating a menu to contain menu items before it is added into a holding menu. The holding menu is finally added into the menu bar.

Code 9.8: WordEditor4 Class

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

public class WordEditor4 extends WordEditor {

  JMenu subMenu;

  public WordEditor4() {

    // create and set menu
    setMenu();
```

Copyright © 2020, Danny Poo.

```
    // create a panel and set layout
    panel.setLayout(new BorderLayout());

    // create text area and set it in a scroll pane
    textArea.setText("");
    textArea.setFont(new Font("Courier New",
                              Font.TRUETYPE_FONT, 12));
    textArea.setLineWrap(true);
    textArea.setWrapStyleWord(true);
    JScrollPane textAreaScrollPane = new JScrollPane(textArea);
    textAreaScrollPane.setVerticalScrollBarPolicy(
                    JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);

    // add text area into panel
    panel.add(textAreaScrollPane);
  }

  private void setMenu() {
    // add File Menu into menuBar
    menuBar.add(menu = new JMenu("File"));
    menuItem = new JMenuItem("Open...",
                             new ImageIcon("gif/open.gif"));
    menuItem.setHorizontalTextPosition(JMenuItem.RIGHT);
    menuItem.setAccelerator(KeyStroke.getKeyStroke(
                KeyEvent.VK_O, ActionEvent.ALT_MASK));
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menuItem = new JMenuItem("Save as...",
                             new ImageIcon("gif/save.gif"));
    menuItem.setMnemonic(KeyEvent.VK_S);
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menuItem = new JMenuItem("Close");
    menuItem.setMnemonic(KeyEvent.VK_C);
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menuItem = new JMenuItem("Print");
    menuItem.setAccelerator(KeyStroke.getKeyStroke('P',
      Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()));
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menu.addSeparator();
    menuItem = new JMenuItem("Exit",
                             new ImageIcon("gif/exit.gif"));
    menuItem.setAccelerator(KeyStroke.getKeyStroke(
                KeyEvent.VK_9, ActionEvent.ALT_MASK));
    menuItem.addActionListener(this);
    menu.add(menuItem);

    // add Edit Menu into menuBar
    menuBar.add(menu = new JMenu("Edit"));
    menuItem = new JMenuItem("Cut",
                             new ImageIcon("gif/cut.gif"));
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menuItem = new JMenuItem("Copy",
                             new ImageIcon("gif/copy.gif"));
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menuItem = new JMenuItem("Paste",
                             new ImageIcon("gif/paste.gif"));
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menu.addSeparator();
    menuItem = new JMenuItem("Clear all");
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menuItem = new JMenuItem("Select all");
    menuItem.addActionListener(this);
    menu.add(menuItem);
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
    // add Change Menu into menuBar
    menuBar.add(menu = new JMenu("Change"));
    subMenu = new JMenu("Word Casing");
    menuItem = new JMenuItem("To Upper Case");
    subMenu.add(menuItem);
    menuItem.addActionListener(this);
    menuItem = new JMenuItem("To Lower Case");
    subMenu.add(menuItem);
    menuItem.addActionListener(this);
    menu.add(subMenu);
    subMenu = new JMenu("Font");
    menuItem = new JMenuItem("Courier New 16");
    subMenu.add(menuItem);
    menuItem.addActionListener(this);
    menuItem = new JMenuItem("Lucida Sans Typewriter 18");
    subMenu.add(menuItem);
    menuItem.addActionListener(this);
    menuItem = new JMenuItem("Verdana 18");
    subMenu.add(menuItem);
    menuItem.addActionListener(this);
    menu.add(subMenu);

    // set menu bar and window listener
    setJMenuBar(menuBar);
    addWindowListener(this);
  }

  private static void createFrameAndShow() {
    //Make sure we have nice window decorations.
    JFrame.setDefaultLookAndFeelDecorated(true);

    //Create and set up the window.
    WordEditor4 frame = new WordEditor4();
    frame.setTitle("Word Editor");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    //Create and set up the content pane.
    JComponent contentPane = (JComponent)frame.getContentPane();
    contentPane.setOpaque(true); //content panes must be opaque
    frame.setContentPane(contentPane);
    contentPane.add(frame.panel);

    //Display the window.
    frame.pack();
    frame.setLocation(300, 100);
    frame.setSize(500, 360);
    frame.setVisible(true);
  }

  public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        createFrameAndShow();
      }
    });
  }
}
```

NOT FOR REDISTRIBUTION

Let us examine the setMenu() method of Code 9.8:

1. We have masked out some of the code with "…" as they are similar to those in WordEditor3.
2. A JMenu object with specified text "Change" is created and added to menuBar.
3. A submenu created as a JMenu object is initialized with the texts "Word Casing".
4. Two separate JMenuItem objects are created and added to the submenu respectively. Action listeners have been added to these menu items.
5. The submenu is added to the JMenu object created.
6. Repeat the above process for adding a "Font" submenu. This submenu includes three menu items each specifying a font type, size and style (default to Plain).

Copyright © 2020, Danny Poo.

9.5.2 Polymorphism in Action

Commands in the submenus are implemented by classes of the same names as the commands. Code 9.9 is a continuation of Code 9.6. As before, we add new classes implementing the commands in the menus. This design approach has greatly reduced the extension effort.

Code 9.9: ApplnWindow Class Continued

```
class ToUpperCase extends ApplnWindow {
  ToUpperCase() {}
  public void execute(WordEditor parentFrame) {
    if (parentFrame.textArea.isEditable()) {
      String s = parentFrame.textArea.getSelectedText();
      if (s != null) {
        s = s.toUpperCase();
        parentFrame.textArea.replaceSelection(s);
      }
    }
  }
}

class ToLowerCase extends ApplnWindow {
  ToLowerCase() {}
  public void execute(WordEditor parentFrame) {
    if (parentFrame.textArea.isEditable()) {
      String s = parentFrame.textArea.getSelectedText();
      if (s != null) {
        s = s.toLowerCase();
        parentFrame.textArea.replaceSelection(s);
      }
    }
  }
}

class CourierNew16 extends ApplnWindow {
  CourierNew16() {}
  public void execute(WordEditor parentFrame) {
    if (parentFrame.textArea.isEditable()) {
      Font font = new Font("Courier New", Font.PLAIN, 16);
      parentFrame.textArea.setFont(font);
    }
  }
}

class LucidaSansTypewriter18 extends ApplnWindow {
  LucidaSansTypewriter18() {}
  public void execute(WordEditor parentFrame) {
    if (parentFrame.textArea.isEditable()) {
      Font font = new Font("Lucida Sans Typewriter", Font.PLAIN, 18);
      parentFrame.textArea.setFont(font);
    }
  }
}

class Verdana18 extends ApplnWindow {
  Verdana18() {}
  public void execute(WordEditor parentFrame) {
    if (parentFrame.textArea.isEditable()) {
      Font font = new Font("Verdana", Font.PLAIN, 18);
      parentFrame.textArea.setFont(font);
    }
  }
}
```

9.6 Adding Radio Buttons and Checkboxes in Menus

In our next enhancement of the “Word Editor”, we will show how we can include radio buttons and checkboxes in menus. In Figure 9.12, a checkbox menu item “Document NOT editable” and two radio buttons “Blue background” and “White background” have been added into the “Change” menu.

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

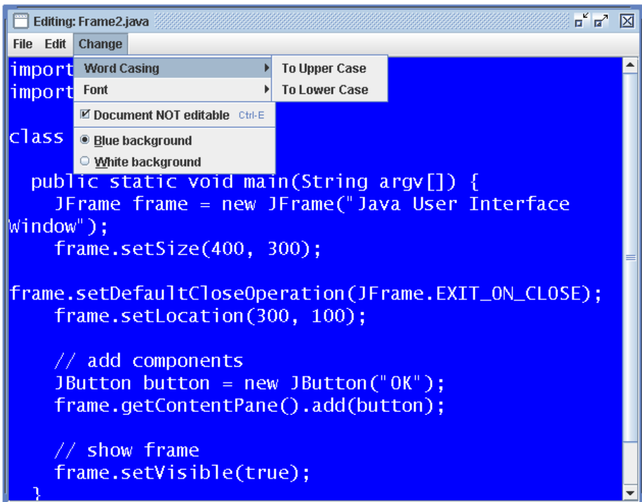


FIGURE 9.12: Add Radio Buttons and Checkboxes in Menus

9.6.1 WordEditor5 – Adding Radio Buttons and Checkboxes

Code 9.10 is the code for WordEditor5, our fifth version of Word Editor, demonstrating the addition of radio buttons and checkboxes in menus.

Code 9.10: WordEditor5 Class

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

public class WordEditor5 extends WordEditor {

  JMenu                subMenu;
  JCheckBoxMenuItem    checkBoxMenuItem;
  JRadioButtonMenuItem radioButtonMenuItem;


  public WordEditor5() {

    // create and set menu
    setMenu();

    // create a panel and set layout
    panel.setLayout(new BorderLayout());

    // create text area and set it in a scroll pane
    textArea.setText("");
    textArea.setFont(new Font("Courier New",
                          Font.TRUETYPE_FONT, 12));
    textArea.setLineWrap(true);
    textArea.setWrapStyleWord(true);
    JScrollPane textAreaScrollPane = new JScrollPane(textArea);
    textAreaScrollPane.setVerticalScrollBarPolicy(
                    JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);

    // add text area into panel
    panel.add(textAreaScrollPane);
  }

  private void setMenu() {
    // add File Menu into menuBar
    menuBar.add(menu = new JMenu("File"));
    menuItem = new JMenuItem("Open...",
                         new ImageIcon("gif/open.gif"));
```

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

```
menuItem.setHorizontalTextPosition(JMenuItem.RIGHT);
menuItem.setAccelerator(KeyStroke.getKeyStroke(
            KeyEvent.VK_O, ActionEvent.ALT_MASK));
menuItem.addActionListener(this);
menu.add(menuItem);
menuItem = new JMenuItem("Save as...",
                         new ImageIcon("gif/save.gif"));
menuItem.setMnemonic(KeyEvent.VK_S);
menuItem.addActionListener(this);
menu.add(menuItem);
menuItem = new JMenuItem("Close");
menuItem.setMnemonic(KeyEvent.VK_C);
menuItem.addActionListener(this);
menu.add(menuItem);
menuItem = new JMenuItem("Print");
menuItem.setAccelerator(KeyStroke.getKeyStroke('P',
  Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()));
menuItem.addActionListener(this);
menu.add(menuItem);
menu.addSeparator();
menuItem = new JMenuItem("Exit",
                         new ImageIcon("gif/exit.gif"));
menuItem.setAccelerator(KeyStroke.getKeyStroke(
            KeyEvent.VK_9, ActionEvent.ALT_MASK));
menuItem.addActionListener(this);
menu.add(menuItem);

// add Edit Menu into menuBar
menuBar.add(menu = new JMenu("Edit"));
menuItem = new JMenuItem("Cut",
                         new ImageIcon("gif/cut.gif"));
menuItem.addActionListener(this);
menu.add(menuItem);
menuItem = new JMenuItem("Copy",
                         new ImageIcon("gif/copy.gif"));
menuItem.addActionListener(this);
menu.add(menuItem);
menuItem = new JMenuItem("Paste",
                         new ImageIcon("gif/paste.gif"));
menuItem.addActionListener(this);
menu.add(menuItem);
menu.addSeparator();
menuItem = new JMenuItem("Clear all");
menuItem.addActionListener(this);
menu.add(menuItem);
menuItem = new JMenuItem("Select all");
menuItem.addActionListener(this);
menu.add(menuItem);

// add Change Menu into menuBar
menuBar.add(menu = new JMenu("Change"));
subMenu = new JMenu("Word Casing");
menuItem = new JMenuItem("To Upper Case");
subMenu.add(menuItem);
menuItem.addActionListener(this);
menuItem = new JMenuItem("To Lower Case");
subMenu.add(menuItem);
menuItem.addActionListener(this);
menu.add(subMenu);
subMenu = new JMenu("Font");
menuItem = new JMenuItem("Courier New 16");
subMenu.add(menuItem);
menuItem.addActionListener(this);
menuItem = new JMenuItem("Lucida Sans Typewriter 18");
subMenu.add(menuItem);
menuItem.addActionListener(this);
menuItem = new JMenuItem("Verdana 18");
subMenu.add(menuItem);
menuItem.addActionListener(this);
menu.add(subMenu);
menu.addSeparator();
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
    checkBoxMenuItem = new JCheckBoxMenuItem("Document NOT editable");
    checkBoxMenuItem.setHorizontalTextPosition(JMenuItem.RIGHT);
    checkBoxMenuItem.setAccelerator(KeyStroke.getKeyStroke('E',
      Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()));
    checkBoxMenuItem.addItemListener(this);
    menu.add(checkBoxMenuItem);
    menu.addSeparator();

    ButtonGroup group = new ButtonGroup();
    radioButtonMenuItem = new JRadioButtonMenuItem("Blue background");
    radioButtonMenuItem.setMnemonic(KeyEvent.VK_B);
    group.add(radioButtonMenuItem);
    radioButtonMenuItem.addItemListener(this);
    menu.add(radioButtonMenuItem);
    radioButtonMenuItem = new JRadioButtonMenuItem("White background");
    radioButtonMenuItem.setSelected(true);
    radioButtonMenuItem.setMnemonic(KeyEvent.VK_W);
    group.add(radioButtonMenuItem);
    radioButtonMenuItem.addItemListener(this);
    menu.add(radioButtonMenuItem);

    // set menu bar and window listener
    setJMenuBar(menuBar);
    addWindowListener(this);
  }

  private static void createFrameAndShow() {
    //Make sure we have nice window decorations.
    JFrame.setDefaultLookAndFeelDecorated(true);

    //Create and set up the window.
    WordEditor5 frame = new WordEditor5();
    frame.setTitle("Word Editor");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    //Create and set up the content pane.
    JComponent contentPane = (JComponent)frame.getContentPane();
    contentPane.setOpaque(true); //content panes must be opaque
    frame.setContentPane(contentPane);
    contentPane.add(frame.panel);

    //Display the window.
    frame.pack();
    frame.setLocation(300, 100);
    frame.setSize(500, 360);
    frame.setVisible(true);
  }

  public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        createFrameAndShow();
      }
    });
  }
}
```

WordEditor5 is built from WordEditor4. We have masked out much of the code so that we can focus our discussion on the addition of radio buttons and checkboxes in menus. Let us examine Code 9.10:

1. We added a separator to the "Change" menu.
2. Instantiates a JCheckBoxMenuItem object (see Section 9.6.4 for a description of JCheckBoxMenuItem class) and initialized with the text "Document NOT editable". When the user checks this box, the document loaded into the JTextArea becomes non-editable i.e. no updates can be done on the document. The implementation of this command, as before, is achieved by a class named DocumentNOTeditable.class.
3. Sets the position of the text "Document NOT editable" to be on the right of image icon representing this command. Since there is no image icon specified for this command, this statement has no visible effect on the menu.
4. As before, sets CTRL-E as the menu keyboard shortcut for this command.

Copyright © 2020, Danny Poo.

NOT FOR REDISTRIBUTION

5. Creates a JRadioButtonMenuItem object (see Section 9.6.5 for a description of JRadioButtonMenuItem class) initialized with the text “Blue background”. A similar object initialized with the text “White background” is created. These two objects are radio buttons; their mnemonics for keyboard activation are set as “B” for “Blue background” and “W” for “White background. They are also set with their own item listeners. A ButtonGroup object is created to associate the two radio buttons so that the user can only select either blue or white background for displaying the document. The two radio button menu items are added to this button group.
6. Although the two radio buttons have been grouped together in a single group, they are still individual components. Adding the group into the menu is not the same as adding the two radio buttons individually into the menu. For correct behaviors, we have to add them into the menu individually as shown.

9.6.2 Event Handling

Event handling on radio buttons and checkboxes is facilitated by the itemStateChanged() method of the item listeners. This method of the java.awt.event.ItemListener interface has been implemented in the superclass, WordEditor. The code for WordEditor is given in Code 9.2.

Code 9.11: WordEditor Class Reproduced

```
private String extract(String s) {
  int i = s.indexOf("text=");
  int j = s.indexOf("]", i);
  String command = s.substring(i+5, j);
  return command;
}

public String formClassName(String s) {
  String command = "";
  StringTokenizer t = new StringTokenizer(s, ". ");
  while (t.hasMoreTokens())
    command = command+t.nextToken();
  return command;
}

public void itemStateChanged(ItemEvent ie) {
  String command = new String();
  command = formClassName(extract(ie.getItem().toString()));
  if (command.equals("DocumentNOTeditable")) {
    if (ie.getStateChange() == ItemEvent.DESELECTED) {
      command = formClassName("DocumentEditable");
    }
  }
  try {
    Class c = Class.forName(command);
    ApplnWindow w = (ApplnWindow) c.newInstance();
    w.execute(this);
  } catch (InstantiationException i) {
      System.out.println("Instance not created");
  } catch (ClassNotFoundException c) {
      System.out.println("Class not found");
  } catch (IllegalAccessException i) {
      System.out.println("Illegal access");
  }
}
```

NOT FOR REDISTRIBUTION

We have reproduced a segment of WordEditor class in Code 9.11 for our discussion on itemStateChanged() method:

1. The itemStateChanged() method is called when a radio button is clicked.
2. Calls the extract() method to grab the command (the text string of the menu item) from the ItemEvent object. formClassName() makes use of the StringTokenizer class to form the text string of the menu item into a command name e.g. “Save as…” becomes “Saveas”.
3. Determines if the command is from the checkbox menu item “Document NOT editable”. If so, it checks if the checkbox has been deselected (i.e. the document is editable) and if so, it forms the class name “DocumentEditable”.

Copyright © 2020, Danny Poo.

4. Uses the static method Class.forName() to return the Class object associated with the class of name equivalent to the command.
5. Instantiates a new instance of the class represented by the Class object produced. Note that all the Class objects must be defined as a subclass of the ApplnWindow class.
6. Calls the execute() method of the object instantiated.

9.6.3 Polymorphic Classes

Code 9.12 shows the last installment of the code for ApplnWindow.java. It includes the class declaration of the menu items (or commands) in the menus.

Code 9.12: ApplnWindow Class Continued

```
class DocumentEditable extends ApplnWindow {
  DocumentEditable() {}
  public void execute(WordEditor parentFrame) {
    parentFrame.textArea.setEditable(true);
  }
}

class DocumentNOTeditable extends ApplnWindow {
  DocumentNOTeditable() {}
  public void execute(WordEditor parentFrame) {
    parentFrame.textArea.setEditable(false);
  }
}

class Bluebackground extends ApplnWindow {
  Bluebackground() {}
  public void execute(WordEditor parentFrame) {
    parentFrame.textArea.setForeground(Color.WHITE);
    parentFrame.textArea.setBackground(Color.BLUE);
  }
}

class Whitebackground extends ApplnWindow {
  Whitebackground() {}
  public void execute(WordEditor parentFrame) {
    parentFrame.textArea.setForeground(Color.BLACK);
    parentFrame.textArea.setBackground(Color.WHITE);
  }
}
```

9.6.4 JCheckBoxMenuItem

The Java Swing version of a checkbox menu item is JCheckBoxMenuItem. Figure 9.13 is the class hierarchy for JCheckBoxMenuItem.

Note that a JCheckBoxMenuItem object is essentially a button (as it extends from javax.swing.AbstractButton) added as a menu item into a list (the menu). This means that a JCheckBoxMenuItem object can respond to item events when selected.

```
java.lang.Object
└── java.awt.Component
    └── java.awt.Container
        └── javax.swing.JComponent
            └── javax.swing.AbstractButton
                └── javax.swing.JMenuItem
                    └── javax.swing.JCheckBoxMenuItem
```

FIGURE 9.13: Class Hierarchy of JCheckBoxMenuItem

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

9.6.4.1 What are the Constructors?

JCheckBoxMenuItem has seven constructor methods (see Table 9.4).

TABLE 9.4: JCheckBoxMenuItem Constructors

No.	Constructor	Description
1	`JCheckBoxMenuItem ()`	Creates a new and unselected JCheckBoxMenuItem object with no set text or icon.
2	`JCheckBoxMenuItem (Action a)`	Creates a new JCheckBoxMenuItem object whose properties are taken from Action a.
3	`JCheckBoxMenuItem (Icon i)`	Creates a new JCheckBoxMenuItem object with the specified Icon i.
4	`JCheckBoxMenuItem (String text)`	Creates a new and unselected JCheckBoxMenuItem object with the specified String text.
5	`JCheckBoxMenuItem (String text, boolean b)`	Creates a new JCheckBoxMenuItem object with the specified text and selection state.
6	`JCheckBoxMenuItem (String text, Icon i)`	Creates a new and unselected JCheckBoxMenuItem object with the specified text and icon.
7	`JCheckBoxMenuItem (String text, Icon i, boolean b)`	Creates a new JCheckBoxMenuItem object with the specified text, icon, and selection state.

9.6.5 JRadioButtonMenuItem

The Java Swing version of a radio button menu item is JRadioButtonMenuItem. Figure 9.14 is the class hierarchy for JRadioButtonMenuItem.

Note that a JRadioButtonMenuItem object is essentially a button (as it extends from javax.swing.AbstractButton) added as a menu item into a list (the menu). This means that a JRadioButtonMenuItem object can respond to item events when selected.

```
java.lang.Object
└── java.awt.Component
     └── java.awt.Container
          └── javax.swing.JComponent
               └── javax.swing.AbstractButton
                    └── javax.swing.JMenuItem
                         └── javax.swing.JRadioButtonMenuItem
```

FIGURE 9.14: Class Hierarchy of JRadioButtonMenuItem

9.6.5.1 What are the Constructors?

JRadioButtonMenuItem has eight constructor methods (see Table 9.5).

TABLE 9.5: JRadioButtonMenuItem Constructors

No.	Constructor	Description
1	`JRadioButtonMenuItem ()`	Creates a new JRadioButtonMenuItem object with no set text or icon.
2	`JRadioButtonMenuItem (Action a)`	Creates a new JRadioButtonMenuItem object whose properties are taken from Action a.
3	`JRadioButtonMenuItem (Icon i)`	Creates a new JRadioButtonMenuItem object with the specified Icon i.
4	`JRadioButtonMenuItem (Icon i, boolean selected)`	Creates a new and unselected JRadioButtonMenuItem object with the specified Icon i and selection state, but no text.
5	`JRadioButtonMenuItem (String text)`	Creates a new JRadioButtonMenuItem object with the

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

		specified String text.
6	`JRadioButtonMenuItem (String text, boolean selected)`	Creates a new JRadioButtonMenuItem object with the specified text and selection state.
7	`JRadioButtonMenuItem (String text, Icon i)`	Creates a new JRadioButtonMenuItem object with the specified text and icon.
8	`JRadioButtonMenuItem (String text, Icon i, boolean selected)`	Creates a new JRadioButtonMenuItem object with the specified text, icon, and selection state.

9.7 Popup Menus

A *popup menu* is an invisible menu. It only appears when the user clicks on the right mouse button in the menu bar or window title areas as shown in Figure 9.15.

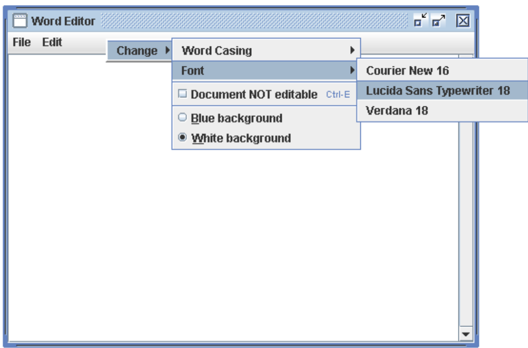


FIGURE 9.15: Popup Menu

9.7.1 WordEditor6 – Adding Popup Menu

In our next example, WordEditor6, we will illustrate how to add and activate a popup menu. We will transform the "Change" menu and its menu items into a popup menu.

Code 9.13: WordEditor6 Class
```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import javax.swing.border.*;
import javax.swing.event.*;

public class WordEditor6 extends WordEditor {

  JMenu                subMenu;
  JCheckBoxMenuItem    checkBoxMenuItem;
  JRadioButtonMenuItem radioButtonMenuItem;
  JPopupMenu           popUpMenu;


  public WordEditor6() {

    // create and set menu
    setMenu();

    // create a panel and set layout
    panel.setLayout(new BorderLayout());

    // create text area and set it in a scroll pane
    textArea.setText("");
    textArea.setFont(new Font("Courier New",
                              Font.TRUETYPE_FONT, 12));
    textArea.setLineWrap(true);
    textArea.setWrapStyleWord(true);
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
    JScrollPane textAreaScrollPane = new JScrollPane(textArea);
    textAreaScrollPane.setVerticalScrollBarPolicy(
                        JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);

    // add text area into panel
    panel.add(textAreaScrollPane);
  }

  private void setMenu() {
    // add File Menu into menuBar
    menuBar.add(menu = new JMenu("File"));
    menuItem = new JMenuItem("Open...",
                             new ImageIcon("gif/open.gif"));
    menuItem.setHorizontalTextPosition(JMenuItem.RIGHT);
    menuItem.setAccelerator(KeyStroke.getKeyStroke(
                KeyEvent.VK_O, ActionEvent.ALT_MASK));
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menuItem = new JMenuItem("Save as...",
                             new ImageIcon("gif/save.gif"));
    menuItem.setMnemonic(KeyEvent.VK_S);
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menuItem = new JMenuItem("Close");
    menuItem.setMnemonic(KeyEvent.VK_C);
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menuItem = new JMenuItem("Print");
    menuItem.setAccelerator(KeyStroke.getKeyStroke('P',
      Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()));
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menu.addSeparator();
    menuItem = new JMenuItem("Exit",
                             new ImageIcon("gif/exit.gif"));
    menuItem.setAccelerator(KeyStroke.getKeyStroke(
                KeyEvent.VK_9, ActionEvent.ALT_MASK));
    menuItem.addActionListener(this);
    menu.add(menuItem);

    // add Edit Menu into menuBar
    menuBar.add(menu = new JMenu("Edit"));
    menuItem = new JMenuItem("Cut",
                             new ImageIcon("gif/cut.gif"));
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menuItem = new JMenuItem("Copy",
                             new ImageIcon("gif/copy.gif"));
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menuItem = new JMenuItem("Paste",
                             new ImageIcon("gif/paste.gif"));
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menu.addSeparator();
    menuItem = new JMenuItem("Clear all");
    menuItem.addActionListener(this);
    menu.add(menuItem);
    menuItem = new JMenuItem("Select all");
    menuItem.addActionListener(this);
    menu.add(menuItem);

    // add Change Menu into menuBar
    menuBar.add(menu = new JMenu("Change"));
    subMenu = new JMenu("Word Casing");
    menuItem = new JMenuItem("To Upper Case");
    subMenu.add(menuItem);
    menuItem.addActionListener(this);
    menuItem = new JMenuItem("To Lower Case");
    subMenu.add(menuItem);
    menuItem.addActionListener(this);
    menu.add(subMenu);
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
    subMenu = new JMenu("Font");
    menuItem = new JMenuItem("Courier New 16");
    subMenu.add(menuItem);
    menuItem.addActionListener(this);
    menuItem = new JMenuItem("Lucida Sans Typewriter 18");
    subMenu.add(menuItem);
    menuItem.addActionListener(this);
    menuItem = new JMenuItem("Verdana 18");
    subMenu.add(menuItem);
    menuItem.addActionListener(this);
    menu.add(subMenu);
    menu.addSeparator();

    checkBoxMenuItem = new JCheckBoxMenuItem("Document NOT editable");
    checkBoxMenuItem.setHorizontalTextPosition(JMenuItem.RIGHT);
    checkBoxMenuItem.setAccelerator(KeyStroke.getKeyStroke('E',
      Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()));
    checkBoxMenuItem.addItemListener(this);
    menu.add(checkBoxMenuItem);
    menu.addSeparator();

    ButtonGroup group = new ButtonGroup();
    radioButtonMenuItem = new JRadioButtonMenuItem("Blue background");
    radioButtonMenuItem.setMnemonic(KeyEvent.VK_B);
    group.add(radioButtonMenuItem);
    radioButtonMenuItem.addItemListener(this);
    menu.add(radioButtonMenuItem);
    radioButtonMenuItem = new JRadioButtonMenuItem("White background");
    radioButtonMenuItem.setSelected(true);
    radioButtonMenuItem.setMnemonic(KeyEvent.VK_W);
    group.add(radioButtonMenuItem);
    radioButtonMenuItem.addItemListener(this);
    menu.add(radioButtonMenuItem);

    popUpMenu = new JPopupMenu();
    popUpMenu.add(menu); // add Change menu
    popUpMenu.setBorder(new BevelBorder(BevelBorder.RAISED));
    //popUpMenu.addPopupMenuListener(new PopUpAlertListener());
    addMouseListener(new MousePopUpListener());

    // set menu bar and window listener
    setJMenuBar(menuBar);
    addWindowListener(this);
  }

  // An inner class to detect mouse events due to popUp trigger
  class MousePopUpListener extends MouseAdapter {
    public void mousePressed(MouseEvent me) {showPopUp(me);}
    public void mouseClicked(MouseEvent me) {showPopUp(me);}
    public void mouseReleased(MouseEvent me) {showPopUp(me);}

    private void showPopUp(MouseEvent me) {
      if (me.isPopupTrigger()) {
        popUpMenu.show(WordEditor6.this, me.getX(), me.getY());
      }
    }
  }

  // An inner class to show occurences of popUp events
  class PopUpAlertListener implements PopupMenuListener {

    public void popupMenuWillBecomeVisible(PopupMenuEvent pe) {
      System.out.println("PopUp menu will become VISIBLE.");
    }

    public void popupMenuWillBecomeInvisible(PopupMenuEvent pe) {
      System.out.println("PopUp menu will become invisible.");
    }

    public void popupMenuCanceled(PopupMenuEvent pe) {
      System.out.println("PopUp menu is CANCELLED/HIDDEN.");
    }
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

```
  }

  private static void createFrameAndShow() {
    //Make sure we have nice window decorations.
    JFrame.setDefaultLookAndFeelDecorated(true);

    //Create and set up the window.
    WordEditor6 frame = new WordEditor6();
    frame.setTitle("Word Editor");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    //Create and set up the content pane.
    JComponent contentPane = (JComponent)frame.getContentPane();
    contentPane.setOpaque(true); //content panes must be opaque
    frame.setContentPane(contentPane);
    contentPane.add(frame.panel);

    //Display the window.
    frame.pack();
    frame.setLocation(300, 100);
    frame.setSize(500, 360);
    frame.setVisible(true);
  }

  public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        createFrameAndShow();
      }
    });
  }
}
```

We have revised WordEditor5 to include the "Change" menu as a popup menu. The new version of WordEditor5 is renamed WordEditor6. Code 9.13 shows the WordEditor6 class:

1. We have masked out quite a bit of code from WordEditor6. The masked out code is similar to those found in WordEditor5 class.
2. We begin our discussion with the "Change" menu in setMenu() method.
3. Instantiates a new JPopupMenu object. popUpMenu has been declared as a JPopupMenu object earlier.
4. The menu object referenced refers to the "Change" menu and is added to popUpMenu.
5. Sets the border of popUpMenu by calling the setBorder() method inherited from javax.swing.JComponent. The javax.swing.border.BevelBorder object created as a parameter to setBorder() method is responsible for defining the insets for the "Change" menu. Do not forget to import the javax.swing.border.* packages.
6. Imports the javax.swing.event.* packages. We need these packages to handle mouse and popup events.
7. A popup menu appears when the user requests for it by clicking the right mouse button on the area the popup menu has been associated with (in this case, the visible frame). In order for the WordEditor6 object (the visible frame) to respond to mouse events, it must register a mouse listener to itself and this is done. The role of the mouse listener is to detect user requests for the popup menu. The MousePopUpListener has been created as an inner class (Note: An **inner class** is a class defined within a class. The scope of an inner class is limited to the class it is defined in. An inner class is thus "invisible" to other classes in the same package. This invisibility enables developers to create more classes within a class but without cluttering the name space of classes in the package) and it extends java.awt.event.MouseAdapter class. The MouseAdapter class is provided as a convenient class for creating mouse listener objects. It is an abstract adapter class that can be extended by subclasses (such as MousePopUpListener) to override some or all of the empty methods the MouseAdapter class has implemented on the java.awt.event.MouseListener interface. MousePopUpListener class overrides three methods: mousePressed(), mouseClicked(), and mouseReleased(). These three methods invoke showPopUp() which tests if the mouse event is a popup menu trigger event. As popup menus are triggered differently on different systems, it is necessary for us to test for popup triggers on mousePressed and mouseReleased for cross-platform compatibility. The show() method of a JPopupMenu object displays the popup menu at the

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

position the right mouse button is clicked. This position is represented as an (x, y) coordinate and passed on as me.getX() and me.getY() values. The first parameter of the show() method represents the popup menu invoker i.e. the WordEditor6 object or the visible frame (shown as WordEditor6.this).

8. Adds a popup menu listener to popUpMenu. Although this statement is not necessary for a popup menu to function, we have added it to show, in the command prompt, when popUpMenu has been activated. The addPopupMenuListener() method requires a javax.swing.event.PopupMenuListener object as its parameter. The PopupMenuListener object implements the PopupMenuListener interface and is instantiated as a PopUpAlertListener object. PopUpAlertListener has been defined as an inner class. It implements the three methods in the PopupMenuListener interface: popupMenuWillBecomeVisible(), popupMenuWillBecomeInvisible(), and popupMenuCanceled(). We have added System.out.println() in these methods to alert us of popup menu activations.

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

APPENDIX: ADDITIONAL CODES

The HTML files describing the various radio stations are provided here.

Edpane.html

```
<html>
<body>
<img src="images/LTPJ.jpg" width="104" height="64"><br>
<font face="Georgia" size=4>
An <b><i>editor pane</i></b> uses specialized editor kits to read, write, display, and
edit text of
different formats. The Swing text package includes editor kits for plain text, HTML, and
RTF.
You can also develop custom editor kits for other formats.
The Java Swing version is <b>JEditorPane</b> - a component for editing various kinds of
contents. <b>The types of contents supported include: <i>text/plain</i>,
<i>text/html</i>, and <i>text/rtf</i></b>:
</font>
<font face="Impact" size=4>
<ul>
<li>text/plain: default type assumed if the type given is not recognised. </li>
<li>text/html: The editor kit used is the class javax.swing.text.html.HTMLEditorKit
which provides HTML 3.2 support. </li>
<li>text/rtf: The editor kit used is the class javax.swing.text.rtf.RTFEditorKit which
provides a limited support of the Rich Text Format. </li>
</ul>
</font>
<p>
</body>
</html>
```

Gold911.html

```
<html>
<body>
<img src="images/gold911.gif" width="90" height="30"><br>
<font face="Georgia" size=5>
<p>
<b>Gold 91.1</b> is an English radio station that is good as gold.
</font>
<p>
<font face="Trebuchet MS" size=6>
This station caters to listeners who enjoy <b><i>English</i></b> pop songs and music
from the 60s, 70s, 80s, and 90s.
</font>
</body>
</html>
```

Love85.html

```
<html>
<body>
<img src="images/love85.gif" width="90" height="30"><br>
<font face="Georgia" size=5>
<p>
<b>Love 85</b> is a Chinese radio station brought to you by your friendly OrangeMedia,
the one and only media company in Orange Land.
</font>
<p>
<font face="Trebuchet MS" size=6>
This station caters to listeners who enjoy <b><i>Chinese</i></b> songs and music from
the 60s, 70s, 80s, and 90s.
</font>
</body>
</html>
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.

Perfect995.html

```
<html>
<body>
<img src="images/perfect995.gif" width="90" height="30"><br>
<font face="Georgia" size=5>
<p>
<b>Perfect 99.5</b> is an English radio station that caters to the young and youthful.
</font>
<p>
<font face="Trebuchet MS" size=6>
This station caters to listeners who enjoy the latest in <b><i>English</i></b> rocks.
</font>
</body>
</html>
```

NOT FOR REDISTRIBUTION

Copyright © 2020, Danny Poo.