CHAPTER

28

GRAPHICS PROGRAMMING USING SWING

In the previous chapter, we learned about Abstract Window Toolkit (AWT), which is used in GUI programming. However, the AWT components internally depend on native methods like C functions and hence problems related to portability arise. Let us have a look at these problems first.

- When a component is created in AWT, it internally calls a native method (for example, a C function) that creates the component internally. This component is called 'peer component'. This peer component is given back and displayed on the screen in AWT. This means AWT internally depends on C code and this is not desirable as we know that C is a system dependant language. AWT is also called 'peer component based' model for this reason.
- The appearance of a component is called its 'look' and how the user interacts with the component is called its 'feel'. The look-and-feel of AWT components change depending on the platform (or operating system). For example, refer code to create a push button in AWT (discussed in Chapter 27). code When this code is executed in Windows it will display windows-type of push button whereas the same code in Unix will display unix-style of push button. It means when a programmer creates a screen with different components, he cannot be sure how his screen will look on a particular system. Its appearance changes from system to system.
- Moreover, AWT components are heavy-weight. It means these components take more system resources like more memory and more processor time.

Due to these reasons, JavaSoft people felt it better to redevelop AWT package without internally taking the help of native methods. Hence, all the classes of AWT are extended to form new classes and a new class library is created. This library is called JFC (Java Foundation Classes).

Java Foundation Classes (JFC)

JFC is an extension of the original AWT. It contains classes that are completely portable, since the entire JFC is developed in pure Java. Noteworthy features of JFC are as follows:

- JFC components are light-weight. This means, they utilize minimum system resources. Their speed is comparatively good and hence JFC programs execute much faster.
- JFC components have same look-and-feel on all platforms. Once a component is created, it looks same on any Operating system. So the programmer can be sure of the look of his screen.
- JFC offers 'pluggable look-and-feel' feature, which allows the programmer to change the look and feel as suited for a platform. Suppose, the programmer wants to display Windows-style push buttons on Windows operating system, and Unix-style buttons on Unix, it is possible.

decides whether to change the model or not. When the model is changed, it tells the view to update itself.

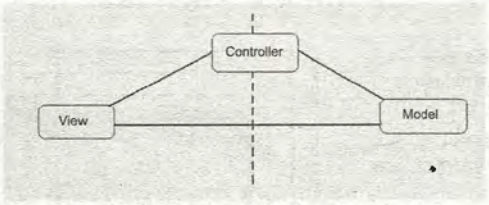


Figure 28.1 MVC architecture

The goal of the MVC architecture is to separate the application object (model), its representation to the user (view), and the way it is controlled by the user (controller). See Figure 28.1.

Before MVC, user interface designs tended to lump such objects together. MVC separates them thus allowing greater flexibility and possibility for reuse. Separating view and model has several advantages, which are as follows:

- **Multiple views using the same model:** Since, the view and model are separated in MVC, it is possible to represent the same model data in several views. For example, the same employee data can be represented in the form of a table or as a histogram or a pie chart. This forms the basis for 'pluggable look and feel' since the user can change the view on different platforms without changing the inner details.
- **Efficient modularity:** It is possible to develop view and model in different environments. For example, we can develop a push button in Java and its model data can be stored using Visual Basic. Also, it is possible to make changes to view or model without affecting the other. Thus debugging will become easy.
- **Easier support for new types of clients:** To support a new type of client, we can simply write a view and controller for it and wire them into the existing enterprise model.

Important Interview Question

Discuss about the MVC architecture in JFC/ swing ?

Model-View-Controller is a model used in swing components. Model represents the data of the component. View represents its appearance and controller is a mediater between the model and the view. MVC represents the separation of model of an object from its view and how it is controlled.

Window Panes

A window pane represents a free area of a window where some text or components can be displayed. For example, we can create a frame using `JFrame` class in `javax.swing`, which contains a free area inside it, as shown in Figure 28.2. This free area is called 'window pane'.

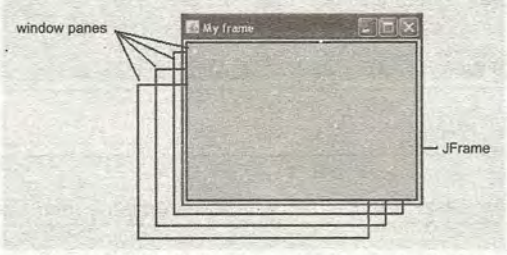


Figure 28.2 Window panes are part of JFrame

We have four types of window panes available in javax.swing package. These panes can be imagi like transparent sheets lying one below the other. So, if we attach any components to any pa they all will be finally displayed on the screen. See Figure 28.3.

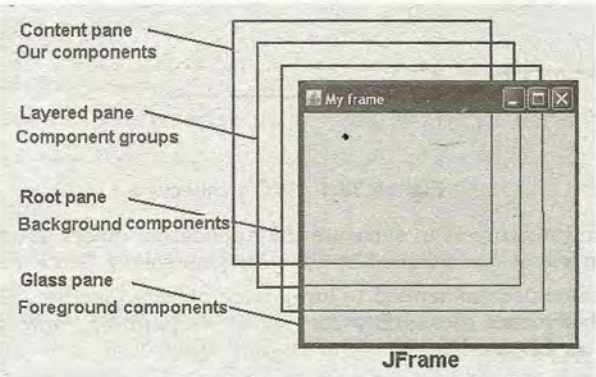


Figure 28.3 Window panes and their use

Let us now have a brief discussion of all the four panes available in swing:

- **Glass pane:** This is the first pane and is very close to the monitor's screen. Any components be displayed in the foreground are attached to this glass pane. To reach this glass pane, we `getGlassPane()` method of `JFrame` class. This method returns Component class object.
- **Root pane:** This pane is below the glass pane. Any components to be displayed in background are displayed in this pane. Root pane and glass pane are used in animation For example, suppose we want to display a flying aeroplane in the sky. The aeroplane can displayed as a `.gif` or `.jpg` file in the glass pane whereas the blue sky can be displayed in root pane in the background. To go to the root pane, we can use `getRootPane()` method `JFrame` class which returns an object of `JRootPane` class.
- **Layered pane:** This pane lies below the root pane. When we want to take several compone as a group, we attach them in the layered pane. We can reach this pane by ca `getLayeredPane()` method of `JFrame` class which returns an object of `JLayeredPane` cl object.
- **Content pane:** This is the bottom most pane of all. Individual components are attached to pane. To reach this pane, we can call `getContentPane()` method of `JFrame` class wh returns Container class object.

Remember, in swing, the components are attached to the window panes only. For example, if want to attach a push button (but object) to the content pane, first of all we should create content pane object by calling `getContentPane()` method as:

```
JFrame jf = new JFrame();  //create JFrame object
Container c = Jf.getContentPane();
//create the content pane, i.e. Container //object
c.add(but);
//add button to content pane
```

Important Interview Question

What are the various window panes available in swing?

There are 4 window panes: Glass pane, Root pane, Layered pane, and Content pane.

Important Classes of javax.swing

Let us have a look at the important classes available in `javax.swing` package. They are shown in Figure 28.4. From the figure, we can understand that the classes of `javax.swing` are derived from the classes of `java.awt` package.

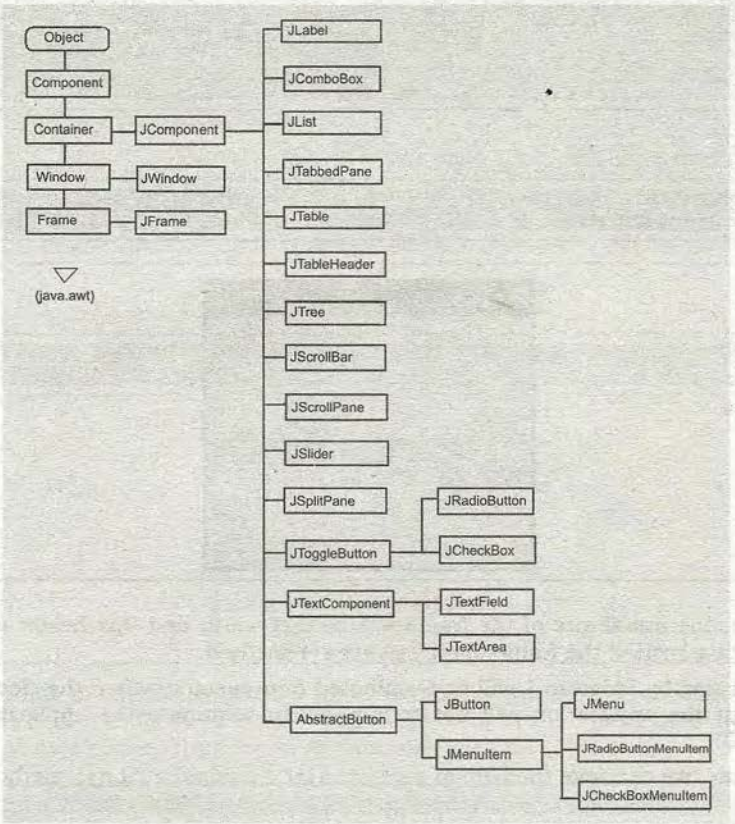


Figure 28.4 Classes of javax.swing

Creating a Frame in Swing

We know that a frame represents a window with a title bar and borders. Frame becomes the basis for creating the screens for an application because all the components go into the frame. To create a frame, we have to create an object to `JFrame` class in swing, as:

- `JFrame jf = new JFrame();  //create a frame without any title`
- `JFrame jf = new JFrame("title");  //create frame with title`

Program 1: Write a program to create a frame by creating an object to `JFrame` class.

```
//A simple frame
import javax.swing.*;
class FrameDemo
{
     public static void main(String args[ ])
```

```
        {
                //create the frame with title
                JFrame obj = new JFrame("My frame");

                //set the size to 200 by 200 px
                obj.setSize(200,200);

                //display the frame
                obj.setVisible(true);

        }
}
```

Output:

```
C:\> javac FrameDemo.java
C:\> java FrameDemo
```

In this program, the initial size of the frame will be 0px width and 0px height and hence it is not visible. So, we have resized the frame using `setSize()` method.

The frame generated by Program 1 will be terminated from memory when the close button is clicked by the user, but the application will not terminate. To terminate the application forcibly, press Control+C in DOS.

To close the frame, we can take the help of `getDefaultCloseOperation()` method of JFrame class as shown here:

```
getDefaultCloseOperation(constant);
```

where the `constant` can be any one of the following:

- `JFrame.EXIT_ON_CLOSE`: This closes the application upon clicking on close button.
- `JFrame.DISPOSE_ON_CLOSE`: This disposes the present frame which is visible on the screen. The JVM may also terminate.
- `JFrame.DO_NOTHING_ON_CLOSE`: This will not perform any operation upon clicking on close button.
- `JFrame.HIDE_ON_CLOSE`: This hides the frame upon clicking on close button.

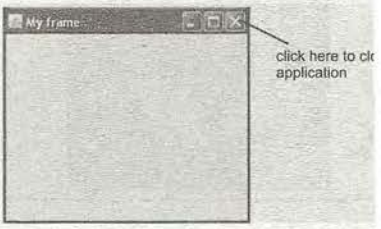
Program 2: Rewrite Program 1 to show how to terminate an application by clicking on the close button of the frame.

```
//A simple frame
import javax.swing.*;
class FrameDemo extends JFrame
{
```

```
        public static void main(String args[])
        {
                //create the frame
                FrameDemo obj = new FrameDemo();

                //set a title for the frame
                obj.setTitle("My swing frame");

                //set the size to 200 by 200 px
                obj.setSize(200,200);

                //display the frame
                obj.setVisible(true);

                //close the application upon clickir
                obj.setDefaultCloseOperation(JFrame.

        }
}
```

Output:

```
C:\> javac FrameDemo.java
C:\> java FrameDemo
```



Let us now think about how to display some text in the frame. V
free area of the frame. To show some background color in the fi
color to the content pane. The code to do this is as follows:

```
c.setBackground(Color.green);  //c represents the
```

Program 3: Write a program to create the frame and display gre

Note

For this purpose, we should set the background color to content pa

```
//A simple frame with background color
import javax.swing.*;
import java.awt.*; //Container class
class FrameDemo extends JFrame
{

        public static void main(String args[])
        {
                //create the frame
                FrameDemo obj = new FrameDemo();

                //create content pane. It is nothing
                Container c = obj.getContentPane();
```

eight and hence it is not

he close button is clicked
pplication forcibly, press

method of JFrame class,

close button.
is visible on the screen.

upon clicking on close

utton.
by clicking on the close

```
                //set green back ground color to c
                c.setBackground(Color.green);

                //set a tsitle for the frame
                obj.setTitle("My swing frame");

                //set the size to 200 by 200 px
                obj.setSize(200,200);

                //display the frame
                obj.setVisible(true);

                //close the application upon clicking on close button of frame
                obj.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        }
}
```

Output:

```
C:\> javac FrameDemo.java
C:\> java FrameDemo
```

Displaying Text in Frame

Swing package provides us the following two ways to display text in an application:

- `paintComponent(Graphics g)` method of `JPanel` class is used to paint the portion of component in swing. We should override this method in our class. In the following example, we are writing our class `MyPanel` as a subclass to `JPanel` and override the `paintComponent` method as:

```
class MyPanel extends JPanel
{
        public void paintComponent(Graphics g)
        {
                super.paintComponent(g);  //call JPanel's method
                g.setColor(Color.red);
                g.setFont(new Font("Helvetica", Font.BOLD, 34));
                g.drawString("Hello Learners!", 50,100);

        }
}
```

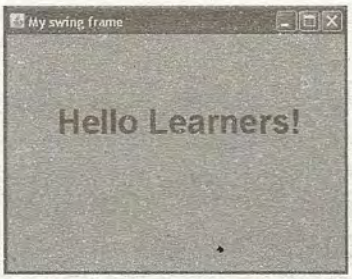
But, we should again call the `paintComponent()` method of `JPanel` from our method using `super.paintComponent()`. This will enable the super class to paint the component's area. The following program depicts how to use `paintComponent()` method to display some text in the frame

Program 4: Write a program to display text in the frame by overriding `paintComponent()` method of `JPanel` class.

```
//A simple frame with background color and text
import javax.swing.*;
import java.awt.*; //Container class
class MyPanel extends JPanel
{
	MyPanel()
	{
		this.setBackground(Color.green);
	}
	public void paintComponent(Graphics g)
	{
		super.paintComponent(g);
		g.setColor(Color.red);
		g.setFont(new Font("Helvetica", Font.BOLD, 34));
		g.drawString("Hello Learners!", 50,100);

	}
}
class FrameDemo extends JFrame
{
	FrameDemo()
	{
		//create content pane
		Container c = this.getContentPane();

		//create MyPanel object and add it to c
		MyPanel mp = new MyPanel();
		c.add(mp);
	}

	public static void main(String args[])
	{
		//create the frame
		FrameDemo obj = new FrameDemo();

		//set a title for the frame
		obj.setTitle("My swing frame");

		//set the size to 300 by 300 px
		obj.setSize(300,300);

		//display the frame
		obj.setVisible(true);

		//close the application upon clicking on close button of frame
		obj.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

	}
}
```

Output:

```
C:\> javac FrameDemo.java
C:\> java FrameDemo
```



- The second way to display some text in swing frame is by using a label. A label represents so constant text to be displayed in the frame. We can use `JLabel` class to create a label as:

```
JLabel lbl = new JLabel("text");
```

Now, this label should be attached to the content pane to display the text in the frame. This shown in the following program.

Program 5: Write a program to display some text in the frame with the help of a label.

```
//A simple frame with background color and text
import javax.swing.*;
import java.awt.*; //Container class

class FrameDemo extends JFrame
{
      //vars
      JLabel lbl;

      FrameDemo()
      {
            //create content pane
            Container c = this.getContentPane();

            //set the layout manager to c
            c.setLayout(new FlowLayout());

            //set background color for content pane c
            c.setBackground(Color.green);

            //create a label with some text
            lbl = new JLabel("Hello Learners!");

            //set font for label
            lbl.setFont(new Font("Helvetica", Font.BOLD, 34));

            //set red color for label
            lbl.setForeground(Color.red);

            //add the label to content pane
            c.add(lbl);

      }

      public static void main(String args[])
      {
            //create the frame
            FrameDemo obj = new FrameDemo();

            //set a title for the frame
            obj.setTitle("My swing frame");
```

```
                //set the size to 300 by 300 px
                obj.setSize(300,300);

                //display the frame
                obj.setVisible(true);

                //close the application upon clicking on close button of frame
                obj.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        }
}
```

Output:

```
C:\> javac FrameDemo.java
C:\> java FrameDemo
```

JComponent Class Methods

`JComponent` class of `javax.swing` package is the subclass of the `Component` class of `java.awt`. So, whatever methods are available in `Component` class are also available to `JComponent`. This is the reason why almost all the methods of AWT are useful in swing also. Additional methods found in `JComponent` are also applicable for the components created in swing.

When a component is created, to display it in the frame, we should not attach it to the frame directly as we did in AWT. On the other hand, the component should be attached to a window pane. For example, to add the component to the content pane, we can write, as:

```
c.add(component);
where c represents the content pane which is represented by Container object.
```

Similarly, to remove the component, we can use `remove()` method, as:

```
c.remove(component);
```

`removeAll()` method removes all the components from the content pane and can be used as:

```
c.removeAll();
```

When components are to be displayed in the frame, we should first set a layout manager which arranges the components in a particular manner in the frame. Layout manager should be set for the content pane, as:

```
c.setLayout(new FlowLayout());
```

Here, c is the Container class object which represents the content pane.

The following methods of JComponent class are very useful while handling the components:

- To set some background color to the component, we can use setBackground() method, as:

```
component.setBackground(Color.yellow);
```

- To set the foreground color to the component, we can use setForeground() method, as:

```
component.setForeground(Color.red);
```

- To set some font for the text displayed on/in the component, we can use setFont() method. We should pass Font class object to this method, as:

```
component.setFont(Font obj);
```

where, Font class object can be created as:

```
Font obj = new Font("fontname", style, size);
```

For example, Font obj =new Font("Dialog", Font.BOLD, 30);

- Tool tip text is the text that is displayed automatically when the mouse is placed on the component. Tool tip text helps to provide some help text about the component. To set the tool tip text, we can use setToolTipText() method, as:

```
component.setToolTipText("This is a swing component.");
```

- Sometimes when the mouse is not useful, the user can use the keyboard to interact with the components. For example, to invoke 'Cancel' button, the user can press Alt+C where C represents the first character of the label of the button. This 'C' is called short cut key or mnemonic. To set a mnemonic, we can use setMnemonic() method, as:

```
component.setMnemonic('C');
```

The preceding statement means that the component can be activated by pressing Alt+C on the keyboard.

- To enable or disable a component, we can use setEnabled() method, as:

```
component.setEnabled(true);
```

Here, true will enable the component to be ready to perform an action, whereas false will disable it.

- To make a component to be visible or invisible, we can use setVisible() method, as:

```
component.setVisible(true);
```

Here, true will make component to be visible, whereas false will make it invisible.

- To know the border of a component, we can use `getBorder()` method, as:

```
component.getBorder();
```

This method returns Border object which represents the border of the component. It returns null if no border is set.

- To know the current height of the component, use `getHeight()` method, as:

```
component.getHeight();
```

This method returns an integer which represents height of component in pixels.

- To know the current width of the component, use `getWidth()` method, as:

```
component.getWidth();
```

This method returns an integer which represents the width of component.

- To know the current x coordinate of the component,

```
component.getX();
```

This method returns integer which represents x coordinate of component in pixels.

- To know the current y coordinate of the component,

```
component.getY();
```

This method returns integer which represents y coordinate of component.

- To set location of the component in the frame, we can use `setBounds()` method, as:

```
component.setBounds(x, y, width, height);
```

This method specifies the x,y coordinates of the component, which represent the width and height of the rectangular area allotted for displaying the component.

- It is possible to set border around the components in swing. The `javax.swing.BorderFactory` class has got methods which are useful to set different borders for the component. These methods return Border interface object. Border interface belongs to `javax.swing.border` package. Border object should be passed to `setBorder()` method to set the border to the component, as:

```
component.setBorder(Border obj);
```

The following are `BorderFactory` methods:

- `createBevelBorder()` Method: To set bevel border, we can use `createBevelBorder()` method. Bevel border is the border which either raises or lowers around the component in 3D form. It has the following forms:

```
BorderFactory.createBevelBorder(BevelBorder.RAISED);
BorderFactory.createBevelBorder(BevelBorder.LOWERED);
```

The above methods will draw bevel border with current background color for highlighting and a bit darker color for shading the shadows around the component.

```
BorderFactory.createBevelBorder(BevelBorder.RAISED, Color.red, Color.green);
```

This method uses raised bevel border with red color for highlighting and green for shading purpose.

- `createEtchedBorder()` Method: Etched border is the border around the component which appears with a shade. To set etched border, we use `createEtchedBorder()` method, as:

```
BorderFactory.createEtchedBorder();
```

The above method creates a border with an etched look using the component's current background color for highlighting and shading.

```
BorderFactory.createEtchedBorder(Color.red, Color.green);
```

This method cretes etched border with red color for high lighting and green for shading purpose.

```
BorderFactory.createEtchedBorder(EtchedBorder.RAISED);
BorderFactory.createEtchedBorder(EtchedBorder.LOWERED);
```

These methods create raised or lowered etched borders.

```
BorderFactory.createEtchedBorder(EtchedBorder.RAISED, Color.red, Color.green);
```

The above method creates an etched border with raised border in red color for highlighting and green for shading.

- `createLineBorder()` Method: To set a simple line as border around the component, we can use `createLineBorder()` method, as:

```
BorderFactory.createLineBorder(Color.red);
```

This method creates a line border with red color.

```
BorderFactory.createLineBorder(Color.red, 5);
```

This method creates a line border with red color and 5 px thickness of the border.

- `createMatteBorder()` Method: Matte border is like line border except that it can be distributed unevenly around the component. To set matte border, we can use `createMatteBorder()` method, as:

```
BorderFactory.createMatteBorder(5,10,15,20, Color.red);
```

This method creates a matte border with thickness 5,10,15,20 px at top, left, bottom, and right of the component. The border appears in red color.

- `createCompoundBorder()` Method: Compound border can use other borders at outside and inside edges of component. To set a compound border, we can use `createCompoundBorder()` method, as:

```
BorderFactory.createCompoundBorder();
```

This will create a compound border without using any other borders at edges of the component.

```
BorderFactory.createCompoundBorder(Border out, Border in);
```

This method creates a compound border with 'out' at outside edge of the component and 'in' at inside edge of component.

- `createEmptyBorder()` Method: To set an empty border which does not take any space around the component, we can use `createEmptyBorder()` method, as:

```
BorderFactory.createEmptyBorder();
```

This creates an empty border without any space around the component.

```
BorderFactory.createEmptyBorder(5,10,15,20);
```

This creates an empty border that takes up space but which does no drawing, specifying the width of the top, left, bottom, and right sides as 5,10,15, and 20 px.

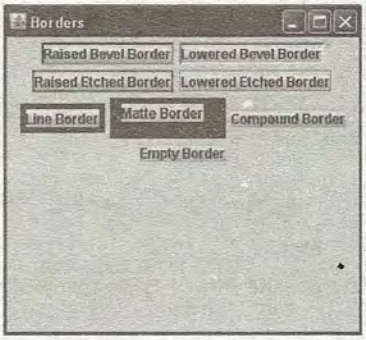
Program 6: Write a program to create some push buttons using `JButton` class and draw different borders around the buttons.

```
//Understanding the borders
import java.awt.*;
import javax.swing.*;
import javax.swing.border.*;

class BorderDemo extends JFrame
{
      //vars
      JButton b1,b2,b3,b4,b5,b6,b7,b8;

      BorderDemo()
      {
            //create content pane c
            Container c = getContentPane();

            //set a layout for content pane
            c.setLayout(new FlowLayout());

            //Create push buttons
            b1 = new JButton("Raised Bevel Border");
            b2 = new JButton("Lowered Bevel Border");
            b3 = new JButton("Raised Etched Border");
            b4 = new JButton("Lowered Etched Border");
            b5 = new JButton("Line Border");
            b6 = new JButton("Matte Border");
            b7 = new JButton("Compound Border");
            b8 = new JButton("Empty Border");

            //set raised bevel border for b1 with high light color:
             red and shadow color:
            //green
            Border bd = BorderFactory.createBevelBorder(BevelBorder.RAISED,
             Color.red, Color.green);
            b1.setBorder(bd);

            //set lowered bevel border for b2 with its current background
             color for
            //hightlight and shadow
            bd = BorderFactory.createBevelBorder(BevelBorder.LOWERED);
            b2.setBorder(bd);

            //set raised etched border for b3 with high light color:red
             and shadow color:
```

```
                //green
                bd = BorderFactory.createEtchedBorder(EtchedBorder.RAISED,
                 Color.red, Color.green);
                b3.setBorder(bd);

                //set lowered etched border for b4 with its current background
                 color for
                //highlight and shadow
                bd = BorderFactory.createEtchedBorder(EtchedBorder.LOWERED);
                b4.setBorder(bd);

                //set line border for b5 with red color and width 5 px
                bd = BorderFactory.createLineBorder(Color.red, 5);
                b5.setBorder(bd);

                //set matte border for b6 with top, left, bottom, right widths as
                 5,10,15,20 px
                //and in red color
                bd = BorderFactory.createMatteBorder(5,10,15,20, Color.red);
                b6.setBorder(bd);

                //set compound border for b7 without any borders inside or outside
                 edges
                bd = BorderFactory.createCompoundBorder();
                b7.setBorder(bd);

                //set empty border for b8 without any space for border
                bd = BorderFactory.createEmptyBorder();
                b8.setBorder(bd);
                //add the buttons to the container
                c.add(b1);
                c.add(b2);
                c.add(b3);
                c.add(b4);
                c.add(b5);
                c.add(b6);
                c.add(b7);
                c.add(b8);
                //close the frame upon clicking
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
        public static void main(String args[])
        {
                //create a frame
                BorderDemo obj = new BorderDemo();
                //set the title and size for frame
                obj.setTitle("Borders");;
                obj.setSize(500,400);
                //display the frame
                obj.setVisible(true);
        }
}
```

Output:

```
C:\> javac BorderDemo.java
C:\> java BorderDemo
```



Important Interview Question

Where are the borders available in swing?

All borders are available in BorderFactory class in javax.swing.border package.

Creating a Push Button with All Features

Let us create a push button in swing and apply different features available in swing like setting colors and font for the button, setting border, short cut key, and tool tip text. We can create a push button using `JButton` class in swing, as:

```
JButton b = new JButton("OK");
Here, a push button with the label "OK"  will be created.
JButton b = new JButton(ImageIcon ii);
This will create a push button with an image on it. The image is specified by
ImageIcon class object.
JButton b = new JButton("OK", ImageIcon ii);
```

Here, the button is created with label "OK" and image `ii`.

It is possible to create components in swing with images on it. It is also possible to create a component in/on another component in swing.

Program 7: Write a program in which we create a push button with a label and image on it and then set different features for the button.

Note

This push button will not perform any action, since there is no action attached to it.

```
//Button with an image, colors, border, tool tip text and shortcut key
import java.awt.*;
import javax.swing.*;
import javax.swing.border.*;

class ButtonDemo extends JFrame
{
      JButton b;

      ButtonDemo()
      {
            //create container
            Container c = getContentPane();
```

```
                //set a layout for container
                c.setLayout(new FlowLayout());

                //store the image into ImageIcon object
                ImageIcon ii = new ImageIcon("car2.gif");

                //create the button with the image
                b = new JButton("Click Me", ii);

                //set background and foreground colors for button
                b.setBackground(Color.yellow);
                b.setForeground(Color.red);

                //set font for the label of button
                b.setFont(new Font("Arial", Font.BOLD, 30));

                //set bevel border for button
                Border bd = BorderFactory.createBevelBorder(BevelBorder.RAISED);
                b.setBorder(bd);
                //set tool tip text for button
                b.setToolTipText("This is a button");
                //set a short cut key for button. Alt+C from keyboard will invoke
                 the button
                b.setMnemonic('C');
                //add the button to the container
                c.add(b);
                //close the frame upon clicking
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
        public static void main(String args[])
        {
                //create a frame
                ButtonDemo obj = new ButtonDemo();
                obj.setTitle("My Button");;
                obj.setSize(500,400);
                obj.setVisible(true);
        }
}
```

Output:

```
C:\> javac ButtonDemo.java
C:\> java ButtonDemo
```

Displaying Image in Swing

Suppose we want to make the push button to perform some action when clicked by the user. For this purpose, we should add action listener to the button and implement `actionPerformed()` method. When the button is clicked, this method is executed. So the action part should be written in this method.

Let us now see how to display an image when the button is clicked. First of all, we take an empty label. A label can be used to display some constant text or an image in the frame. For example,

```
JLabel lbl = new JLabel() ;
```

The above statement will create an empty label. Later, we can send text to this label using `setText()` method. We can also set some image to this label using `setIcon()` method.

```
JLabel lbl = new JLabel("text");
```

This will create the label with text.

```
JLabel lbl = new JLabel("text", alignment);
```

This will create a label with text and the text will be aligned. The alignment can be `JLabel.CENTER`, `JLabel.LEFT`, and `JLabel.RIGHT`.

```
JLabel lbl = new JLabel(ImageIcon ii);
```

This will create a label with an image on it. The image is provided by the ImageIcon object.

```
JLabel lbl = new JLabel("text", ImageIcon ii);
```

This will create the label with text as well as image.

Program 8: Write a program to create a push button as we did in Program 7. When the button is clicked an image is displayed in the frame.

```
//Button which displays image when clicked
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;

class ButtonDemo1 extends JFrame implements ActionListener
{
     JButton b;
     JLabel lbl;

     ButtonDemo1()
     {
          //create container
          Container c = getContentPane();

          //set a layout for container
          c.setLayout(new FlowLayout());

          //store the image into ImageIcon object
          ImageIcon ii = new ImageIcon("car2.gif");

          //create the button with the image
          b = new JButton("Click Me", ii);

          //set background and foreground colors for button
          b.setBackground(Color.yellow);
          b.setForeground(Color.red);

          //set font for the label of button
          b.setFont(new Font("Arial", Font.BOLD, 30));

          //set bevel border for button
```

```
                Border bd = BorderFactory.createBevelBorder(BevelBorder.RAISED);
                b.setBorder(bd);

                //set tool tip text for button
                b.setToolTipText("This is a button");

                //set a short cut key for button. Alt+C will invoke the button
                b.setMnemonic('C');
                //add the button to the container
                c.add(b);

                //add action listener to button
                b.addActionListener(this);

                //create an empty label and add to the content pane
                lbl = new JLabel();
                c.add(lbl);

                //close the frame upon clicking
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }

        public void actionPerformed(ActionEvent ae)
        {
                //set some image to the label. This image is displayed when the
                 button is
                //clicked
                ImageIcon ii = new ImageIcon("car2.gif");
                lbl.setIcon(ii);
        }

        public static void main(String args[])
        {
                //create a frame
                ButtonDemo1 obj = new ButtonDemo1();

                obj.setTitle("My Button");;
                obj.setSize(500,400);
                obj.setVisible(true);
        }
}
```

Output:

```
C:\> javac ButtonDemo1.java
C:\> java ButtonDemo1
```

Creating Components in Swing

Let us now discuss some common components and how to create them in swing. Already we have seen how to create a push button and a label. To create check boxes, we can use `JCheckBox` class, as:

```
JCheckBox cb = new JCheckBox();
```

This statement creates a check box without any label and image.

```
JCheckBox cb = new JCheckBox("label");
```

This creates a check box with the label.

```
JCheckBox cb = new JCheckBox(ImageIcon ii);
```

This creates a check box with an image. The image is loaded from `ImageIcon` object.

```
JCheckBox cb = new JCheckBox("label", ImageIcon ii);
```

This creates the check box with label and image.

```
JCheckBox cb = new JCheckBox("label", status);
```

Here, `status` can be 'true' or 'false'. If it is 'true', the check box appears as if it is selected by default.

To know which check box is selected by the user, we should first go to the model of the check box by calling `getModel()` method,as:

```
Model m = cb.getModel();
```

Now this model object contains the information on whether the check box is selected or not. This information can be obtained by using `isSelected()` method on Model object as:

```
boolean x = m.isSelected();
```

This method returns true if the check box is selected, otherwise it returns false.

We can combine both the methods and write, as:

```
cb.getModel().isSelected();
```

To create radio buttons, we can use `JRadioButton` class, as:

```
JRadioButton rb = new JRadioButton();
```

This will create a radio button without any label or image.

```
JRadioButton rb = new JRadioButton("label");
```

This creates a radio button with the indicated label.

```
JRadioButton rb = new JRadioButton(ImageIcon ii);
```

This creates a radio button with an image.

```
JRadioButton rb = new JRadioButton("label",
```

This creates a radio button with label and image.

```
JRadioButton rb = new JRadioButton("label",
```

This creates a radio button with label and the status can
button appears as if it is selected by default.

After creating the radio buttons, we should add them to B
buttons form a group and hence JVM will allow the user
This can be done as:

```
ButtonGroup bg = new ButtonGroup();
bg.add(rb);  //add radio button to button gr
```

To know whether a radio button is selected by the user or

```
rb.getModel().isSelected();
```

This method gives true if the radio button is selected other

To create a text field, we can take the help of JTextField

```
JTextField tf = new JTextField();
```

This constructs a text field without any text within it.

```
JTextField tf = new JTextField("text");
```

This creates a text field with text displaying in it.

```
JTextField tf = new JTextField(15);
```

This creates a text field with a width of 15 characters.

```
JTextField tf = new JTextField("text", 15);
```

This creates a text field with text and a width of 15 charac

A text field can accommodate only one line of text, wherea
lines of text. A text area can be created as:

```
JTextArea ta = new JTextArea();
```

This creates a text area without any text in it.

```
JTextArea ta = new JTextArea("text");
```

This creates a text area with text within it.

```
JTextArea ta = new JTextArea(5, 15);
```

This creates a text area with 5 rows and 15 characters in each row.

```
JTextArea ta = new JTextArea("text", 5, 15);
```

This creates a text area with text and with 5 rows and 15 characters per row.

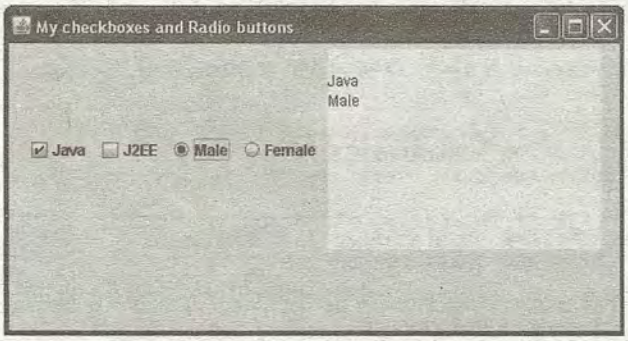
Program 9: Write a program that helps in creating some check boxes and radio buttons. When the user clicks on a check box or radio button, the selected option text will be displayed in a text area.

```
//Check boxes, radio buttons and Text area
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

class CheckRadio extends JFrame implements ActionListener
{
      //vars
      JCheckBox cb1, cb2;
      JRadioButton rb1, rb2;
      JTextArea ta;
      ButtonGroup bg;
      String msg="";

      CheckRadio()
      {
            //create the content pane
            Container c = getContentPane();

            //set flow layout to content pane
            c.setLayout(new FlowLayout());

            //create a text area with 10 rows and 20 chars per row
            ta = new JTextArea(10,20);

            //create two check boxes
            cb1 = new JCheckBox("Java", true);
            cb2 = new JCheckBox("J2EE");

            //create two radio buttons
            rb1 = new JRadioButton("Male", true);
            rb2 = new JRadioButton("Female");

            //create a button group and add the radio buttons to it
            bg = new ButtonGroup();
            bg.add(rb1);
            bg.add(rb2);

            //add the checkboxes, radio buttons, textarea to the container
            c.add(cb1);
            c.add(cb2);
            c.add(rb1);
            c.add(rb2);
            c.add(ta);

            //add action listeners. We need not add listener to text area
            //since the user clicks on the checkboxes or radio buttons only
            cb1.addActionListener(this);
            cb2.addActionListener(this);
            rb1.addActionListener(this);
            rb2.addActionListener(this);

            //close the frame upon clicking
            setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      }

      public void actionPerformed(ActionEvent ae)
```

```
        {
                //know which components are selected by user
                if(cb1.getModel().isSelected()) msg+="\nJava";
                if(cb2.getModel().isSelected()) msg+="\nJ2EE";
                if(rb1.getModel().isSelected()) msg+="\nMale";
                else msg+="\nFemale";
                //display the selected message in text area
                ta.setText(msg);

                //reset the message to empty string
                msg="";
        }

        public static void main(String args[])
        {
                //create frame
                CheckRadio cr = new CheckRadio();
                cr.setTitle("My checkboxes and Radio buttons");
                cr.setSize(500,400);
                cr.setVisible(true);
        }
}
```

Output:

```
C:\> javac CheckRadio.java
C:\> java CheckRadio
```



Setting the Look and Feel of Components

Swing provides the programmer the facility to change the look and feel of components bei displayed on any system. This is called 'plaf' (pluggable look and feel). 'Look' refers to t appearance of the component on the screen and 'feel' represents how the user can interact with t component. There are 3 types of look and feel available in swing—Metal look and feel, Motif lo and feel, and Window look and feel. They are defined as classes in `javax.swing.plaf` package. default, swing programs use 'metal' look and feel. It is possible to set any look and feel for t swing components. For this purpose, we should use `UIManager.setLookAndFeel()` method. this method we should pass one of the following strings:

- For getting metal look and feel, we use:

```
"javax.swing.plaf.metal.MetalLookAndFeel"
```

- For getting motif look and feel, we use:

```
"com.sun.java.swing.plaf.motif.MotifLookAndFeel"
```

- For getting windows look and feel, we use:

```
"com.sun.java.swing.plaf.windows.WindowsLookAndFeel"
```

After this, we should update the contents on the content pane, using `updateComponentTreeUI()` method, as:

```
SwingUtilities.updateComponentTreeUI(c);
```

Where, `c` represents the content pane which is nothing but the Container object.

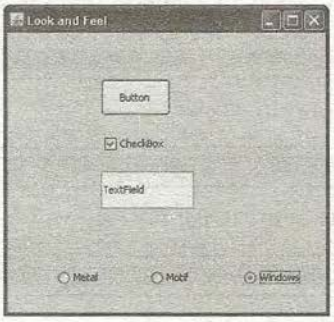
Program 10: Write a program that changes the look and feel of the component.

```
//Changing the look and feel of components
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.plaf.*;
class LookFeel extends JFrame implements ItemListener
{
      //vars
      JButton b;
      JCheckBox cb;
      JTextField t;
      JRadioButton r1, r2, r3;
      ButtonGroup bg;
      Container c;

      LookFeel()
      {
            //create content pane
            c = this.getContentPane();

            //set flow layout to c
            c.setLayout(null);

            //create components
            b = new JButton("Button");
            cb = new JCheckBox("CheckBox");
            t = new JTextField("TextField", 15);
            r1 = new JRadioButton("Metal");
            r2 = new JRadioButton("Motif");
            r3 = new JRadioButton("Windows");

            //create ButtonGroup object and add radio buttons to specify
            //that they belong to same group
            bg = new ButtonGroup();
            bg.add(r1);
            bg.add(r2);
            bg.add(r3);

            //set the location of components in content pane
            b.setBounds(100,50,75,40);
            cb.setBounds(100,100,100,40);
            t.setBounds(100,150,100,40);
            r1.setBounds(50,250,100,30);
            r2.setBounds(150,250,100,30);
            r3.setBounds(250,250,100,30);

            //add the components to content pane
```

```
                c.add(b);
                c.add(cb);
                c.add(t);
                c.add(r1);
                c.add(r2);
                c.add(r3);
                //add item listeners to radio buttons
                r1.addItemListener(this);
                r2.addItemListener(this);
                r3.addItemListener(this);

                //close the frame
                this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }

        public void itemStateChanged(ItemEvent ie)
        {
                try{
                //know which radio button is selected and accordingly change
                 the look and
                // feel

                if(r1.getModel().isSelected())
                UIManager.setLookAndFeel("javax.swing.plaf.metal.
                 MetalLookAndFeel");
                if(r2.getModel().isSelected())
                UIManager.setLookAndFeel(
                 "com.sun.java.swing.plaf.motif.MotifLookAndFeel");
                if(r3.getModel().isSelected())
                UIManager.setLookAndFeel
                ("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");

                //change the look and feel in the content pane
                SwingUtilities.updateComponentTreeUI(c);
                }catch(Exception e){}
        }

        public static void main(String args[])
        {
                //create the frame
                LookFeel lf = new LookFeel();
                lf.setSize(400,400);
                lf.setTitle("Look and Feel");
                lf.setVisible(true);
        }
}
```

Output:

```
C:\> javac LookFeel.java
C:\> java LookFeel
```



In this program, we create 3 components: a push button, a check box and a text field. Their look and feel will be changed when the user selects one of the radio buttons labeled as 'Metal', 'Motif' and 'Windows'.

JTable Class

`JTable` class is useful to create a table. A table represents several rows and columns of data. To create a table, we can create JTable class object as:

```
JTable tab = new JTable(data, columnnames);
```

Here, `data` represents the data of the table in the form of a two-dimensional array and `columnnames` represents the names of the individual columns that can be a one-dimensional array. Alternately, we can also use a Vector class object to represent data and another Vector object to represent columnnames.

Suppose we want to create a row for the table using a Vector, we can create an object to Vector class as:

```
Vector row = new Vector();
```

Now, using `add()` method, we can add column data to this row, as:

```
row.add(columndata);
```

Now this row should be added to the table's data part, as:

```
Vector data = new Vector();
data.add(row);
```

Let us see the methods which help us to work with `JTable`.

- To know the number of rows in the table:

```
int n = tab.getRowCount();
```

- To know the number of columns in the table:

```
int n = tab.getColumnCount();
```

- To know the name of the column when column position number is given:

```
String name = tab.getColumnName(int columnnumber);
```

- To know the height of a table row, in pixels:

```
int n = tab.getRowHeight();
```

- To set the rows height in pixels in the table:

```
tab.setRowHeight(int height);
```

- To know the index of the first selected column:

```
int n = tab.getSelectedColumn();
```

This method returns `-1` if no column is selected.

- To know the indices of all selected columns:

```
int x[] = tab.getSelectedColumns();
```

- To know the index of the first selected row:

```
int n = tab.getSelectedRow();
```

This method returns -1 if no row is selected.

- To know the indices of all selected rows:

```
int x[] = tab.getSelectedRows();
```

- To know which object is there in the table at particular row and column position:

```
Object x = tab.getValueAt(int row, int column);
```

- To set an object in a particular row and column of the table:

```
tab.setValueAt(Object obj, int row, int column);
```

- A grid represents rows and columns of the table. To set the grid color:

```
tab.setGridColor(Color.red);
```

- To return the table header used by the table, we can use `getTableHeader()` method as:

```
JTableHeader  head = tab.getTableHeader();
```

Please note that `JTableHeader` class is defined in `javax.swing.table` package.

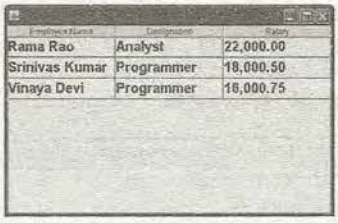
Program 11: Write a program that creates a table with some rows and columns.

```
//JTable demo
import java.awt.*;  //Container
import javax.swing.*;  //JTable
import javax.swing.table.*;  //JTableHeader
import javax.swing.border.*; //Border
import java.util.*; //Vector

class JTableDemo extends JFrame
{

      JTableDemo()
      {
            //take Vector object to represent data of table
            Vector<Vector> data= new Vector<Vector>();

            //take another Vector object to represent a row
            Vector<String> row= new Vector<String>();

            //add 3 column's data to row
            row.add("Rama Rao");
            row.add("Analyst");
            row.add("22,000.00");

            //add the row to data of the table
            data.add(row);

            //create another row
            row= new Vector<String>();
            row.add("Srinivas Kumar");
            row.add("Programmer");
            row.add("18,000.50");

            //add the second row also to data
            data.add(row);

            //create third row
            row= new Vector<String>();
```

```
                row.add("Vinaya Devi");
                row.add("Programmer");
                row.add("16,000.75");

                //add the second row also to data
                data.add(row);

                //Create another vector object for column names
                Vector<String> cols = new Vector<String>();
                cols.add("Employee Name");
                cols.add("Designation");
                cols.add("Salary");

                //do not add column names to data of table

                //create the table
                JTable tab = new JTable(data,cols);

                //set green line border to the table
                tab.setBorder(BorderFactory.createLineBorder(Color.green, 2));

                //set some font to the table
                tab.setFont(new Font("Arial", Font.BOLD, 20));

                //set row height to 30px
                tab.setRowHeight(30);

                //set grid color to red
                tab.setGridColor(Color.red);

                //get the table header into head
                JTableHeader head = tab.getTableHeader();

                //create content pane
                Container c = getContentPane();

                //set border layout to content pane
                c.setLayout(new BorderLayout());

                //add head of the table at top and remaining table below the top
                c.add("North",head);
                c.add("Center",tab);

        }

        public static void main(String args[])
        {
                //create the frame
                JTableDemo demo = new JTableDemo();
                demo.setSize(500,400);
                demo.setVisible(true);

                //close the frame
                demo.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        }
}
```

Output:

```
C:\> javac JTableDemo.java
C:\> java JTableDemo
```



In this program, to create rows of the table, we are using Vector class objects. These objects are stored again in another Vector class object which forms the data for the table.

In Program 11, we used `BorderLayout` to arrange the table header and its body in the content pane. `BorderLayout` is useful to arrange the components in the four borders of the screen identified as North, East, South and West and also in the center of the screen which is identified as Center. This is shown in Figure 28.5. In this program, the table header is added in the North and the table in the Center of the content pane. Also note that in the output, the columns can be dragged and moved from one place to another place in the table. For example, the first column 'Employee Name' can be swapped with the second column 'Designation' by dragging and dropping it on the second column.

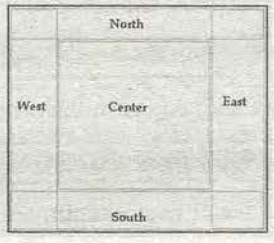


Figure 28.5 BorderLayout

JTabbedPane Class

A pane represents a frame area. A tabbed pane represents a frame with tabs attached to it. `JTabbedPane` is useful to create a tabbed pane, such that on each tab sheet a group of components can be added. The user can choose any component from the tab sheet.

To create a tabbed pane, we can simply create an object to `JTabbedPane`, as:

```
JTabbedPane jtp = new JTabbedPane();
```

The following statement creates an empty tabbed pane with the specified tab placement of any of these—JTabbedPane.TOP, JTabbedPane.BOTTOM, JTabbedPane.LEFT, or JTabbedPane.RIGHT.

```
JTabbedPane jtp = new JTabbedPane(int tabplacement);
```

The following methods can be used to work with JTabbedPane:

- To add tab sheets to the tabbed pane:

```
jtp.add("title", object);
jtp.addTab("title", object);
jtp.addTab("title", ImageIcon ii, object);
```

Here, `title` represents tab sheet title and `object` represents a component or it may represent a group of components collectively. In the later case, it represents an object of `JPanel` class which contains a group of components. To create an object of `JPanel`, we should first create a class that extends `JPanel` and create an object to that class. For example,

```
class MyPanel extends JPanel
```

Now, pass `MyPanel` class's object to `addTab()` method.

- To remove a tab and its components from the tabbed pane:

```
jtp.removeTabAt(int index);
```

- To remove a specified component from the tabbed pane:

```
jtp.remove(Component c);
```

- To remove all the tabs and their corresponding components:

```
jtp.removeAll();
```

- To get the component when the position of the component is given:

```
Component c = jtp.getComponentAt(int index);
```

- To get the currently selected component object in the tabbed pane:

```
Component c = jtp.getSelectedComponent();
```

- To get the selected component's index or position number:

```
int x = jtp.getSelectedIndex();
```

- To know the number of tab sheets present in the tabbed pane:

```
int x = jtp.getTabCount();
```

- To set a component at a particular position in the tabbed pane:

```
jtp.setComponentAt(int index, Component c);
```

Program 12: Write a program to create a tabbed pane with two tab sheets. In the first tab sheet, we display some push buttons with names of capital cities. In the second tab sheet, we display some checkboxes with names of countries.

```
//Tabbed Pane
import java.awt.*;
import javax.swing.*;

class JTabbedPaneDemo extends JFrame
{
      JTabbedPaneDemo()
      {
            //create content pane
            Container c = getContentPane();

            //create tabbed pane
```

```
                JTabbedPane jtp = new JTabbedPane();

                //add two tab sheets. CapitalsPanel and CountriesPanel are classes
                 which
                //extend JPanel and contain a group of components.
                jtp.addTab("Capitals", new CapitalsPanel());
                jtp.addTab("Countries", new CountriesPanel());

                //add the tabbed pane to content pane
                c.add(jtp);
        }

        public static void main(String args[])
        {
                //create the frame
                JTabbedPaneDemo demo = new JTabbedPaneDemo();
                demo.setTitle("JTabbed pane");
                demo.setSize(300,400);
                demo.setVisible(true);

                //close frame
                demo.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
}

//the components of this class go into Capitals tab sheet
class CapitalsPanel extends JPanel
{
        CapitalsPanel()
        {
                //create 3 push buttons and add to panel
                JButton b1 = new JButton("Washington");
                JButton b2 = new JButton("London");
                JButton b3 = new JButton("Tokyo");

                add(b1);
                add(b2);
                add(b3);

        }
}
//the components of this class appear in Countries tab sheet
class CountriesPanel extends JPanel
{
        CountriesPanel()
        {
                //create 3 check boxes and add to panel
                JCheckBox c1 = new JCheckBox("UnitedStates");
                JCheckBox c2 = new JCheckBox("Britain");
                JCheckBox c3 = new JCheckBox("Japan");

                add(c1);
                add(c2);
                add(c3);
        }
}
```

Output:

```
C:\> javac JTabbedPaneDemo.java
C:\> java JTabbedPaneDemo
```

JSplitPane Class

`JSplitPane` is used to create a split pane which divides two (and only two) components.

To create a split pane:

```
JSplitPane sp = new JSplitPane(orientation, component1, component2);
```

Here, orientation is:

`JSplitPane.HORIZONTAL_SPLIT` to align the components from left to right.

`JSplitPane.VERTICAL_SPLIT` to align the components from top to bottom.

The following methods are useful to work with `JSplitPane`:

- Setting the divider location between the components:

```
sp.setDividerLocation(int pixels);
```

- Getting the divider location:

```
int n = sp.getDividerLocation();
```

- To get the top or left side component:

```
Component obj = sp.getTopComponent();
```

- To get the bottom or right side component:

```
Component obj = sp.getBottomComponent();
```

- To remove a component from the split pane:

```
sp.remove(Component obj);
```

- To remove a component at a specified location:

```
sp.remove(int index);
```

- To set a component as top component in the split pane:

```
sp.setTopComponent(Component obj);
```

- To set a component as bottom component:

```
sp.setBottomComponent(Component obj);
```

- To set left component in the split pane:

```
sp.setLeftComponent(Component obj);
```

- To set right component in the split pane:

```
sp.setRightComponent(Component obj);
```

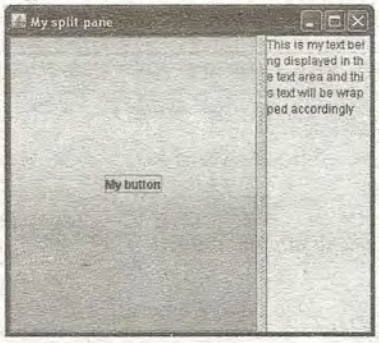
Program 13: Write a program that creates a split pane, which divides the frame into two parts horizontally. In the left part, we create a button and in the right part, we create a text area. When the button is clicked, the text is displayed inside the text area.

```
//Split pane  with text area and button.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class JSplitPaneDemo extends JFrame implements ActionListener
{
      //vars
      String str="This is my text being displayed in the text area" + " and
       this text will be wrapped accordingly";
      JButton b;
      JTextArea ta;
      JSplitPane sp;

      JSplitPaneDemo()
      {
            //create content pane
            Container c = getContentPane();

            //set border layout to content pane
            c.setLayout(new BorderLayout());

            //create a push button and text area
            b= new JButton("My button");
            ta= new JTextArea();

            //set wrapping of the line for text area
            ta.setLineWrap(true);

            //create horizontal split pane that contains b,ta
            sp = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, b,ta);

            //set the divider location at 300 pixels in split pane
            sp.setDividerLocation(300);

            //add split pane in the center of container
            c.add("Center",sp);

            //add action listener to the button
            b.addActionListener(this);

            //close frame
            setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      }
      public void actionPerformed(ActionEvent ae)
      {
            //when button clicked, set the string to the text area
            ta.setText(str);
```

```
        }
        public static void main(String args[])
        {
                //create the frame
                JSplitPaneDemo spd = new JSplitPaneDemo();
                spd.setSize(400,400);
                spd.setTitle("My split pane");
                spd.setVisible(true);
        }
}
```

Output:

```
C:\> javac JSplitPaneDemo.java
C:\> java JSplitPaneDemo
```



JTree Class

`JTree` class is useful to create a tree structure where a set of nodes can be displayed in a hierarchical manner. Each node may represent an item or some text.

To create a tree, we can create an object to `JTree`, as:

```
JTree tree = new JTree(root);
```

Here, `root` represents root node of the tree from where other nodes will span. This root node and other nodes can be created using `DefaultMutableTreeNode` class, as:

```
DefaultMutableTreeNode node = new DefaultMutableTreeNode("Item");
```

We can also create a tree by passing a Hashtable object that contains key and value pairs to the `JTree` object, as:

```
JTree tree = new JTree(Hashtable obj);
```

Similarly, a tree can be created by passing a Vector object that contains other objects, to the `JTree` object, as:

```
JTree tree = new JTree(Vector obj);
```

The following methods can be used to work with `JTree`:

- To add the nodes to root node, we can use `add()` method:

```
root.add(node);
```

- To find the path of selected item in the tree, `getNewLeadSelectionPath()` method of `TreeSelectionEvent` class is useful. This method returns `TreePath` object.

```
TreePath tp = tse.getNewLeadSelectionPath();
```

- To find the selected item in the tree, we can use `getLastPathComponent()` method of `TreePath` class.

```
Object comp = tp.getLastPathComponent();
```

Here, `comp` represents the component or node selected by the user.

- To know the path number (this represents the level):

```
int n = tp.getPathCount();
```

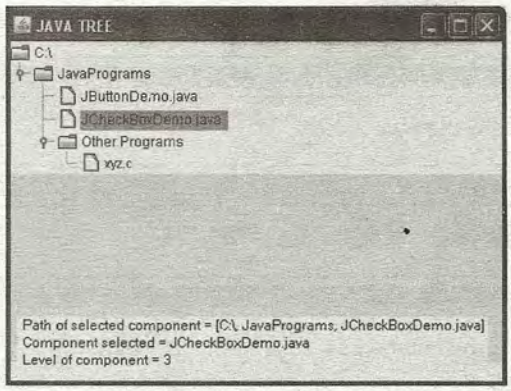
Program 14: Write a program to create a `JTree` with a root node and other nodes spanning from root node.

```
//JTree demo
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*; //TreeSelectionListener
import javax.swing.tree.*; //TreePath

class JTreeDemo extends JFrame implements TreeSelectionListener
{
     //vars
     DefaultMutableTreeNode root,dir1,dir2,file1,file2,file3;
     JTree tree;
     Container c;
     String msg="";
     JTextArea ta;

     JTreeDemo()
     {
          //create content pane c
          c = getContentPane();

          //set border layout to c
          c.setLayout(new BorderLayout());

          //create root node
          root = new DefaultMutableTreeNode("C:\\");

          //create other nodes
          dir1 = new DefaultMutableTreeNode("JavaPrograms");
          dir2 = new DefaultMutableTreeNode("Other Programs");
          file1 = new DefaultMutableTreeNode("JButtonDemo.java");
          file2 = new DefaultMutableTreeNode("JCheckBoxDemo.java");
          file3 = new DefaultMutableTreeNode("xyz.c");

          //add dir1 to root node
          root.add(dir1);

          //add other nodes to dir1
```

```
                dir1.add(file1);
                dir1.add(file2);
                dir1.add(dir2);

                //add file3 as a node in dir2
                dir2.add(file3);

                //create the tree from root node
                tree = new JTree(root);

                //add the tree to container
                c.add("North", tree);

                //create 3 empty labels and add to container
                ta = new JTextArea();
                c.add("South", ta);

                //add tree selection listener to the tree
                tree.addTreeSelectionListener(this);

                //close the frame
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }

        //this method belongs to tree selection listener
        public void valueChanged(TreeSelectionEvent tse)
        {
                //let us find out the newely selected item path
                TreePath tp = tse.getNewLeadSelectionPath();
                msg += "\nPath of selected component = "+ tp;

                Object comp = tp.getLastPathComponent();
                msg += "\nComponent selected = " + comp;

                int n = tp.getPathCount();
                msg += "\nLevel of component = "+ n;

                //send the user selection to the label
                ta.setText(msg);
                msg = "";
        }

        public static void main(String args[])
        {
                //create the frame
                JTreeDemo td = new JTreeDemo();
                td.setSize(400,300);
                td.setTitle("JAVA TREE");
                td.setVisible(true);
        }
}
```

Output:

```
C:\> javac JTreeDemo.java
C:\> java JTreeDemo
```



In this program, we create a tree with a root node `C:\`. The child node is `JavaPrograms` and other nodes are `JButtonDemo.java`, `JCheckBoxDemo.java`, and `Other Programs`. This `Other Programs` has another node `xyz.c`. When the user clicks on any node, the selected node's information is displayed in the text area.

JComboBox Class

`JComboBox` allows us to create a combo box, with a group of items which are displayed as a drop-down list. The user can select a single item only.

To create a combo box, we can simply create an object to `JComboBox`, as:

```
JComboBox box = new JComboBox();
```

This preceding `statement creates an empty combo box`

```
JComboBox box = new JComboBox(Object arr[]);
```

This preceding statement creates a `JComboBox` that contains the elements in the specified `array arr[]`.

```
JComboBox box = new JComboBox(Vector v);
```

This statement creates a `JComboBox` that contains the elements in the specified Vector `v`.

Using `JComboBox` becomes easy with the following methods:

- To add the items to the combo box, we can use `addItem()` method:

```
box.addItem("India");
```

- To retrieve the selected item from the combo box:

```
object obj = box.getSelecedItem();
```

- To retrieve the selected item's index:

```
int i = box.getSelectedIndex();
```

- To get the item of the combo box upon giving its index:

```
Object obj = box.getItemAt(int index);
```

- To get number of items in the combo box:

```
int n = box.getItemCount();
```

- To remove an item obj from the combo box:

```
box.removeItem(Object obj);
```

- To remove an item from the combo box when index is given:

```
box.removeItemAt(int index);
```

- To remove all items from the combo box:

```
box.removeAllItems();
```

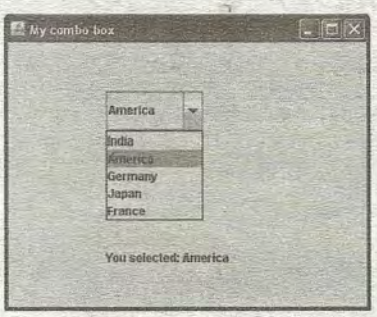
Program 15: Write a program to create a combo box with names of some countries. The user can select any one name from the list and the selected country name is displayed again in the frame.

```
//JComboBox demo
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

class JComboBoxDemo extends JFrame implements ItemListener
{
      //vars
      JComboBox box;
      JLabel lbl;

      JComboBoxDemo()
      {
            //create content pane
            Container c = getContentPane();

            //do not set any layout to c
            c.setLayout(null);

            //create an empty combobox
            box = new JComboBox();

            //add items to it
            box.addItem("India");
            box.addItem("America");
            box.addItem("Germany");
            box.addItem("Japan");
            box.addItem("France");

            //set the location of combo box
            box.setBounds(100,50,100,40);

            //add combo box to the container
            c.add(box);

            //create an empty label
            lbl = new JLabel();
```

```
                //set the location of label
                lbl.setBounds(100,200,200,40);

                //add the label to content pane
                c.add(lbl);

                //attach item listener to combo box
                box.addItemListener(this);

                //close the frame
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }

        public void itemStateChanged(ItemEvent ie)
        {
                //know which item is selected
                String str = (String)box.getSelectedItem();

                //display the selected item in the label
                lbl.setText("You selected: "+str);
        }

        public static void main(String args[])
        {
                //create the frame
                JComboBoxDemo demo = new JComboBoxDemo();
                demo.setTitle("My combo box");
                demo.setSize(500,400);
                demo.setVisible(true);
        }
}
```

Output:

```
C:\> javac JComboBoxDemo.java
C:\> java JComboBoxDemo
```



JList Class

`JList` class is useful to create a list which displays a list of items and allows the user to select one or more items.

To create a list, we can simply create an object to `JList`, as:

```
JList lst = new JList();
```

This creates an empty list.

```
JList lst = new JList(Object arr[]);
```

This creates a list with the elements of the array `arr[]`.

```
JList lst = new JList(Vector v);
```

This constructs a `JList` that displays the elements in the specified Vector.

The following methods are useful to work with `JList`:

- To know which item is selected in the list, when only a single item is selected:

```
Object item = lst.getSelectedValue();
```

- To know the selected item's index whne only a single item is selected:

```
int index = lst.getSelectedIndex();
```

- To get all the selected items into an array `arr[]`:

```
Object arr[] = lst.getSelectedValues();
```

- To get the indexes of all the selected items into an array `arr[]`:

```
int  arr[] = lst.getSelectedIndices();
```

Program 16: Write a program to create a list box with names of some countries such that the use can select any one or more items from the list and the selected country names are displayed aga in the frame.

```
//JComboBox demo
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;

class JListDemo extends JFrame implements ListSelectionListener
{
      //vars
      JList lst;
      JLabel lbl;
      Object arr[];
      String msg="";

      JListDemo()
      {
            //create content pane
            Container c = getContentPane();

            //do not set any layout to c
            c.setLayout(null);

            //create an array with items list
            String items[] = {"India", "America", "Germany", "Japan",
             "France"};

            //create a list box with the items
            lst = new JList(items);

            //set the location of list box
```

```
                lst.setBounds(100,50,100,100);

                //add list to the container
                c.add(lst);

                //create an empty label
                lbl = new JLabel();

                //set the location of label
                lbl.setBounds(50,200,400,40);

                //add the label to content pane
                c.add(lbl);

                //attach item listener to list box
                lst.addListSelectionListener(this);

                //close the frame
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }

        public void valueChanged(ListSelectionEvent le)
        {
                //know which items are selected
                arr = lst.getSelectedValues();

                //retrieve selected items and add to string msg
                for(int i=0; i<arr.length; i++)
                msg += (String)arr[i];
                //display the selected items in the label
                lbl.setText("selected: "+msg);
                //reset the string
                msg="";
        }
        public static void main(String args[])
        {
                //create the frame
                JListDemo demo = new JListDemo();
                demo.setTitle("My combo lst");
                demo.setSize(500,400);
                demo.setVisible(true);
        }
}
```

Output:

```
C:\> javac JListDemo.java
C:\> java JListDemo
```

As you can see, the list box is displayed in the output. To select the items sequentially, we can use Shift+clicking on the items. To select randomly, we can use Control+clicking on the items.

JMenu Class

A menu represents a group of items or options for the user to select from. JMenu is used to create a menu with some options. After creating the menu, it should be added to the menu bar. See Figure 28.6 to get an idea of the menu bar, menu, and menu items.

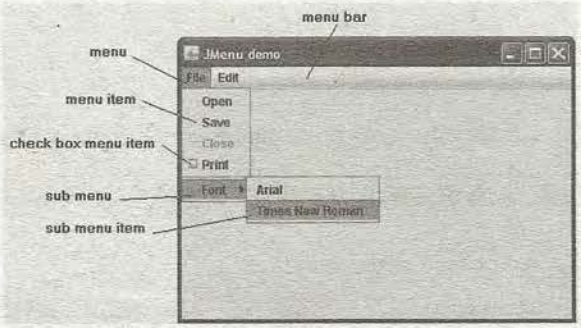


Figure 28.6 File and Edit are menus attached to menu bar

To create a menu, the following steps should be used:

- Create a menu bar using `JMenuBar` class object.

```
JMenuBar mb = new JMenuBar();
```

- Attach this menu bar to the container.

```
c.add(mb);
```

- Create separate menus to attach to the menu bar. For example, we want to create file menu with the name "`File`".

```
JMenu file= new JMenu("File");
```

- Attach this menu to the menu bar.

```
mb.add(file);
```

- A menu consists of a group of menu items. These menu items can be created using `JMenuItem` class. Sometimes, we can use a check box or a radio button as a menu item. For creating such check box, we should use `JCheckBoxMenuItem` and for creating radio button as a menu item we should use `JRadioButtonMenuItem`.

```
JMenuItem op = new JMenuItem("Open");
JCheckBoxMenuItem pr = new JCheckBoxMenuItem("Print");
```

- Attach the menu item to the menu.

```
file.add(op);
```

This completes creation of a menu. It is possible to create a menu inside another menu. It is called submenu. To create a submenu, one can follow the steps:

- Create a menu.

```
JMenu font= new JMenu("Font");
```

Here, `font` represents a submenu to be attached as an item in a menu.

- Now attach it to a menu.

```
file.add(font);
```

- Create menu items using `JMenuItem`, `JCheckBoxMenuItem`, or `JRadioButtonMenuItem`.

```
JMenuItem f1 = new JMenuItem("Arial");
```

- Attach menu items to the submenu.

```
font.add(f1);
```

To disable an item in the menu, we can use `setEnabled(false)` method and to enable it, we can use `setEnabled(true);`

To display a horizontal line which separates a group of items from another group in the menu, we can use `addSeparator()` method.

It is the duty of the programmer to know which item has been selected by the user. It helps him to write the code representing the further action depending on the user selection. In case of menu items, the selected item can be known, by using `isArmed()` method. This method returns true if the item is selected, otherwise false. In case of check box menu items or radio button menu items, we can use `getModel()` method first on the item so that it gets the Model object which contains the state of the item. Then `isSelected()` method can tell whether the item is selected or not by referring to the data inside the Model object. These things can be observed in Program 17.

Program 17: Write a program to create a menu with several menu items.

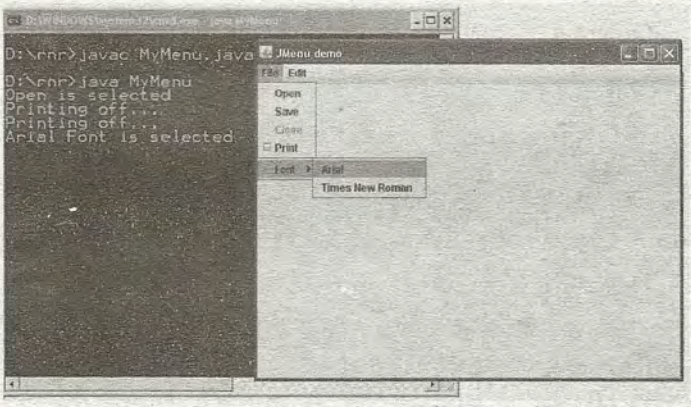
```
//Menu creation
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class MyMenu extends JFrame implements ActionListener
{
      //vars
      JMenuBar mb;
      JMenu file,edit,font;
      JMenuItem op,sa,cl,cp,pt,f1,f2;
      JCheckBoxMenuItem pr;

      MyMenu()
      {
            //create container
            Container c = getContentPane();
            c.setLayout(new BorderLayout());

            //create a menubar
            mb = new JMenuBar();

            //add menubar to container
            c.add("North",mb);

            //create the File,Edit menus
```

```
        //and attach them to menubar
        file= new JMenu("File");
        edit= new JMenu("Edit");
        mb.add(file);
        mb.add(edit);

        //create menu items
        op= new JMenuItem("Open");
        sa= new JMenuItem("Save");
        cl= new JMenuItem("Close");
        cp= new JMenuItem("Copy");
        pt= new JMenuItem("Paste");

        //add Open,Save,Close to File menu
        //and Copy,Paste to Edit menu
        file.add(op);
        file.add(sa);
        file.add(cl);
        edit.add(cp);
        edit.add(pt);

        //Make close disabled
        cl.setEnabled(false);

        //create Print checkbox and add it to File menu
        pr= new JCheckBoxMenuItem("Print");
        file.add(pr);

        //add a separator (horizontal line) to File menu
        file.addSeparator();

        //Create a Font submenu and add it to File menu
        font = new JMenu("Font");
        file.add(font);

        //create menu items
        f1 = new JMenuItem("Arial");
        f2 = new JMenuItem("Times New Roman");

        //add menu items to sub menu
        font.add(f1);
        font.add(f2);

        //attach action listeners to all menu items
        op.addActionListener(this);
        sa.addActionListener(this);
        cl.addActionListener(this);
        cp.addActionListener(this);
        pt.addActionListener(this);
        pr.addActionListener(this);
        f1.addActionListener(this);
        f2.addActionListener(this);

        //close frame
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    //this method is executed when a menu item is clicked
    public void actionPerformed(ActionEvent ae)
    {
        //know which menu item is clicked
        if(op.isArmed()) System.out.println("Open is selected");
        if(sa.isArmed()) System.out.println("Save is selected");
        if(cl.isArmed()) System.out.println("Close is selected");
        if(cp.isArmed()) System.out.println("Copy is selected");
        if(pt.isArmed()) System.out.println("Paste is selected");

        if(pr.getModel().isSelected()) System.out.println("Printing
```

```
                on...");
            else System.out.println("Printing off...");

            if(f1.isArmed()) System.out.println("Arial Font is selected");
            if(f2.isArmed()) System.out.println("Times New Roman is selected");

        }

        public static void main(String args[])
        {
            //create the frame
            MyMenu mm = new MyMenu();
            mm.setTitle("JMenu demo");
            mm.setSize(500,400);
            mm.setVisible(true);
        }
    }
```

Output:

```
C:\> javac MyMenu.java
C:\> java MyMenu
```



In this program, we create two menus—file and edit and add them to the menu bar. There is a submenu Font which is added in the file menu. When the user clicks on any menu items, at DOS prompt, we displayed the clicked items.

In Program 17, we are merely catching which item is selected by the user and displaying that item's name at DOS prompt. We should go further depending on the user selection. For example, if the user has clicked on 'open' item in File menu, then the file should be actually opened and displayed in a separate window. How to achieve this? We can write a method like `openFile()` where we display a file dialog box which displays all the files available in the system. Then the user can select any file he wants to open. That filename and the path should be found and its contents should be read using some input stream like `FileReader` and then the contents should be sent to a text area in another frame. Thus the user can view the contents in another frame.

For this purpose, we need `JFileChooser` class. `JFileChooser` class has method to display a file open dialog box, as:

```
int i =fc.showOpenDialog(this);  //fc is JFileChooser object
```

Similarly, we can also display a file save dialog box where the user can select a file name and attempt to save data in the file, as:

```
int i =fc.showSaveDialog(this);  //to handle File -> Save item
```

When the user approves a file in the dialog box, then the `getSelectedFile()` method will help to get the filename into File object as:

```
File f = fc.getSelectedFile();
```

Now, the selected file name can be obtained by using `f.getName()` and its path from `f.getPath()`. This method gives the path and the filename. So, that full path and filename can be sent to a `FileReader`, as:

```
BufferedReader br = new BufferedReader(new FileReader(fname));
```

Here, the `FileReader` is again connected to the `BufferedReader`. Using `br.readLine()` method we can read the contents of the file string by string and can display them in a text area. This text area can be displayed along with the content in another frame for the user to view. This is done in the next program

Program 18: Write a program to create a menu and handle the file open event for the user.

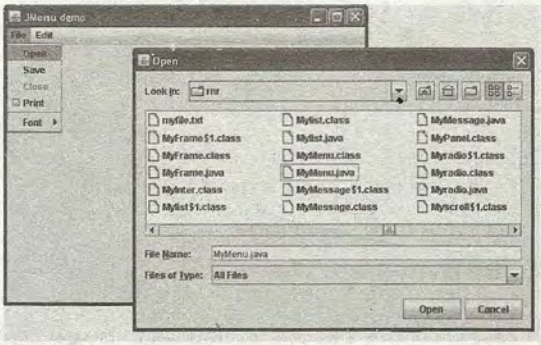
```
//Menu creation
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
class MyMenu extends JFrame implements ActionListener
{
      //vars
      JMenuBar mb;
      JMenu file,edit,font;
      JMenuItem op,sa,cl,cp,pt,f1,f2;
      JCheckBoxMenuItem pr;

      MyMenu()
      {
            //create container
            Container c = getContentPane();
            c.setLayout(new BorderLayout());

            //create a menubar
            mb = new JMenuBar();

            //add menubar to container
            c.add("North",mb);

            //create the File,Edit menus
            //and attach them to menubar
            file= new JMenu("File");
            edit= new JMenu("Edit");
            mb.add(file);
            mb.add(edit);

            //create menu items
            op= new JMenuItem("Open");
            sa= new JMenuItem("Save");
            cl= new JMenuItem("Close");
            cp= new JMenuItem("Copy");
            pt= new JMenuItem("Paste");
```

```
        //add Open,Save,Close to File menu
        //and Copy,Paste to Edit menu
        file.add(op);
        file.add(sa);
        file.add(cl);
        edit.add(cp);
        edit.add(pt);

        //Make close disabled
        cl.setEnabled(false);

        //create Print checkbox and add it to File menu
        pr= new JCheckBoxMenuItem("Print");
        file.add(pr);

        //add a separator (horizontal line) to File menu
        file.addSeparator();

        //Create a Font submenu and add it to File menu
        font = new JMenu("Font");
        file.add(font);

        //create menu items
        f1 = new JMenuItem("Arial");
        f2 = new JMenuItem("Times New Roman");

        //add menu items to sub menu
        font.add(f1);
        font.add(f2);

        //attach listeners to all menu items
        op.addActionListener(this);
        sa.addActionListener(this);
        cl.addActionListener(this);
        cp.addActionListener(this);
        pt.addActionListener(this);
        pr.addActionListener(this);
        f1.addActionListener(this);
        f2.addActionListener(this);

        //close frame
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    //this method is executed when a menu item is clicked
    public void actionPerformed(ActionEvent ae)
    {
        //know which menu item is clicked
        if(op.isArmed()) this.openFile();
        if(sa.isArmed()) //this.saveFile();

        if(cl.isArmed()) System.out.println("Close is selected");
        if(cp.isArmed()) System.out.println("Copy is selected");
        if(pt.isArmed()) System.out.println("Paste is selected");

        if(pr.getModel().isSelected()) System.out.println("Printing
         on...");
        else System.out.println("Printing off...");

        if(f1.isArmed()) System.out.println("Arial Font is selected");
        if(f2.isArmed()) System.out.println("Times New Roman is selected");

    }
    //this method is called when File->Open is selected
    void openFile()
    {
        //create an object to JFileChooser class
        JFileChooser fc = new JFileChooser();
```

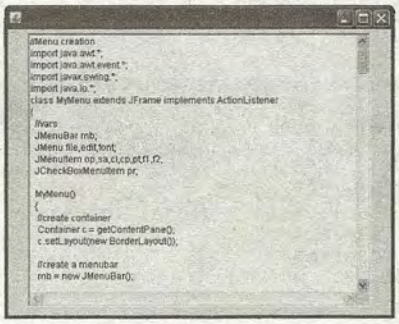
```
                //display file open dialog box
                int i =fc.showOpenDialog(this);

                //if the user selected a file name then
                if(i == JFileChooser.APPROVE_OPTION) {
                //get the selected file into File object
                File f = fc.getSelectedFile();

                //The file name is given by f.getName();
                //File name with path is given by f.getPath();
                String fname = f.getPath();

                //open another frame and pass the fname to it
                OpenFrame of = new OpenFrame(fname);

                of.setSize(500,400);
                of.setVisible(true);
        }
}

public static void main(String args[])
{
        //create the frame
        MyMenu mm = new MyMenu();
        mm.setTitle("JMenu demo");
        mm.setSize(500,400);
        mm.setVisible(true);
}
}

//this is another class which creates another frame
//to display file contents
class OpenFrame extends JFrame
{
        //catch the file name
        OpenFrame(String fname)
        {
                //create content pane
                Container c = getContentPane();
                c.setLayout(new FlowLayout());

                //create a text area and add to content pane
                TextArea ta = new TextArea(22,60);
                c.add(ta);

                //vars
                String str="";
                String str1="";

                try{
                //create reader to read from file
                BufferedReader br = new BufferedReader(new FileReader(fname));

                //read string by string and add to str1
                while((str = br.readLine()) != null)
                str1+= str+"\n";

                //display the file content in text area
                ta.setText(str1);

                //close the file
                br.close();
                }catch(Exception e){}

        }
}
```

Output:

```
C:\> javac MyMenu.java
C:\> java MyMenu
```



When the File -> Open option is selected by the user, the Open dialog box appears as shown in the output. In this dialog box, if the user selects a file with the name `MyMenu.java`, then the contents of the file will appear in a separate frame as shown in the next figure:



```
/*Menu creation
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
class MyMenu extends JFrame implements ActionListener
{
 //vars
 JMenuBar mb;
 JMenu file,edit,font;
 JMenuItem op,sa,cl,cp,pt,f1,f2;
 JCheckBoxMenuItem pr;

 MyMenu()
 {
  //create container
  Container c = getContentPane();
  c.setLayout(new BorderLayout());

  //create a menubar
  mb = new JMenuBar();
```

JToggleButton Class

A toggle button looks like a push button, but has two states—pushed and released. When the user clicks on the button, it goes into pushed state and some task can be performed. When the user clicks the same button once again, it goes into released state and another task may be performed. `JToggleButton` class is useful to create a toggle button, as:

```
JToggleButton but = new JToggleButton("label");
```

This creates a toggle button with a label on it.

```
JToggleButton but = new JToggleButton("label", ImageIcon obj);
```

This creates a toggle button with label and image.

`isSelected()` method is useful to determine the state of the toggle button. If this method returns true, the button is selected, otherwise it is not selected.

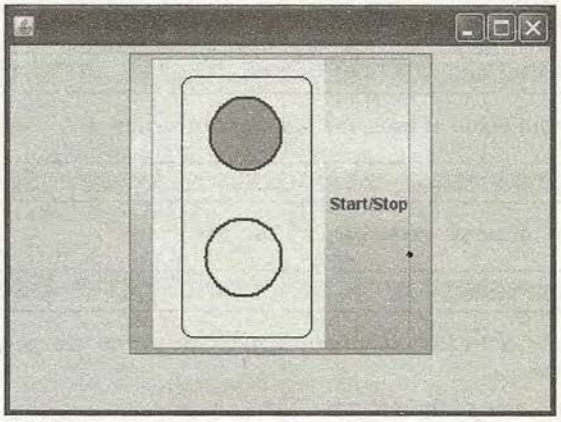
Program 19: Write a program to show the functioning of a toggle button.

```
//A toggle button with start and stop images
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class JTButton extends JFrame implements ActionListener
{
      //vars
      JToggleButton but;
      ImageIcon img1;

      JTButton()
      {
            //create content pane with flow layout
            Container c = getContentPane();
            c.setLayout(new FlowLayout());

            //image with start signal
            img1 = new ImageIcon("start.gif");

            //create toggle button with start image
            but = new JToggleButton("Start/Stop", img1);

            //add button to content pane
            c.add(but);

            //add action listener to button
            but.addActionListener(this);
      }

      public void actionPerformed(ActionEvent ae)
      {
            //image with stop signal
            ImageIcon img2 = new ImageIcon("stop.gif");

            //if toggle button is selected display stop signal image
            //else display start signal image
            if(but.isSelected())
            but.setIcon(img2);
            else but.setIcon(img1);

      }

      public static void main(String args[])
      {
            //create the frame
            JTButton demo = new JTButton();
            demo.setSize(400,400);
            demo.setVisible(true);
            demo.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      }
}
```

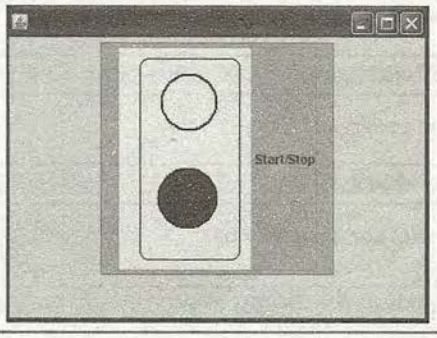
Output:

```
C:\> javac JTButton.java
C:\> java JTButton
```

In this program, we create a toggle button with an image `'start.gif'`. When the button is clicked again, it displays the image `'stop.gif'`. Again when the button is clicked, it displays `'start.gif'`. Like this, it toggles between the two images.



When the button is clicked, it displays the image as shown here:



JProgressBar Class

A progress bar visually displays the progress of some task. As the task progresses towards completion, the progress bar displays the task's percentage of completion. This percentage is represented visually by a rectangle which starts out empty and gradually gets filled as the task progresses. Progress bar can be displayed horizontally or vertically. `JProgressBar` is used to create the progress bar, as:

```
JProgressBar bar = new JProgressBar();
```

This creates a horizontal progress bar.

```
JProgressBar bar = new JProgressBar(orientation);
```

Where the orientation can be `SwingConstants.HORIZONTAL (0)` for displaying horizontal progress bar or `SwingConstants.VERTICAL (1)` for displaying vertical progress bar.

```
JProgressBar bar = new JProgressBar(alignment, int min, int max);
```

Here, the `min` and `max` represent the starting and ending values of the progress bar.

- To know the minimum value of progress bar, we can use:

```
int n = bar.getMinimum();
```

- To know the maximum value of progress bar, we can write:

```
int n = bar.getMaximum();
```

- To get the alignment of the progress bar, we can use:

```
int n = bar.getOrientation();
```

This method returns `0` if `HORIZONTAL` orientation is used for the progress bar. It returns `1` if `VERTICAL` alignment is there.

- To know the current value of the progress bar:

```
int n = bar.getValue();
```

- To set the value of the progress bar to a particular value:

```
bar.setValue(int value);
```

- To set the minimum of the progress bar:

```
bar.setMinimum(int value);
```

- To set the maximum value of the progress bar:

```
bar.setMaximum(int value);
```

- To set the alignment, we can use:

```
bar.setOrientation(int orientation);
```

Here, if the orientation given is `0`, it is set to `HORIZONTAL`, if it is `1` then it is set to `VERTICAL`.

- To display the percentage of progress, we can use `setStringPainted()` method, as:

```
bar.setStringPainted(true);
```

When `true` is passed to this method, progress percentage will be displayed as a string, otherwise not.

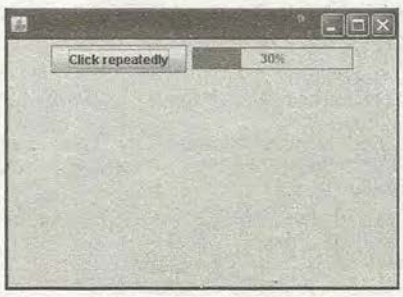
Program 20: Write a program to create a push button and a progress bar, such that everytime, the button is clicked by the user the progress bar progresses by 5 units.

```
//JToggleButton
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
class ProgressDemo extends JFrame implements ActionListener
{
     //vars
     JButton b;
     JProgressBar bar;
```

```
        ProgressDemo()
        {
                //create content pane with flow layout
                Container  c = getContentPane();
                c.setLayout(new FlowLayout());

                //create a button
                b = new JButton("Click repeatedly");

                //create a progress bar
                bar = new JProgressBar();

                //set gray as foreground color
                bar.setForeground(Color.gray);

                //display the percentage string
                bar.setStringPainted(true);

                //add button and progress bar to c
                c.add(b);
                c.add(bar);

                //add action listener to button
                b.addActionListener(this);
        }

        public void actionPerformed(ActionEvent ae)
        {
                //every time the button is clicked,
                //increment the progress bar value by 5
                bar.setValue(bar.getValue()+5);

        }

        public static void main(String args[])
        {
                //create the frame
                ProgressDemo d = new ProgressDemo();
                d.setSize(400,400);
                d.setVisible(true);
                d.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
}
```

Output:

```
C:\> javac ProgressDemo.java
C:\> java ProgressDemo
```



JToolBar Class

`JToolBar` class is useful to create a tool bar. A tool bar is a bar to which some components can be attached. It is possible to attach push buttons, radio buttons, check boxes, list boxes, etc. to the tool bar. To create a tool bar:

```
JToolBar tb = new JToolBar();
```

This creates a tool bar with HORIZONTAL orientation.

```
JToolBar tb = new JToolBar(int orientation);
```

Here, orientation is 0 for HORIZONTAL orientation and 1 for VERTICAL orientation.

```
JToolBar tb = new JToolBar(String str, int orientation);
```

This creates a tool bar with the name `str` and the specified orientation.

After creating the tool bar, we can add components to it, using `add()` method, as:

```
tb.add(component);
```

Program 21: Write a program to create a tool bar with 3 push buttons added to it. The push buttons are created with images to denote new, open, and print options. When the user clicks on any button, the selected button is displayed in the label.

```
//JToolBar
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
class JToolBarDemo extends JFrame implements ActionListener
{
      //vars
      String str;
      Container c;
      JToolBar tb;
      JButton b1,b2,b3;
      JLabel lbl;

      JToolBarDemo()
      {
            //create content pane
            c = getContentPane();

            //set border layout to c
            c.setLayout(new BorderLayout());

            //create tool bar
            tb = new JToolBar();

            //set etched border around the tool bar
            tb.setBorder(BorderFactory.createEtchedBorder(Color.green,
            Color.black));

            //load images into ImageIcon objects
            ImageIcon img1, img2, img3;
            img1 = new ImageIcon("new.gif");
            img2 = new ImageIcon("open.gif");
            img3 = new ImageIcon("print.gif");

            //create 3 push buttons with images
            b1 = new JButton(img1);
```

```
                b2 = new JButton(img2);
                b3 = new JButton(img3);

                //add the push buttons to the tool bar
                tb.add(b1);
                tb.add(b2);
                tb.add(b3);

                //add the tool bar in c at top
                c.add("North", tb);

                //create a label and add to c at.center
                lbl = new JLabel();
                lbl.setFont(new Font("SansSerif", Font.PLAIN, 30));
                c.add("Center", lbl);

                //add action listener to buttons in tool bar
                b1.addActionListener(this);
                b2.addActionListener(this);
                b3.addActionListener(this);
        }

        public void actionPerformed(ActionEvent ae)
        {
                //know which button is clicked
                if(ae.getSource() == b1) str = "New clicked";
                if(ae.getSource() == b2) str = "Open clicked";
                if(ae.getSource() == b3) str = "Print clicked";

                //display the string in the label
                lbl.setText(str);

        }

        public static void main(String args[])
        {
                //create the frame
                JToolBarDemo tbd = new JToolBarDemo();
                tbd.setSize(400,400);
                tbd.setVisible(true);
                tbd.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
}
```

Output:

```
C:\> javac JToolBarDemo.java
C:\> java JToolBarDemo
```

JColorChooser Class

This class is useful to create a color chooser dialog box with several colors so that the user can select any color. To create a color chooser dialog box, we can use `showDialog()` method of `JColorChooser` class, as:

```
Color color = JColorChooser.showDialog(this, "Select a color", selectedcolor);
```

`showDialog()` method takes 3 arguments—the parent component of the dialog, the title for the dialog, and the initial color set when the color dialog is shown. In the color dialog, if the user selects a color and then press OK button, then the selected color is returned by the `showDialog()` method. This selected color can be used for any purpose by the user.

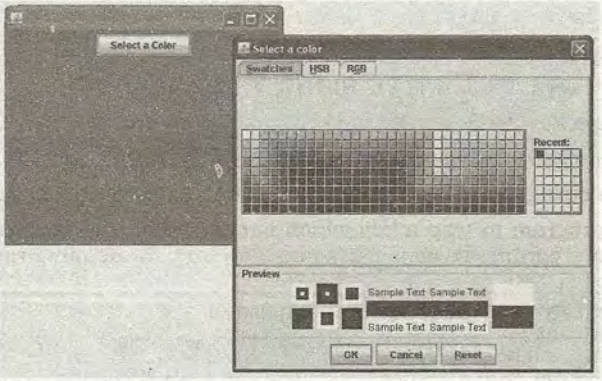
Program 22: Write a program to create a color chooser dialog box for the user to select a color from and the selected color is displayed as background color for the frame.

```
//JColorChooser demo
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class JColorChooserDemo extends JFrame implements ActionListener
{
	//vars
	JButton b;
	Container c;

	JColorChooserDemo()
	{
		//create the content pane
		c = getContentPane();
		c.setLayout(new FlowLayout());

		//creat a push button
		b = new JButton("Select a Color");

		//add button to content pane
		c.add(b);

		//add action listener to button
		b.addActionListener(this);
	}

	public void actionPerformed(ActionEvent ae)
	{
		//take the initial color as null
		Color selectedcolor= null;

		//create the color chooser with dialog box to select a color
		Color color = JColorChooser.showDialog(this, "Select a color",
		 selectedcolor);

		//if color is not null then some color is selected
		if(color != null)
		{
			//get the selected color
			selectedcolor = color;
		}

		//show back ground color of frame with the selected color
		c.setBackground(color);
	}

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

```
                //create the frame
                JColorChooserDemo demo = new JColorChooserDemo();
                demo.setSize(400,400);
                demo.setVisible(true);
                demo.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
    }
```

Output:

```
C:\> javac JColorChooserDemo.java
C:\> java JColorChooserDemo
```



Handling Keyboard Events

A user interacts with the application by pressing keys either on the keyboard or by using mouse. A programmer should know which key the user has pressed on the keyboard or whether the mouse is moved, pressed, or released. These are also called 'events'. Knowing these events will enable the programmer to write his code according to the key pressed or mouse event.

KeyListener interface of `java.awt.event` package helps to know which key is pressed or released by the user. It has 3 methods:

- `public void keyPressed(KeyEvent ke)` : This method is called when a key is pressed on the keyboard. This include any key on the keyboard along with special keys like function keys, shift, alter, caps lock, home, end, etc.
- `public void keyTyped(KeyEvent ke)` : This method is called when a key is typed on the keyboard. This is same as `keyPressed()` method but this method is called when general keys like A to Z or 1 to 9, etc. are typed. It cannot work with special keys.
- `public void keyReleased(KeyEvent ke)` : This method is called when a key is released.

`KeyEvent` class has the following methods to know which key is typed by the user:

- `char getKeyChar()` : This method returns the key name (or character) related to the key pressed or released.
- `int getKeyCode()` : This method returns an integer number which is the value of the key pressed by the user.

The following are the key codes for the keys on the keyboard. They are defined as constants in `KeyEvent` class. Remember VK represents Virtual Key.

- To represent keys from a to z: VK_A to VK_Z
- To represent keys from 1 to 9: VK_0 to VK_9
- To represent keys from F1 to F12: VK_F1 to VK_F12
- To represent Home, End: VK_HOME, VK_END
- To represent PageUp, PageDown: VK_PAGE_UP, VK_PAGE_DOWN
- To represent Insert, Delete: VK_INSERT, VK_DELETE
- To represent caps lock: VK_CAPS_LOCK
- To represent alter key: VK_ALT
- To represent Control key: VK_CONTROL
- To represent Shift key: VK_SHIFT
- To represent Tab key: VK_TAB
- To represent arrow keys: VK_LEFT, VK_RIGHT, VK_UP, VK_DOWN
- To represent Escape key: VK_ESCAPE
- static String getKeyText(int keyCode)

This method returns a string describing the keyCode such as HOME, F1, or A.

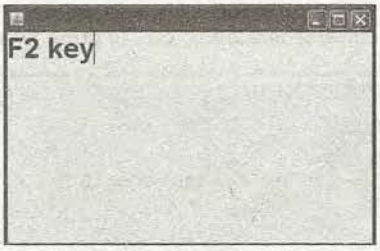
Program 23: Write a program to trap a key which is pressed on the keyboard and display its name in the text area. In this program, we consider some keys only for demonstration purpose.

```
//To catch some of the keys of the keyboard
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class KeyBoardEvents extends JFrame implements KeyListener
{
      //vars
      Container c;
      JTextArea ta;
      String str="";

      KeyBoardEvents()
      {
            //create content pane
            c = getContentPane();

            //create a text area and set some font to it
            ta = new JTextArea("Press a key");
            ta.setFont(new Font("Arial", Font.BOLD, 30));

            //add text area to content pane
            c.add(ta);

            //add key listener to text area
            ta.addKeyListener(this);
      }

      public void keyPressed(KeyEvent ke)
      {
            //get the key code of the key pressed on keyboard
            int keycode = ke.getKeyCode();

            //find which key is pressed
            if(keycode == KeyEvent.VK_F1) str += "F1 key";
            if(keycode == KeyEvent.VK_F2) str += "F2 key";
            if(keycode == KeyEvent.VK_F3) str += "F3 key";
            if(keycode == KeyEvent.VK_PAGE_UP) str += "Page Up";
            if(keycode == KeyEvent.VK_PAGE_DOWN) str += "Page Down";
            if(keycode == KeyEvent.VK_ALT) str += "Alter";
```

```
                if(keycode == KeyEvent.VK_HOME) str += "Home";
                if(keycode == KeyEvent.VK_END) str += "End";
                ta.setText(str);
                str="";
        }

        public void keyReleased(KeyEvent ke)
        {}

        public void keyTyped(KeyEvent ke)
        {}

        public static void main(String args[])
        {
                //create the frame
                KeyBoardEvents kbe = new KeyBoardEvents();

                kbe.setSize(400,400);
                kbe.setVisible(true);
                kbe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
}
```

Output:

```
C:\> javac KeyBoardEvents.java
C:\> java KeyBoardEvents
```



Handling Mouse Events

The user may click, release, drag, or move a mouse while interacting with the application. If the programmer knows what the user has done, he can write the code according to the mouse event. To trap the mouse events, `MouseListener` and `MouseMotionListener` interfaces of `java.awt.event` package are used.

MouseListener interface has the following methods:

- `void mouseClicked(MouseEvent e)`: Void MouseClickedThis method is invoked when the mouse button has been clicked (pressed and released) on a component.
- `void mouseEntered(MouseEvent e)`: This method is invoked when the mouse enters a component.
- `void mouseExited(MouseEvent e)`: This method is invoked when the mouse exits a component.

- `void mousePressed(MouseEvent e)`: This method is invoked when a mouse button has been pressed on a component.
- `void mouseReleased(MouseEvent e)`: This method is invoked when a mouse button has been released on a component.

`MouseMotionListener` interface has the following methods:

- `void mouseDragged(MouseEvent e)`: This method is invoked when a mouse button is pressed on a component and then dragged.
- `void mouseMoved(MouseEvent e)`: This method is invoked when the mouse cursor has been moved onto a component but no buttons have been pushed.

The `MouseEvent` class has the following methods:

- `int getButton()`: This method returns a value representing a mouse button, when it is clicked It returns 1 if left button is clicked, 2 if middle button, and 3 if right button is clicked.
- `int getClickCount()`: This method returns the number of mouse clicks associated with this event.
- `int getX()`: This method returns the horizontal x position of the event relative to the source component.
- `int getY()`: This method returns the vertical y position of the event relative to the source component.

Program 24: Write a program to create a text area and display the mouse event when the button on the mouse is clicked, when the mouse is moved, etc. is done by the user.

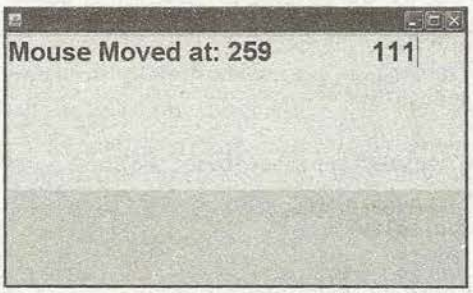
```
//Mouse events
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class MouseEvents extends JFrame implements MouseListener,

MouseMotionListener
{
      //vars
      String str="";
      JTextArea ta;
      Container c;
      int x,y;

      MouseEvents()
      {
            //create content pane
            c = getContentPane();
            c.setLayout(new FlowLayout());

            //create a text area and set some font to it
            ta = new JTextArea("Click the mouse or move it",5,20);
            ta.setFont(new Font("Arial", Font.BOLD, 30));

            //add text area to content pane
            c.add(ta);

            //add mouse listener, mouse motion listener to text area
            ta.addMouseListener(this);
            ta.addMouseMotionListener(this);

      }

      public void mouseClicked(MouseEvent me)
```

```
{
        //know which button of mouse is clicked
        int i = me.getButton();
        if(i==1)
        str += "Clicked Button: Left";
        else if(i==2)
        str += "Clicked Button: Middle";
        else if(i==3)
        str += "Clicked Button: Right";

        this.display();
}

public void mouseEntered(MouseEvent me)
{
        str += "Mouse entered";
        this.display();
}

public void mouseExited(MouseEvent me)
{
        str += "MouseExited";
        this.display();
}

Public void mousePressed(MouseEvent me)
{
        x = me.getX();
        y = me.getY();
        str += "Mouse Pressed at: "+x+"\t"+y;
        this.display();
}

public void mouseReleased(MouseEvent me)
{
        x = me.getX();
        y = me.getY();
        str += "Mouse Released at: "+x+"\t"+y;
        this.display();
}

public void mouseDragged(MouseEvent me)
{
        x = me.getX();
        y = me.getY();
        str += "Mouse Dragged at: "+x+"\t"+y;
        this.display();
}

public void mouseMoved(MouseEvent me)
{
        x = me.getX();
        y = me.getY();
        str += "Mouse Moved at: "+x+"\t"+y;
        this.display();
}

public void display()
{
        ta.setText(str);
        str="";
}

public static void main(String args[])
{
        //create the frame
        MouseEvents mes = new MouseEvents();
        mes.setSize(400,400);
```

```
                mes.setVisible(true);
                mes.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            }
    }
```

Output:

```
C:\> javac MouseEvents.java
C:\> java MouseEvents
```



Conclusion

Since AWT components are heavy-weight and dependent internally on native methods, swing has been invented. Swing components are light weight and take very less resources of the system. The screens designed in swing look same on all operating systems. But, if the programmer wishes to provide a different look and feel depending on the operating system, he can do so. This is the flexibility in swing. Also, swing has more and more number of features which made it to be programmers' favorite package. This is the reason that today almost all software development companies look for swing programmers for their projects.