GRAPHICS PROGRAMMING USING AWT

CHAPTER 27

Whenever an application (or software) is created, the user can work with it in two ways. The first way is where the user can remember some commands on which the application is built and type those commands to achieve the respective tasks. For example, the user may have to type a PRINT command to send a file contents to a printer. Here, the user should know the syntax and correct usage of the PRINT command. Then only he can interact with the application properly. This is called CUI (Character User Interface) since the user has to use characters or commands to interact with the application. The main disadvantage in CUI is that the user has to remember several commands and their use with correct syntax. Hence, CUI is not user-friendly. A person who does not know any thing about computers will find this CUI very difficult.

The second way is where the user need not remember any commands but interacts with any application by clicking on some images or graphics. For example, if the user wants to print a file, he can click on a printer image and the rest of the things will be taken care of by the application. The user has to tell how many copies he wants and printing continues. This is very easy for the user since the user remembers only some symbols or images, like a magnifying glass symbol for searching, a briefcase symbol for a directory, etc. This environment where the user can interact with an application through graphics or images is called GUI (Graphics User Interface). GUI has the following advantages:

- It is user-friendly. The user need not worry about any commands. Even a layman will be able to work with the application developed using GUI.
- It adds attraction and beauty to any application by adding pictures, colors, menus, animation, etc. We can observe almost all websites lure their visitors on Internet since they are developed attractively using GUI.
- It is possible to simulate the real life objects using GUI. For example, a calculator program may actually display a real calculator on the screen. The user feels that he is interacting with a real calculator and he would be able to use it without any difficulty or special training. So, GUI eliminates the need of user training.
- GUI helps to create graphical components like push buttons, radio buttons, check boxes, etc. and use them effectively in our programs.

AWT

Abstract Window Toolkit (AWT) represents a class library to develop applications using GUI. The `java.awt` package got classes and interfaces to develop GUI and let the users interact in a more friendly way with the applications. Figure 27.1 shows some important classes of `java.awt` package.

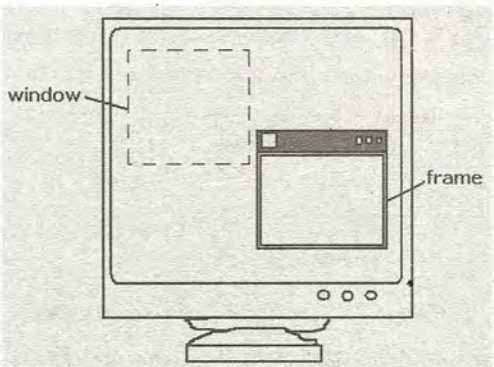


Figure 27.2 A window and a frame on the monitor

Creating a Frame

A frame becomes the basic component in AWT. The frame has to be created before any other component. The reason is that all other components can be displayed in a frame. There are three ways to create a frame, which are as follows:

- Create a `Frame` class object,

```
Frame f = new Frame();
```

- Create a `Frame` class object and pass its title also,

```
Frame f = new Frame("My frame");
```

- The third way is to create a subclass `MyFrame` to the `Frame` class and create an object to the subclass as:

```
class MyFrame extends Frame
MyFrame f = new MyFrame();
```

Since, `MyFrame` is a subclass of `Frame` class, its object `f` contains a copy of `Frame` class and hence `f` represents the frame.

In all these cases, a frame with initial size of 0 pixels width and 0 pixels height will be created, which is not visible on the screen. You may be wondering what these pixels are. A pixel (short for picture element) represents any single point or dot on the screen. Any data or pictures which are displayed on the screen are composed of several dots called pixels. Nowadays, monitors can accommodate 800 pixels horizontally and 600 pixels vertically. So the total pixels seen on one screen would be 800x600 = 480,000 pixels. This is called 'screen resolution'. See Figure 27.3. The more the screen resolution, the more clarity a picture will have on the screen when displayed. This is the reason most laptops use 1024x768 pixels resolution.

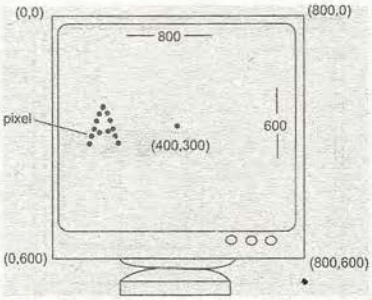


Figure 27.3 Screen coordinates in pixels

Since, the size of our frame would be 0px width and 0px height, it is not visible and hence we should increase its size so that it would be visible to us. This is done by `setSize()` method, as:

```
f.setSize(400,350);
```

Here, the frame's width is set to 400 px and height to 350 px. Then, we can display the frame, using `setVisible()` method, as:

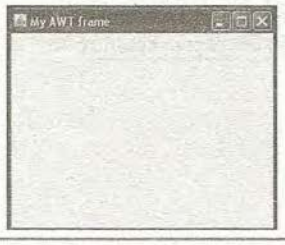
```
f.setVisible(true);
```

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

```
//Creating a frame - version 1
import java.awt.*;
class MyFrame
{
      public static void main(String args[])
      {
            //create a frame
            Frame f = new Frame("My AWT frame");

            //set the size of the frame
            f.setSize(300,250);

            //display the frame
            f.setVisible(true);
      }
}
```

Output:

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



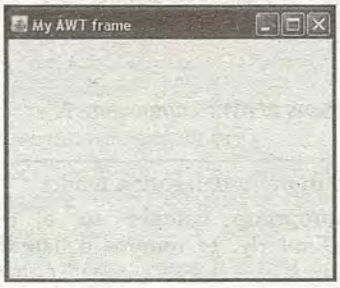
The same program can be rewritten in a different way, as shown in Program 2.

Program 2: Write a program to create a frame by creating an object to the subclass of `Frame` class.

```
//Creating a frame - version 2
import java.awt.*;
class MyFrame extends Frame
{
      //call super class constructor to store title
      MyFrame(String str)
      {
            super(str);
      }

      public static void main(String args[])
      {
            //create a frame with title
            MyFrame f = new MyFrame("My AWT frame");

            //set the size of the frame
            f.setSize(300,250);

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

      }
}
```

Output:

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



This frame can be minimized, maximized and resized, but cannot be closed. Even if we click on close button of the frame, it will not perform any closing action. Now the question is how to close the frame? Closing a frame means attaching action to the component. To attach actions to the components, we need 'event delegation model'. Let us discuss what it is.

Event Delegation Model

When we create a component, generally the component is displayed on the screen but is not capable of performing any actions. For example, we created a push button, which can be displayed but cannot perform any action, even when someone clicks on it. But user expectation will be different. A user wants the push button to perform some action. Hence, he clicks on the button. Clicking like this is called event. An event represents a specific action done on a component. Clicking, double clicking, typing data inside the component, mouse over, etc. are all examples of events.

When an event is generated on the component, the component will not know about it because it cannot listen to the event. To let the component understand that an event is generated on it, we should add some listener to the components. A listener is an interface which listens to an event coming from a component. A listener will have some abstract methods which need to be implemented by the programmer.

When an event is generated by the user on the component, the event is not handled by the component. On the other hand, the component sends (delegates) that event to the listener attached to it. The listener will not handle the event. It hands over (delegates) the event to an appropriate method. Finally, the method is executed and the event is handled. This is called 'event delegation model'. See Figure 27.4.

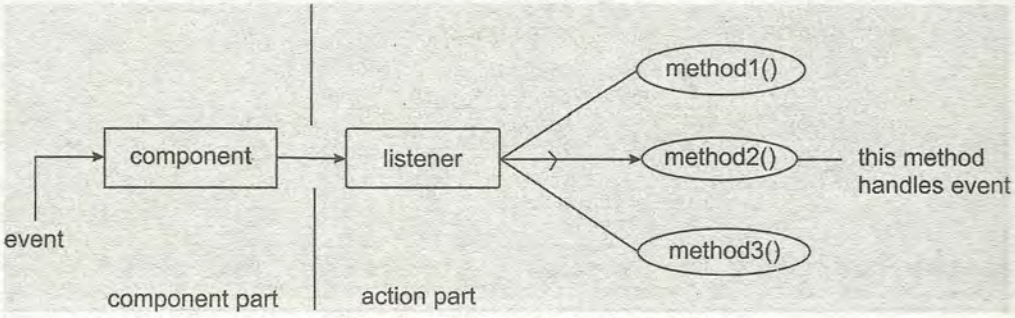


Figure 27.4 Event delegation model

Important Interview Question

What is event delegation model?

Event delegation model represents that when an event is generated by the user on a component, it is delegated to a listener interface and the listener calls a method in response to the event. Finally, the event is handled by the method.

Which model is used to provide actions to AWT components?

Event delegation model.

So, the following steps are involved in event delegation model:

- We should attach an appropriate listener to a component. This is done using `addxxxListener()` method. Similarly, to remove a listener from a component, we can use `removexxxListener()` method.
- Implement the methods of the listener, especially the method which handles the event.
- When an event is generated on the component, then the method in step 2 will be executed and the event is handled.

What is the advantage of event delegation model? In this model, the component is separated from the action part. So there are two advantages:

- The component and the action parts can be developed in two separate environments. For example, we can create the component in Java and the action logic can be developed in VisualBasic.
- We can modify the code for creating the component without modifying the code for action part of the component. Similarly, we can modify the action part without modifying the code for the component. Thus, we can modify one part without effecting any modification to other part. This makes debugging and maintenance of code very easy.

Closing the Frame

We know Frame is also a component. We want to close the frame by clicking on its close button. Let us follow these steps to see how to use event delegation model to do this:

- We should attach a listener to the frame component. Remember, all listeners are available in `java.awt.event` package. The most suitable listener to the frame is 'window listener'. It can be attached using `addWindowListener()` method as:

```
f.addWindowListener(WindowListener obj);
```

Please note that the `addWindowListener()` method has a parameter that is expecting object of `WindowListener` interface. Since it is not possible to create an object to an interface, we should create an object to the implementation class of the interface and pass it to the method.

- Implement all the methods of the `WindowListener` interface. The following methods are found in `WindowListener` interface:

```
public void windowActivated(WindowEvent e)
public void windowClosed(WindowEvent e)
public void windowClosing(WindowEvent e)
public void windowDeactivated(WindowEvent e)
public void windowDeiconified(WindowEvent e)
public void windowIconified(WindowEvent e)
public void windowOpened(WindowEvent e)
```

In all the preceding methods, `WindowListener` interface calls the `public void windowClosing()` method when the frame is being closed. So, implementing this method alone is enough, as:

```
public void windowClosing(WindowEvent e)
{
     //close the application
     System.exit(0);
}
```

For the remaining methods, we can provide empty body.

- So, when the frame is closed, the body of this method is executed and the application gets closed. In this way, we can handle the frame closing event.

These steps are shown in Program 3.

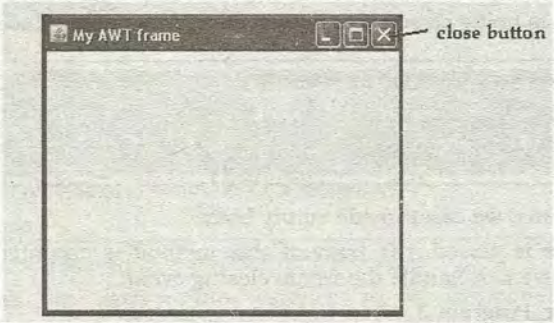
Program 3: Write a program which first creates a frame and then closes it on clicking the close button.

```
//Creating a frame and closing it.
import java.awt.*;
import java.awt.event.*;
class MyFrame extends Frame
{

     public static void main(String args[])
     {
          //create a frame with title
          MyFrame f = new MyFrame();

          //set a title for the frame
          f.setTitle("My AWT frame");

          //set the size of the frame
          f.setSize(300,250);
```

```
                //display the frame
                f.setVisible(true);

                //close the frame
                f.addWindowListener(new Myclass());

        }
}
class Myclass implements WindowListener
{
        public void windowActivated(WindowEvent e){}
        public void windowClosed(WindowEvent e){}
        public void windowClosing(WindowEvent e)
        {
                System.exit(0);
        }
        public void windowDeactivated(WindowEvent e){}
        public void windowDeiconified(WindowEvent e){}
        public void windowIconified(WindowEvent e){}
        public void windowOpened(WindowEvent e){}

}
```

Output

```
C:\> javac MyFrame.java
C:\> java MyFrame
CLICK ON CLOSE BUTTON, THE FRAME CLOSES
```



In this program, we not only create a frame but also close the frame when the user clicks on close button. For this purpose, we use `WindowListener` interface.

Here, we had to mention all the methods of `WindowListener` interface, just for the sake of method. This is really cumbersome. There is another way to escape this. There is a cl `WindowAdapter` in `java.awt.event` package, that contains all the methods of the `WindowListe` interface with an empty implementation (body). If we can extend `Myclass` from this `WindowAdap` class, then we need not write all the methods with empty implementation. We can write only t method which interests us. This is shown Program 4.

Program 4: Write a program to close the frame using `WindowAdapter` class.

```
//Creating a frame and closing it.
import java.awt.*;
import java.awt.event.*;
class MyFrame extends Frame
{
```

```
        public static void main(String args[])
        {
                //create a frame with title
                MyFrame f = new MyFrame();

                //set a title for the frame
                f.setTitle("My AWT frame");

                //set the size of the frame
                f.setSize(300,250);

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

                //close the frame
                f.addWindowListener(new Myclass());

        }
}
class Myclass extends WindowAdapter
{
        public void windowClosing(WindowEvent e)
        {
                System.exit(0);
        }
}
```

Output

```
C:\> javac MyFrame.java
C:\> java MyFrame
CLICK ON CLOSE BUTTON, THE FRAME CLOSES
```

Important Interview Question

What is an adapter class?

An adapter class is an implementation class of a listener interface which contains all methods implemented with empty body. For example, WindowAdapter is an adapter class of WindowListener interface. Adapter classes reduce overhead on programming while working with listener interfaces.

Please observe Program 4. In this, even the code of `Myclass` can be copied directly into `addWindowListener()` method, as:

```
f.addWindowListener(new WindowAdapter()
{
        public void windowClosing(WindowEvent e)
        {
                System.exit(0);
        }
});
```

This looks a bit confusing, but it is correct. We copy the code of `Myclass` into the method of `MyFrame` class. But, in the preceding code, we cannot find the name of `Myclass` anywhere in the code. It means the name of `Myclass` is hidden in `MyFrame` class and hence `Myclass` is an inner class in `MyFrame` class whose name is not mentioned. Such an inner class is called 'anonymous inner class'.

Important Interview Question

What is anonymous inner class?

Anonymous inner class is an inner class whose name is not mentioned, and for which only one object is created.

Let us now rewrite Program 4 again using anonymous inner class concept to close the frame.

Program 5: Write a program to close the frame using an anonymous inner class.

```
//Creating a frame and closing it.
import java.awt.*;
import java.awt.event.*;
class MyFrame extends Frame
{

      public static void main(String args[])
      {
            //create a frame with title
            MyFrame f = new MyFrame();

            //set a title for the frame
            f.setTitle("My AWT frame");

            //set the size of the frame
            f.setSize(300,250);

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

            //close the frame. Here Myclass name is not mentioned
            //but its object is passed to the method.
            f.addWindowListener(new WindowAdapter()
            {
                  public void windowClosing(WindowEvent e)
                  {
                        System.exit(0);
                  }
            });

      }
}
```

So far, we discussed the following three ways to close the frame:

- By implementing all the methods of `WindowListener` interface.
- By using `WindowAdapter` class and by implementing only the required method.
- By directly copying the code of an anonymous inner class.

Uses of a Frame

Once, the frame is created, we can use it for any of the purposes mentioned here:

- To draw some graphical shapes like dots, lines, rectangles, etc. in the frame.
- To display some text in the frame.
- To display pictures or images in the frame.
- To display components like push buttons, radio buttons, etc. in the frame.

Drawing in the Frame

Graphics class of `java.awt` package has the following methods which help to draw various shapes.

- `drawLine(int x1, int y1, int x2, int y2)`

This method is useful to draw a line connecting (`x1,y1`) and (`x2,y2`).

- `drawRect(int x, int y, int w, int h)`

This method draws outline of a rectangle. The left top corner of the rectangle starts at (`x,y`), the width is `w`, and the height is `h`.

- `drawRoundRect(int x, int y, int w, int h, int arcw, int arch)`: This method draws the outline of a rectangle with rounded corners. The rectange's top left corner starts at (`x,y`), the width is `w`, and height is `h`. The rectangle will have rounded corners. The rounding is specified by `arcw` and `arch`. `arcw` represents horizontal diameter of the arc at the corner and `arch` represents vertical diameter of the arc at the corner.
- `drawOval(int x, int y, int w, int h)`: This method draws a circle or ellipse bounded in the region of a rectangle starting at (`x,y`), with width `w` and height `h`.
- `drawArc(int x, int y, int w, int h, int sangle, int aangle)`: Draws an arc bounded by a rectangle region of (`x,y`), with width `w` and height `h`. Here, `sangle` represents the beginning angle measured from 3 o'clock position in the clock. `aangle` represents the arc angle relative to start angle. `sangle + aangle = end angle`. If the `sangle` is positive, the arc is drawn counter clock wise and if it is negative, then it is drawn clockwise. To understand the arcs, see Figure 27.5

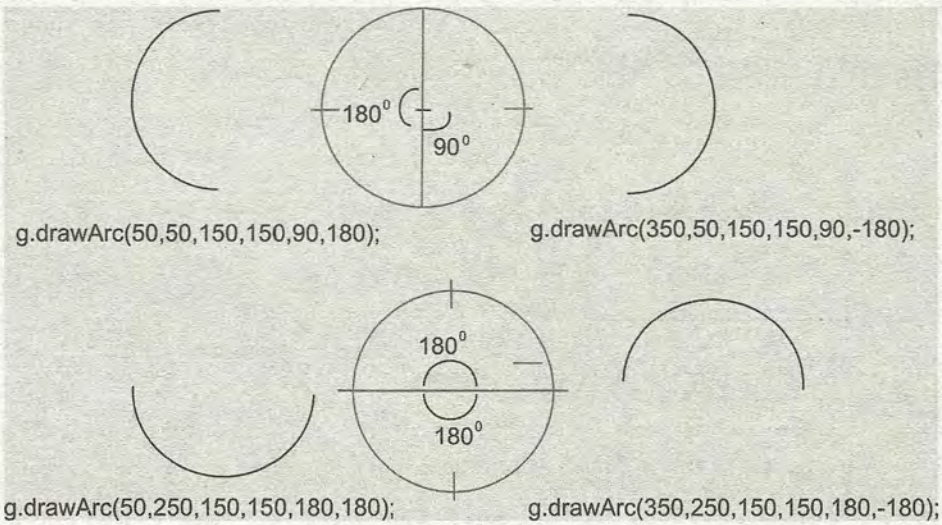


Figure 27.5 Understanding arcs

- `drawPolygon(int x[], int y[], int n)`: This method draws a polygon that connects pairs of coordinates mentioned by the arrays `x[]` and `y[]`. Here, `x[]` is an array which holds `x` coordinates of points and `y[]` is an array which holds `y` coordinates. `n` represents the number of pairs of coordinates.

To draw any of these shapes, we need `paint()` method of `Component` class, which refreshes the frame contents automatically when a drawing is displayed. This method is useful whenever we want to display some new drawing or text or images in the frame. The `paint()` method is automatically called when a frame is created and displayed.

Program 6: Write a program to draw a smiling face using the methods of `Graphics` class.

```
//Drawing a smiling face in a frame
import java.awt.*;
import java.awt.event.*;
class Draw1 extends Frame
{
      Draw1()
      {
            //close the frame
            this.addWindowListener(new WindowAdapter()
            {
                  public void windowClosing(WindowEvent e)
                  {
                        System.exit(0);
                  }
            });
      }

      //to refresh the frame contents
      public void paint(Graphics g)
      {
            //set blue color for drawing
            g.setColor(Color.blue);

            //display a rectangle to contain drawing
            g.drawRect(40,40,200,200);

            //face
            g.drawOval(90,70,80,80);

            //eyes
            g.drawOval(110,95,5,5);
            g.drawOval(145,95,5,5);

            //nose
            g.drawLine(130,95,130,115);

            //mouth
            g.drawArc(113,115,35,20,0,-180);
      }

      public static void main(String args[])
      {
            //create the frame
            Draw1 d = new Draw1();

            //set the size and title
            d.setSize(400,400);
            d.setTitle("My drawing");

            //display the frame
            d.setVisible(true);
      }
}
```

Output:

```
C:\> javac Draw1.java
C:\> java Draw1
```

Filling with Colors

To fill any shape with a desired color, first of all we should set a color using `setColor()` method. Then any of the following methods will draw those respective shapes by filling with the color.

- `fillRect(int x, int y, int w, int h)`: This method draws a rectangle and fills it with the specified color.
- `fillRoundRect(int x, int y, int w, int h, int arcw, int arch)`: This method draws filled rectangle with rounded corners.
- `fillOval(int x, int y, int w, int h)`: This method is useful to create an oval (circle or ellipse) which is filled with a specified color.
- `fillArc(int x, int y, int w, int h, int sangle, int aangle)`: This method draws an arc and fills it with a specified color.
- `fillPolygon(int x[], int y[], int n)` : This method draws and fills a polygon with a specified color.

Let us rewrite Program 6, using filled shapes and see the output.

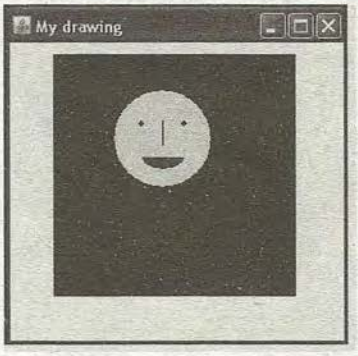
Program 7: Write a program that allows you to fill the shapes with some colors.

```
//Drawing a smiling face in a frame with filled colors
import java.awt.*;
import java.awt.event.*;
class Draw2 extends Frame
{
      Draw2()
      {
            //close the frame
            this.addWindowListener(new WindowAdapter()
            {
                  public void windowClosing(WindowEvent e)
                  {
                        System.exit(0);
                  }
            });
      }

      public void paint(Graphics g)
      {
            //set blue color
            g.setColor(Color.blue);

            //display a rectangle to contain drawing
            g.fillRect(40,40,200,200);

            //set yellow color
            g.setColor(Color.yellow);
```

```
                //face
                g.fillOval(90,70,80,80);

                //set black color
                g.setColor(Color.black);

                //eyes
                g.fillOval(110,95,5,5);
                g.fillOval(145,95,5,5);

                //nose
                g.drawLine(130,95,130,115);

                //set red color
                g.setColor(Color.red);

                //mouth
                g.fillArc(113,115,35,20,0,-180);
        }

        public static void main(String args[])
        {
                //create the frame
                Draw2 d = new Draw2();

                //set the size and title
                d.setSize(400,400);
                d.setTitle("My drawing");

                //display the frame
                d.setVisible(true);
        }
}
```

Output:

```
C:\> javac Draw2.java
C:\>java Draw2
```



Let us write a Java program to see how to create a polygon. Now, a polygon has several sides and create each side, we need x and y coordinates to connect by a straight line. All the x coordina and y coordinates can be stored in two arrays as:

```
        int x[] = {40,200,40,100};
        int y[] = {40,40,200,200};
```

Now, if a polygon is drawn, it starts at (40, 40) and connects it with (200,40), from there a line connects it to (40, 200) and finally ends at (100,200). So here, totally 4 pairs of coordinates are there, and hence the polygon can be drawn using:

```
g.drawPolygon(x,y,4);
```

Or, to get a filled polygon, we can use:

```
g.fillPolygon(x, y, 4);
```

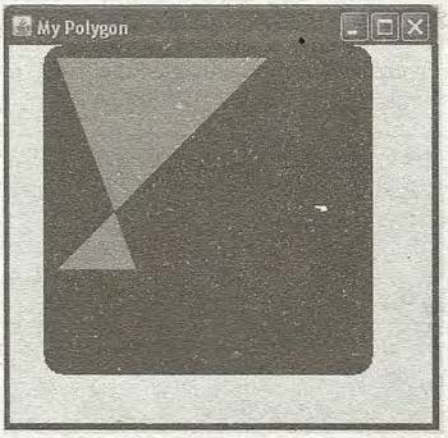
Program 8: Write a program to create a polygon that is filled with green color. This polygon must be created inside a rounded rectangle which is filled with red color.

```
//Drawing a smilie in a frame
import java.awt.*;
import java.awt.event.*;
class DrawPoly extends Frame
{
	DrawPoly()
	{
		//close the frame
		this.addWindowListener(new WindowAdapter()
		{
			public void windowClosing(WindowEvent e)
			{
				System.exit(0);
			}
		});
	}

	public void paint(Graphics g)
	{
		//set red color
		g.setColor(Color.red);

		//display a filled rounded rectangle
		g.fillRoundRect(30,30,250,250,30,30);

		//set green color
		g.setColor(Color.green);

		//take x and y coordinates in arrays
		int x[] = {40,200,40,100};
		int y[] = {40,40,200,200};

		//there are 4 pairs of x,y coordinates
		int num = 4;

		//create filled polygon connecting the coordinates
		g.fillPolygon(x, y, num);

	}

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

		//set the size and title
		d.setSize(400,400);
		d.setTitle("My Polygon");

		//display the frame
		d.setVisible(true);
```

```
        }
    }
```

Output:

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



Finally, another Java program to depict a small figure is shown in Program 9.

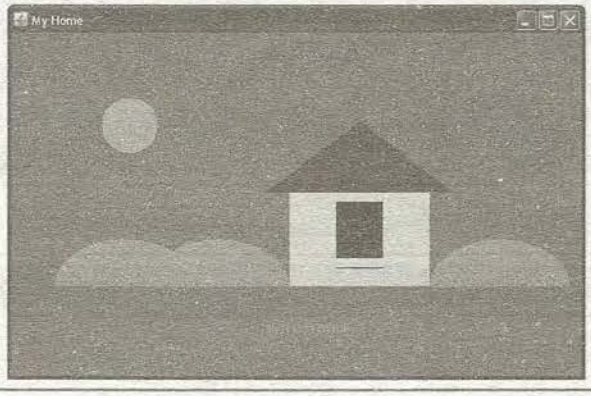
Program 9: Write a program to draw a home with moon at back ground.

```
//My Home
import java.awt.*;
import java.awt.event.*;
class Home extends Frame
{
    Home()
    {
        this.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        });
    }

    public void paint(Graphics g)
    {
        //store x,y coordinates in x[] and y[]
        int x[] = {375,275,475};
        int y[] = {125,200,200};
        int n = 3; //no. of pairs

        //set gray background for frame
        this.setBackground(Color.gray);


        //set yellow color for rectangle - house
        g.setColor(Color.yellow);
        g.fillRect(300,200,150,100);

        //set blue color for another rectangle - door
```

```
                g.setColor(Color.blue);
                g.fillRect(350,210,50,60);

                //draw a line - line below the door
                g.drawLine(350,280,400,280);

                //set dark gray for polygon - roof
                g.setColor(Color.darkGray);
                g.fillPolygon(x,y, n);

                //set cyan color for oval - moon
                g.setColor(Color.cyan);
                g.fillOval(100,100,60,60);

                //set green for arcs - grass
                g.setColor(Color.green);
                g.fillArc(50,250,150,100,0,180);
                g.fillArc(150,250,150,100,0,180);
                g.fillArc(450,250,150,100,0,180);

                //draw a line - the bottom most line of drawing
                g.drawLine(50,300,600,300);

                //display some text
                g.drawString("My Happy Home", 275, 350);
        }

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

                //set the size and title
                h.setSize(500,400);
                h.setTitle("My Home");

                //display the frame
                h.setVisible(true);
        }
}
```

Output:

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



Displaying Dots

To display a dot or point on the screen, we can take the help of `drawLine()` method. For example to display a point at (300,300) coordinates, we should use `drawLine()` method in such a way that the line is drawn starting from the same point and ending at the same point, as:

```
drawLine(300,300,300,300);
```

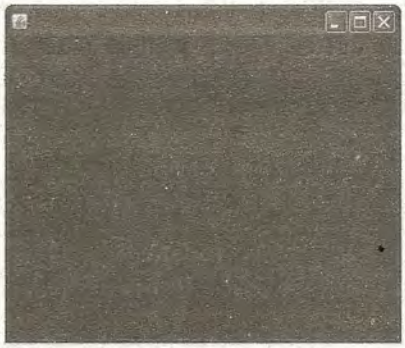
This will display a dot at (300,300). The following program illustrates how to draw several white dots on the black screen.

Program 10: Write a program to display several dots on the screen continuously.

```
//Displaying a group of dots on black screen
import java.awt.*;
import java.awt.event.*;
class Points extends Frame
{

      public void paint(Graphics g)
      {
            //set white color for dots
            g.setColor(Color.white);

            for (;;) //display dots forever
            {
            //generate x, y coordinates randomly. Maximum 800 and 600 px
            int x = (int) (Math.random() * 800);
            int y = (int) (Math.random() * 600);

            //Use drawLine() to display a dot
            g.drawLine(x, y, x, y);
            try{
                  //make a time delay of 20 milliseconds
                  Thread.sleep(20);
            }catch(InterruptedException ie){}

            }
      }
      public static void main(String args[])
      {
            //create frame
            Points obj = new Points();
            //set black background color for frame
            obj.setBackground(Color.black);
            //set the size and title for frame
            obj.setSize(500,400);
            obj.setTitle("Random dots");
            //display the frame
            obj.setVisible(true);

      }
}
```

Output:

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



Displaying text in the frame

To display some text or strings in the frame, we can take the help of `drawstring()` method of `Graphics` class, as:

```
g.drawString("Hello", x,y);
```

Here, the string `"Hello"` will be displayed starting from the coordinates `(x, y)`.

If we want to set some color for the text, we can use `setColor()` method of Graphics class, as:

```
g.setColor(Color.red);
```

There are two ways to set a color in AWT. The first way is by directly mentioning the needed color name from `Color` class, as `Color.red`, `Color.yellow`, `Color.cyan`, etc. All the standard colors are declared as constants in `Color` class, as shown in the Table 27.1:

Table 27.1

Color.black	Color.blue	Color.cyan
Color.pink	Color.red	Color.orange
Color.magenta	Color.darkGray	Color.gray
Color.lightGray	Color.green	Color.yellow
Color.white		

The second way to mention any color is by combining the three primary colors: red, green, and blue while creating `Color` class object, as:

```
Color c = new Color(r,g,b);
```

Here, r,g,b values can change from 0 to 255. 0 represents no color. 10 represents low intensity whereas 200 represents high intensity of color. Thus,

```
Color c = new Color(255,0,0); //red color
Color c = new Color(255,255,255); //white color
Color c = new Color(0,0,0);  //black color
```

This `Color` class object `c` should be then passed to `setColor()` method to set the color.

To set some font to the text, we can use `setFont()` method of `Graphics` class, as:

```
g.setFont(Font object);
```

This method takes `Font` class object, which can be created as:

```
Font f = new Font("SansSerif", Font.BOLD, 30);
```

Here, `"SansSerif"` represents the font name, `Font.BOLD` represents the font style and `30` represents the font size in pixels. There are totally 3 styles that we can use:

```
Font.BOLD
Font.ITALIC
Font.PLAIN
```

We can also combine any two styles, for example, to use bold and italic, we can write `Font.BOLD + Font.ITALIC`

Program 11: Write a program to display some text in the frame using `drawString()` method.

```
//Frame with background color and message
import java.awt.*;
import java.awt.event.*;
class Message extends Frame
{
	Message()
	{
		//close the frame when close button clicked
		addWindowListener(new WindowAdapter(){
		public void windowClosing(WindowEvent we)
		{
			System.exit(0);
		}
		});
	}//end of constructor

	public void paint(Graphics g)
	{
		//set background color for frame
		this.setBackground(new Color(100,20,20));

		//set font for the text
		Font f = new Font("Arial", Font.BOLD+Font.ITALIC,30);
		g.setFont(f);

		//set foreground color
		g.setColor(Color.green);

		//display the message
		g.drawString("Hello, How are U? ", 100,100);

	}

	public static void main(String args[])
	{
		Message m = new Message();
		m.setSize(400,300);
		m.setTitle("This is my text");
		m.setVisible(true);
	}
}
```

Output:

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

Knowing the Available Fonts

It is possible to know which fonts are available in our system. When we know the available fonts, we can use any of the font names to create `Font` class object. First of all, we should get the local graphics environment, as: `GraphicsEnvironment ge = GraphicsEnvironment.getLocalGraphicsEnvironment();`

Now, this `ge` contains the available font names which can be retrieved as:

```
String fonts[] = ge.getAvailableFontFamilyNames();
```

Program 12: Write a program to know which fonts are available in a local system.

```
//knowing the available fonts
import java.awt.*;
class Fonts
{
     public static void main(String args[])
     {

          //get the local graphics environment information into
          //GraphicsEnvironment object ge
          GraphicsEnvironment ge =
          GraphicsEnvironment.getLocalGraphicsEnvironment();

          //From ge, get available font family names into fonts[]
          String fonts[] = ge.getAvailableFontFamilyNames();

          System.out.println("Available fonts on this system: ");

          //retreive one by one the font names from fonts[] and display
          for(int i=0; i<fonts.length; i++)
          System.out.println(fonts[i]);
     }
}
```

Output:

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

```
C:\> java Fonts
Available fonts on this system:
Agency FB
Arial
Arial Black
Arial Narrow
Arial Rounded MT Bold
AS-TTDurga
Astro
Blackadder ITC
BN-TTDurga
Bodoni MT
Bodoni MT Black
Bodoni MT Condensed
Book Antiqua
Bookman Old Style
Bookshelf Symbol 7
Bradley Hand ITC
Calisto MT
Castellar
Century Gothic
Century Schoolbook
Comic Sans MS
Courier New
:
:
```

Displaying Images in the Frame

We can display images like `.gif` and `.jpg` files in the frame. For this purpose, we should fol these steps:

- Load the image into `Image` class object using `getImage()` method of Toolkit class.

```
Image img = Toolkit.getDefaultToolkit().getImage("diamonds.gif");
```

Here, `Toolkit.getDefaultToolkit()` method creates a default `Toolkit` class object. Using t object, we call `getImage()` method and this method loads the image `diamonds.gif` into `img` obje

- But, loading the image into `img` will take some time. JVM uses a separate thread to load image into `img` and continues with the rest of the program. So, there is a possibilit completing the program execution before the image is completely loaded into `img`. In this ca a blank frame without any image will be displayed. To avoid this, we should make JVM wa the image is completely loaded into `img` object. For this purpose, we need `MediaTracker` cl Add the image to `MediaTracker` class and allot an identification number to it starting f 0,1,...

```
MediaTracker track = new MediaTracker(this);
track.addImage(img,0);  //0 is the id number of image
```

- Now, `MediaTracker` keeps JVM waiting till the image is loaded completely. This is don `waitForID()` method.

```
track.waitForID(0);
```

This means wait till the image with the id number 0 is loaded into `img` object. Similarly, w several images are there, we can use `waitForID(1)`, `waitForID(2)`, etc.,

- Once the image is loaded and available in `img`, then we can display the image using `drawImage()` method of `Graphics` class, as:

```
g.drawImage(img,50,50,null);
```

Here, `img` is the image that is displayed at (50,50) coordinates and 'null' represents `ImageObserver` class object which is not required. `ImageObserver` is useful to store history of how the image is loaded into the object. Since, this is not required, we can use just null in its place. Another alternative for `drawImage()` is:

```
g.drawImage(img,50,50, 200, 250, null);
```

Here, additional 200, 250 represent the width and height of the image. They help to increase or decrease the size of the image to fit in the area allotted for it in the frame.

- To display an image in the title bar of the frame, we can use `setIconImage()` method of `Frame` class, as:

```
setIconImage(img);
```

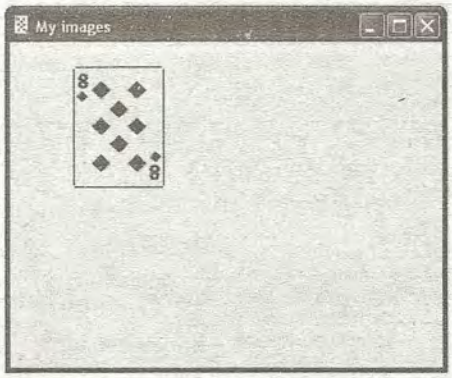
Program 13: Write a program to display an image in the frame and also in the title bar of the frame.

```
//Displaying an image in the frame and also in the title bar
import java.awt.*;
import java.awt.event.*;
class Images extends Frame
{
      //take a static type Image class object
      static Image img;

      Images()
      {
            //load an image into Image object
            img=Toolkit.getDefaultToolkit().getImage("diamonds.gif");

            //wait till the image is loaded into img object
            //for this purpose, create MediaTracker
            MediaTracker track = new MediaTracker(this);

            //add image to MediaTracker
            track.addImage(img,0);
            try{
                  //let the JVM wait till the image is loaded completely
                  track.waitForID(0);
            }catch(InterruptedException ie){}

            //close the frame
            addWindowListener(new WindowAdapter()
            {
                  public void windowClosing(WindowEvent we)
                  {
                        System.exit(0);
                  }
            });
      }

      public void paint(Graphics g)
      {
            //display the image in the frame at 50,50 pixels
            g.drawImage(img,50,50,null);

      }
```

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

                //set the size and title
                i.setSize(500,400);
                i.setTitle("My images");

                //display the same image in the title bar of frame
                i.setIconImage(img);

                //display the frame
                i.setVisible(true);
        }
}
```

Output:

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



Component Class Methods

A component is a graphical representation of an object on the screen. For example, push buttons, radio buttons, menus, etc. are components. Even frame is also a component. There is Component class available in `java.awt` package which contains the following methods which are applicable to any component.

- `Font getFont()`: This method returns the font of the component.
- `void setFont(Font f)`: This method sets a particular font `f` for the text of the component.
- `Color getForeground()`: This method gives the foreground color of the component.
- `void setForeground(Color c)`: This method sets a foreground color `c` to the component.
- `Color getBackground()`: Gets the background color of the component.
- `void setBackground(Color c)`: Sets the background color `c` for the component.
- `String getName()`: Returns the name of the component.
- `void setName(String name)`: Sets a new name for the component.

- `int getHeight()`: Returns the height of the component in pixels as an integer.
- `int getWidth()`: Returns the width of the component in pixels as an integer.
- `Dimension getSize()`: Returns the size of the component as an object of `Dimension` class. `Dimenison.width` and `Dimenision.height` will provide the width and height of the component.
- `int getX()`: Returns the current x coordinate of the component's origin.
- `int getY()`: Returns the current y coordinate of the component's origin.
- `Point getLocation()`: Gets the location of the component in the form of a point specifying the component's top-left corner.
- `void setLocation(int x, int y)`: Moves the component to a new location specified by (x,y).
- `void setSize(int width, int height)`: Resizes the component so that it has new width and height as passed to `setSize()` method.
- `void setVisible(boolean b)`: Shows or hides the component depending on the value of parameter `b`. `setVisible(true)` will display the component and `setVisible(false)` hides the component.
- `void setEnabled(boolean b)`: Enables or disables the component, depending on the value of the parameter `b`. `setEnabled(true)` will enable the component to function and `setEnabled(false)` will disable it.
- `void setBounds(int x, int y, int w, int h)`: This method allots a rectangular area starting at (x,y) coordinates and with width `w` and height `h`. The component is resized to this area before its display. This method is useful to specify the location of the component in the frame.

After creating a component, we should add the component to the frame. For this purpose, `add()` method is used.

```
f.add(component);
```

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

```
f.remove(component);
```

Push Buttons

Button class is useful to create Push buttons. A Push button is useful to perform a particular action.

- To create a push button with a label, we can create an object to `Button` class, as:

```
Button b = new Button();  //a button without any label is created.
Button b = new Button("label");  //a button with label is created.
```

- To get the label of the button, use `getLabel()`:

```
String l = b.getLabel();
```

- To set the label of the button:

```
b.setLabel("label");
```

Here, the "label" is set to the button b.

- When, there are several buttons, naturally the programmer should know which button is clicked by the user. For this purpose, getActionCommand() method of ActionEvent class is useful.

```
String s = ae.getActionCommand();
```

Here, s represents the label of the button clicked by the user.

- To know the source object which has been clicked by the user, we can use, getSource() method of ActionEvent class, as:

```
Object obj = ae.getSource();
```

Remember that only displaying the push buttons will not perform any actions. This means they cannot handle any events. To handle the events, we should use event delegation model. According to this model, an appropriate listener should be added to the push button. When the button is clicked, the event is passed to the listener and the listener calls a method which handles the event. With the push buttons, ActionListener is a suitable listener. To handle ActionListener to the button, we can use addxxxListener() method, as:

```
b.addActionListener(ActionListener obj);
```

Similarly, to remove action listener from the button, we can use removexxxListener() method, as:

```
b.removeActionListener(ActionListener obj);
```

In the two methods, addActionListener() and removeActionListener(), we are passing ActionListener object. Since ActionListener is an interface, we cannot directly create an object to it, we should pass object of implementation class of the interface in this case.

Let us write a program to create push buttons and also add some actions to the buttons. The following steps can be followed:

- First, set a layout manager using setLayout() method. In our program, we do not want to use any layout, hence we can pass null to setLayout() as:

```
this.setLayout(null);
```

Since our class Mybuttons extends Frame class, this represents Mybuttons class object or Frame class.

- Then create the push buttons by creating objects to Button class.
- Since, we are not using any layout manager, we should specify where to attach the buttons in the frame using setBounds() method.
- Then add the buttons to the frame using add() method.
- Add ActionListener to the buttons so that when we click on any button, the listener will handle the event by calling actionPerformed() method.

Program 14: Write a program that helps in creating 3 push buttons bearing the names of 3 colors. When a button is clicked, that particular color is set as background color in the frame.

```
//Push buttons
import java.awt.*;
import java.awt.event.*;
class Mybuttons extends Frame implements ActionListener
```

```
{
        //vars
        Button b1,b2,b3;

        Mybuttons()
        {
                //do not set any layout
                this.setLayout(null);

                //create 3 push buttons
                b1 = new Button("Yellow");
                b2 = new Button("Blue");
                b3 = new Button("Pink");

                //set the locations of buttons in the frame
                b1.setBounds(100,100,70,40);
                b2.setBounds(100,160,70,40);
                b3.setBounds(100,220,70,40);

                //add the buttons to the frame
                this.add(b1);
                this.add(b2);
                this.add(b3);

                //add action listener to the buttons
                b1.addActionListener(this);
                b2.addActionListener(this);
                b3.addActionListener(this);

                //close the frame
                addWindowListener(new WindowAdapter()
                {
                        public void windowClosing(WindowEvent we)
                        {
                                System.exit(0);
                        }
                });

        }//end of constructor

        //this method is called when a button is clicked
        public void actionPerformed(ActionEvent ae)
        {
                //know the label of the button clicked by user
                String str= ae.getActionCommand();

                //change the frame's background color depending on the button
                clicked
                if(str.equals("Yellow")) this.setBackground(Color.yellow);
                if(str.equals("Blue")) this.setBackground(Color.blue);
                if(str.equals("Pink")) this.setBackground(Color.pink);

        }

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

Output:

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

In the preceding program, observe the following statement:

```
this.setLayout(null);
```

Here, `this` represents current class object. Since the current class `Mybuttons` is a subclass of `Frame` class, we know that `Mybuttons` class object has a copy of `Frame` class object within it. So any `Frame` class methods can be invoked using `this`. So, `setLayout()` method of `Frame` class can be called using

```
this.setLayout()
```

In fact, even if `this` is also not used, there will not be any problem, since anyhow a method by default acts on present class object only.

`setLayout()` is useful to set a layout for the frame. What is this layout? A layout represents the manner of arranging components in the frame. All layouts are represented as implementation classes of `LayoutManager` interface. For example, the following layouts are available in AWT:

- `FlowLayout`
- `BorderLayout`
- `GridLayout`
- `CardLayout`
- `GridBagLayout`
- `BoxLayout` (belongs to `javax.swing` package)

We will discuss about layout managers in a later chapter. The following points are helpful to understand how to work with a layout manager:

- To set a layout for our components, we can pass the layout class object to the `setLayout` method as:

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

The preceding statement will set `FlowLayout` to the frame. It means `FlowLayout` will take the responsibility of arranging the components in the frame as a flow or as a line one after the other.

```
                b2.addActionListener(this);
                b3.addActionListener(this);

                //close the frame
                addWindowListener(new WindowAdapter()
                {
                        public void windowClosing(WindowEvent we)
                        {
                                System.exit(0);
                        }
                });

        }//end of constructor

        //this method is called when a button is clicked
        public void actionPerformed(ActionEvent ae)
        {
                //know the the button clicked by user

                if(ae.getSource() == b1) setBackground(Color.yellow);
                if(ae.getSource() == b2) setBackground(Color.blue);
                if(ae.getSource() == b3) setBackground(Color.pink);

        }

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

Output:

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

Listeners and Listener Methods

For working with push buttons, `ActionListener` is more suitable. Similarly, for other components other listeners are also available. All listeners are available in `java.awt.event` package. Table 27.2 summarizes the components, suitable listeners for the component, and the methods in the listener interface to be implemented when using that listener

Check Boxes

A check box is a square shaped box which displays an option to the user. The user can select one or more options from a group of check boxes. Let us see how to work with check boxes.

- To create a check box, we can create an object to `Checkbox` class, as:

```
Checkbox cb = new Checkbox();  //create a check box without any label
Checkbox cb = new Checkbox("label");  //with a label
Checkbox cb = new Checkbox("label", state);// if state is true, then the check
//box appears as if it is selected by default, else not selected.
```

- To get the state of a check box:

```
boolean b = cb.getState();
```

If the check box is selected, this method returns true, else false.

- To set the state of a check box:

```
cb.setState(true);
```

The check box `cb` will now appear as if it is selected.

- To get the label of a check box:

```
String s = cb.getLabel();
```

If `cb` is the check box object, then `getLabel()` will give its label.

- `void setLabel(String label)`

This method sets a new label to the check box.

- To get the selected check box label into an array we can use `getSelectedObjects()` method. This method returns an array of size 1 only.

```
Object x[ ] = cb.getSelectedObjects();
```

Here, `x[0]` stores the label of the check box selected by the user, if selected. Other wise, it stores null.

For example, there are 2 check boxes as:

```
Checkbox c1 = new Checkbox("One");
Checkbox c2 = new Checkbox("Two");

x = c1.getSelectedObjects();  //x is an array of Object type
if(x[0] == null) System.out.println("The check box is not selected");
else System.out.println("Selected check box label is : "+ x[0]);
```

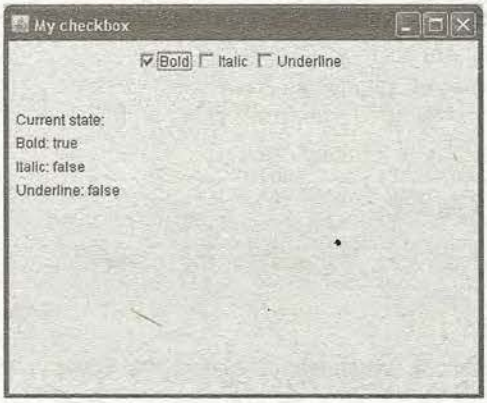
Program 16: Write a program to create 3 check boxes to display Bold, Italic, and Underline to the user.

```
//Checkbox demo
import java.awt.*;
import java.awt.event.*;
class Mycheckbox extends Frame implements ItemListener
{
      //vars
      String msg="";
```

```
        Checkbox c1,c2,c3;

        Mycheckbox()
        {
                //set flow layout manager
                setLayout(new FlowLayout());

                //display 3 checkboxes
                c1 = new Checkbox("Bold",true);
                c2 = new Checkbox("Italic");
                c3 = new Checkbox("Underline");

                //add the check boxes to the frame
                add(c1);
                add(c2);
                add(c3);

                //add item listener to the check boxes
                c1.addItemListener(this);
                c2.addItemListener(this);
                c3.addItemListener(this);

                //close the frame
                addWindowListener(new WindowAdapter()
                {
                        public void windowClosing(WindowEvent we)
                        {
                                System.exit(0);
                        }
                });
        } //end of constructor

        //this method is called the user clicks on a check box
        public void itemStateChanged(ItemEvent ie)
        {
                repaint(); //call paint() method
        }

        //display current state of checkboxes
        public void paint(Graphics g)
        {
                g.drawString("Current state: ", 10,100);
                msg = "Bold: "+c1.getState();
                g.drawString(msg, 10,120);
                msg = "Italic: "+c2.getState();
                g.drawString(msg, 10,140);
                msg = "Underline: "+c3.getState();
                g.drawString(msg, 10, 160);
        }

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

Output:

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



In this program, when the user clicks on any check box, the event is handled by `ItemListener` attached to the check boxes and hence the code of `itemStateChanged(ItemEvent ie)` gets executed.

Radio Button

A radio button represents a round shaped button, such that only one can be selected from a group of buttons. Radio buttons can be created using `CheckboxGroup` class and `Checkbox` classes. First of all, we should create a `CheckboxGroup` class object. While creating a radio button, we should pass CheckboxGroup object to the Checkbox class. It represents the group to which the radio button belongs. When the same CheckboxGroup object is passed to different radio buttons, then all those radio buttons will be considered as belonging to same group and hence the user is allowed to select only one from them.

- To create a radio button, pass `CheckboxGroup` object to `Checkbox` class object:

```
CheckboxGroup cbg = new CheckboxGroup();
Checkbox cb = new Checkbox("label", cbg, state);
```

Here, if state is true then the radio button appears to be already selected by default. If the state is false, then the radio button appears normal as if it is not selected.

- To know which radio button is selected by the user:

```
Checkbox cb = cbg.getSelectedCheckbox();
```

- To know the selected radio button's label:

```
String label = cbg.getSelectedCheckbox().getLabel();
```

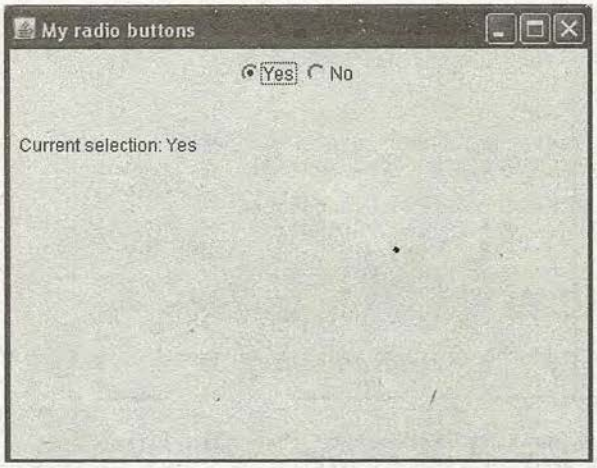
Program 17: Write a program that creates 2 radio buttons 'Yes' and 'No'. The user selects a button from them, and displays the selected button label.

```
//Radio buttons demo
import java.awt.*;
import java.awt.event.*;
class Myradio extends Frame implements ItemListener
{
    //vars
```

```
        String msg="";
        CheckboxGroup cbg;
        Checkbox y,n;

        Myradio()
        {
                //set the layout to flow layout
                setLayout(new FlowLayout());

                //create CheckboxGroup object
                cbg = new CheckboxGroup();

                //create 2 radio buttons
                y = new Checkbox("Yes",cbg,true);
                n = new Checkbox("No", cbg,false);

                //add the radio buttons to frame
                add(y);
                add(n);

                //add item listener to the radio buttons
                y.addItemListener(this);
                n.addItemListener(this);

                //close the frame
                addWindowListener(new WindowAdapter()
                {
                        public void windowClosing(WindowEvent we)
                        {
                                System.exit(0);
                        }
                });
        } //end of constructor

        //this method is called when a radio button is clicked
        public void itemStateChanged(ItemEvent ie)
        {
                repaint();  //call paint()
        }

        //display the selected radio label
        public void paint(Graphics g)
        {
                msg= "Current selection: ";
                msg+= cbg.getSelectedCheckbox().getLabel();
                g.drawString(msg,10,100);
        }

        public static void main(String args[])
        {
                //create frame
                Myradio mr = new Myradio();
                mr.setTitle("My radio buttons");
                mr.setSize(400,400);
                mr.setVisible(true);
        }
}
```

Output:

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



TextField

A TextField represents a long rectangular box where the user can type a single line of text. We can also display a line of text in the text field also.

- To create a TextField:

```
TextField tf = new TextField();  // a blank text field is created
TextField tf = new TextField(25);  //25 characters width of text field
TextField tf = new TextField("default text", 25); //default text is displayed
when  //the text field is displayed.
```

- To retrieve the text from a `TextField`:

```
String s = tf.getText();
```

- To set the text to a `TextField`:

```
tf.setText("text");
```

- To hide the text being typed into the `TextField` by a character `char`:

```
tf.setEchoChar('char');
```

Now, the original characters typed in the text field are not displayed. In their place, the `char` is displayed. This is useful to hide important text like credit card numbers, passwords, etc.

TextArea

A TextArea is similar to a text field, but it can accommodate several lines of text. For example, if the user wants to type his address which contains several lines, he can use a text area.

- To create a TextArea:

```
TextArea ta = new TextArea(); //an empty text area
```

```
TextArea ta = new TextArea(rows,cols); //text area with some rows and
//columns
TextArea ta = new TextArea("string"); //text area with predefined string
```

- To retrieve the text from a TextArea:

```
String s = ta.getText();
```

- To set the text to a TextArea:

```
ta.setText("text");
```

- To append the given text to the text area's current text:

```
ta.append("text");
```

- To insert the specified text at the specified position in this text area:

```
ta.insert("text", position);
```

Label

A Label is a constant text that is generally displayed along with a TextField or TextArea.

- To create a label:

```
Label l = new Label();   //create an empty label
Label l = new Label("text", alignment constant);
Here, the alignmentconstant may be one of the following:
Label.RIGHT, Label.LEFT, Label.CENTER
```

When the label is displayed, there would be some rectangular area allotted for the label. In this area, the label is aligned towards right, left, or center as per the alignment constant.

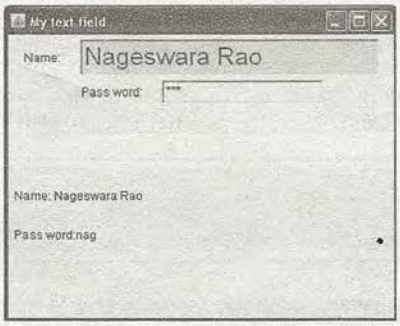
Program 18: Write a program to create two labels and two text fields for entering name and passwords. The password typed by the user in the text field is hidden.

```
//TextFields with a Labels
import java.awt.*;
import java.awt.event.*;
class MyText extends Frame implements ActionListener
{
    //vars
    TextField name,pass;

    MyText()
    {
        //set layout to flow layout
        setLayout(new FlowLayout());

        //create 2 labels
        Label n = new Label("Name: ", Label.LEFT);
        Label p = new Label("Pass word: ", Label.LEFT);

        //create text fields for name and password
        name = new TextField(20);
        pass = new TextField(20);

        //hide the password by *
```

```
                pass.setEchoChar('*');

                //use background,foreground colors and font for name textfield
                name.setBackground(Color.yellow);
                name.setForeground(Color.red);
                Font f = new Font("Arial",Font.PLAIN,25);
                name.setFont(f);

                //add the labels and textfields to frame
                add(n);
                add(name);
                add(p);
                add(pass);

                //add action listener to text fields
                name.addActionListener(this);
                pass.addActionListener(this);

                //close the frame
                addWindowListener(new WindowAdapter()
                {
                        public void windowClosing(WindowEvent we)
                        {
                                System.exit(0);
                        }
                });

        }//end of constructor

        //this method is executed when enter is clicked
        //display the text entered into the text fields
        public void actionPerformed(ActionEvent ae)
        {
                //create Graphics class object
                Graphics g = this.getGraphics();

                g.drawString("Name: "+name.getText(), 10,200);
                g.drawString("Pass word:"+pass.getText(),10,240);
        }


        public static void main(String args[])
        {
                //create the frame
                MyText mt = new MyText();
                mt.setTitle("My text field");
                mt.setSize(400,400);
                mt.setVisible(true);
        }
}
```

Output:

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



Choice Class

`Choice` class is useful to display a choice menu. It is a pop-up list of items and the user can select only one item from the available items.

- To create a Choice menu:

```
Choice ch = new Choice(); //create empty choice menu
```

- Once a choice menu is created, we should add items to it using `add()` method, as:

```
ch.add("item");
```

- To know the name of the item selected from the Choice menu:

```
String s = ch.getSelectedItem();
```

- Index of items in the Choice menu starts from 0 onwards. To know the index of the currently selected item:

```
int i = ch.getSelectedIndex();
This method returns -1 if nothing is selected.
```

- To get the item string, given the item index number,

```
String item = ch.getItem(int index);
```

- To know the number of items in the Choice menu,

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

- To remove an item from the choice menu at a specified position,

```
ch.remove(int position)
```

- To remove an item from the choice menu,

```
ch.remove(String item)
```

- To remove all items from the choice menu,

```
ch.removeAll();
```

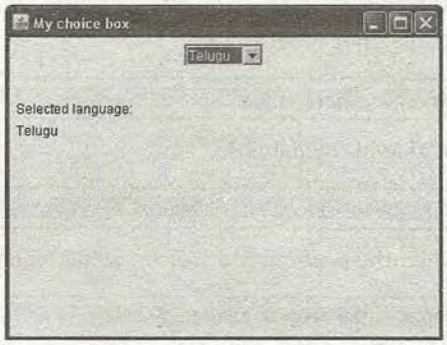
Program 19: Write a program to create a choice menu with names of some languages from which the user has to select any one item. The selected item must also be displayed in the frame.

```
//Choice box demo
import java.awt.*;
import java.awt.event.*;
class Mychoice extends Frame implements ItemListener
{
	//vars
	String msg;
	Choice ch;

	Mychoice()
	{
		//set flow layout to frame
		setLayout(new FlowLayout());

		//create an empty choice menu
		ch = new Choice();

		//add some items to choice menu
		ch.add("English");
		ch.add("Hindi");
		ch.add("Telugu");
		ch.add("Sanskrit");
		ch.add("French");

		//add the choice menu to frame
		add(ch);

		//add item listener to choice menu
		ch.addItemListener(this);

		//close the frame
		addWindowListener(new WindowAdapter()
		{
			public void windowClosing(WindowEvent we)
			{
				System.exit(0);
			}
		});

	}

	//this method is called when any item is clicked
	public void itemStateChanged(ItemEvent ie)
	{

		//call paint() method
		repaint();

	}

	//display selected item from the choice menu
	public void paint(Graphics g)
	{
		g.drawString("Selected language: ", 10,100);
		msg = ch.getSelectedItem();
		g.drawString(msg,10,120);
	}

	public static void main(String args[])
	{
		//create a frame
```

```
                Mychoice mc = new Mychoice();
                mc.setTitle("My choice box");
                mc.setSize(400,350);
                mc.setVisible(true);
        }
    }
```

Output:

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



List Class

This class is useful to create a list box which is similar to choice menu. A list box presents the user with a scrolling list of text items. The user can select one or more items from the list box.

- To create a list box, we can create an object to `List` class:

```
List lst = new List();
```

This statement creates a list box. The user can select only one item from the available items.

```
List lst = new List(3);
```

This statement creates a list box which displays initially 3 rows. The rest of the rows can be seen by clicking on the scroll button.

```
List lst = new List(3, true);
```

This list box initially displays 3 items. The next parameter `true` represents that the user can select more than one item from the available items. If it is `false`, then the user can select only one item.

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

```
lst.add("item");
```

- To get all the selected items from the list box:

```
String x[] = lst.getSelectedItems();
```

- To get a single selected item from the list box:

```
String x = lst.getSelectedItem();
```

- To get the selected items' position numbers:

```
int x[] = lst.getSelectedIndexes();
```

- To get a single selected item position number:

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

- To get the number of visible lines (items) in this list:

```
int x = lst.getRows();
```

- To get all the items available in the list box:

```
String x[] = lst.getItems();
```

- To get the item name when the position number (index) is known:

```
String item = lst.getItem(int index);
```

- To know how many number of items are there in the list box:

```
int x = lst.getItemCount();
```

- To remove an item at a specified position from the list:

```
lst.remove(int position);
```

- To remove an item whose name is given:

```
lst.remove(String item);
```

- To remove all items from the list:

```
lst.removeAll();
```

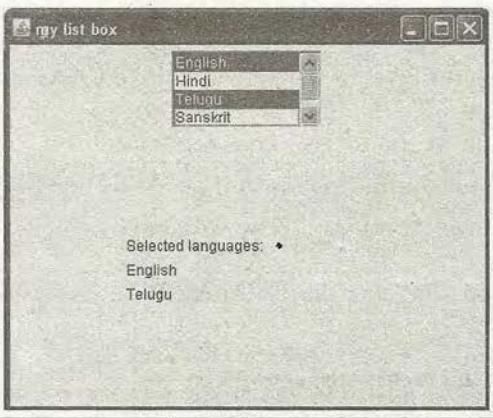
Program 20: Write a program to create a list box with names of some languages from where the user can select one or more items.

```
//List box demo
import java.awt.*;
import java.awt.event.*;

class Mylist extends Frame implements ItemListener
{
     //vars
     int[] msg;
     List lst;

     Mylist()
     {
          //set flow layout manager
```

```
        setLayout(new FlowLayout());

        //create an empty list box that displays 4 items initially
        //and multiple selection is also enabled
        lst = new List(4,true);

        //add items to the list box
        lst.add("English");
        lst.add("Hindi");
        lst.add("Telugu");
        lst.add("Sanskrit");
        lst.add("French");

        //add the list box to frame
        add(lst);

        //add item listener to the list box
        lst.addItemListener(this);

        //frame closing
        addWindowListener(new WindowAdapter()
        {
                public void windowClosing(WindowEvent we)
                {
                        System.exit(0);
                }
        });

    } //end of constructor

    public void itemStateChanged(ItemEvent ie)
    {
        //call the paint() method
        repaint();
    }

    public void paint(Graphics g)
    {
        g.drawString("Selected languages: ",100,200);

        //get the selected items position numbers into msg[]
        msg = lst.getSelectedIndexes();

        //know each selected item's name and display
        for(int i=0; i<msg.length; i++)
        {
                String item = lst.getItem(msg[i]);
                g.drawString(item, 100, 220+i*20);
        }
    }
    public static void main(String args[])
    {
        //create the frame
        Mylist ml = new Mylist();
        ml.setTitle("my list box");
        ml.setSize(400,400);
        ml.setVisible(true);
    }
}
```

Output:

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



In this program, the position numbers of the selected items are stored in an array and displayed in the frame.

Scrollbar Class

`Scrollbar` class is useful to create scrollbars that can be attached to a frame or text. `Scrollbars` are used to select continuous values between a specified minimum and maximum. Scrollbars can be arranged vertically or horizontally.

- To create a scrollbar we can create an object to `Scrollbar` class , as:

```
Scrollbar sb = new Scrollbar(alignment, start, step, min, max);
```

Here,

```
alignment:  Scrollbar.VERTICAL, Scrollbar.HORIZONTAL
start: starting value (Ex: 0)
step: step value (Ex: 30)  //represents scrollbar length
min: minimum value (Ex: 0)
max: maximum value (Ex: 300)
```

- To know the location of a scrollbar, we can use `getValue()` method that gives the position of the scrollbar in pixels, as:

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

- To update the scrollbar position to a new position, we can use `setValue()` method, as:

```
sb.setValue(int position);
```

- To get the maximum value of the scrollbar:

```
int x = sb.getMaximum();
```

- To get the minimum value of the scrollbar:

```
int x = sb.getMinimum();
```

- To get the alignment of the scrollbar:

```
int x = getOrientation();
```

This method returns 0 if the scrollbar is aligned HORIZONTAL and returns 1 if it is aligned VERTICAL.

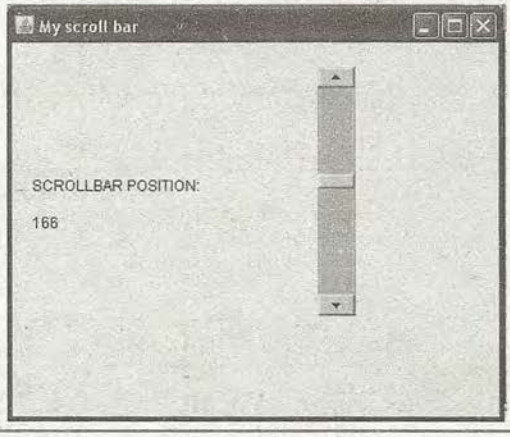
Program 21: Write a program to create a vertical scrollbar with scroll button length 30 px and with the starting and ending positions ranging from 0 to 400px.

```
//Creating a vertical scrollbar
import java.awt.*;
import java.awt.event.*;

class Myscroll extends Frame implements AdjustmentListener
{
      //vars
      String msg="";
      Scrollbar s1;

      Myscroll()
      {
            //do not set any layout
            setLayout(null);

            //create a vertical scrollbar
            s1 = new Scrollbar(Scrollbar.VERTICAL, 0,30,0,400);

            //set the location of scrollbar in the frame
            s1.setBounds(250,50,30,200);

            //add it to frame
            add(s1);

            //add adjustment listener to scrollbar
            s1.addAdjustmentListener(this);

            //frame closing
            addWindowListener(new WindowAdapter()
            {
                  public void windowClosing(WindowEvent we)
                  {
                        System.exit(0);
                  }
            });
      }

      public void adjustmentValueChanged(AdjustmentEvent ae)
      {
            repaint();  //call paint()
      }

      public void paint(Graphics g)
      {
            //display the position of scrollbar
            g.drawString("SCROLLBAR POSITION: ", 20,150);
            msg += s1.getValue();
            g.drawString(msg, 20,180);
            msg="";
      }

      public static void main(String args[])
      {
            //create the frame
            Myscroll ms = new Myscroll();
            ms.setTitle("My scroll bar");
            ms.setSize(400,400);
```

```
                ms.setVisible(true);
        }
    }
```

Output:

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



Knowing the Keys on Keyboard

To know which key is pressed on the keyboard, we can take the help of `KeyListener` interface. This interface has the following methods:

- `public void keyPressed(KeyEvent ke)` : This method is called when a key on the keyboard is pressed.
- `public void keyTyped(KeyEvent ke)` : This method is called when a key on the keyboard is typed. This method is applicable to the keys, which can be displayed and generally does not denote the special keys like function keys, control keys, shift keys, etc.
- `public void keyReleased(KeyEvent ke)` : This method is called when a key on the keyboard is released.

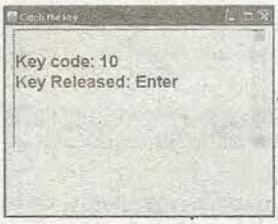
Program 22: Write a program to trap the key code and key name typed by the user on the keyboard and display them in a text area.

```
//Catching which key is pressed
import java.awt.*;
import java.awt.event.*;

class Keys extends Frame implements KeyListener
{
    //vars
    TextArea ta;
    String msg="";

    Keys()
    {
        //set flow layout
        setLayout(new FlowLayout());
```

```
        //create a text area to display the key code
        ta= new TextArea(5,25);

        //set some font and foreground color to text area
        Font f = new Font("SansSerif", Font.BOLD, 25);
        ta.setFont(f);
        ta.setForeground(Color.red);

        //add text area to frame
        add(ta);

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

        //close the frame
        addWindowListener(new WindowAdapter()
        {
                public void windowClosing(WindowEvent we)
                {
                        System.exit(0);
                }
        });
    }
    public void keyPressed(KeyEvent ke)
    {
        //get the code of the key pressed
        int keycode= ke.getKeyCode();
        msg += "\nKey code: "+keycode;

        //get the name of the key from the code
        String keyname= ke.getKeyText(keycode);
        msg += "\nKey pressed: "+ keyname;

        //display the key code and key name in text area
        ta.setText(msg);
        msg="";
    }

    public void keyTyped(KeyEvent ke)
    {  }

    public void keyReleased(KeyEvent ke)
    {
        //get the key code released
        int keycode= ke.getKeyCode();
        msg += "\nKey code: "+keycode;

        //get the key name from the code
        String keyname= ke.getKeyText(keycode);
        msg += "\nKey Released: "+ keyname;

        //display key code and key name in text area
        ta.setText(msg);
        msg="";

    }

    public static void main(String args[])
    {
        //create the frame
        Keys ks = new Keys();
        ks.setTitle("Catch the key");
        ks.setSize(400,400);
        ks.setVisible(true);
    }
}
```

Output:

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



Working with Several Frames

In software development, it is necessary to create several frames to display some screens to the user and to accept user input. Similarly, there may be several other screens which display the results to the user. So, we should know how to generate one frame from the other one.

Suppose, we create a frame with the name 'Frame1'. When a button like 'Next' is clicked on Frame1 then it should display 'Frame2'. There is another button 'Back' in Frame2, which when clicked takes us back to the first frame.

To create a new frame 'Frame2', we should write the following code in `Frame1` class:

```
Frame2 f2 = new Frame2();
f2.setSize(400,400);
f2.setVisible(true);
```

To make Frame2 terminate from memory, we can use `dispose()` method. To do this, write the following code in `Frame2` class:

```
this.dispose();
```

Program 23: Write a program to create a frame 'Frame1' with Next and Close buttons.

```
//This is Frame1
import java.awt.*;
import java.awt.event.*;
class Frame1 extends Frame implements ActionListener
{
      //vars
      Button b1,b2;
      Frame1()
      {
            setLayout(null);

            //create two buttons
            b1= new Button("Next");
            b2= new Button("Close");

            //set the location of buttons
            b1.setBounds(100,100,70,40);
            b2.setBounds(200,100,70,40);

            //add them to frame
```

```
                add(b1);
                add(b2);

                //add action listener to buttons
                b1.addActionListener(this);
                b2.addActionListener(this);

        }
        public void actionPerformed(ActionEvent ae)
        {
                //if Next button is clicked, display Frame2
                if(ae.getSource() == b1)
                {
                        //create Frame2 object and display
                        Frame2 f2 = new Frame2();
                        f2.setSize(400,400);
                        f2.setVisible(true);
                }
                else {
                        //if Close button is clicked, close application
                        System.exit(0);
                }
        }
        public static void main(String args[])
        {
                //create Frame1
                Frame1 f1 = new Frame1();
                f1.setSize(500,500);
                f1.setTitle("First frame");
                f1.setVisible(true);
        }
}
```

Note

DO NOT COMPILE THIS CODE TILL FRAME2.JAVA IS COMPILED.

Here, in this program when the user clicks the Next button, Frame2 is displayed. When the user clicks the Close button, the application is closed.

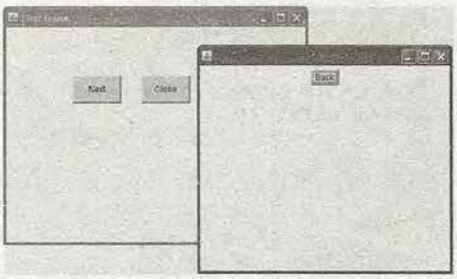
Program 24: Write a program to create Frame2 with Back button, such that when the user clicks Back button, Frame2 is closed and we see the Frame1 only.

```
//This is Frame2
import java.awt.*;
import java.awt.event.*;
class Frame2 extends Frame implements ActionListener
{
        //create a button
        Button b;

        Frame2()
        {
                //set layout to flow layout
                setLayout(new FlowLayout());

                //create the button
                b= new Button("Back");

                //add it to frame
                add(b);

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

```
        public void actionPerformed(ActionEvent ae)
        {
                //remove this frame from memory
                this.dispose();
        }
}
```

Output:

```
C:\> javac Frame2.java
C:\> java Frame1.java
C:\> java Frame1
```



Is it possible to send some data from Frame1 to Frame2 ? Yes, it is. Now, suppose we want to pass the roll number (10) and name 'Srinu' of a student to Frame2. We can do that at the time of creating Frame2 object by passing the data as shown here:

```
//this code should be written in Frame1
Frame2 f2 = new Frame2(10, "Srinu");
f2.setSize(400,400);
f2.setVisible(true);
```

Now, to catch the data in Frame2, we should take a parameterized constructor with two parameters which accepts the values coming from Frame1 as:

```
//instance vars in Frame2
int rno;
String name;

//constructor in Frame2
Frame2(int rno, String name)
{
     this.rno = rno;
     this.name = name;
}
```

Now, the values `10` and `"Srinu"` will be received by the constructor through its parameters and stored into the instance variables. These values can be used in `Frame2` class as the user wishes.

Conclusion

AWT provides the features to create and use graphics, images, components, etc. in our Java programs. These graphics features are very important especially in making the application more user-friendly and more attractive on Internet. We can also use `javax.swing` package for developing graphics programs which we see in later chapters.