

Subject: Programming in Java

Topic:Frame,Dialog,Menus

Notes By: Bishow Paudel

©nec 2003

Frame

A **frame** is a powerful features of awt.You can create your application using the Frame class.a Frame has a title bar,an Optional menu bar and a resizable border.As it is derived from java.awt.Container ,you can add components to a frame using add() method.The BorderLayout is a default Layout of the frame.A Frame receives mouse event KeyBoards Event and Focus Events.The constructor of Frame class receives the tilte of the frame as the parameter.The String is displayed on the title of the frame

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

After the window is created ,it can be displayed by calling the setVisible() method.The window can be sized by calling setSize() method.The program displays a frame.

```
import java.awt.*;
import java.applet.*;
import java.net.*;
public class HelloJavaParam
{
    public static void main(String arg[])
    {
        Frame f=new Frame("this is a TEST FRAME");
        f.setSize(200,300); //default size is (0,0)
        f.setBackground(Color.cyan);
        f.setVisible(true);
    }
}
```

You can add components to an frame using add() method.The default Layout of the frame is BorderLayout.

The following example show a frame with different graphical component(*save file as test.java since it contain the **main** method*)

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
class test extends Frame implements ActionListener
{
    test(String s)
    {
        super(s);
        setLayout(null);
        Button b=new Button("Exit");
        add(b);
        b.setBounds(150,250,40,20);
        b.addActionListener(this);
    }

    public void actionPerformed(ActionEvent ae)
```

```

    {
        System.exit(0);
    }

    public void paint(Graphics g)
    {
        setBackground(Color.red);
        Color c=new Color(64,68,100);
        g.setColor(c);
        g.fillOval(60,60,150,150);
        g.setColor(Color.white);
        g.fillOval(80,100,50,50);
        g.fillOval(140,100,50,50);
        int x[]={130,120,150};
        int y[]={160,180,180};
        g.fillPolygon(x,y,3);
        Font f=new Font("Courier",Font.BOLD,15);
        g.setFont(f);
        g.drawString("this is a graphics Demo",40,220);
    }
}

class testframe
{
    public static void main(String arg[])
    {
        test t=new test("this is a test frame");
        t.setSize(300,300);
        t.setVisible(true);
    }
}

```

Closable Frame:

When using a frame window , your program must remove that window from the screen when it is closed. To intercept a window close event ,you must implement the **windowClosing()** method of the **WindowListener** interface. Inside **windowClosing()** ,you must remove the window from screen..After running the following application user can close the frame /application by clicking the close box of the window.

```

import java.awt.*;
import java.applet.*;
import java.awt.event.*;

class test extends Frame implements ActionListener,WindowListener
{
    test(String s)
    {
        super(s);
        setLayout(null);
        Button b=new Button("Exit");
        add(b);
    }
}

```

```

        b.setBounds(150,250,40,20);
        b.addActionListener(this); //button is registered for the ActionEvent
        addWindowListener(this); //the frame itself is passed for the WindowEvent
    }
    //we now implement all the methods defined in the WindowListener Interface
    public void actionPerformed(ActionEvent ae)
    {
        System.exit(0);
    }
    public void windowOpened(WindowEvent we)
    {
    }
    public void windowClosed(WindowEvent we)
    {
    }
    public void windowClosing(WindowEvent we)
    {System.exit(0);
    }

    public void windowIconified(WindowEvent we)
    {
    }

    public void windowDeiconified(WindowEvent we)
    {
    }
    public void windowActivated(WindowEvent we)
    {
    }
    public void windowDeactivated(WindowEvent we)
    {
    }

    public void paint(Graphics g)
    {
        setBackground(Color.red);
        Color c=new Color(64,68,100);
        g.setColor(c);
        g.fillOval(60,60,150,150);
        g.setColor(Color.white);
        g.fillOval(80,100,50,50);
        g.fillOval(140,100,50,50);
        int x[]={130,120,150};
        int y[]={160,180,180};
        g.fillPolygon(x,y,3);
        Font f=new Font("Courier",Font.BOLD,15);
        g.setFont(f);
        g.drawString("this is a graphics Demo",40,220);
    }
}

```

```

class testframe
{
    public static void main(String arg[])
    {
        test t=new test("this is a test frame");
        t.setSize(300,300);
        t.setVisible(true);
    }
}

```

The above code can be reduced as shown below. Here as shown below the without implementing the WindowListener interface the frame can be made closeable .

```

import java.awt.*;
import java.applet.*;
import java.awt.event.*;

class test extends Frame
{
    test(String s)
    {
        super(s);
    }

    public void paint(Graphics g)
    {
        setBackground(Color.red);
        Color c=new Color(64,68,100);
        g.setColor(c);
        g.fillOval(60,60,150,150);
        g.setColor(Color.white);
        g.fillOval(80,100,50,50);
        g.fillOval(140,100,50,50);
        int x[]={130,120,150};
        int y[]={160,180,180};
        g.fillPolygon(x,y,3);
        Font f=new Font("Courier",Font.BOLD,15);
        g.setFont(f);
        g.drawString("this is a graphics Demo",40,220);
    }
}

class testframe
{
    public static void main(String arg[])
    {
        test t=new test("this is a test frame");
        t.setSize(300,300);
        t.setVisible(true);
        t.addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent we)
            { System.exit(0);
            }
        }
    }
}

```

```

    }); //Single line of code starting from t.addWindo....
}
}

```

Menus

Implementing a menu in a `Frame` involves connections among a number of different objects: `MenuBar`, `Menu`, `MenuItem`, and the optional `CheckboxMenuItem`. Several of these classes implement the `MenuContainer` interface. A menubar contains one or more `MenuObject`. Each `Menu Object` contains a list of `MenuItem` object. Each `MenuItem` object represent something that can be selected by the user. A hierarchies of nested submenu can be created. It is also possible to include checkable menu items. These are menu option of type `CheckboxMenuItem` and will have a check mark next to them when they are selected.

```

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

class TestMenu extends Frame implements ActionListener
{
    MenuItem quitnow; //Global object decleration WHY????
    Label output; //Global object decleration WHY????

    TestMenu(String title)
    {
        super("My menu test");
        MenuBar mb=new MenuBar();
        Menu calculate=new Menu("Calculate");
        // calculate.setMnemonic('C');
        MenuItem fact=new MenuItem("Factorial");
        //fact.setMnemonic('F');
        MenuItem square=new MenuItem("Square");
        // fact.setMnemonic('S');
        calculate.add(fact);
        calculate.add(square);
        Menu exit=new Menu("Exit");
        // exit.setMnemonics('F');
        Menu quit=new Menu("quit");
        quitnow=new MenuItem("Quit now");
        MenuItem cancel=new MenuItem("Cancel");
        quit.add(quitnow);
        quit.add(cancel);
        exit.add(quit);
        mb.add(calculate);
        mb.add(exit);
        setMenuBar (mb);
        output=new Label();//blank label that contains the output

        add(output, BorderLayout.CENTER);
        square.addActionListener(this);
        fact.addActionListener(this);
        quitnow.addActionListener(this);
    }
}

```

```

    }
public void actionPerformed(ActionEvent e)
{
/*    getSource()    return    the    source    component    where    as
getActionCommand()    returns    the    Label    of    the    action    generating
component.
*/
    if (e.getSource()==quitnow)
    {System.exit(0);
    }

    if(e.getActionCommand().equals("Square"))
    {
        int x=Integer.parseInt(JOptionPane.showInputDialog("Please
enter the number"));
        int sq=x*x;
        output.setText("The OutPut "+sq);
    }
    if(e.getActionCommand().equals("Factorial"))
    {
        int x=Integer.parseInt(JOptionPane.showInputDialog("Please
enter the number"));
        int fact=1;
        for(int i=1;i<=x;i++)
            fact=fact*i;

        output.setText("The Output "+fact);
    }
}

public static void main (String arg[])
{
    TestMenu tm=new TestMenu("Menu Test");
    tm.setSize(200,200);
    tm.setVisible(true);
    tm.addWindowListener(new WindowAdapter()
    {
        public void windowClosing(WindowEvent we)
        {
            System.exit(0);
        }
    }); //Single line of code starting from t.addWindo....

}
}

```

DialogBox:

The Dialog class provides a special type of display window that is normally used for pop-up messages or input from the user. They are similar to frame window, except that dialog box are always a child windows of top-level window (Applet or Frame). Also Dialog Box don't have a menu Bar. If the parent Frame is iconified, the Dialog disappears until the Frame is de-iconified. If the Frame is destroyed, so are all the associated dialogs.

Two Commonly used Constructors are:

1. Dialog (Frame *parentwindow*, Boolean *mode*)

Here Parent window is the owner of the dialog box. If mode is true, the dialog box is modal. (When a modal dialog box is active, all the input is directed to it until it is closed.) ie. If modal is true, the Dialog grabs all the user input of the program until it is closed. If modal is false, there is no special behavior associated with the Dialog.

2. Dialog(Frame *parentWindow*, String *Title*, Boolean *mode*)

This version of the constructor creates an instance of Dialog with parent as the Frame owning it and a window title of title. If mode is true, the Dialog grabs all the user input of the program until it is closed. If modal is false, there is no special behavior associated with the Dialog. Initially, the Dialog will be resizable.

eg:

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

```
class TestDialog extends Dialog implements ActionListener
{
    Frame parent;
    TestDialog(Frame parent, String title, boolean f)
    {
        super(parent, title, f);
        this.parent=parent;
        setLayout(new FlowLayout()); //layout for frame window
        setSize(200,200);
        setLocation(200,200);
        setVisible(true);
        add(new Label("Press this button"));
        Button b;
        add(b=new Button("cancel"));
        b.addActionListener(this);
    }
    public void actionPerformed(ActionEvent t)
    {
        dispose(); //dispose the current window from where the action is generated
        //parent.setVisible(true);
    }
    public void paint(Graphics g)
    {g.drawString("this is a dialog box", 20, 20);
    }
}
```

```
public class testframe1 extends Frame implements ActionListener
{
    testframe1(String s)
    {super(s);
    setLayout(new FlowLayout());
    Button b;
    add(b=new Button("click for DialogBox"));
    add(new Label("this is a main window"));
    b.addActionListener(this);
    }
    public void actionPerformed(ActionEvent tt)
```

```
    {
        JOptionPane.showMessageDialog(null,"The coming window is a
dialog box");
        // test.setVisible(false); //Try this
        TestDialog t=new TestDialog(this,"this is a test
dialog",false);
        //t.setVisible(true);
    }
public static void main(String arg[])
{
    testframe1 test=new testframe1("hi this is a test frame");
    test.setSize(200,200);
    test.setVisible(true);
    test.addWindowListener(new WindowAdapter()
    {public void windowClosing(WindowEvent we)
    {
        System.exit(0);
    }
    });
}
}
```

