Chapter-2.4
User interface components with Swing and Event Handling

Introduction to java swing

- Swing is a GUI widget toolkit which is an API for providing agraphical user interface (GUI) for Java programs.
- It has more powerful and flexible components than AWT.
- In addition to familiar components such as buttons, check boxes and labels, Swing provides several advanced components such as tabbed panel, scroll panes, trees, tables, Dialog, and lists.
- So Swing is a set of classes that provides more powerful and flexible components.
- **The swing related class are contained in javax.swing package.**

Advantagesof swing over AWT // key features of Swing

1. **Swing provides a *native look and feel* that emulates the look and feel of several platforms, and also supports a *pluggable look and feel* that allows applications to have a look and feel unrelated to the underlying platform.**
2. **Swing components are light weight while AWT components are heavy weight in terms of system resource utilization.** (Swing is build on the foundation of AWT).
3. **Swing was developed to provide a more sophisticated set of GUI components than the earlier Abstract Window Toolkit (AWT).**
4. **Swing component implement double buffering and automatic repainting thereby providing better looks and performance.**

Similarities between Swing and AWT

1. Swing application uses AWT's layout manager. However, it also introduces new layout manager like Springs, Grouplayout manager etc.
2. Swing application uses AWT event handeling.
3. Swing uses AWT to create an operating system window.
4. Both provide GUI widget toolkits.

Differences between Swing and AWT

1. Swing has more powerful and flexible components than AWT components.
2. AWT components are heavy weight components while Swing components are light weigtht.
3. AWT components provides platform dependent look/native look and feel while Swing components provides pluggable look and feel.
4. Swing components supports tool-tips text while AWT doesn't.
5. The Swing related classes are contained in **javax. swing** package while AWT components are contained in **java.awt** package.

Swing Is Built on the AWT

- AWT is a portable GUI library for Java applications/applets. The AWT provides the connection between our application and the native GUI while Java Swing implements a set of GUI components that build on AWT technology and it can provide a pluggable look and feel. Java Swing is implemented entirely in the Java programming language.
- Heavy-weight, it means the code will take comparatively more time to load and it will consume more System resources.
- AWT is considered to be heavy-weight because its components are dependent on the underlying Operating System. For instance, When we create an object of java.awt.Checkbox class, its underlying Operating System will generate a checkbox for us. This is also the reason, AWT components are platform dependent.
- On the other hand, most of the Java Swing components are implemented in Java itself. Some of the top level components like windows are dependent on the Operating System. But still, the overall program is comparatively light-weight than AWT.

bheshmathapa@gmail.com

- Although Swing eliminates a number of the limitations inherent in the AWT, Swing does not replace it. Instead, Swing is built on the foundation of the AWT. This is why the AWT is still a crucial part of Java.
- Swing also uses the same event handling mechanism as the AWT. Therefore, a basic understanding of the AWT and of event handling is required to use Swing.

Swing and the MVC Design Pattern

- In general, a visual swing component is a composite of three distinct aspects:
  • The way that the component looks when rendered on the screen
  • The way that the component reacts to the user
  • The state information associated with the component
- No matter what architecture is used to implement a swing component, it must implicitly contain these three parts.
- Over the years, one component architecture has proven itself to be effective: ***Model-View-Controller*, or MVC for short.**
- In MVC terminology, the *model* corresponds to the state information associated with the component. For example, in the case of a check box, the model contains a field that indicates if the box is checked or unchecked.
- The *view* determines how the component is displayed on the screen, including any aspects of the view that are affected by the current state of the model.
- The *controller* determines how the component reacts to the user. For example, when the user clicks a check box, the controller reacts by changing the model to reflect the user's choice (checked or unchecked). This then results in the view being updated.

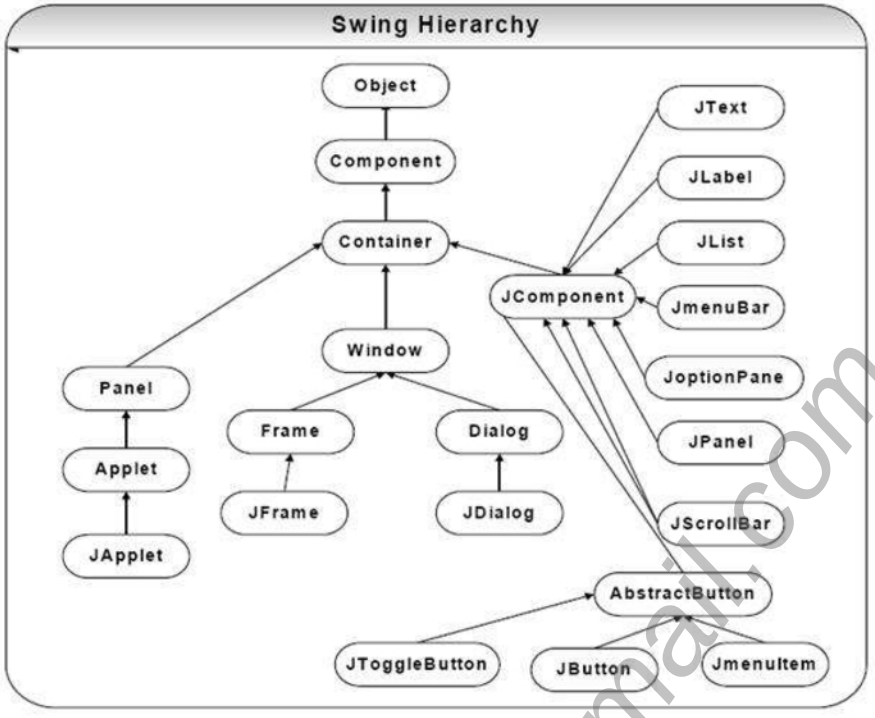
A Model-View-Controller Analysis of Swing Buttons

- Classes implementing the ButtonModel interface can define **the state of button**.
- Swing library contains a single class, called DefaultButtonModel, that implements this interface

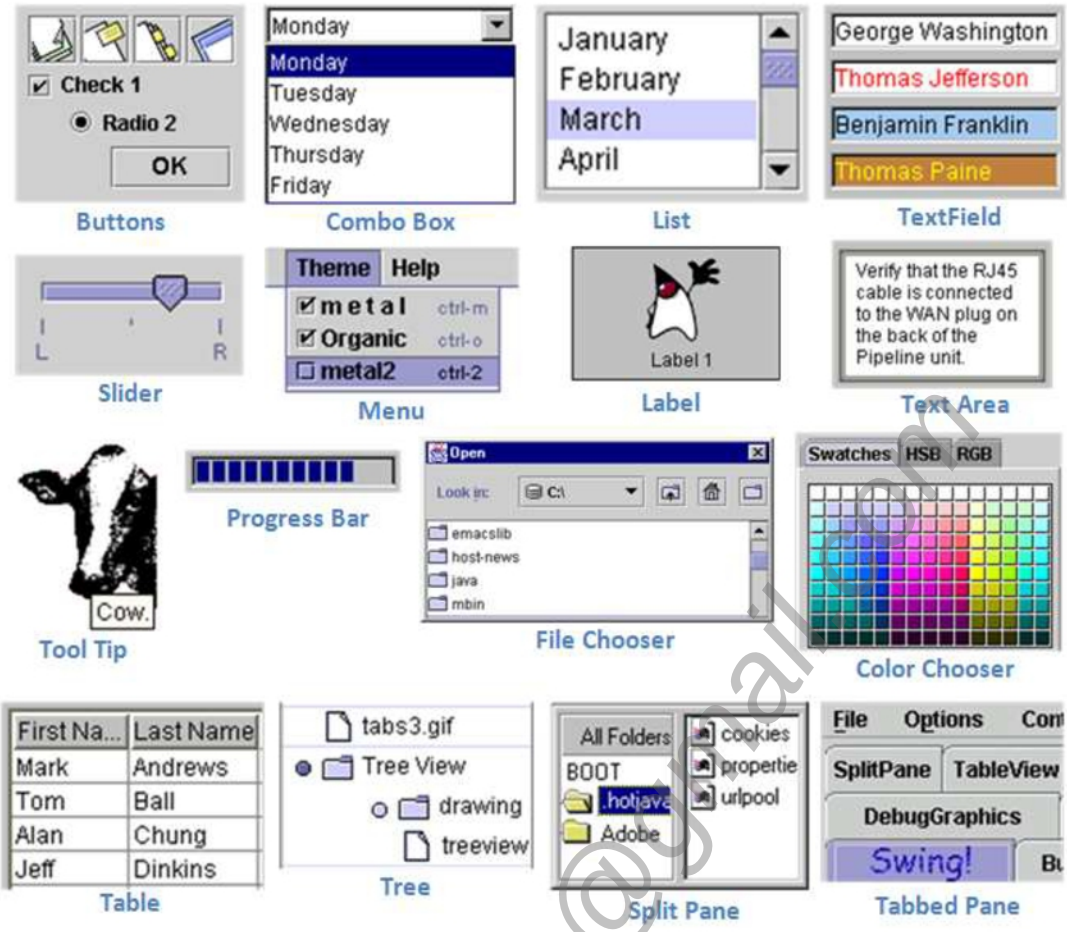
```
import java.awt.*;
import javax.swing.*;
public class Demo extends JFrame
{
        JButton b;
        public Demo()
        {
                setSize(500,300);
                b=new JButton("Submit");

                DefaultButtonModel d=new DefaultButtonModel();
                d=(DefaultButtonModel)b.getModel();   // Model
                d.setEnabled(false);   //controller


                setLayout(new FlowLayout());
                add(b);

                show();          //view
        }
        public static void main(String [] args) throws Exception
        {
                new Demo();
        }
}
```

So, in the above program, we updated the model of Button by chaning the property Enabled to false. This will be reflected in the view of button when we display the frame which consists of this Button.

Swing Hierarchy



1. **Container class**
   Container classes are the components that can hold other components. Swing container classes are
   i. JWindow
   ii. JDialog
   iii. JFrame
   iv. JApplet
2. **Component class**
   The swing API provides large number of GUI componetnts. Some of the mostly used components are listed below.
   i. JLabel
   ii. JTextArea
   iii. JTextField
   iv. JPasswordField
   v. JButton
   vi. JCheckBox
   vii. JComboBox
   viii. JRadioButton
   ix. Jlist
   x. JOptionPane
   xi. JMenuBar
   xii. JTree: encapsulates a tree based control
   xiii. ImageIcon: encapsulates an icon
   xiv. JTabbedPane: encapsulates a tabbed window



JFrame (see previous chapter for theory)

New method added

i. setDefaultCloseOperation(int operation)

The setDefaultCloseOperation() method can be used to specify one of several options for the close button

Frame.EXIT_ON_CLOSE — Exit the application.

Frame.HIDE_ON_CLOSE — Hide the frame, but keep the application running.

Frame.DISPOSE_ON_CLOSE — Dispose of the frame object, but keep the application running.

Frame.DO_NOTHING_ON_CLOSE — Ignore the click.

Frame.HIDE_ON_CLOSE is assigned by default.

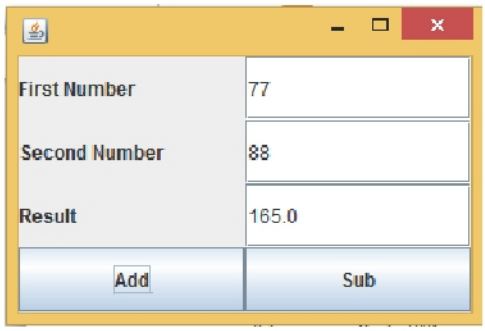
JLabel, JTextField, JButton (See previouse chapter for theory)

Write a complete GUI program using Swing components to add and subtract two numbers using GridLayout manager.

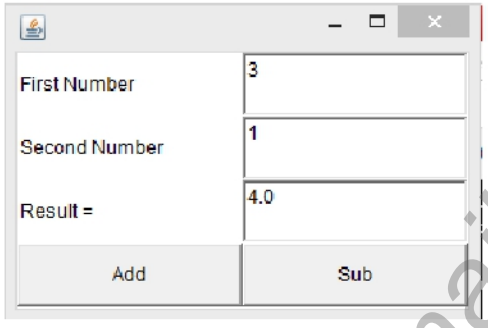
```
import javax.swing.*;
import java.awt.event.*;  // For using awt Event classes
import java.awt.*;  // For using GridLayout Class
public class Demo extends JFrame implements ActionListener
{
        JLabel lbl_fn,lbl_sn,lbl_res;
        JButton btn_add,btn_sub;
        JTextField txt_fn,txt_sn,txt_res;
```

```
        public Demo()
        {
                setSize(300,200);
                setLayout(new GridLayout(4,2));
                lbl_fn=new JLabel("First Number");
                add(lbl_fn);
                txt_fn=new JTextField(20);
                add(txt_fn);
                lbl_sn=new JLabel("Second Number");
                add(lbl_sn);
                txt_sn=new JTextField(20);
                add(txt_sn);
                lbl_res=new JLabel("Result");
                add(lbl_res);
                txt_res=new JTextField(20);
                add(txt_res);
                btn_add=new JButton("Add");
                btn_add.addActionListener(this);
                add(btn_add);
                btn_sub=new JButton("Sub");
                btn_sub.addActionListener(this);
                add(btn_sub);
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);// close the frame when the user presses the
                                                                close button on the frame
                setVisible(true);
        }

        public static void main(String [] args)
        {
                new Demo();
        }

        public void actionPerformed(ActionEvent e)// overriding method
        {
                double fn=Double.parseDouble(txt_fn.getText());
                double sn=Double.parseDouble(txt_sn.getText());
                if(e.getSource()==btn_add)
                {
                        double sum=fn+sn;
                        txt_res.setText(Double.toString(sum));
                }
                else if(e.getSource()==btn_sub)
                {
                        double dif=fn-sn;
                        txt_res.setText(Double.toString(dif));
                }
        }
}
```

Output:



Remember the output of same program using AWT components looked like this



Write a complete GUI program using Swing components to add and subtract two numbers using GridLayout manager. Use annynomous inner class for event handling.

```
import javax.swing.*;
import java.awt.event.*;  // For using awt Event classes
import java.awt.*;  // For using GridLayout Class
public class Demo extends JFrame
{
	JLabel lbl_fn,lbl_sn,lbl_res;
	JButton btn_add,btn_sub;
	JTextField txt_fn,txt_sn,txt_res;

	public Demo()
	{
		setSize(300,200);
		setLayout(new GridLayout(4,2));
		lbl_fn=new JLabel("First Number");
		add(lbl_fn);
		txt_fn=new JTextField(20);
		add(txt_fn);
		lbl_sn=new JLabel("Second Number");
		add(lbl_sn);
		txt_sn=new JTextField(20);
		add(txt_sn);
		lbl_res=new JLabel("Result");
		add(lbl_res);
		txt_res=new JTextField(20);
		add(txt_res);
		btn_add=new JButton("Add");
		btn_add.addActionListener( new ActionListener()
		{
			public void actionPerformed(ActionEvent e)
			{
				double fn=Double.parseDouble(txt_fn.getText());
				double sn=Double.parseDouble(txt_sn.getText());
				double sum=fn+sn;
				txt_res.setText(Double.toString(sum));
			}
```

```
                }
                );

                add(btn_add);
                btn_sub=new JButton("Sub");
                btn_sub.addActionListener( new ActionListener()
                {
                        public void actionPerformed(ActionEvent e)
                        {
                                double fn=Double.parseDouble(txt_fn.getText());
                                double sn=Double.parseDouble(txt_sn.getText());
                                double dif=fn-sn;
                                txt_res.setText(Double.toString(dif));
                        }
                }
                );

                add(btn_sub);
                setVisible(true);
        }

        public static void main(String [] args)
        {
                new Demo();
        }
}
```

Write a complete GUI program using Swing components to find the sum of two numbers using GridLayout manager. Your program display output if you press any key in the keyboard. Use annynomous inner class and adpater for event handling.

```
import javax.swing.*;
import java.awt.event.*;  // For using awt Event classes
import java.awt.*;  // For using GridLayout Class
public class Demo extends JFrame
{
        JLabel lbl_fn,lbl_sn,lbl_res;
        JTextField txt_fn,txt_sn,txt_res;
        public Demo()
        {
                setSize(300,200);
                setLayout(new GridLayout(3,2));
                lbl_fn=new JLabel("First Number");
                add(lbl_fn);
                txt_fn=new JTextField(20);
                add(txt_fn);
                txt_fn.addKeyListener(new KeyAdapter()
                {
                        public void keyReleased(KeyEvent e)
                        {
                                double fn=Double.parseDouble( txt_fn.getText());
                                double sn=Double.parseDouble( txt_sn.getText());
                                double res=fn+sn;
                                txt_res.setText(Double.toString(res));
                        }
                }
                );


                lbl_sn=new JLabel("Second Number");
```

```
                add(lbl_sn);
                txt_sn=new JTextField(20);
                add(txt_sn);
                txt_sn.addKeyListener(new KeyAdapter()
                {
                        public void keyReleased(KeyEvent e)
                        {
                                double fn=Double.parseDouble( txt_fn.getText());
                                double sn=Double.parseDouble( txt_sn.getText());
                                double res=fn+sn;
                                txt_res.setText(Double.toString(res));
                        }
                }
                );
                lbl_res=new JLabel("Result");
                add(lbl_res);
                txt_res=new JTextField(20);
                add(txt_res);
                setVisible(true);
        }

        public static void main(String [] args)
        {
                new Demo();
        }
}
```

JOptionPane

The JOptionPane class is used to show simple pop up **modal** dialog window to the user. We can use four different methods to create different types of Dialog box as listed below.

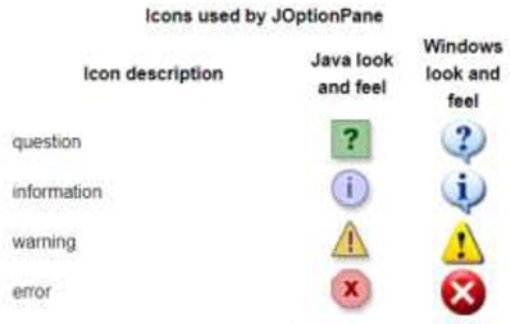
i. **showMessageDialog()**: simply show message text to the user. It consists of one button labelled "OK".

Syntax: 1. JOptionPane.showMessageDialog(Component f, String message);

2. JOptionPane.showMessageDialog(Component f, String message, String title, int messagetype)

Message type can be one of the below type

- ERROR_MESSAGE
- INFORMATION_MESSAGE
- WARNING_MESSAGE
- QUESTION_MESSAGE
- PLAIN_MESSAGE



Write a GUI program to input name, roll and marks in three different subjects. Show message dialog to display the information of student along with total marks obtained.

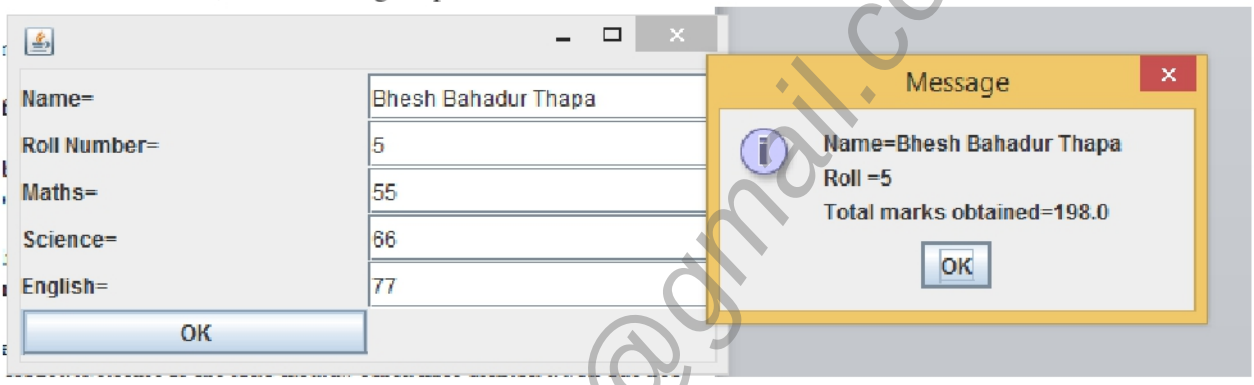
```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
```

```
public class Demo extends JFrame implements ActionListener
{
    JLabel lbl_name,lbl_roll,lbl_fs,lbl_ss,lbl_ts;
    JButton btn_ok;
    JTextField txt_name,txt_roll,txt_fs,txt_ss,txt_ts;
    public Demo()
    {
            setSize(400,200);
            setLayout(new GridLayout(6,2));
            lbl_name=new JLabel("Name=");
            add(lbl_name);
            txt_name=new JTextField(20);
            add(txt_name);
            lbl_roll=new JLabel("Roll Number=");
            add(lbl_roll);
            txt_roll=new JTextField(20);
            add(txt_roll);
            lbl_fs=new JLabel("Maths=");
            add(lbl_fs);
            txt_fs=new JTextField(20);
            add(txt_fs);
            lbl_ss=new JLabel("Science=");
            add(lbl_ss);
            txt_ss=new JTextField(20);
            add(txt_ss);
            lbl_ts=new JLabel("English=");
            add(lbl_ts);
            txt_ts=new JTextField(20);
            add(txt_ts);
            btn_ok=new JButton("OK");
            btn_ok.addActionListener(this);
            add(btn_ok);
            setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            setVisible(true);
    }

    public static void main(String [] args)
    {
            new Demo();
    }
    public void actionPerformed(ActionEvent e)// overriding method
    {
            String name=txt_name.getText();
            String roll=txt_roll.getText();
            double fs=Double.parseDouble(txt_fs.getText());
            double ss=Double.parseDouble(txt_ss.getText());
            double ts=Double.parseDouble(txt_ts.getText());
            double total=fs+ss+ts;
```

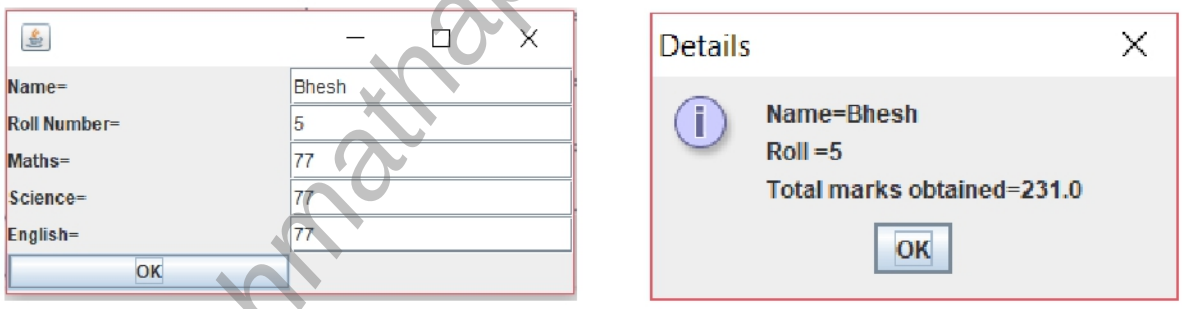
```
                if(e.getSource()==btn_ok)
                {
                        JOptionPane.showMessageDialog(null,"Name="+name +"\nRoll ="+roll+"\nTotal marks
                        obtained="+total);

                        /*
                        JOptionPane.showMessageDialog(null,"Name="+name +"\nRoll ="+roll+"\nTotal marks
                        obtained="+total,"Details",JOptionPane.INFORMATION_MESSAGE);
                        */
                }
        }
}
```

Output:

When the user click the ok button, the following output is observed.



OR



ii. **showInputDialog()**: prompt the user to input the text in textfield. It consists of two buttons labelled "OK " and "Cancel".

Syntax: 1. JOptionPane.showInputDialog(String message)

or

JOptionPane.showInputDialog(Component parent, String message)

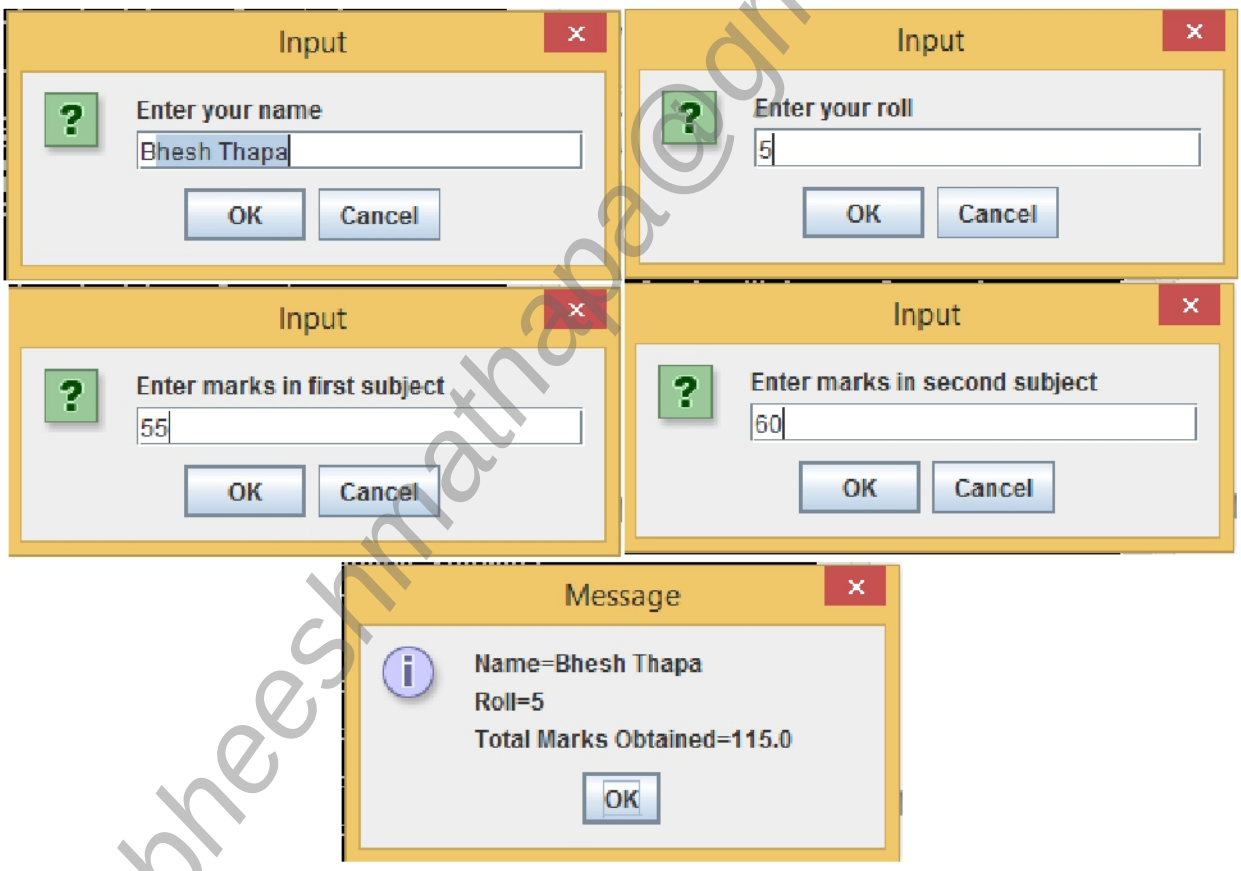
Shows a dialog with textfield, Ok button, cancel button, default question icon and default "input" title, asking the user to type a string. Shows a dialog with textfield, Ok button, cancel button, default question icon and default "input" title, asking the user to type a string.

2. JOptionPane.showInputDialog(Component Parent, String message, String title, int messagetype, ImageIcon ic, Object [] possibleValues, Object defaultvalue);

Note: if ImageIcon ic, is not assigned i.e. null then icon will be determined by message type by default.

WAP to input name, roll and marks in three different subject and display the information of student along with total marks obtained.

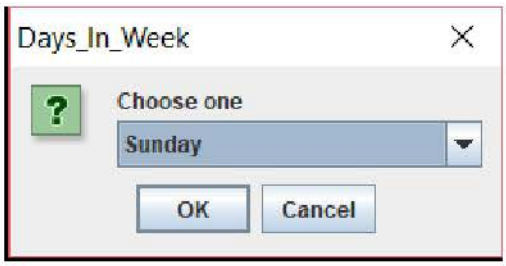
```
import javax.swing.*;
public class Demo1
{
	public Demo1()
	{
		String name=JOptionPane.showInputDialog("Enter your name");
		String roll=JOptionPane.showInputDialog("Enter your roll");
		String fs=JOptionPane.showInputDialog("Enter marks in first subject");
		String ss=JOptionPane.showInputDialog("Enter marks in second subject");
		double total=Double.parseDouble(fs)+Double.parseDouble(ss);
		JOptionPane.showMessageDialog(null,"Name="+name+"\nRoll="+roll+"\nTotal Marks Obtained="+total);
	}


	public static void main(String [] args)
	{
		new Demo1();
	}
}
```

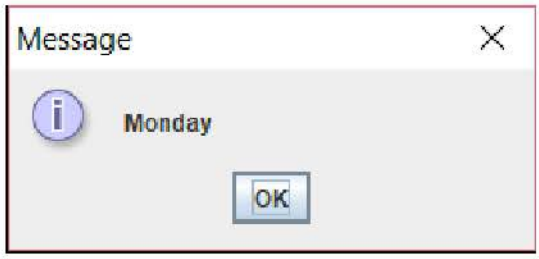


Display a dialog that allows user to select a day in a week.

```
import javax.swing.*;
public class Demo
{
	public Demo()
	{
		String[] possibleValues = { "Sunday", "Monday",
		"Tuesday","Wednesday","Thursday","Friday","Saturday" };
		Object selectedValue = JOptionPane.showInputDialog(null,"Choose one", "Days_In_Week",
```

```
                JOptionPane.QUESTION_MESSAGE, null,
                possibleValues, possibleValues[0]);
                String val=(String)selectedValue;
                JOptionPane.showMessageDialog(null,val);

        }

        public static void main(String [] args)
        {
                new Demo();
        }
}
```





iii. **showConfirmDialog()**: prompt the user for yes, no or cancel input. It consists of three buttons labelled "Yes","No" and "Cancel"

The constructors are

1. public static int showConfirmDialog(Component parent, String message) :
2. public static int showConfirmDialog(Component parent, String message, String title, int optionType)


```
//default icon, custom title
int n = JOptionPane.showConfirmDialog(
    frame,
    "Would you like green eggs and ham?",
    "An Inane Question",
    JOptionPane.YES_NO_OPTION);
```

Note: Option type can be,

optionType - an integer designating the options available on the dialog: DEFAULT_OPTION, YES_NO_OPTION, YES_NO_CANCEL_OPTION, or OK_CANCEL_OPTION

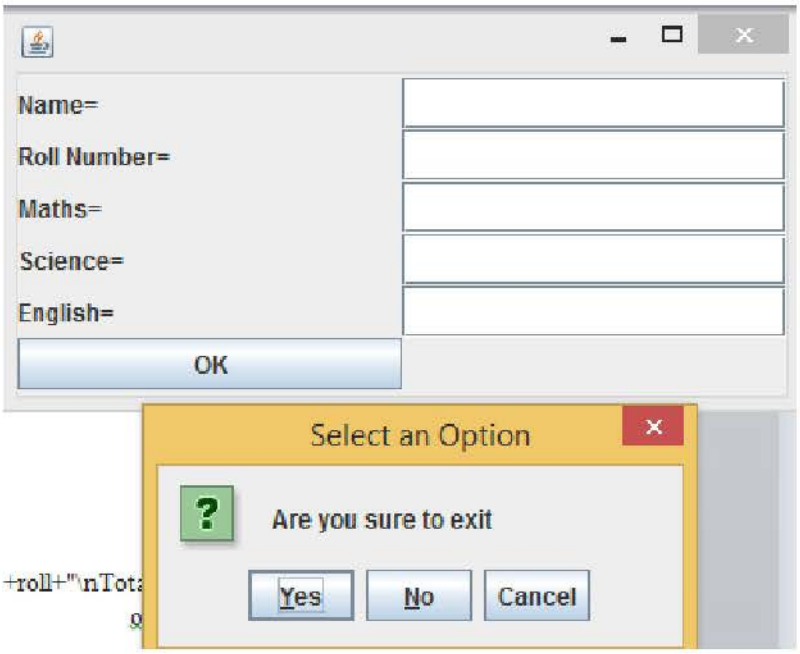
Write a GUI program which handles the window event that prompts the user whether or not to close the frame when the user presses the close button in the frame.

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class Demo2 extends JFrame implements ActionListener
{
        JLabel lbl_name,lbl_roll,lbl_fs,lbl_ss,lbl_ts;
        JButton btn_ok;
        JTextField txt_name,txt_roll,txt_fs,txt_ss,txt_ts;
        public Demo2()
        {
                setSize(400,200);
                setLayout(new GridLayout(6,2));
                setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
```

```
        addWindowListener(new WindowAdapter()
        {
                public void windowClosing(WindowEvent e)
                {
                        int a=JOptionPane.showConfirmDialog(null,"Are you sure to exit");//a=0,1 and 2 for yes
                        ,no and cancel
                        /*
                        JOptionPane j=new JOptionPane();
                        int a=j.showConfirmDialog(null,"Are you sure to exit");
                        */
                        if(a==0)
                        {
                                System.exit(0); // Parameter o represents succsssful termination, 1 or -1 for
                                                unsuccessful termination
                        }
                }
        }
        );
        lbl_name=new JLabel("Name=");
        add(lbl_name);
        txt_name=new JTextField(20);
        add(txt_name);
        lbl_roll=new JLabel("Roll Number=");
        add(lbl_roll);
        txt_roll=new JTextField(20);
        add(txt_roll);
        lbl_fs=new JLabel("Maths=");
        add(lbl_fs);
        txt_fs=new JTextField(20);
        add(txt_fs);
        lbl_ss=new JLabel("Science=");
        add(lbl_ss);
        txt_ss=new JTextField(20);
        add(txt_ss);
        lbl_ts=new JLabel("English=");
        add(lbl_ts);
        txt_ts=new JTextField(20);
        add(txt_ts);
        btn_ok=new JButton("OK");
        btn_ok.addActionListener(this);
        add(btn_ok);
        setVisible(true);
}

public static void main(String [] args)
{
        new Demo2();
}
```

```
    public void actionPerformed(ActionEvent e)// overriding method
    {
        String name=txt_name.getText();
        String roll=txt_roll.getText();
        double fs=Double.parseDouble(txt_fs.getText());
        double ss=Double.parseDouble(txt_ss.getText());
        double ts=Double.parseDouble(txt_ts.getText());
        double total=fs+ss+ts;
        if(e.getSource()==btn_ok)
        {
            JOptionPane.showMessageDialog(null,"Name="+name +"\nRoll ="+roll+"\nTotal marks

        obtained="+total);
        }
    }
}
```

Output: So if the user clicks the close button in the frame then following output is observed. If the user clicks yes button then the application exists otherwise not.



iv. **showOptionDialog**(): creates **customizable** dialog. Using this method, we can easily specify the title text, message text,Buttons types, and Dialog icon for the Dialog box.

The constructor is

1. public static int showOptionDialog(Component parentComponent, String message, String title, int optionType, int messageType, Icon icon, Object[] options, Object initialValue)

- **parentComponent** - determines the Frame in which the dialog is displayed; if null, or if the parentComponent has no Frame, a default Frame is used
- **message** - the message text to display
- **title - the title string for the dialog**
- **optionType** - an integer designating the options available on the dialog: DEFAULT_OPTION, YES_NO_OPTION, YES_NO_CANCEL_OPTION, or OK_CANCEL_OPTION
- **messageType** - an integer designating the kind of message this is, primarily used to determine the icon from the pluggable Look and Feel: ERROR_MESSAGE, INFORMATION_MESSAGE, WARNING_MESSAGE, QUESTION_MESSAGE, or PLAIN_MESSAGE

- **icon -** the icon to display in the dialog
- **options** - an array of objects indicating the possible choices the user can make; if the objects are components, they are rendered properly; non-String objects are rendered using their toString methods; if this parameter is null, the options are determined by the Look and Feel
- **initial value** that represents the default selection for the dialog; only meaningful if options is used; can be null


```
Object[] options = {"Yes, please",
                    "No way!"};
int n = JOptionPane.showOptionDialog(frame,
    "Would you like green eggs and ham?",
    "A Silly Question",
    JOptionPane.YES_NO_OPTION,
    JOptionPane.QUESTION_MESSAGE,
    null,     //do not use a custom Icon
    options,  //the titles of buttons
    options[0]); //default button title
```

JOptionPane.showOptionDialog(null, "Click OK to continue", "Warning", JOptionPane.YES_NO_OPTION, JOptionPane.WARNING_MESSAGE, null, null, null);

Example:

```
import javax.swing.*;
public class Demo3
{
        public static void main(String [] args)
        {
                Object[] options = { "Red", "Green", "Yellow" };
                int a=JOptionPane.showOptionDialog(null, "Click OK to continue", "Warning",
                JOptionPane.DEFAULT_OPTION, JOptionPane.WARNING_MESSAGE, null, options,
                options[0]);
                if(a==0)
                {
                //write code for Red option
                }
                else if(a==1)
                {
                //write code for Green option
                }
                else
                {
                //write code for Yellow option
                }

        }
}
```

Output:

Design a GUI form using swing with a textfield, a text label for displaying the input message "Input any String", and three buttons with caption checkPalindrome, Reverse, FIndVowels. Write a complete program for above scenario and for checking palindrome in first button, reverse it after clicking second button and extract the vowels from it after clicking third button.

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class Demo extends JFrame implements ActionListener
{
	JButton btn_pal, btn_rev, btn_vow;
	JTextField txt_input;
	JLabel lbl_msg;
	public Demo()
	{
		setLayout(new FlowLayout());
		lbl_msg=new JLabel("Input any String");
		txt_input=new JTextField(20);
		btn_pal=new JButton("Check Palindrome");
		btn_pal.addActionListener(this);
		btn_rev=new JButton("Reverse");
		btn_rev.addActionListener(this);
		btn_vow=new JButton("Find Vowels");
		btn_vow.addActionListener(this);
		add(lbl_msg);
		add(txt_input);
		add(btn_pal);
		add(btn_rev);
		add(btn_vow);
		setSize(400,200);
		setVisible(true);
	}
	public static void main(String [] args)
	{
		new Demo();
	}

	public void actionPerformed(ActionEvent e)
	{
		String inputstring=txt_input.getText();
		if(e.getSource()==btn_rev)
		{
			StringBuilder sb = new StringBuilder(inputstring);
			String rev=sb.reverse().toString();
			JOptionPane.showMessageDialog(null,"Reverse String="+rev);
		}
		else if(e.getSource()==btn_pal)
		{
			StringBuilder sb = new StringBuilder(inputstring);
			String rev=sb.reverse().toString();
			if(inputstring.equals(rev))
			{
				JOptionPane.showMessageDialog(null,"The input String is Palindrome");
			}
```

```
                else
                {
                        JOptionPane.showMessageDialog(null,"The input String is Not Palindrome");
                }
        }
        else
        {
                String totvow="";
                for (int i=0;i<inputstring.length();i++)
                {
                        char c=inputstring.charAt(i);
                        if(Character.isLetter(c))
                        {
                                if(c=='a'|| c=='e'|| c=='i'|| c=='o'||c=='u'|| c=='A'||
        c=='E'|| c=='I'|| c=='O'||c=='U')
                                {
                                        totvow=totvow+c;
                                }
                        }
                }
                JOptionPane.showMessageDialog(null,"Extracted Vowel="+totvow);
        }
    }
}
```

JDilaog: Creating Dialog

- Up to now we have used JOptionPane class to show a simple dialog.
- Now we are going to show how to create such a dialog by hand.
- **Dialog control represents a top-level window with a title and a border used to take some form of input from the user.**

- Some of the constructors that are used to create Dialog are
  i. public JDialog (JFrame owner): Constructs an initially invisible, modeless Dialog with the specified owner Frame and an empty title.
  ii. public JDialog(JFrame owner, String title):Constructs an initially invisible, modeless Dialog with the specified owner frame and title.
  iii. public JDialog (JFrame owner, boolean modal): Constructs an initially invisible Dialog with the specified owner Frame and modality and an empty title.
  iv. **public JDialog (JFrame owner, String title, boolean model):** Constructs an initially invisible Dialog with the specified owner Frame, title and modality

The Dailog Class has the following methods

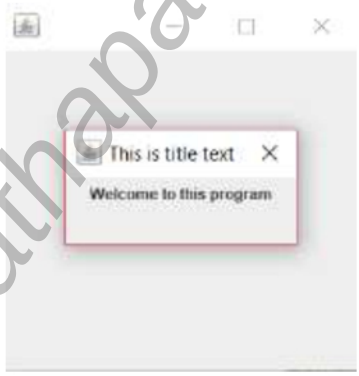
i. void setTitle(String title): Sets the title of the Dialog
ii. setSize(int width, int height): Set the dimensions of this Dialog
iii. String getTitle(): Returns the title of the Dialog.
iv. void show(): shows the current Dialog //Depreciated
v. void hide(): hides the current Dialog//Depreciated
vi. void setVisible(boolean b): shows or hides the Dialog based on specified boolean value
vii. void setResizable(boolean resizable) :Sets whether this dialog is resizable by the user.
viii. void setModel(boolean model) :Specifies whether this dialog should be model or model less.
ix. boolean isUndecorated() :Indicates whether this dialog is undecorated.
x. void setUndecorated(boolean undecorated) :Disables or enables decorations(title bar and boundary) for this dialog.

xi. void dispose(): Dispose/Close this Dialog

- Also dialog box can be implemented by extending the JDialog class

Example:

```
import javax.swing.*;
import java.awt.*;
public class DialogDemo extends JFrame
{
        JDialog d;
        JLabel b;
        public DialogDemo()
        {
                setSize(300,300);// frame size
                d=new JDialog(this,"This is title text",false);// model less dialog
                d.setSize(200,100);// dialog size
                d.setLayout(new FlowLayout());
                b=new JLabel("Welcome to this program");
                d.add(b);// add label to dialog
                setVisible(true);//show frame
                d.setLocationRelativeTo(this); //display dialog centred to Frame
                d.setVisible(true);// show dialog
        }
        public static void main(String [] args)
        {
                new DialogDemo();
        }
}
```



Data Exchange

- The most common reason to put up a dialog box is to get information from the user.
- Now the question is how to transfer data in and out of a dialog box.

Example: Program to demonstrate transferring data from frame to Dialog

Create a GUI frame with a text field and "OK" button. When user clicks on "Ok" button, display a Dialog having a label with text that is entered in the text field.

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class Demo extends JFrame implements ActionListener
{
        JButton btn_ok;
        JTextField txt_name;
        JLabel lbl_nm, lbl_dis;
        JDialog diag;
```

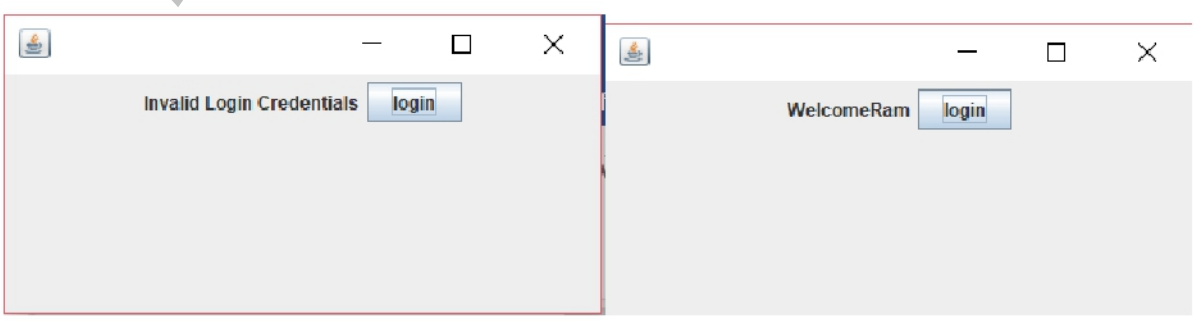
```
	public Demo()
	{
		lbl_nm=new JLabel("Name: ");
		txt_name=new JTextField(10);
		btn_ok=new JButton("Ok");
		btn_ok.addActionListener(this);
		setSize(300,100);
		setLayout(new FlowLayout());
		add(lbl_nm);
		add(txt_name);
		add(btn_ok);
		setLocationRelativeTo(null);//display frame centered to window
		setVisible(true);
	}

	public static void main(String [] args)
	{
		new Demo();
	}
	public void actionPerformed(ActionEvent e)
	{
		if(e.getSource()==btn_ok)
		{

			diag=new JDialog(this,"Alert",true);  //model Dialog
			diag.setSize(200,100);
			diag.setLocationRelativeTo(this);//display dialog centered to parent
			diag.setLayout(new FlowLayout());
			lbl_dis=new JLabel();
			lbl_dis.setText(txt_name.getText());
			diag.add(lbl_dis);
			diag.setVisible(true);
		}
	}
}
```

Example: Program to demonstrate transferring data from Dialog to Frame
Create A GUI with a label and login button. When the user presses the login button, allow user to input username and password from modal Dialog. Upon clicking the submit button in Dialog, the program must validate if the user is valid user or not. Finally display suitable message like "welcome Ram" in the label added in Frame.

Username: Ram Password: javaiseasy

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class Demo extends JFrame implements ActionListener
{
	JLabel lbl_msg,lbl_un, lbl_pwd;
	JButton btn_login;
	JButton btn_submit;
	JTextField txt_un;
	JPasswordField txt_pwd;
	JDialog d;
	public Demo()
	{
		setSize(400,200);
		setLayout(new FlowLayout());
		lbl_msg=new JLabel();
		btn_login=new JButton("login");
		add(lbl_msg);
```

```
            add(btn_login);
            btn_login.addActionListener(this);
            setVisible(true);
        }
        public void actionPerformed(ActionEvent e)
        {
            if(e.getSource()==btn_login)
            {
                d=new JDialog(this,"Login",true);
                d.setSize(300,200);
                d.setLayout(new GridLayout(3,2));
                lbl_un=new JLabel("Username");
                lbl_pwd=new JLabel("Password");
                txt_un=new JTextField(20);
                txt_pwd=new JPasswordField();
                btn_submit=new JButton("Submit");
                btn_submit.addActionListener(this);
                d.add(lbl_un);
                d.add(txt_un);
                d.add(lbl_pwd);
                d.add(txt_pwd);
                d.add(btn_submit);
                d.setVisible(true);
            }
            else if(e.getSource()==btn_submit)
            {
                String uname=txt_un.getText();
                String pwd=txt_pwd.getText();
                if(uname.equals("Ram") && pwd.equals("javaiseasy"))
                {
                    String msg="Welcome"+uname;
                    lbl_msg.setText(msg);
                    d.dispose();
                }

                else
                {
                    String msg="Invalid Login Credentials";
                    lbl_msg.setText(msg);
                    d.dispose();
                }
            }
        }

        public static void main(String[] args)
        {
            new Demo();
        }
}
```



File Chooser Dialog

- File choosers provide a GUI for navigating the file system, and then either choosing a file or directory from a list, or entering the name of a file or directory.
- Consturctor used to create file chooser dialog

  JFileChooser fc = new JFileChooser();
- Some useful methods are
  i. int showOpenDialog(Frame parent),: File open **model** dialogs for selecting single file
  ii. int showOpenMultipleDialog(Frame parent) : File open **model** dialogs for selecting multiple files
  iii. int showSaveDialog(Frame parent) : File save **model** dialogs
  iv. File getSelectedFile(): obtains the currently selected file
  v. **File []** getSelectedFiles() : Returns a list of selected files if the file chooser is set to allow multiple selection.

Note:

- We can use the **FileNameExtensionFilter** class for filtering the desired file. The constructor is defined as
  FileNameExtensionFilter filter = new FileNameExtensionFilter("JPG & GIF Images", "jpg", "gif");
  And use **addChoosableFileFilter(FileNameExtensionFilter obj)** method to add filter to required file chooser

  fc.addChoosableFileFilter(filter);
- we must import the below package for using this class

  import javax.swing.filechooser.FileNameExtensionFilter;

Create a GUI with a textarea, browse button and Save Button button. The program must allow the user to chose a text file with extension .txt and display the content in the text area. Finally, allow the user to save the file.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.filechooser.FileNameExtensionFilter;
public class Demo extends JFrame implements ActionListener
{
	JButton btn_browse, btn_save;
	JTextArea txt;

	public Demo()
	{
		setSize(300,200);
		setLayout(new FlowLayout());
		btn_browse=new JButton("Browse");
		btn_save=new JButton("Save");
		txt=new JTextArea(5,20);
		add(btn_browse);
		btn_browse.addActionListener(this);
		add(btn_save);
		btn_save.addActionListener(this);
		add(txt);
		setVisible(true);
	}

	public static void main(String [] args)
	{
		new Demo();
	}

	public void actionPerformed(ActionEvent e)
```

```
        {
                JFileChooser fc=new JFileChooser();
                FileNameExtensionFilter textFilter = new FileNameExtensionFilter("Text Files","txt");
                fc.addChoosableFileFilter(textFilter);
                if(e.getSource()==btn_browse)
                {
                        int ret=fc.showOpenDialog(this);  // o means selected 1 means cancelled
                        if(ret==0)
                        {
                                try
                                {
                                        File f=fc.getSelectedFile();
                                        FileReader fr=new FileReader(f);
                                        int c;
                                        String dat="";
                                        while ( (c=fr.read())!=-1)
                                        {
                                                dat=dat+ (char)c;
                                        }
                                        txt.setText(dat);
                                        fr.close();
                                }
                                catch(Exception ex)
                                {}
                        }
                }
                else if(e.getSource()==btn_save)
                {
                        int ret=fc.showSaveDialog(this); //o for save option clicked, 1 for others
                        if(ret==0)
                        {
                                try
                                {
                                        File f = fc.getSelectedFile();
                                        FileWriter fw=new FileWriter(f);
                                        String msg=txt.getText();
                                        fw.write(msg);
                                        fw.close();
                                }
                                catch(Exception ex)
                                {}
                        }
                }
        }
}
```

Color Chooser Dialog

- Swing provides a component called JColorChooser which can be used it to let users pick a color value.
- The constructor used for creating color chooser dialog is
    1. JColorChooser() : It is used to create a color chooser panel with white color initially.
- The Methods used for showing color chooser dialog is
    1. Color showDialog(Component c, String title, Color initialColor): It is used to show the color chooser dialog box.

A GUI consists of TextArea and browse color button. Upon clicking the browse color button the user must be allowed to select a color and finlally apply the selected color as background color for textarea.

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

```
import java.awt.*;
public class Demo extends JFrame implements ActionListener
{
	JButton btn_browse;
	JColorChooser cc;
	JTextArea txt;
	public Demo()
	{
		setSize(500,500);
		setLayout(new FlowLayout());
		txt=new JTextArea(5,10);
		btn_browse=new JButton("Browse Color");
		btn_browse.addActionListener(this);
		add(txt);
		add(btn_browse);
		setVisible(true);
	}
	public void actionPerformed(ActionEvent e)
	{
		cc=new JColorChooser();
		Color color=cc.showDialog(this,"Select a color",Color.RED);
		txt.setBackground(color);
	}
	public static void main(String [] args)
	{
		new Demo();
	}
}
```

WAP to allow user to set the background color of frame as selected by the user in color chooser dialog.

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class Demo extends JFrame implements ActionListener
{
	JButton btn_browse;
	JColorChooser cc;
	Container c;
	public Demo()
	{
		c=getContentPane(); // Returns the contentPane object for this frame.
		setSize(500,500);
		setLayout(new FlowLayout());
		btn_browse=new JButton("Browse Color");
		btn_browse.addActionListener(this);
		add(btn_browse);
		setVisible(true);

	}
	public void actionPerformed(ActionEvent e)
	{
		cc=new JColorChooser();
		Color color=cc.showDialog(this,"Select a color",Color.RED);
		c.setBackground(color);  //apply the background color to the container i.e. Frame
	}
	public static void main(String [] args)
	{
		new Demo();
	}
}
```

JPasswordField

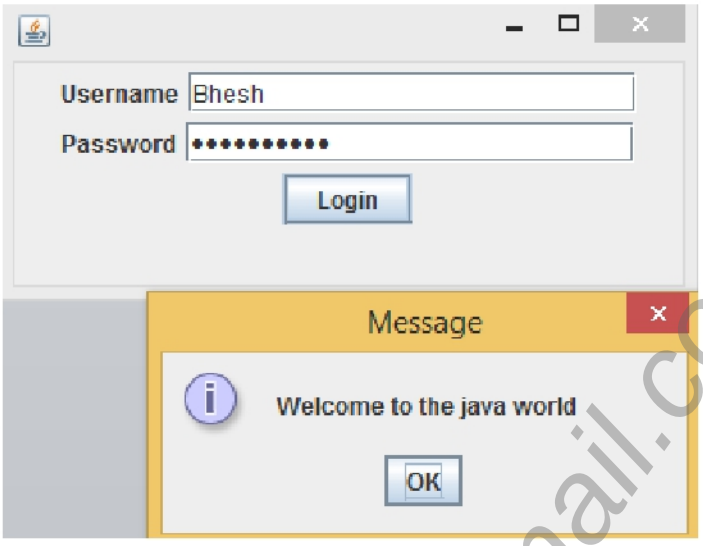
A JPasswordField is a JTextField with echo character enabled.

Write a GUI program to input username and password. If the user input is "Bhesh" and "javaiseasy" for username and password field then display the message "Welcome to the java world" other wise display "You are not valid user".

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class Demo4 extends JFrame implements ActionListener
{
	JLabel lbl_username,lbl_pwd;
	JTextField txt_username;
	JPasswordField txt_pwd;
	JButton btn_login;
	public Demo4()
	{
		setSize(350,150);
		setLayout(new FlowLayout());
		lbl_username=new JLabel("Username");
		add(lbl_username);
		txt_username=new JTextField(20);
		add(txt_username);
		lbl_pwd=new JLabel("Password");
		add(lbl_pwd);
		txt_pwd=new JPasswordField(20);
		add(txt_pwd);
		btn_login=new JButton("Login");
		btn_login.addActionListener(this);
		add(btn_login);
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setVisible(true);
	}

	public static void main(String [] args)
	{
		new Demo4();
	}

	public void actionPerformed(ActionEvent e)// overriding method
	{
		if(e.getSource()==btn_login)
		{
			String uname=txt_username.getText();
			String pwd=txt_pwd.getText();
			if(uname.equals("Bhesh") && pwd.equals("javaiseasy"))
			{
				JOptionPane.showMessageDialog(null,"Welcome to the java world");
			}
			else
			{
				JOptionPane.showMessageDialog(null,"You are not valid user");
			}
```

```
                    }
                }
        }
```

Output: if the user input Bhesh and javaiseasy for the textfield and passwordfield then following output is observed when the user clicks login button.



ImageIcon, JLabel, JTextField and ToolTipText

We can use ImageIcon class to create Icon from image. The constructors for creating ImageIcon is

ImageIcon(String file_name)

We can use this icon to set image for JButton and JLabels.

JLabel are subclass of JComponent class.

The constructors for creating JLabel are

JLabel(String txt)

JLabel(Icon i)

JLabel (String txt, Icon I, int align)

JTextField are subclass of JText component which extends JComponent.

ToolTip is used to show the text which is shown when hovering mouse. To set the tool tip text for any components we use **setToolTipText(String txt)** method.

The Swing Buttons

i. JButton

JButton are subclass of AbstractButton class which extends JComponent class.

The constructor for creating JButtons are

JButton(Icon i)

JButton(String txt)

JButton(String txt, ImageIcon i)

The program demonstrates how to set background image and tool tip text to label and button.

Example

```
import java.awt.*;
import javax.swing.*;
public class Demo extends JFrame
{
        JButton btn_fb;
        JLabel lbl_pic,lbl_picandtext;
        ImageIcon ii;
```

```
        public Demo()
        {
                setSize(300,100);
                setLayout(new FlowLayout());
                ii=new ImageIcon("facebook_icon.png");
                lbl_pic=new JLabel(ii);
                lbl_pic.setToolTipText("facebook");
                add(lbl_pic);
                lbl_picandtext=new JLabel("Facebook",ii,JLabel.CENTER);
                                                        //Allignment can be LEFT or RIGHTor CENTER
                lbl_picandtext.setToolTipText("Facebook");
                add(lbl_picandtext);
                btn_fb=new JButton(ii);
                btn_fb.setToolTipText("Facebook");
                add(btn_fb);
                setVisible(true);
        }

        public static void main(String [] args)
        {
                new Demo();
        }
}
```

Output:

Note: Constructors for JLabel

JLabel(Icon)
JLabel(Icon, int)
JLabel(String)
JLabel(String, Icon, int)
JLabel(String, int)
JLabel()

i. **JCheckBox**

- The JCheckBox class provides the functionality of a check box.
- The constructor used to create check box is

  i. JCheckBox(String str): It creates a check box that has the text specified by str as a label.

  ii. JCheckBox(String str, ImageIcon im) : It creates a check box that has the text specified by str as a label and a image icon.

  iii. JCheckBox(String text, boolean selected) :Creates a check box with text and specifies whether or not it is initially selected.

iv. JCheckBox(String text, Icon icon, boolean selected) :Creates a check box with text and icon, and specifies whether or not it is initially selected.

- Some methods provided by JCheckBOx class are
  - String getLabel(): Returns the label of checkbox. //Depreciated
  - String getText(): Returns the label of checkbox.
  - void setLabel(String txt): Sets the label text to txt value in the argument//Depreciated
  - void setText (String txt): Sets the label text to txt value in the argument
  - boolean isSelected(): Returns true if the checkbox is checked. (Note: getState() in case of AWT)
  - void setSelected(boolean st):Sets the state of the checkbox to specified state Note: setState() in case of AWT)

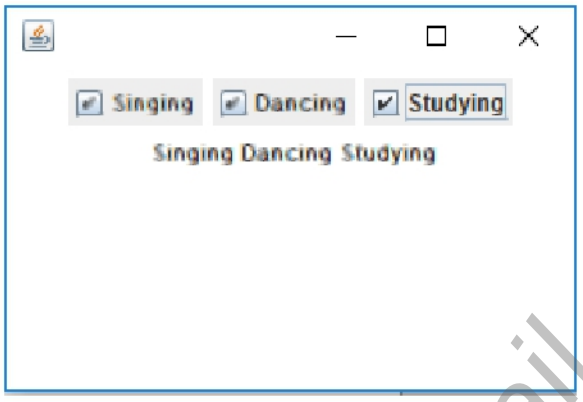
- **A JCheckBox generates ItemEvent, when the state of CheckBox is changed.**

Example:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class Demo1 extends Frame implements ItemListener
{
	JCheckBox cb_sing,cb_dance,cb_study;
	JLabel lb;
	public Demo1()
	{
		setSize(300,200);
		cb_sing=new JCheckBox("Singing");
		cb_sing.addItemListener(this);
		cb_dance=new JCheckBox("Dancing");
		cb_dance.addItemListener(this);
		cb_study=new JCheckBox("Studying");
		cb_study.addItemListener(this);
		lb=new JLabel("");
		setLayout(new FlowLayout());
		add(cb_sing);
		add(cb_dance);
		add(cb_study);
		add(lb);
		setVisible(true);
	}


	public static void main(String [] args)
	{
		new Demo1();
	}

	public void itemStateChanged(ItemEvent e)
	{
			String txt="";
			if(cb_sing.isSelected())
			{
				txt=txt+" " +cb_sing.getLabel();
			}
			if (cb_dance.isSelected())
			{
				txt=txt+" " +cb_dance.getLabel();
			}
			if(cb_study.isSelected())
			{
```

```
                txt=txt+" " +cb_study.getLabel();
            }
            lb.setText(txt);

        }
    }
```

Output:



iv. **JRadioButton**

- Radiobuttons are a group of mutually exclusive buttons, in which only one button can be selected at any one time.
- Some of the constructors provided by JRadioButton are

  JRadioButton(String str)
  JRadioButon(String str, boolean state)
  JRadioButton(String str, ImageIcon im, boolean state)
- Some methods provided by JRadioButton Class are
  i. void setLabel(String s) : It is used to set specified text on button.//Depreciated
  ii. void setText(String s) : It is used to set specified text on button.
  iii. String getLabel() : It is used to return the text of the button. //Depreciated
  iv. String getText() : It is used to return the text of the button.
  v. void setEnabled(boolean b): It is used to enable or disable the button.
  vi. void setIcon(Icon b): It is used to set the specified Icon on the button.
  vii. Icon getIcon(): It is used to get the Icon of the button.
  viii. boolean isSelected(): Returns true if the checkbox is checked.
  ix. void setSelected(boolean st):Sets the state of the checkbox to specified state

- **A JRadio Button generates action events each time the button selection changes.**

Example:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class Demo3 extends JFrame implements ActionListener
{
    JRadioButton rbtn_male,rbtn_female;
    JLabel lbl;
    ButtonGroup bg;
    public Demo3()
    {
        setSize(200,100);
        setLayout(new FlowLayout());
        rbtn_male=new JRadioButton("Male");
        rbtn_male.addActionListener(this);
        rbtn_female=new JRadioButton("Female");
        rbtn_female.addActionListener(this);
        lbl=new JLabel("None Selected");
```

```
                bg=new ButtonGroup();
                bg.add(rbtn_male);
                bg.add(rbtn_female);
                add(rbtn_male);
                add(rbtn_female);
                add(lbl);
                setVisible(true);
        }

        public static void main(String args[])
        {
                new Demo3();
        }
        public void actionPerformed(ActionEvent e)
        {
                if(rbtn_male.isSelected())
                {
                        lbl.setText("Male selected");
                }
                else
                {
                        lbl.setText("Female selected");
                }
        }
}
```

Output:

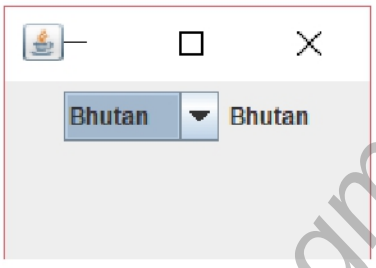
JComboBox

- Swing provides a combo box (a combination of a text field and a drop-down list) through the JComboBox class.
- A combo box normally displays one entry, but it will also display a drop-down list that allows a user to select a different entry.
- One of the constructor for creating JComboBox is
  JComboBox(Object[ ] items)
- Some of the methods provided by JComboBox class are
  - addItem(Object obj) : for dynamically adding new list of items
  - Object getSelectedItem(): obtain the item selected in the combobox
  - int getSelectedIndex(): return the index of selected item
- JComboBox also generates an **item event** when the state of selection changes, which occurs when an item is selected or deselected

Example:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class Demo6 extends JFrame implements ItemListener
{
        JComboBox cb;
        JLabel lbl;
        public Demo6()
        {
                setSize(200,300);
                setLayout(new FlowLayout());
                String [] items={"Nepal","India","Bhutan","Pakistan"};
```

```
                cb=new JComboBox(items);
                cb.addItemListener(this);
                lbl=new JLabel("None selected");
                add(cb);
                add(lbl);
                setVisible(true);
        }

        public static void main(String args[])
        {
                new Demo6();
        }
        public void itemStateChanged(ItemEvent e)
        {
                lbl.setText((String)(cb.getSelectedItem()));
        }
}
```



JTable and JScrollPane

- The JTable is used to display and edit regular two-dimensional tables of cells i.e. Table are used to display the data in the form of rows and column.
- JTables are typically placed inside of a JScrollPane.
- By default, a JTable will adjust its width such that a horizontal scrollbar is unnecessary.
- The constructor for creating table is

  Jtable(Object [][]data , Object []column_header)

  Where data is two dimensional information to be filled in rows and column of table, and header is one dimensional array infromtion with column header name.

- The constructor for creating scrollpane is

  JScrollPane(Component c)

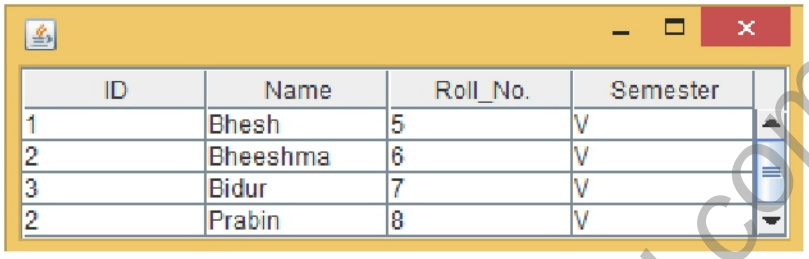
  Where, c is component to be added to the scrollpane,
- A Scrollpane is a component tha present a rectangular area in which a component may be viewed.
- We can add different components like table, panel, TextArea etc in scrollpane.

Example:

```
import javax.swing.*;
import java.awt.*;
public class Demo extends JFrame
{
        JTable table;
        JScrollPane jsp;
        public Demo()
        {
                setSize(400,125);
                String [] header={"ID","Name","Roll_No.","Semester"};
                String[][]data={{"1","Bhesh","5","V"},{"2","Bheeshma","6","V"},{"3","Bidur","7","V"},{"2","Pra
                bin","8","V"}};
                table=new JTable (data, header);
                jsp=new JScrollPane(table);
```

```
                add(jsp);
                setVisible(true);
        }
        public static void main(String [] args)
        {
                new Demo();
        }
}
```

Output:



WAP to demonstrate how scrollpane can be used with TextArea.

```
import javax.swing.*;
import java.awt.*;
public class Demo extends JFrame
{
        JTextArea txt;
        JScrollPane jsp;
        public Demo()
        {
                setSize(200,200);
                txt=new JTextArea(500,500);
                jsp=new JScrollPane(txt);
                add(jsp);
                setVisible(true);
        }
        public static void main(String [] args)
        {
                new Demo();
        }
}
```

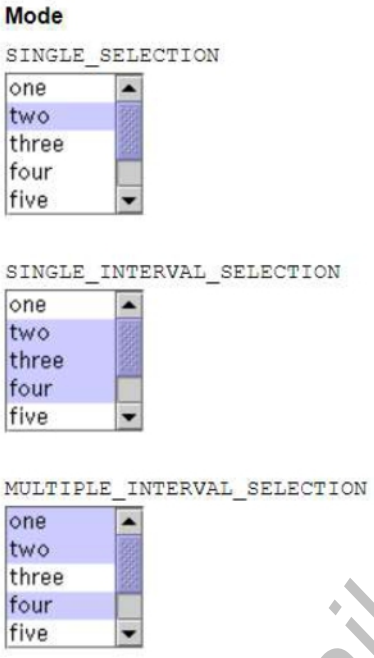
JList

- In Swing, the basic list class is called JList.
- It supports the selection of one or more items from a list.
- Although the list often consists of strings, it is possible to create a list of just about any object that can be displayed.
- JList provides several constructors. The one used here is
  JList(Object[ ] items)
- A JList generates a **ListSelectionEvent** when the user makes or changes a selection.
- This event is also generated when the user deselects an item.
- It is handled by implementing **ListSelectionListener** This listener specifies only one method, called valueChanged( ), which is shown here:

  void valueChanged(ListSelectionEvent le).

Some of the methods provided by Jlist are given below

- void setSelectionMode(int mode) where mode can be SINGLE_SELECTION, SINGLE_INTERVAL_SELECTION MULTIPLE_INTERVAL_SELECTION(default)



- int getSelectedIndex() :returns the index of item selected in Jlist
- int [] getSelectedIndices(): returns the array of index of items selected in Jlist
- Object getSelectedValue() : returns the value associated with the selection.
- Object [] getSelectedValues() : returns the value associated with the selection.

Example:

```
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;
public class Demo4 extends JFrame
{
        JList lst;
        JLabel lbl;
        public Demo4()
        {
                setSize(200,500);
                setLayout(new FlowLayout());
                String [] items={"Nepal","India","Pakistan","Bhutan","America","China","Australia"};
                lst=new JList(items);
                lbl=new JLabel("None selected");
                lst.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
                add(lst);
                add(lbl);
                lst.addListSelectionListener(new ListSelectionListener()
                {
                        public void valueChanged(ListSelectionEvent le)
                        {
                                int idx = lst.getSelectedIndex();
                                lbl.setText("Current selection: " + items[idx]);

                                /* OR
                                String sel=(String)lst.getSelectedValue();
                                lbl.setText("Current Selecteion: "+sel);
                                */
                        }
                }
                );
                setVisible(true);
        }
```

```
        public static void main(String args[])
        {
                new Demo4();
        }
}
```

Output:

JTabbedPane

- JTabbedPane manages a set of components by linking them with tabs. Selecting a tab causes the component associated with that tab to come to the forefront. The constructor for creating tabbed pane is

  JTabbedPane()

- Tabs are added by calling the following method

  void addTab(String name, Component comp)

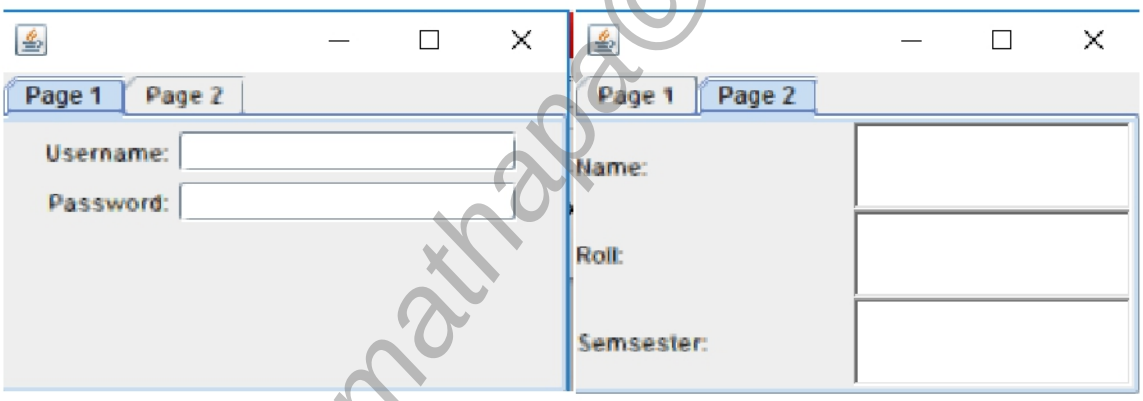
  Here, name is the name for the tab, and comp is the component that should be added to the tab. Often, the component added to a tab is a JPanel that contains a group of related components. This technique allows a tab to hold a set of components.

Example:

```
import java.awt.*;
import javax.swing.*;
class Tabbed1 extends   JFrame
{
        JTabbedPane tp;
        JPanel          panel1;
        JPanel          panel2;

        public Tabbed1()
        {
                setSize( 300, 200 );

                // Create the tab pages1
                panel1 = new JPanel();
                panel1.setLayout( new FlowLayout() );
                JLabel label1 = new JLabel( "Username:" );
                panel1.add( label1 );
                JTextField field = new JTextField(15);
                panel1.add( field );
                JLabel label2 = new JLabel( "Password:" );
                panel1.add( label2 );
                JPasswordField fieldPass = new JPasswordField(15);
                panel1.add( fieldPass );
```

```
                //create tabpage2
                panel2 = new JPanel();
                panel2.setLayout( new GridLayout( 3, 2 ) );
                panel2.add( new JLabel( "Name:" ) );
                panel2.add( new TextField() );
                panel2.add( new JLabel( "Roll:" ) );
                panel2.add( new TextField() );
                panel2.add( new JLabel( "Semsester:" ) );
                panel2.add( new TextField());

                // Create a tabbed pane
                tp = new JTabbedPane();
                tp.addTab( "Page 1", panel1 );
                tp.addTab( "Page 2", panel2 );
                add(tp);
                setVisible( true );
        }

        public static void main( String args[] )
        {

                new Tabbed1();

        }
}
```

Output:



Assignment

- Write a complete GUI program to calculate simple interest.
- Write a complete GUI program to check if the number entered by user is palindrome or not.
- Create a GUI with a button name btn_add and three textfields named txt_fn, txt_sn and txt_res. When the user press the button the sum of two numbers in txt_fn and txt_sn should be displayed in txt_res. **The program must validate for blank field, display warning message immediately and delete it, if the user press other then numeric charcter, enter or backspace in the textfields and prompt whether on not to end the program when the user press close button in window. Use adapter class for event handling and FlowLayout as layout manager.**

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class Demo2 extends JFrame
{
        JLabel lbl_fn, lbl_sn, lbl_res;
        JTextField txt_fn, txt_sn,txt_res;
        JButton btn_add;
```

```
public Demo2()
{
	JButton b=new JButton("a");
	setSize(750,100);
	setLayout(new FlowLayout());
	setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
	addWindowListener(new WindowAdapter()
	{
		public void windowClosing(WindowEvent e)
		{
			int a=JOptionPane.showConfirmDialog(null,"Are you sure to exit?");
			if(a==0)
			{
				System.exit(0);
			}
		}
	}
	);
	lbl_fn=new JLabel("First Num:");
	lbl_sn=new JLabel("Second Num:");
	lbl_res=new JLabel("Result:");
	txt_fn=new JTextField(10);
	txt_fn.addKeyListener(new KeyAdapter()
	{
		public void keyReleased(KeyEvent e)
		{
			char a=e.getKeyChar();
			int b=e.getKeyCode();
			if(Character.isDigit(a)|| a=='.' || b==KeyEvent.VK_ENTER  ||
								b==KeyEvent.VK_BACK_SPACE)
			{

			}
			else
			{
				JOptionPane.showMessageDialog(null,"Enter only number");
				String txt=txt_fn.getText();
				txt_fn.setText(txt.substring(0,txt.length()-1));
			}

		}
		}
	);

	txt_sn=new JTextField(10);
	txt_sn.addKeyListener(new KeyAdapter()
	{
		public void keyReleased(KeyEvent e)
		{
			char a=e.getKeyChar();
			int b=e.getKeyCode();
			if(Character.isDigit(a)|| a=='.' || b==KeyEvent.VK_ENTER  ||
								b==KeyEvent.VK_BACK_SPACE)
			{
			}
			else
			{
				JOptionPane.showMessageDialog(null,"Enter only number");
				String txt=txt_sn.getText();
				txt_sn.setText(txt.substring(0,txt.length()-1));
```

```
                        }

                    }
                }
                );

                txt_res=new JTextField(10);
                btn_add=new JButton("Add");
                add(lbl_fn);
                add(txt_fn);
                add(lbl_sn);
                add(txt_sn);
                add(lbl_res);
                add(txt_res);
                add(btn_add);
                btn_add.addActionListener(new ActionListener()
                {
                public void actionPerformed(ActionEvent e)
                        {
                                if(txt_fn.getText().equals("") || txt_sn.getText().equals(""))
                                {
                                        JOptionPane.showMessageDialog(null,"Blank field not accepted");
                                }
                                else
                                {
                                        double a=Double.parseDouble(txt_fn.getText());
                                        double b=Double.parseDouble(txt_sn.getText());
                                        double c=a+b;
                                        txt_res.setText(Double.toString(c));
                                }
                        }
                }
                );
                setVisible(true);
        }

        public static void main(String []args)
        {
                new Demo2();


        }
}
```

Internal Fram

JMenu(with submenu)

Check Box and Radio Buttons in Menu Items

Keyboard Mnemonics and Accelerators

Borders

- Explain the following compoinents with suitable programming example
  1. JMenu
  2. Sliders
  3. Borders
  4. Split Pane, Desktop Pane and **Internal Frame**
  5. Progress Bars