Unit 11 : Java Applications

11.1 About AWT & Swing
11.2 About JFrame (a Top Level Window in Swing)
11.3 Swing Components (JLabel, About Text Component like JTextField, JButton,
Event Handling in Swing Applications,
Layout Management using Flow Layout, Border Layout, Grid Layout,
Using JPanel, Choice Components like JCheckBox, JRadioButton, Borders Components,
JComboBox & its events, JList & its events with MVC Patterns, Key and Mouse Event Handling,
Menus in Swing,
JTextArea, Dialog Boxes in Swing,
JTable for Displaying Data in Tabular form,
MDI using JDeskto Pane & JInternalFrame)
11.4 Using like NetBeans, JBuilder for building java applications using Drag & Drop
11.5 Adapter Classes

What is AWT?

AWT is the standard abbreviation for Abstract Window Toolkit. It is a platform-dependent API for developing GUI (Graphical User Interface) or window-based applications using Java. It was devised by Sun Microsystems in 1995.

It is a heavy-weight API as it is usually generated by the system's host operating system. It comes with a significantly high number of classes and methods, primarily created for designing and managing GUI.

Components of AWT

A component is an object with a graphical representation that can be displayed on the screen and is allowed to interact with the user. The Component class is the abstract parent of the non-menu-related AWT components. The prime AWT Components are:

1. **Button (java.awt.Button)**

   To create a Button object, merely create an instance of the Button class by calling any of the constructors. The most commonly used constructor of the Button class takes a String argument that gives the Button object a text title.

2. **Checkboxes (java. awt.Checkbox)**

   Checkboxes have two states, namely on and off. The state of the button is returned as the Object argument when a Checkbox event occurs. To find out the state of a checkbox object we can use getState() that returns a true or false value. We can also get the label of the checkbox using getLabel() that returns a String object.

3. **Radio Buttons (java. awt.CheckboxGroup)**

   It is a group of checkboxes, where only one of the items in the group can be selected at a time.

4. **Choice Buttons (java.awt.Choice)**

   Similar to a radio button, where we are allowed to make a selection, however, it requires less space and allows us to add items to the menu dynamically using the addItem() method.

5. **Labels (java.awt.Label)**

   Used for adding a text description to a point on the applet or application.

6. **TextFields (java.awt.TextField)**

   These are areas where the user can enter text. They are used for displaying and receiving text messages. We can make this text field read-only or editable. We can use the setEditable(false) to set a text field read-only.

What is Swing?

Swing is another Java toolkit which is a lightweight graphical user interface (GUI), used for creating various Java applications. The Swing components are platform-independent. It allows designers to create buttons and scroll bars.

Swing comes with packages for devising desktop applications using Java. Swing components are programmed using the robust Java programming language. It is an important subset of the Java Foundation Classes (JFC).

Components of Swing

The prime 12 AWT Components are:

1. **ImageIcon**: The ImageIcon component is used for creating an icon sized-image from an image residing at the source URL.

2. **JButton**: JButton class is used for creating a push-button on the UI. The button can contain some display text or images. It generates an event when clicked or double-clicked.

3. **JLabel**: JLabel class is used for rendering a read-only text label or images on the UI. It does not generate any events.

4. **JTextField**: JTextField renders an editable single-line text box. A user is allowed to input non-formatted text in the box.

5. **JPasswordField**: JPasswordField is a subclass of JTextField class. It renders a text box that masks the user input text with bullet points. This is used for inserting passwords into the application.

6. **JCheckBox**: JCheckBox renders a check-box with a label. The check-box has two states – on/off. When selected, the state is on and a small tick is displayed in the box.

7. **JRadioButton**: JRadioButton is used to render a group of radio buttons in the UI. A designer can select one choice from the group.

8. **JList**: JList component Renders a scrollable list of elements. A designer can select a value or multiple values from the list. This select behaviour is defined in the code by the developer.

9. **JComboBox**: JComboBox class is used to render a dropdown of the list of options.

10. **JFileChooser**: JFileChooser class renders a file selection utility. This component allows a designer to select a file from the local system.

11. **JTabbedPane**: JTabbedPane is another essential component that allows the designer to switch between tabs in an application. This is a highly useful UI component as it allows the designer to browse more content without navigating to different pages.

12. **JSlider**: The JSlider component displays a slider which the designer can drag to change its value. The constructor takes three arguments – minimum value, maximum value, an initial value.

Difference Between AWT and Swing

Basis	AWT	Swing
Meaning	Java AWT is an Application programming interface for developing GUI applications using Java.	Swing comes from the set of Java Foundation Classes and is used for creating various applications.
Weight	The Java AWT components are usually heavy weighted.	The Java Swing components are usually light weighted.
Functionality	Java AWT has lesser functionalities in comparison to Swing.	Java Swing has wider functionality in comparison to AWT.
Execution Time	The execution time of AWT is quite higher than Swing.	The execution time of Swing is quite lower than that of AWT.
Platform Support	The components of Java AWT are typically platform-dependent.	The components of Java Swing are typically platform-independent.
MVC pattern	AWT doesn't support the MVC pattern.	Swing supports the MVC pattern.
Power	AWT components are comparatively less powerful.	Swing components are comparatively more powerful.

11.2 About JFrame (a Top Level Window in Swing)

Frame is a top-level container that provides a window on the screen.

- It is an extended version of **Java.awt.Frame** that adds support for the JFC/Swing architecture.
- JFrame is the core class of **javax.swing** package and is used to develop GUI (Graphical User Interface) in which various visual objects like Text Field, Radio Button, Scroll Bar, Check Box etc are embedded. This GUI is called Window pane.
- JFrame is a window with a title Bar, Border, a resizable Border, menu Bar, Buttons that close or iconify the window and a content pane that acts as a container.
- JFrame contains a **JRootpane**, which is where all of the components are added, and this JRootpane is shared among all container objects. When components are added to a JFrame, they are added to the content pane(JRootpane).
- The JFrame is typically the outermost container used in GUI applications.

How to create a JFrame?

JFrame(): This helps in creating a frame which is invisible.
JFrame(String Title): Helps in creating a frame with a title.
JFrame(GraphicsConfiguration gc): Creates a frame with blank title and the graphics configuration of screen.

Example:

```
JFrame F = new JFrame();
// Or overload the constructor and give it a title:
JFrame F1 = new JFrame("Red Alert!");
```

Example:

```
import javax.swing.*;
import java.awt.*;
public class myWindowJFrame {
    public static void main(String[] args){
        JFrame f= new JFrame();
        f.setTitle("myWindow");
        f.setSize(640,480);
        //Container c= f.getContentPane();
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JLabel label=new JLabel("label");
        f.add(label);
        f.setVisible(true);
    }
}
```

11.3 Swing Components (JLabel, About Text Component like JTextField, JButton, Event Handling in Swing Applications, Layout Management using Flow Layout, Border Layout, Grid Layout, Using JPanel, Choice Components like JCheckBox, JRadioButton, Borders Components, JComboBox & its events, JList & its events with MVC Patterns, Key and Mouse Event Handling, Menus in Swing, JTextArea, Dialog Boxes in Swing, JTable for Displaying Data in Tabular form, MDI using JDeskto Pane & JInternalFrame)

---

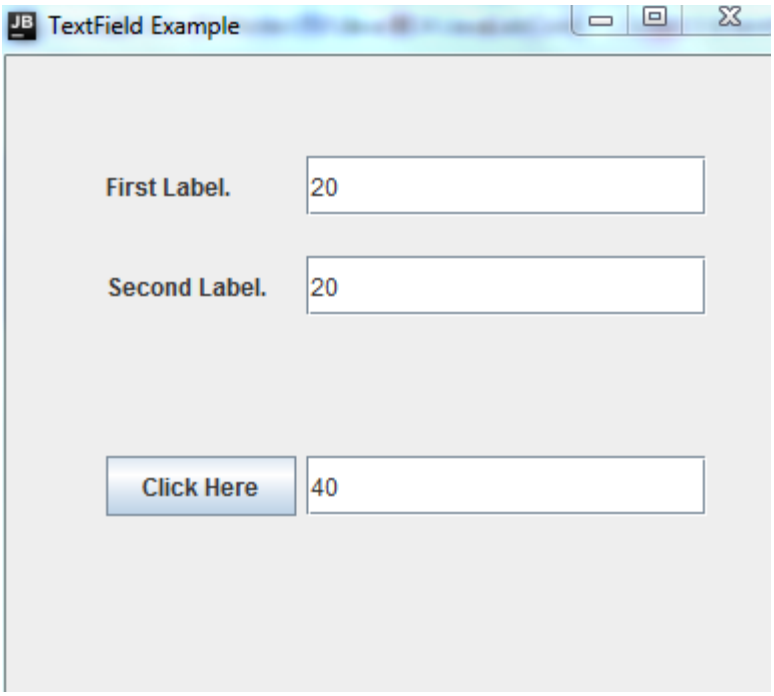
a. **Jlabel**
b. **JTextField**
C. **JButton**

```
import javax.swing.*;
class TextFieldExample
{
    public static void main(String args[])
    {
        JFrame f= new JFrame("TextField Example");
        JTextField t1,t2;
        t1=new JTextField("Text 1 Placehold.");
        t1.setBounds(150,50, 200,30);
        t2=new JTextField("Text 2 Placehold");
        t2.setBounds(150,100, 200,30);
        f.add(t1); f.add(t2);
//JButton
        JButton b=new JButton("Click Here");
        b.setBounds(50,200,95,30);
        f.add(b);
//JLabel
        JLabel l1,l2;
        l1=new JLabel("First Label.");
        l1.setBounds(50,50, 100,30);
        l2=new JLabel("Second Label.");
        l2.setBounds(50,100, 100,30);
        f.add(l1); f.add(l2);

        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

d. Event Handling in Swing Applications

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

public class EventHandelJButton {
    static JTextField t1,t2,t3;
     public static void  main  ( String[] args ) {
        JFrame f= new JFrame("TextField Example");

        t1=new JTextField("Text 1 Placehold.");
        t1.setBounds(150,50, 200,30);
        t2=new JTextField("Text 2 Placehold");
        t2.setBounds(150,100, 200,30);
        t3=new JTextField("Text 3 Placehold");
        t3.setBounds(150,200, 200,30);
        f.add(t1); f.add(t2);
        f.add(t3);

        //JLabel
        JLabel l1,l2;
        l1=new JLabel("First Label.");
        l1.setBounds(50,50, 100,30);
        l2=new JLabel("Second Label.");
        l2.setBounds(50,100, 100,30);
        f.add(l1); f.add(l2);
        //
        //JButton
        JButton b=new JButton("Click Here");
        b.setBounds(50,200,95,30);
        b.addActionListener(new ActionListener(){
             public void actionPerformed(ActionEvent e){
                 int num1 = Integer.parseInt(t1.getText());
                 int num2 = Integer.parseInt(t2.getText());
                 int sum = num1 + num2;
                    String txt = Integer.toString(sum);
                 t3.setText(txt);
             }
         });
         f.add(b);
        //
        f.setSize(400,400);
        f.setLayout(null);
```

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



Example 2:

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

public class EventHandelButton extends Frame implements ActionListener {
    JTextField tf;
    JLabel l;
    JButton b;

    EventHandelButton () {
        tf = new JTextField();
        tf.setBounds(50, 50, 150, 20);
        l = new JLabel();
        l.setBounds(50, 100, 250, 20);
        b = new JButton("Find IP");
        b.setBounds(50, 150, 95, 30);
        b.addActionListener(this);
        add(b);
        add(tf);
        add(l);
        setSize(400, 400);
        setLayout(null);
        setVisible(true);
    }

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

```
        public void actionPerformed ( ActionEvent e ) {
            try {
                String host = tf.getText();
                String ip = java.net.InetAddress.getByName(host).getHostAddress();
                l.setText("IP of " + host + " is: " + ip);
            } catch (Exception ex) {
                System.out.println(ex);
            }
        }
    }
```

e. Layout Management using Flow Layout, Border Layout, Grid Layout,

The LayoutManagers are used to arrange components in a particular manner. The **Java LayoutManagers** facilitates us to control the positioning and size of the components in GUI forms. LayoutManager is an interface that is implemented by all the classes of layout managers. There are the following classes that represent the layout managers:

1. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout

Java FlowLayout

The Java FlowLayout class is used to arrange the components in a line, one after another (in a flow). It is the default layout of the applet or panel.

Fields of FlowLayout class

1. **public static final int LEFT**

2. **public static final int RIGHT**
3. **public static final int CENTER**
4. **public static final int LEADING**
5. **public static final int TRAILING**

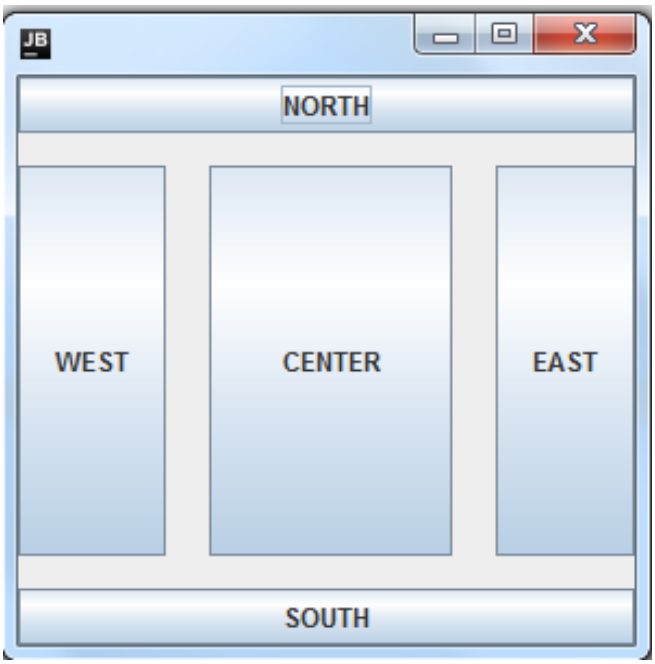
```
// import statement
import java.awt.*;
import javax.swing.*;

public class FlowLayoutExample1
{
    JFrame frameObj;

    // constructor
    FlowLayoutExample1()
    {
        // creating a frame object
        frameObj = new JFrame();
        // creating the buttons
        JButton b1 = new JButton("1");
        JButton b2 = new JButton("2");
        JButton b3 = new JButton("3");
        JButton b4 = new JButton("4");

        // adding the buttons to frame
        frameObj.add(b1); frameObj.add(b2); frameObj.add(b3); frameObj.add(b4);

        // parameterized constructor is used
        // where alignment is left
        // horizontal gap is 20 units and vertical gap is 25 units.
        frameObj.setLayout(new FlowLayout(FlowLayout.LEFT, 20, 25));
        frameObj.setSize(300, 300);
        frameObj.setVisible(true);
    }
    // main method
    public static void main(String argvs[])
    {
        new FlowLayoutExample1();
    }
}
```

Java BorderLayout

The BorderLayout is used to arrange the components in five regions: north, south, east, west, and center. Each region (area) may contain one component only. It is the default layout of a frame or window. The BorderLayout provides five constants for each region:

1. **public static final int NORTH**
2. **public static final int SOUTH**
3. **public static final int EAST**
4. **public static final int WEST**
5. **public static final int CENTER**

```
// import statement
import java.awt.*;
import javax.swing.*;
public class BorderLayoutExample
{
    JFrame jframe;
    // constructor
    BorderLayoutExample()
    {
        // creating a Frame
        jframe = new JFrame();
        // create buttons
        JButton btn1 = new JButton("NORTH");
        JButton btn2 = new JButton("SOUTH");
        JButton btn3 = new JButton("EAST");
        JButton btn4 = new JButton("WEST");
        JButton btn5 = new JButton("CENTER");
        // creating an object of the BorderLayout class using
        // the parameterized constructor where the horizontal gap is 20
        // and vertical gap is 15. The gap will be evident when buttons are placed
        // in the frame
        jframe.setLayout(new BorderLayout(20, 15));
        jframe.add(btn1, BorderLayout.NORTH);
        jframe.add(btn2, BorderLayout.SOUTH);
        jframe.add(btn3, BorderLayout.EAST);
        jframe.add(btn4, BorderLayout.WEST);
        jframe.add(btn5, BorderLayout.CENTER);
        jframe.setSize(300,300);
        jframe.setVisible(true);
    }
    // main method
    public static void main(String argvs[])
    {
        new BorderLayoutExample();
    }
}
```

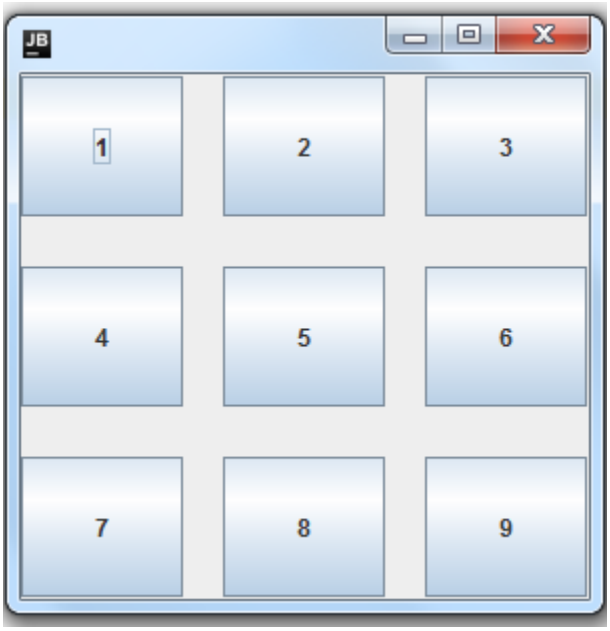


Java GridLayout

The Java GridLayout class is used to arrange the components in a rectangular grid. One component is displayed in each rectangle.

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

public class GridLayoutExample1
{

    JFrame frameObj;

    // constructor
    GridLayoutExample1()
    {
        frameObj = new JFrame();

// creating 9 buttons
        JButton btn1 = new JButton("1");
        JButton btn2 = new JButton("2");
        JButton btn3 = new JButton("3");
        JButton btn4 = new JButton("4");
        JButton btn5 = new JButton("5");
        JButton btn6 = new JButton("6");
        JButton btn7 = new JButton("7");
        JButton btn8 = new JButton("8");
        JButton btn9 = new JButton("9");
```

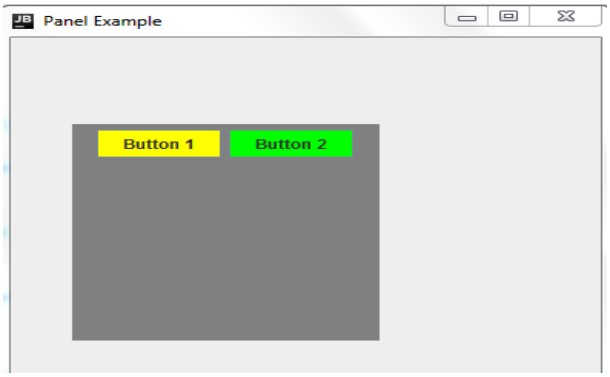
```
// adding buttons to the frame
// since, we are using the parameterless constructor, therefore;
// the number of columns is equal to the number of buttons we
// are adding to the frame. The row count remains one.
        frameObj.add(btn1); frameObj.add(btn2); frameObj.add(btn3);
        frameObj.add(btn4); frameObj.add(btn5); frameObj.add(btn6);
        frameObj.add(btn7); frameObj.add(btn8); frameObj.add(btn9);
// setting the grid layout
// a 3 * 3 grid is created with the horizontal gap 20
// and vertical gap 25
        frameObj.setLayout(new GridLayout(3, 3, 20, 25));
        frameObj.setSize(300, 300);
        frameObj.setVisible(true);
    }
    // main method
    public static void main(String argvs[])
    {
        new GridLayoutExample1();
    }
}
```



e. Using JPanel,

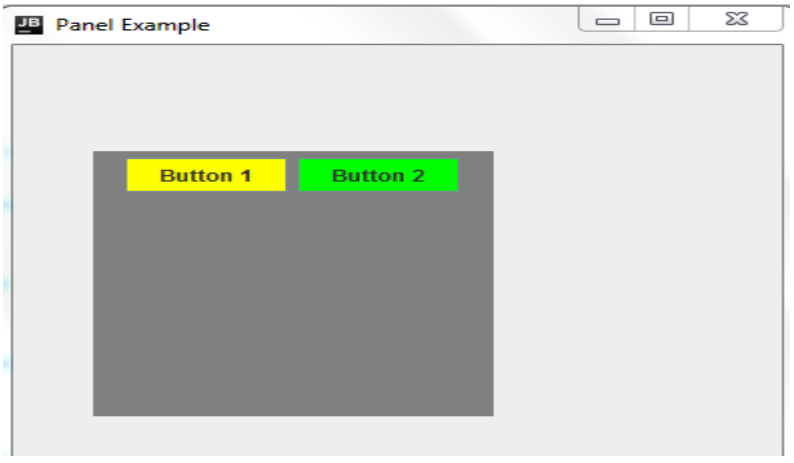
JPanel

```
import java.awt.*;
import javax.swing.*;
public class PanelExample {
    PanelExample()
    {
        JFrame f= new JFrame("Panel Example");
        JPanel panel=new JPanel();
        panel.setBounds(40,80,200,200);
        panel.setBackground(Color.gray);
        JButton b1=new JButton("Button 1");
        b1.setBounds(50,100,80,30);
        b1.setBackground(Color.yellow);
        JButton b2=new JButton("Button 2");
        b2.setBounds(100,100,80,30);
        b2.setBackground(Color.green);
        panel.add(b1); panel.add(b2);
        f.add(panel);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new PanelExample();
    }
}
```



f. Choice Components like JCheckBox, JRadioButton, Borders Components,

```
// JcheckBox

import javax.swing.*;
import java.awt.event.*;
public class CheckBoxExample
{
    CheckBoxExample(){
        JFrame f= new JFrame("CheckBox Example");
        final JLabel label = new JLabel();
        label.setHorizontalAlignment(JLabel.CENTER);
        label.setSize(400,100);
        f.add(label);
        JCheckBox checkbox1 = new JCheckBox("C++");
        checkbox1.setBounds(150,100, 50,50);
        f.add(checkbox1);
        checkbox1.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {
                label.setText("C++ Checkbox: "
                        + (e.getStateChange()==1?"checked":"unchecked"));
            }
        });

        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new CheckBoxExample();
    }
}
```



//JRadioButton

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

class RadioButtonExample extends JFrame implements ActionListener {
    JRadioButton rb1, rb2;
    JButton b;

    RadioButtonExample () {
        rb1 = new JRadioButton("Male");
        rb1.setBounds(100, 50, 100, 30);
        rb2 = new JRadioButton("Female");
        rb2.setBounds(100, 100, 100, 30);
        ButtonGroup bg = new ButtonGroup();
        bg.add(rb1);
        bg.add(rb2);
        b = new JButton("click");
        b.setBounds(100, 150, 80, 30);
        b.addActionListener(this);
        add(rb1);
        add(rb2);
        add(b);
        setSize(300, 300);
        setLayout(null);
        setVisible(true);
    }

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

    public void actionPerformed ( ActionEvent e ) {
        if (rb1.isSelected()) {
            JOptionPane.showMessageDialog(this, "You are Male.");
        }
        if (rb2.isSelected()) {
            JOptionPane.showMessageDialog(this, "You are Female.");
        }
    }
}
```

//Borders Components

// JComboBox & its events

```
import javax.swing.*;
import java.awt.event.*;
public class ComboBoxExample {
    JFrame f;
    ComboBoxExample(){
        f=new JFrame("ComboBox Example");

        final JLabel label = new JLabel();
        label.setHorizontalAlignment(JLabel.CENTER);
        label.setSize(400,100);
        f.add(label);
        //

        //
        String country[]={"India","Aus","U.S.A","England","Newzealand"};
        JComboBox cb=new JComboBox(country);
        cb.setBounds(50, 50,90,20);
        cb.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String data = "Country Selected: "
                        + cb.getItemAt(cb.getSelectedIndex());
                label.setText(data);
            }
        });

        f.add(cb);
        f.setLayout(null);
        f.setSize(400,500);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new ComboBoxExample();
    }
}
```

//JList & its events

```
import javax.swing.*;
import java.awt.event.*;
public class ListExample {
    ListExample () {
        JFrame f = new JFrame();
        final JLabel label = new JLabel();
        label.setSize(500,100);
        JButton b=new JButton("Show");
        b.setBounds(200,150,80,30);

        DefaultListModel<String> l1 = new DefaultListModel<>();
        l1.addElement("Item1");
        l1.addElement("Item2");
        l1.addElement("Item3");
        l1.addElement("Item4");
        JList<String> list = new JList<>(l1);
        list.setBounds(100, 100, 75, 75);

        b.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String data = "";
                if (list.getSelectedIndex() != -1) {
                    data = "Item Selected: " + list.getSelectedValue();
                    label.setText(data);
                }
                label.setText(data);
            }
        });

        f.add(list);
        f.add(label);
        f.add(b);

        f.setSize(400, 400);
        f.setLayout(null);
        f.setVisible(true);
    }

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

Java JMenuBar, JMenu and JMenuItem

// Menus in Swing

```
import javax.swing.*;
import java.awt.event.*;
public class MenuExample2 implements ActionListener{
    JFrame f;
    JMenuBar mb;
    JMenu file,edit,help;
    JMenuItem cut,copy,paste,selectAll;
    JTextArea ta;
    MenuExample2(){
        f=new JFrame();
        cut=new JMenuItem("cut");
        copy=new JMenuItem("copy");
        paste=new JMenuItem("paste");
        selectAll=new JMenuItem("selectAll");
        cut.addActionListener(this);
        copy.addActionListener(this);
        paste.addActionListener(this);
        selectAll.addActionListener(this);
        mb=new JMenuBar();
        file=new JMenu("File");
        edit=new JMenu("Edit");
        help=new JMenu("Help");
        edit.add(cut);edit.add(copy);edit.add(paste);edit.add(selectAll);
        mb.add(file);mb.add(edit);mb.add(help);
        ta=new JTextArea();
        ta.setBounds(5,5,360,320);
        f.add(mb);f.add(ta);
        f.setJMenuBar(mb);
        f.setLayout(null);
        f.setSize(400,400);
        f.setVisible(true);
    }
    public void actionPerformed(ActionEvent e) {
        if(e.getSource()==cut)
            ta.cut();
        if(e.getSource()==paste)
            ta.paste();
        if(e.getSource()==copy)
            ta.copy();
        if(e.getSource()==selectAll)
            ta.selectAll();
    }
    public static void main(String[] args) {
        new MenuExample2();
    }
}
```

//JTextArea,

```
import javax.swing.*;
import java.awt.event.*;
public class TextAreaExample implements ActionListener{
    JLabel l1,l2;
    JTextArea area;
    JButton b;
    TextAreaExample() {
        JFrame f= new JFrame();
        l1=new JLabel();
        l1.setBounds(50,25,100,30);
        l2=new JLabel();
        l2.setBounds(160,25,100,30);
        area=new JTextArea();
        area.setBounds(20,75,250,200);
        b=new JButton("Count Words");
        b.setBounds(100,300,120,30);
        b.addActionListener(this);
        f.add(l1);f.add(l2);f.add(area);f.add(b);
        f.setSize(450,450);
        f.setLayout(null);
        f.setVisible(true);
    }
    public void actionPerformed(ActionEvent e){
        String text=area.getText();
        String words[]=text.split("\\s");
        l1.setText("Words: "+words.length);
        l2.setText("Characters: "+text.length());
    }
    public static void main(String[] args) {
        new TextAreaExample();
    }
}
```

//Dialog Boxes in Swing,

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class DialogExample {
    private static JDialog d;
    DialogExample() {
        JFrame f= new JFrame();
        d = new JDialog(f , "Dialog Example", true);
        d.setLayout( new FlowLayout() );
        JButton b = new JButton ("OK");
        b.addActionListener ( new ActionListener()
        {
            public void actionPerformed( ActionEvent e )
            {
                DialogExample.d.setVisible(false);
            }
        });
        d.add( new JLabel ("Click button to continue."));
        d.add(b);
```

```
            d.setSize(300,300);
            d.setVisible(true);
        }
        public static void main(String args[])
        {
            new DialogExample();
        }
    }
```

//JTable for Displaying Data in Tabular form,

```
    import javax.swing.*;
    public class TableExample {
        JFrame f;
        TableExample(){
            f=new JFrame();
            String data[][]={ {"101","Amit","670000"},
                    {"102","Jai","780000"},
                    {"101","Sachin","700000"}};
            String column[]={"ID","NAME","SALARY"};
            JTable jt=new JTable(data,column);
            jt.setBounds(30,40,200,300);
            JScrollPane sp=new JScrollPane(jt);
            f.add(sp);
            f.setSize(300,400);
            f.setVisible(true);
        }
        public static void main(String[] args) {
            new TableExample();
        }
    }
```

MDI using JDesktop Pane & JInternalFrame

Java JDesktopPane

The JDesktopPane class, can be used to create "multi-document" applications. A multi-document application can **have many windows** included in it. We do it by making the contentPane in the main window as an instance of the JDesktopPane class or a subclass. Internal windows add instances of JInternalFrame to the JdesktopPane instance. The internal windows are the instances of JInternalFrame or its subclasses.

```
import java.awt.BorderLayout;
import java.awt.Container;
import javax.swing.JDesktopPane;
import javax.swing.JFrame;
import javax.swing.JInternalFrame;
import javax.swing.JLabel;
public class JDPaneDemo extends JFrame
{
    public JDPaneDemo()
    {
        CustomDesktopPane desktopPane = new CustomDesktopPane();
        Container contentPane = getContentPane();
        contentPane.add(desktopPane, BorderLayout.CENTER);
        desktopPane.display(desktopPane);

        setTitle("JDesktopPane Example");
        setSize(300,350);
        setVisible(true);
    }
    public static void  main(String args[])
    {
        new JDPaneDemo();
    }
}
class CustomDesktopPane extends JDesktopPane
{
    int numFrames = 3,  x = 30, y = 30;
    public void display(CustomDesktopPane dp)
    {
        for(int  i = 0; i < numFrames ; ++i )
        {
            JInternalFrame jframe = new JInternalFrame("Internal Frame " + i ,  true, true,
true, true);

            jframe.setBounds(x, y, 250, 85);
            Container c1 = jframe.getContentPane( ) ;
            c1.add(new JLabel("I love my country"));
            dp.add( jframe );
            jframe.setVisible(true);
            y += 85;
        }
    }
}
```

Java Adapter Classes

Java adapter classes *provide the default implementation of listener interfaces*. If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces. So it *saves code*.

Pros of using Adapter classes:

- It assists the unrelated classes to work combinedly.
- It provides ways to use classes in different ways.
- It increases the transparency of classes.
- It provides a way to include related patterns in the class.
- It provides a pluggable kit for developing an application.
- It increases the reusability of the class.

The Adapter classes with their corresponding listener interfaces are given below.

java.awt.event Adapter classes

Adapter class	Listener interface
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
HierarchyBoundsAdapter	HierarchyBoundsListener

```
// importing the necessary libraries

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

// class which inherits the MouseMotionAdapter class
public class MouseMotionAdapterExample extends MouseMotionAdapter {
    // object of Frame class
    Frame f;

    // class constructor
    MouseMotionAdapterExample () {
// creating the frame with the title
        f = new Frame("Mouse Motion Adapter");
// adding MouseMotionListener to the Frame
        f.addMouseMotionListener(this);
        // setting the size, layout and visibility of the frame
        f.setSize(300, 300);
        f.setLayout(null);
        f.setVisible(true);
    }

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

    // overriding the mouseDragged() method
    public void mouseDragged ( MouseEvent e ) {
// creating the Graphics object and fetching them from the Frame object using getGraphics()
method
        Graphics g = f.getGraphics();
// setting the color of graphics object
        g.setColor(Color.ORANGE);
// setting the shape of graphics object
        g.fillOval(e.getX(), e.getY(), 20, 20);
    }
}
```

Key and Mouse Event Handling,

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The java.awt.event package provides many event classes and Listener interfaces for event handling.

Java Event classes and Listener interfaces

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

Steps to perform Event Handling

Following steps are required to perform event handling:

1. Register the component with the Listener

Registration Methods

For registering the component with the Listener, many classes provide the registration methods. For example:

- **Button**
  - public void addActionListener(ActionListener a){}
- **MenuItem**

    - public void addActionListener(ActionListener a){}
- **TextField**
    - public void addActionListener(ActionListener a){}
    - public void addTextListener(TextListener a){}
- **TextArea**
    - public void addTextListener(TextListener a){}
- **Checkbox**
    - public void addItemListener(ItemListener a){}
- **Choice**
    - public void addItemListener(ItemListener a){}
- **List**
    - public void addActionListener(ActionListener a){}
    - public void addItemListener(ItemListener a){}

I did it

I can do it

I'm going to try

How can I do?

I want to do

I can't

I don't

Believe in yourself!

THE 7 C'S OF SUCCESS

- Confidence.
- Conception.
- Concentration.
- Consistency.
- Character.
- Commitment.
- Capacity to enjoy.