

UNIT-6

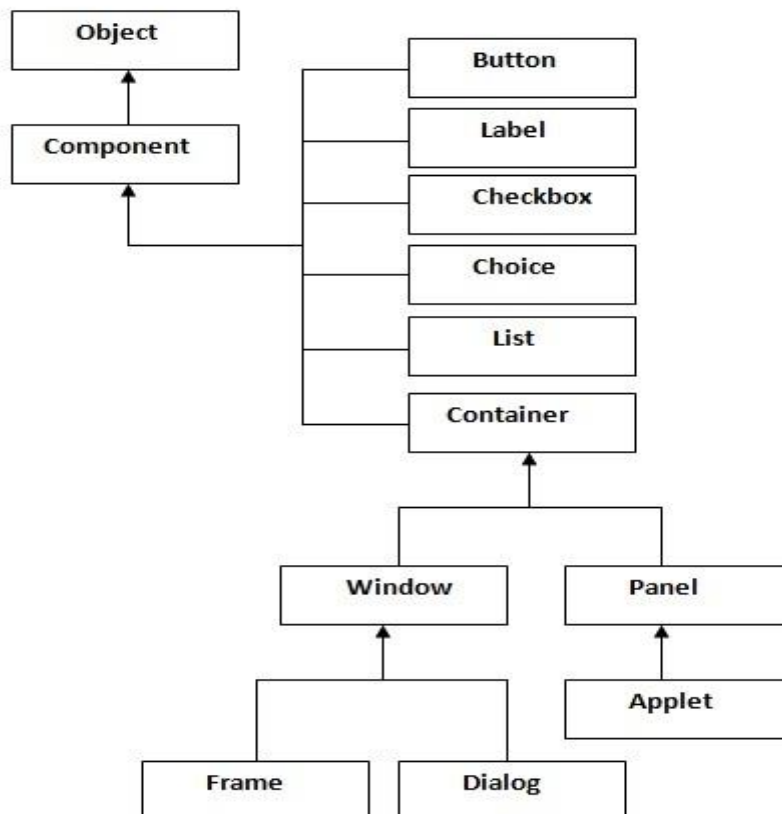
GUI Programming with Java & Event Handling

GUI Programming with Java - The AWT class hierarchy, Components and Containers, Introduction to Swing, Swing vs AWT, Hierarchy for Swing Components, Containers - JFrame, JApplet, JDialog, JPanel, Swing components- JButton, JLabel, JTextField, JTextArea, JRadio buttons, JList, JTable, Layout manager, executable jar file in Java.

Event Handling - Events, Event sources, Event classes, Event Listeners, Delegation event model, handling mouse and keyboard events, Adapter classes, Inner classes.

AWT:

- **AWT** (Abstract Window Toolkit) is an API to develop GUI or window-based applications in java.
- AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.
- The java.awt package provides classes for AWT api such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.
- **AWT class Hierarchy**
The hierarchy of AWT classes are given below.



Container:

- The Container is a component in AWT that can contain other components like buttons, textfields, labels etc. The classes that extend the Container class are known as container such as Frame, Dialog and Panel.
- **Window**
The window is the container that has no borders and menu bars. You must use frame, dialog or another window for creating a window.
- **Panel**
The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.
- **Frame**
The Frame is the container that contains title bar and can have menu bars. It can have other components like button, textfield etc.

Methods of Component class

Method	Description
public void add(Component c)	inserts a component on this component.
public void setSize(int width,int height)	sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	defines the layout manager for the component.
public void setVisible(boolean status)	changes the visibility of the component, by default false.

Creating a awt Frame with Button:

There are two ways to create a frame in AWT.

- By extending Frame class (inheritance)
- By creating the object of Frame class (association)

By extending Frame class

```
import java.awt.*;
class FirstFrame extends Frame
{
    FirstFrame()
    {
        Button b=new Button("click me");
        b.setBounds(30,100,80,30); // setting button position
        add(b); //adding button into frame
        setSize(300,300); //frame size 300 width and 300 height
        setLayout(null); //no layout manager
        setVisible(true); //frame will be visible, by default not visible
    }
    public static void main(String args[])
    {
        FirstFrame f=new FirstFrame();
    }
}
```

Output:



By creating the object of Frame class:

```
import java.awt.*;
class FirstFrame
{
    public static void main(String args[])
    {
        Frame f=new Frame();
        Button b=new Button("click me");
        b.setBounds(30,100,80,30);// setting button position
        f.add(b);//adding button into frame
        f.setSize(300,300);//frame size 300 width and 300 height
        f.setLayout(null);//no layout manager

        f.setVisible(true);//frame will be visible,by default not visible
    }
}
```

Output:



Note: awt Frames are not closed when clicked on close button. If we want to close awt frame we have to use Event handling concept.

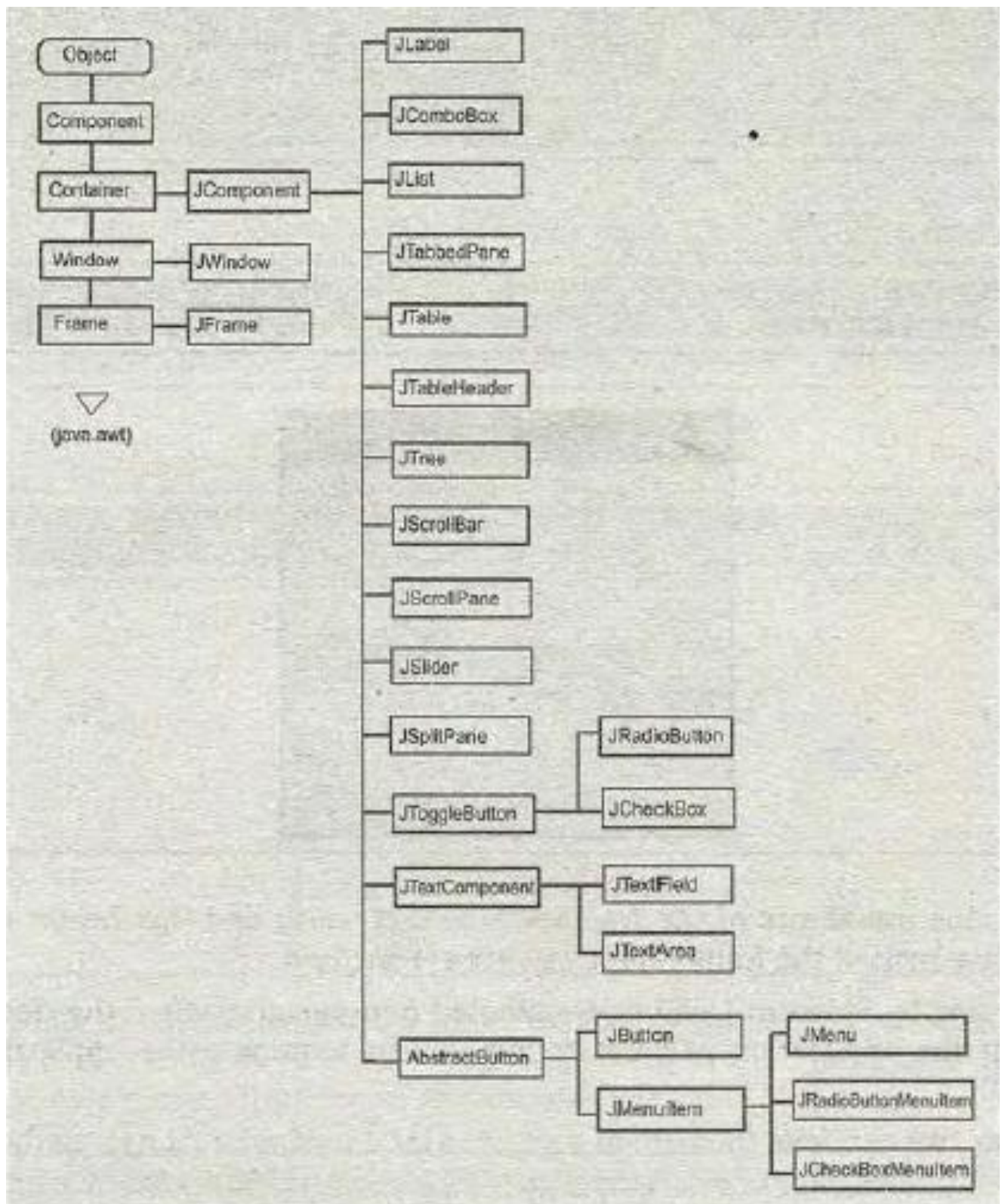
Swings:

- **Swing** is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.
- Unlike AWT, Java Swing provides platform-independent and lightweight components.
- The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Difference between AWT and Swing

Java AWT	Java Swing
AWT components are platform-dependent .	Java swing components are platform-independent .
AWT components are heavyweight .	Swing components are lightweight .
AWT doesn't support pluggable look and feel .	Swing supports pluggable look and feel .
AWT provides less components than Swing.	Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
AWT doesn't follows MVC	Swing follows MVC . (Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.

Hierarchy of Java Swing classes



Containers:

JFrame:

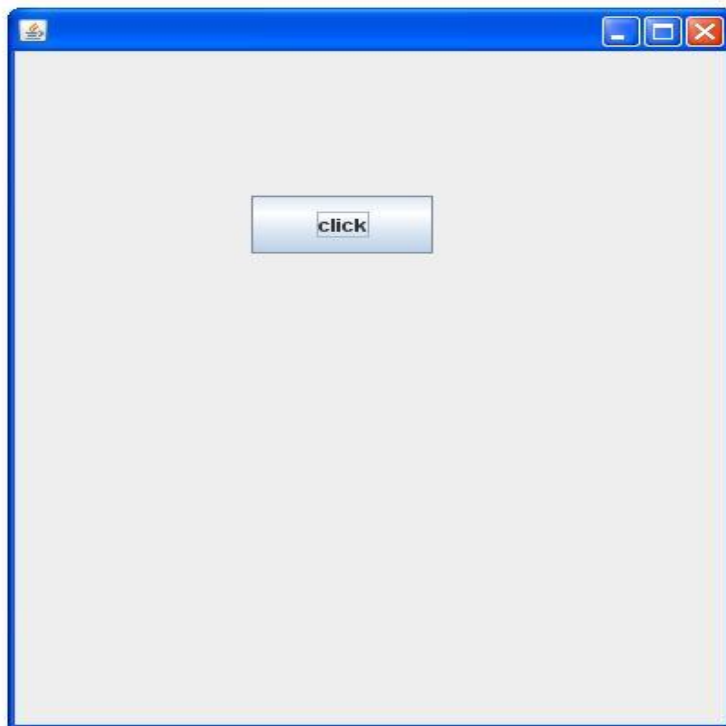
-
- The class **JFrame** is an extended version of **java.awt.Frame** that adds support for the JFC/Swing component architecture.

- **Example:**

```
import javax.swing.*;
public class FirstSwingExample
{
    public static void main(String[] args)
    {
        JFrame f=new JFrame();//creating instance of JFrame
        JButton b=new JButton("click");//creating instance of JButton
        b.setBounds(130,100,100, 40);//x axis, y axis, width, height

        f.add(b);//adding button in JFrame
        f.setSize(400,500);//400 width and 500 height
        f.setLayout(null);//using no layout managers
        f.setVisible(true);//making the frame visible
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //closing
        frame
    }
}
```

OutPut:



JApplet:

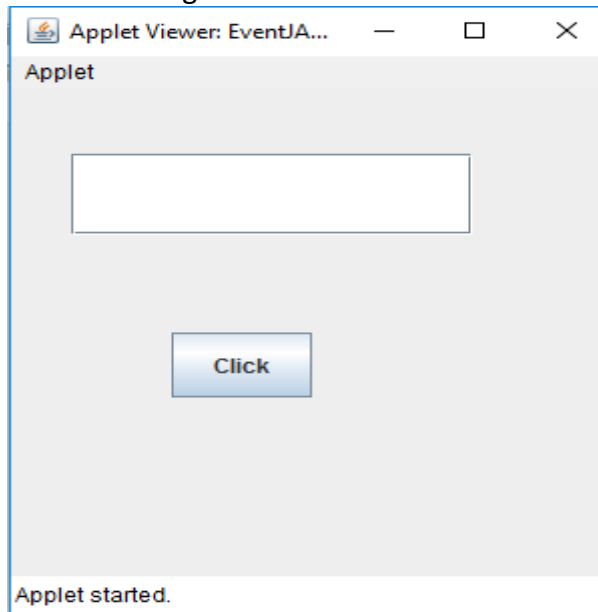
- As we prefer Swing to AWT. Now we can use JApplet that can have all the controls of swing. The JApplet class extends the Applet class.

- **Example:**

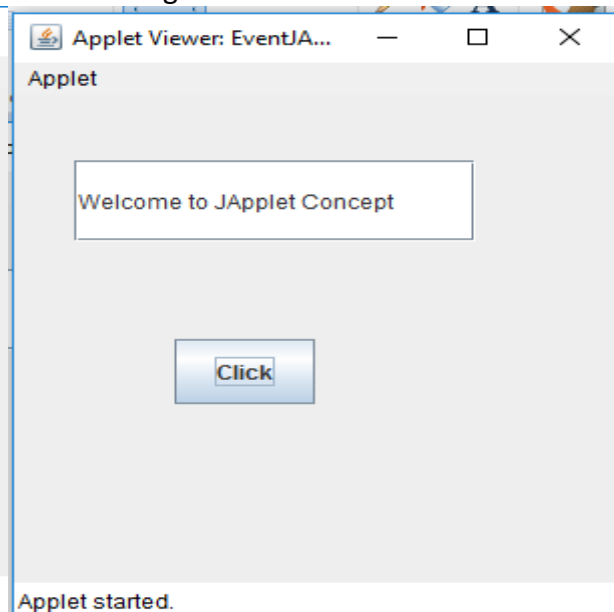
```
import java.applet.*;
import javax.swing.*;
import java.awt.event.*;
public class EventJApplet extends JApplet implements ActionListener
{
    JButton b;
    JTextField tf;
    public void init()
    {
        tf=new JTextField();
        tf.setBounds(30,40,150,20);
        b=new JButton("Click");
        b.setBounds(80,150,70,40);
        add(b);add(tf);
        b.addActionListener(this);
        setLayout(null);
    }
    public void actionPerformed(ActionEvent e)
    {
        tf.setText("Welcome");
    }
}
/*
<applet code="EventJApplet.class" width="300" height="300">
</applet>
*/
```

Output:

Before clicking the button



After clicking the button



JDialog:

- The JDialog control represents a top level window with a border and a title used to take some form of input from the user. It inherits the Dialog class.
- Unlike JFrame, it doesn't have maximize and minimize buttons.
- JDialog Exists inside "javax.swing" package.

Commonly used Constructors:

Constructor	Description
JDialog()	It is used to create a modeless dialog without a title and without a specified Frame owner.
JDialog(Frame owner)	It is used to create a modeless dialog with specified Frame as its owner and an empty title.
JDialog(Frame owner, String title, boolean modal)	It is used to create a dialog with the specified title, owner Frame and modality.

Example:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class DialogDemo
{
    DialogDemo()
    {
        JDialog d = new JDialog();
        JLabel l=new JLabel ("This is Dialog Box");
        d.add(l);
        d.setLayout(new FlowLayout());
        d.setSize(300,300);
        d.setVisible(true);
        d.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
    }
    public static void main(String args[])
    {
        new DialogDemo();
    }
}
```

Output:



JPanel:

- The JPanel is a simplest container class. It provides space in which an application can attach any other component.
- It inherits the JComponents class. It doesn't have title bar.

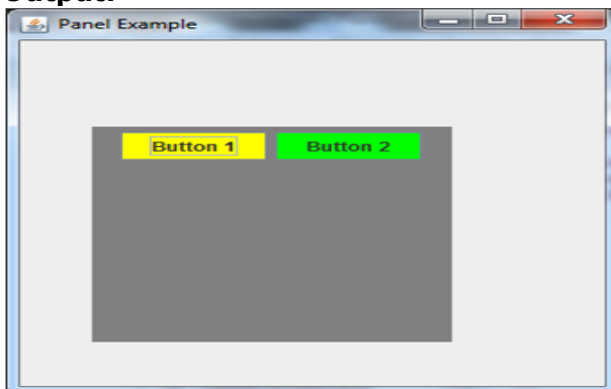
Commonly used Constructors:

Constructor	Description
JPanel()	create a new JPanel with a double buffer and a flow layout.
JPanel(boolean isDoubleBuffered)	create a new JPanel with FlowLayout with some strategy.
JPanel(LayoutManager layout)	create a new JPanel with the specified layout manager.

Example:

```
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();
    }
}
```

Output:



Swing components:

JButton:

The JButton class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed. It inherits AbstractButton class.

Commonly used Constructors:

Constructor	Description
JButton()	It creates a button with no text and icon.
JButton(String s)	It creates a button with the specified text.
JButton(Icon i)	It creates a button with the specified icon object.

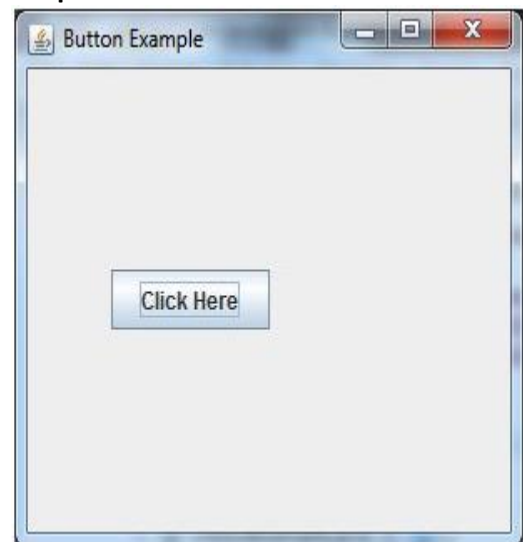
Commonly used methods:

Methods	Description
void setText(String s)	It is used to set specified text on button
String getText()	It is used to return the text of the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void addActionListener(ActionListener a)	It is used to add the action listener to this object.

Example

```
import javax.swing.*;
public class ButtonExample
{
    public static void main(String[] args)
    {
        JFrame f=new JFrame("Button Example");
        JButton b=new JButton("Click Here");
        b.setBounds(50,100,95,30);
        f.add(b);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

Output:



JLabel

The object of JLabel class is a component for placing text in a container. It is used to display a single line of read only text. The text can be changed by an application but a user cannot edit it directly. It inherits JComponent class.

Commonly used Constructors:

Constructor	Description
JLabel()	Creates a JLabel instance with no image and with an empty string for the title.
JLabel(String s)	Creates a JLabel instance with the specified text.
JLabel(Icon i)	Creates a JLabel instance with the specified image.

Commonly used Methods:

Methods	Description
String getText()	It returns the text string that a label displays.
void setText(String text)	It defines the single line of text this component will display.
void setHorizontalAlignment(int alignment)	It sets the alignment of the label's contents along the X axis.

Example:

```
import javax.swing.*;
class LabelExample
{
    public static void main(String args[])
    {
        JFrame f= new JFrame("Label Example");
        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(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

Output:



JTextField:

- The object of a JTextField class is a text component that allows the editing of a single line text. It inherits JTextComponent class.

Commonly used Constructors:

Constructor	Description
JTextField()	Creates a new TextField
JTextField(String text)	Creates a new TextField initialized with the specified text.
JTextField(String text, int columns)	Creates a new TextField initialized with the specified text and columns.
JTextField(int columns)	Creates a new empty TextField with the specified number of columns.

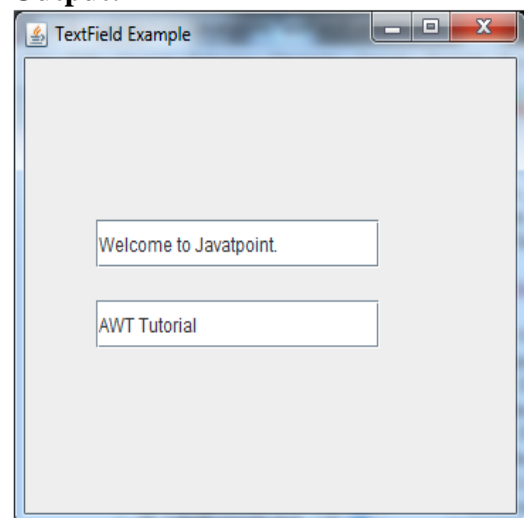
Commonly used Methods:

Methods	Description
void addActionListener(ActionListener l)	add the specified action listener to receive action events from this textfield.
Action getAction()	It returns the currently set Action for this ActionEvent source, or null if no Action is set.
void setFont(Font f)	set the current font.
void removeActionListener(ActionListener l)	remove the specified action listener so that it no longer receives action events from this

Example:

```
import javax.swing.*;
class TextFieldExample
{
    public static void main(String args[])
    {
        JFrame f= new JFrame("TextField Example");
        JTextField t1,t2;
        t1=new JTextField("Welcome to Javatpoint.");
        t1.setBounds(50,100, 200,30);
        t2=new JTextField("AWT Tutorial");
        t2.setBounds(50,150, 200,30);
        f.add(t1); f.add(t2); f.setSize(300,400);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

Output:



JTextArea:

- The object of a JTextArea class is a multi line region that displays text. It allows the editing of multiple line text. It inherits JTextComponent class

Commonly used Constructors:

Constructor	Description
JTextArea()	Creates a text area that displays no text initially.
JTextArea(String s)	Creates a text area that displays specified text initially.
JTextArea(int row, int column)	Creates a text area with the specified number of rows and columns that displays no text initially.
JTextArea(String s, int row, int column)	Creates a text area with the specified number of rows and columns that displays specified text.

Commonly used Methods:

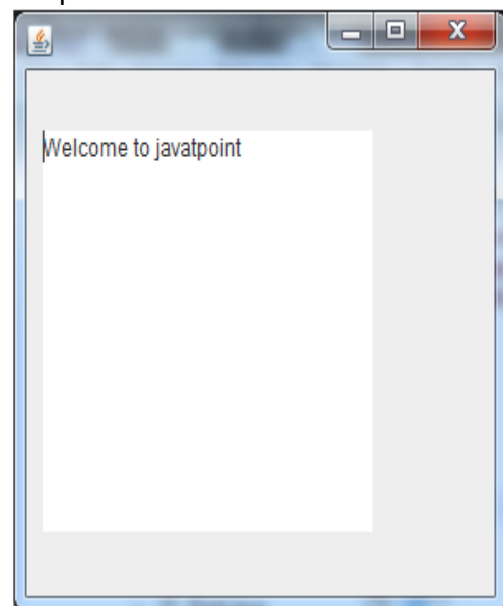
Methods	Description
void setRows(int rows)	It is used to set specified number of rows.
void setColumns(int cols)	It is used to set specified number of columns.
void setFont(Font f)	It is used to set the specified font.
void insert(String s, int position)	It is used to insert the specified text on the specified position.
void append(String s)	It is used to append the given text to the end of the document.

Example:

```
import javax.swing.*;
public class TextAreaExample
{
    JFrame f;
    JTextArea area;
    TextAreaExample()
    {
        f= new JFrame();
        area=new JTextArea("Welcome to javatpoint")

        area.setBounds(10,30, 200,200);
        f.add(area);
        f.setSize(300,300); f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new TextAreaExample();
    }
}
```

Output:



JRadioButton:

- The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz.
- It should be added in ButtonGroup to select one radio button only.

Commonly used Constructors:

Constructor	Description
JRadioButton()	Creates an unselected radio button with no text.
JRadioButton(String s)	Creates an unselected radio button with specified text.
JRadioButton(String s, boolean selected)	Creates a radio button with the specified text and selected status.

Commonly used Methods:

Methods	Description
void setText(String s)	It is used to set specified text on button.
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int a)	It is used to set the mnemonic on the button.
void addActionListener(ActionListener a)	It is used to add the action listener to this object.

Example:

```
import javax.swing.*;
public class RadioButtonExample
{
    JFrame f;
    RadioButtonExample()
    {
        f=new JFrame();
        JRadioButton r1=new JRadioButton("A) Male");
        JRadioButton r2=new JRadioButton("B) Female");
        r1.setBounds(75,50,100,30);
        r2.setBounds(75,100,100,30);
        ButtonGroup bg=new ButtonGroup();
        bg.add(r1);
        bg.add(r2);
        f.add(r1);
        f.add(r2);
        f.setSize(300,300);
    }
}
```

```

        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String[] args)
    {
        new RadioButtonExample();
    }
}

```

Output:



JList:

- The object of JList class represents a list of text items. The list of text items can be set up so that the user can choose either one item or multiple items. It inherits JComponent class.

Commonly used Constructors:

Constructor	Description
JList()	Creates a JList with an empty, read-only, model.
JList(ary[] listData)	Creates a JList that displays the elements in the specified array.

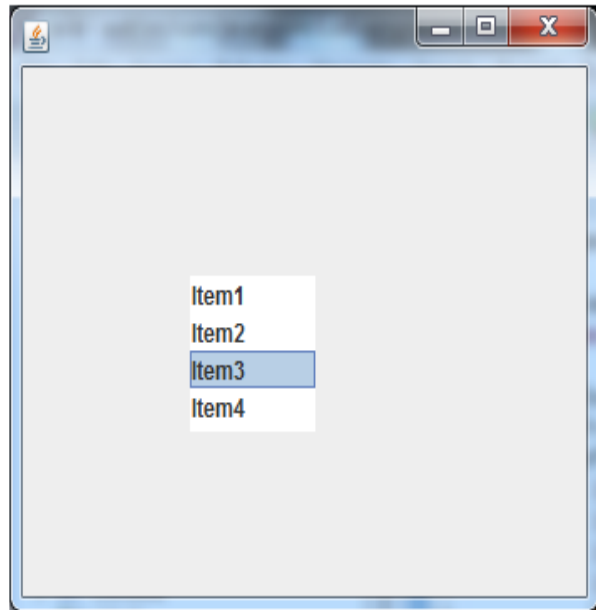
Commonly used Methods:

Methods	Description
Void addListSelectionListener (ListSelectionListener listener)	It is used to add a listener to the list, to be notified each time a change to the selection occurs.
int getSelectedIndex()	It is used to return the smallest selected cell index.
ListModel getModel()	It is used to return the data model that holds a list of items displayed by the JList component.
void setListData(Object[] listData)	It is used to create a read-only ListModel from an array of objects.

Example:

```
import javax.swing.*;
public class ListExample
{
    //DefaultListModel model;
    ListExample()
    {
        JFrame f= new JFrame();
        /*
            model=new DefaultListModel();
            model.addElement("Item1");
            model.addElement("Item2");
            model.addElement("Item3");
            model.addElement("Item4");
            JList lst=new JList(model);
        */
        String items[]={"Item1"," Item2",
                        " Item3","
Item4"};
        JList lst=new JList(items);
        lst.setBounds(100,50,100,150);

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

Output:**JTable:**

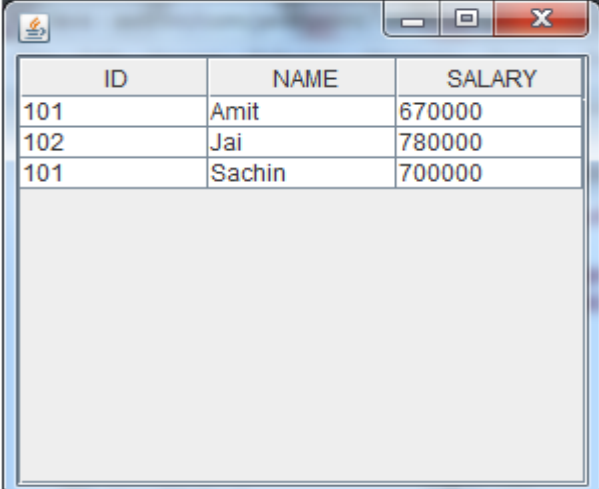
The JTable class is used to display data in tabular form. It is composed of rows and columns.

Commonly used Constructors:

Constructor	Description
JTable()	Creates a table with empty cells.
JTable(Object[][] rows, Object[] columns)	Creates a table with the specified data.

Example:

```
import javax.swing.*;
public class TableExample
{
    JFrame f;
    TableExample()
    {
        f=new JFrame();
        String data[][]={ {"101","Amit","670000"},
                           {"102","Jai","780000"},
                           {"103","Sachin","700000"}
                          };
        String head[]={"ID","NAME","SALARY"};
        JTable jt=new JTable(data,head);
        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();
    }
}
```

Output:

ID	NAME	SALARY
101	Amit	670000
102	Jai	780000
103	Sachin	700000

Layout managers:

- Basically we have 3-types of Layout managers in java.
 - FlowLayout
 - BorderLayout
 - GridLayout

FlowLayout:

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

Fields of FlowLayout class:

```
public static final int LEFT  
public static final int RIGHT  
public static final int CENTER
```

Constructors of FlowLayout class:

FlowLayout(): creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.

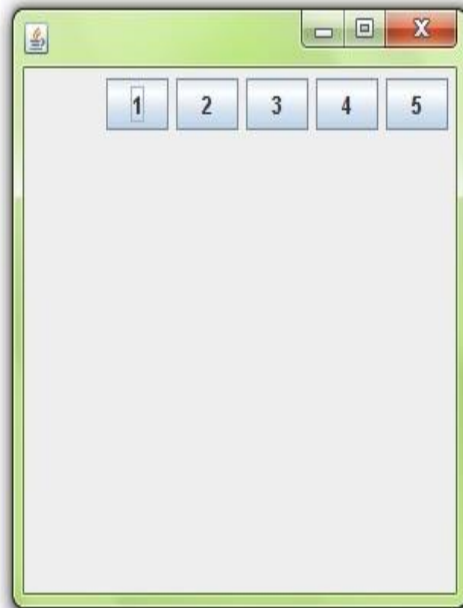
FlowLayout(int align): creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.

FlowLayout(int align, int hgap, int vgap): creates a flow layout with the given alignment and the given horizontal and vertical gap.

Example:

```
import java.awt.*;  
import javax.swing.*;  
public class MyFlowLayout  
{  
    JFrame f;  
    FlowLayout fl;  
    MyFlowLayout()  
    {  
        f=new JFrame();  
        JButton b1=new JButton("1");  
        JButton b2=new JButton("2");  
        JButton b3=new JButton("3");  
        JButton b4=new JButton("4");  
        JButton b5=new JButton("5");  
        f.add(b1);f.add(b2);f.add(b3);  
        f.add(b4);f.add(b5);  
        fl=new  
FlowLayout(FlowLayout.RIGHT);  
        f.setLayout(fl);  
        f.setSize(600,600);  
f.setVisible(true);  
    }  
    public static void main(String[] args)  
    {  
        new MyFlowLayout();  
    }  
}
```

Output:



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 frame or window. The BorderLayout provides five constants for each region:

```
public static final int NORTH
public static final int SOUTH
public static final int EAST
public static final int WEST
public static final int CENTER
```

Constructors of BorderLayout class:

BorderLayout(): creates a border layout but with no gaps between the components.

BorderLayout(int hgap, int vgap): creates a border layout with the given horizontal and vertical gaps between the components.

Example:

```
import java.awt.*;
import javax.swing.*;
public class Border
{
    JFrame f;
    Border()
    {
        f=new JFrame();
        JButton b1=new JButton("NORTH");
        JButton b2=new JButton("SOUTH");
        JButton b3=new JButton("EAST");
        JButton b4=new JButton("WEST");
        JButton b5=new JButton("CENTER");
        f.add(b1,BorderLayout.NORTH);
        f.add(b2,BorderLayout.SOUTH);
        f.add(b3,BorderLayout.EAST);
        f.add(b4,BorderLayout.WEST);
        f.add(b5,BorderLayout.CENTER);
        f.setSize(300,300);
        f.setVisible(true);
    }
    public static void main(String[] args)
    {
        new Border();
    }
}
```

Output:



GridLayout:

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

Constructors of GridLayout class:

GridLayout(): creates a grid layout with one column per component in a row.

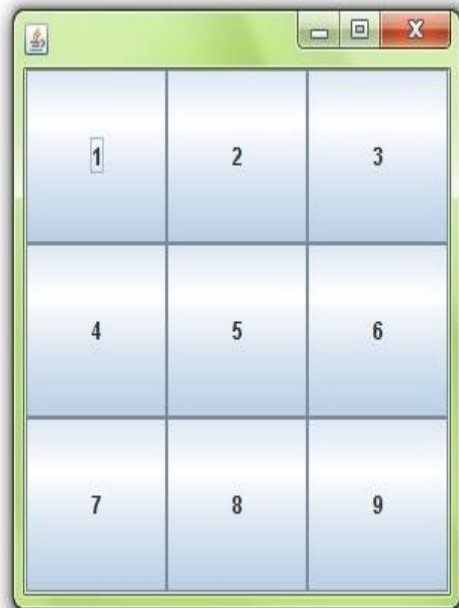
GridLayout(int rows, int columns): creates a grid layout with the given rows and columns but no gaps between the components.

GridLayout(int rows, int columns, int hgap, int vgap): creates a grid layout with the given rows and columns alongwith given horizontal and vertical gaps.

Example:

```
import java.awt.*;
import javax.swing.*;
import java.io.*;
public class MyGridLayout
{
    JFrame f;
    MyGridLayout()
    {
        f=new JFrame();
        JButton b1=new JButton("1");
        JButton b2=new JButton("2");
        JButton b3=new JButton("3");
        JButton b4=new JButton("4");
        JButton b5=new JButton("5");
        JButton b6=new JButton("6");
        JButton b7=new JButton("7");
        JButton b8=new JButton("8");
        JButton b9=new JButton("9");
        f.add(b1);f.add(b2);f.add(b3);
        f.add(b4);f.add(b5);
        f.add(b6);f.add(b7);
        f.add(b8);f.add(b9);
        //grid layout of 3 rows and 3 columns
        f.setLayout(new GridLayout(3,3));
        f.setSize(300,300);
        f.setVisible(true);
    }
    public static void main(String[] args)
    {
        new MyGridLayout();
    }
}
```

Output:



Executable jar file in Java:

- The **jar (Java Archive)** tool of JDK provides the facility to create the executable jar file.
- An executable jar file calls the main method of the class if you double click it.
- To create the executable jar file, you need to create **.mf file**, also known as manifest file.

Creating manifest file

- To create manifest file, you need to write Main-Class, then colon, then space, then class name then enter.
- For example: myfile.mf (type the below in notepad save with an extension **.mf**)

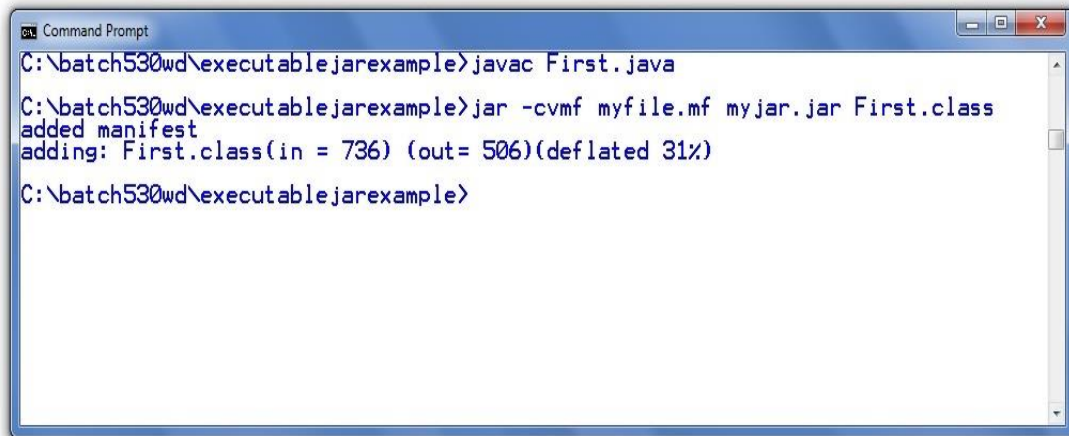
Main-Class: First

Creating executable jar file using jar tool

- The jar tool provides many switches, some of them are as follows:
 1. **-c** creates new archive file
 2. **-v** generates verbose output. It displays the included or extracted resource on the standard output.
 3. **-m** includes manifest information from the given mf file.
 4. **-f** specifies the archive file name
 5. **-x** extracts files from the archive file
- Write **jar** then **switches** then **mf_file** then **jar_file** then **.classfile** as given below:

jar -cvmf myfile.mf myjar.jar First.class

- It is shown in the image given below:



```
Command Prompt
C:\batch530wd\executablejarexample>javac First.java
C:\batch530wd\executablejarexample>jar -cvmf myfile.mf myjar.jar First.class
added manifest
adding: First.class(in = 736) (out= 506)(deflated 31%)
C:\batch530wd\executablejarexample>
```

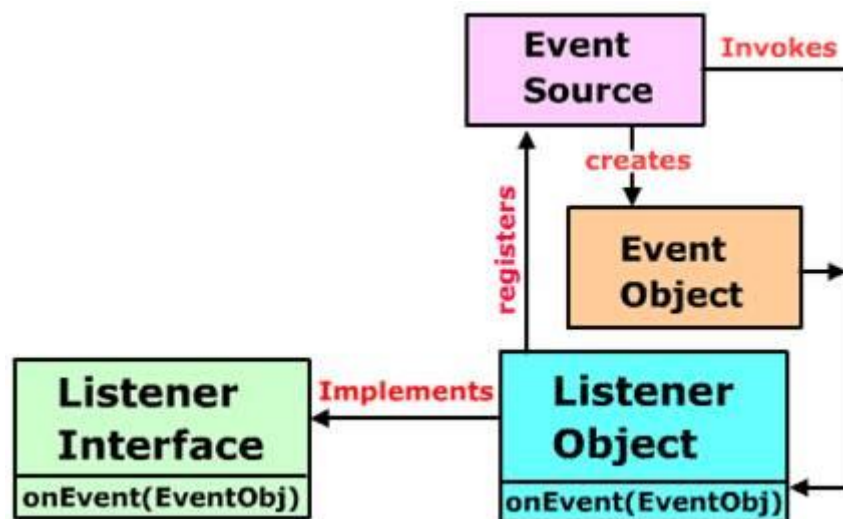
- Now it will create the executable jar file. If you double click on it, it will call the main method of the First class.
- We are assuming that a window based application using AWT or SWING. *First.java*

Event handling:

- Event is the change in the state of Object or Source
- Event handling is the mechanism that controls the event and decides what should happen if an event occurs.

Delegation event model:

- The event model is based on the Event Source and Event Listeners.
- **Event Listener** is an object that receives the messages / events.
- **Event Source** is any object which creates the message / event.
- The Event Delegation model is based on – The Event Classes, The Event Listeners, Event Objects.



- There are three participants in event delegation model in Java;
Event Source – the class which broadcasts the events
Event Listeners – the classes which receive notifications of events
Event Object – the class object which describes the event.

EVENTS	SOURCE	LISTENERS
Action Event	Button, List, MenuItem, Text field	ActionListener
Component Event	Component	Component Listener
Focus Event	Component	FocusListener
Item Event	Checkbox, CheckboxMenuItem, Choice, List	ItemListener
Key Event	when input is received from keyboard	KeyListener
Text Event	Text Component	TextListener
Window Event	Window	WindowListener
Mouse Event	Mouse related event	MouseListener

Listener interfaces:

1. ActionListener
2. ItemListener
3. KeyListener
4. MouseMotionListener
5. MouseListener.

ActionListener:

registration method:

```
public void addActionListener(ActionListener a);
```

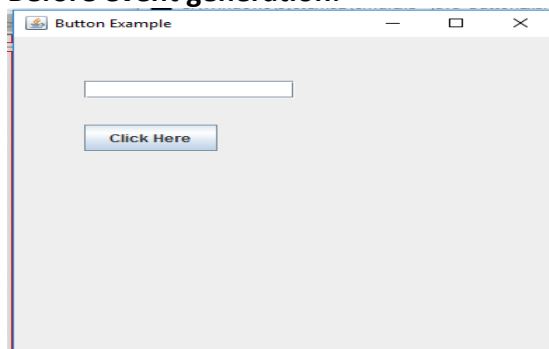
performing event method:

```
public void actionPerformed(ActionEvent e)
{
    // event code
}
```

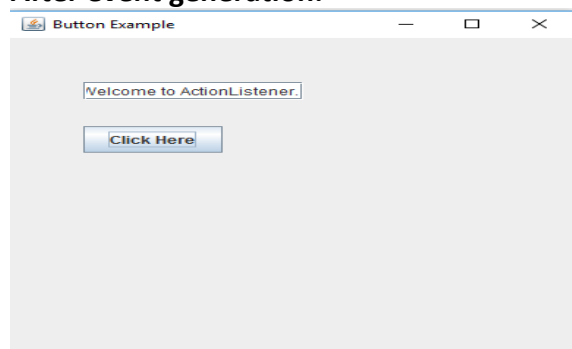
Example:

```
import java.awt.event.*;
import javax.swing.*;
public class ButtonExample1 implements ActionListener
{
    JTextField tf;
    ButtonExample1()
    {
        JFrame f=new JFrame("Button Example");
        tf=new JTextField();
        tf.setBounds(50,50, 150,20);
        JButton b=new JButton("Click Here");
        b.setBounds(50,100,95,30);
        f.add(b);f.add(tf);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
        b.addActionListener(this);
    }
    public void actionPerformed(ActionEvent e)
    {
        tf.setText("Welcome to ActionListener.");
    }
    public static void main(String args[])
    {
        new ButtonExample1();
    }
}
```

Before event generation:



After event generation:



ItemListener:

registration method:

```
public void addItemListener(ItemListener i);
```

performing event method:

```
public void itemStateChanged(ItemEvent e)
{
    // event code
}
```

KeyListener:

registration method:

```
public void addKeyListener(KeyListener k);
```

performing event methods:

```
public void keyPressed(KeyEvent e)
{
    // event code
}
public void keyReleased(KeyEvent e)
{
    // event code
}
public void keyTyped(KeyEvent e)
{
    // event code
}
```

Note: the above 3-methods should be overridden, other wise compile time error raised.
i.e suppose if an interface having 4-methods , then for the all 4-methods implementation must be provide in a class.

MouseMotionListener:

registration method:

```
public void addMouseMotionListener(MouseMotionListener mm);
```

performing event methods:

```
public void mouseMoved(MouseEvent me)
{
    // event code
}

public void mouseDragged(MouseEvent e)
{
    // event code
}
```

MouseListener:

registration method:

```
public void addMouseListener(MouseListener mm);
```

performing event methods:

```
public void mousePressed(MouseEvent me)
{
    // event code
}
public void mouseClicked(MouseEvent e)
{
    // event code
}
```



```

    public void mouseEntered(MouseEvent e)
    {
        // event code
    }
    public void mouseReleased(MouseEvent e)
    {
        // event code
    }
    public void mouseExited(MouseEvent e)
    {
        // event code
    }
}

```

- The main drawback of Listener interfaces are we have to provide implementation part for every method inside a interface.
- This is overcome by using **Adapter classes**.

Example program for MouseListener and MouseMotionListener to handle mouse events:

```

import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;
public class AppletMouseXY extends Applet implements MouseListener,
MouseMotionListener
{
    int x, y;
    String str="";
    public void init()
    {
        addMouseListener(this);
        addMouseMotionListener(this);
    }
    // override ML 5 abstract methods
    public void mousePressed(MouseEvent e)
    {
        x = e.getX();
        y = e.getY();
        str = "Mouse Pressed";
        repaint();
    }
    public void mouseReleased(MouseEvent e)
    {
        x = e.getX();
        y = e.getY();
        str = "Mouse Released";
        repaint();
    }
    public void mouseClicked(MouseEvent e)
    {
        x = e.getX();
        y = e.getY();
        str = "Mouse Clicked";
        repaint();
    }
}

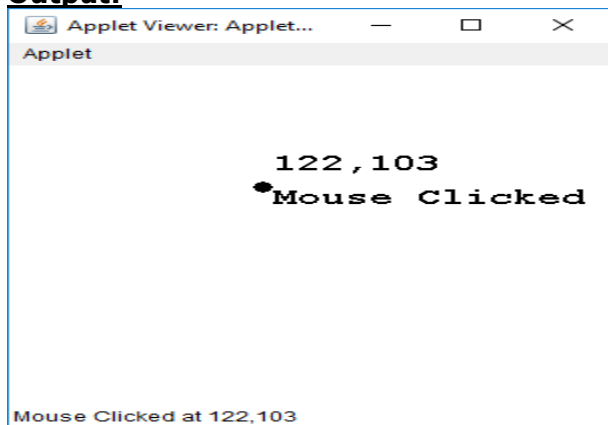
```

```

public void mouseEntered(MouseEvent e)
{
    x = e.getX();
    y = e.getY();
    str = "Mouse Entered";
    repaint();
}
public void mouseExited(MouseEvent e)
{
    x = e.getX();
    y = e.getY();
    str = "Mouse Exited";
    repaint();
}
// override two abstract methods of MouseMotionListener
public void mouseMoved(MouseEvent e)
{
    x = e.getX();
    y = e.getY();
    str = "Mouse Moved";
    repaint();
}
public void mouseDragged(MouseEvent e)
{
    x = e.getX();
    y = e.getY();
    str = "Mouse dragged";
    repaint();
}
// called by repaint() method
public void paint(Graphics g)
{
    g.setFont(new Font("Monospaced", Font.BOLD, 20));
    g.fillOval(x, y, 10, 10);
    g.drawString(x + "," + y, x+10, y -10);
    g.drawString(str, x+10, y+20);
    showStatus(str + " at " + x + "," + y);
}
}
/*
<applet code="AppletMouseXY.class" width="300" height="300">
</applet>
*/

```

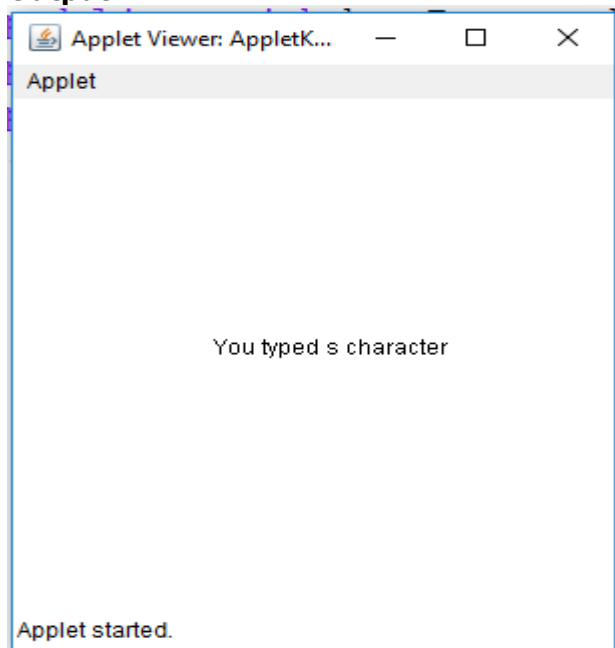
Output:



Example for KeyListener to handle Keyboard events:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class AppletKeyListener extends Applet implements KeyListener
{
    char ch;
    String str;
    public void init() // link the KeyListener with Applet
    {
        addKeyListener(this);
    }
    // override all the 3 abstract methods of KeyListener interface
    public void keyPressed(KeyEvent e){ }
    public void keyReleased(KeyEvent e){ }
    public void keyTyped(KeyEvent e)
    {
        ch = e.getKeyChar();
        str="You typed "+ch+" character";
        repaint();
    }
    public void paint(Graphics g)
    {
        g.drawString(str, 100, 150);
    }
}
/*
<applet code="AppletKeyListener.class" width="300" height="300">
</applet>
*/
```

Output:



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*.
- The adapter classes are found in **java.awt.event**, and **javax.swing.event** packages.
- The Adapter classes with their corresponding listener interfaces are given below.

java.awt.event Adapter classes

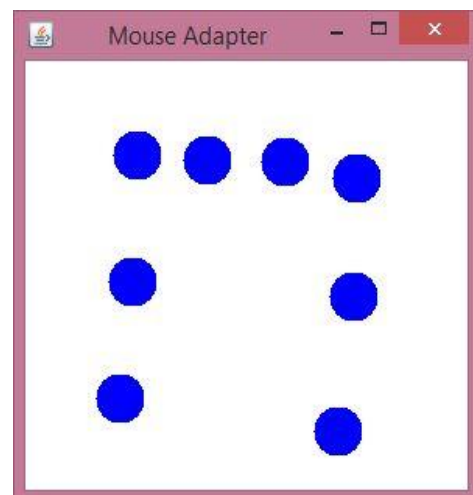
Adapter class	Listener interface
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener

Example program for MouseAdapter:

Example:

```
import java.awt.*;
import java.awt.event.*;
public class MouseAdapterExample extends MouseAdapter
{
    Frame f;
    MouseAdapterExample()
    {
        f=new Frame("Mouse Adapter");
        f.addMouseListener(this);
        f.setSize(300,300);
        f.setLayout(null); f.setVisible(true);
    }
    public void mouseClicked(MouseEvent e)
    {
        Graphics g=f.getGraphics();
        g.setColor(Color.BLUE);
        g.fillOval(e.getX(),e.getY(),30,30);
    }
    public static void main(String[] args)
    {
        new MouseAdapterExample();
    }
}
```

Output:



Inner classes:

- **Java inner class** or nested class is a class i.e. declared inside the class or interface.
- We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable.
- Additionally, it can access all the members of outer class including private data members and methods.

- **Syntax of Inner class**

```
class Outer_class
{
    //code
    class Inner_class
    {
        //code
    }
}
```

- **Advantage of inner classes**

There are basically three advantages of inner classes in java. They are as follows:

- 1) **It can access all the members (data members and methods) of outer class** including private.
- 2) Used **to develop more readable and maintainable code** because it logically group classes and interfaces in one place only.
- 3) **Code Optimization:** It requires less code to write.

- **Example:**

```
class TestMemberOuter1
{
    private int data=30;
    class Inner
    {
        void msg()
        {
            System.out.println("data is "+data);
        }
    }
    public static void main(String args[])
    {
        TestMemberOuter1 obj=new TestMemberOuter1();
        TestMemberOuter1.Inner in=obj.new Inner();
        in.msg();
    }
}
```

Output:

data is 30
