UNIT 3. GUI- Components- 2 (Swings):

- 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.

JButton class declaration

Let's see the declaration for javax.swing.JButton class.

1. **public class** JButton **extends** AbstractButton **implements** Accessible

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 of AbstractButton class:

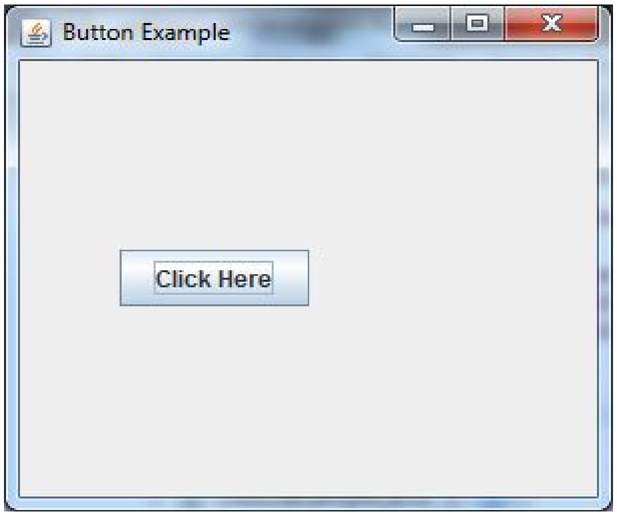
Methods	Description
void setText(String s)	It is used to set specified text on button
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	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.

Java JButton 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:

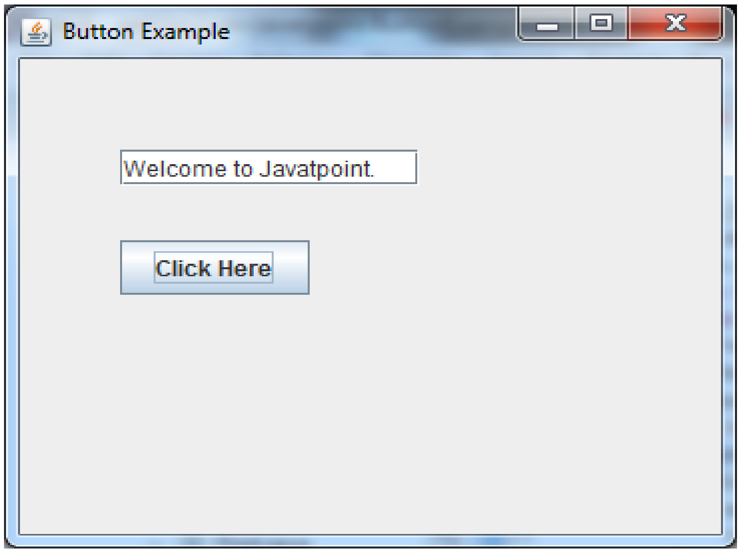


Java JButton Example with ActionListener

```
import java.awt.event.*;
import javax.swing.*;
public class ButtonExample {
```

```
public static void main(String[] args) {
    JFrame f=new JFrame("Button Example");
    final JTextField tf=new JTextField();
    tf.setBounds(50,50, 150,20);
    JButton b=new JButton("Click Here");
    b.setBounds(50,100,95,30);
    b.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent e){
            tf.setText("Welcome to Javatpoint.");
        }
    });
    f.add(b);f.add(tf);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```

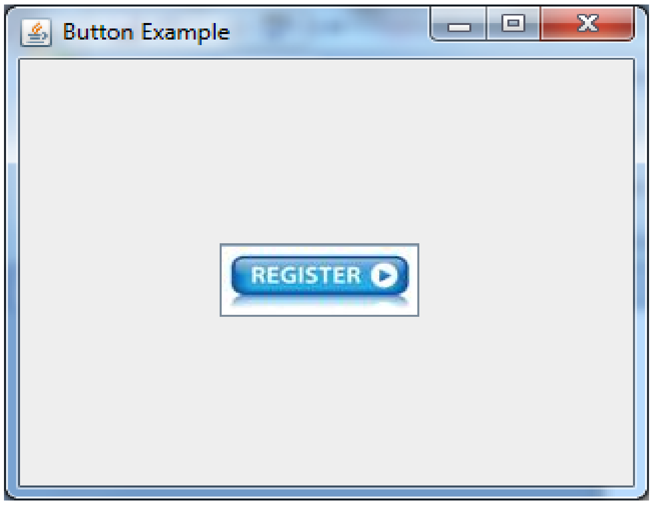
Output:



Example of displaying image on the button:

```
import javax.swing.*;
public class ButtonExample{
ButtonExample(){
JFrame f=new JFrame("Button Example");
JButton b=new JButton(new ImageIcon("D:\\icon.png"));
b.setBounds(100,100,100, 40);
f.add(b);
f.setSize(300,400);
f.setLayout(null);
f.setVisible(true);
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   }
public static void main(String[] args) {
   new ButtonExample();
}
}
```

Output:



- <u>JLabel:</u>

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.

JLabel class declaration

Let's see the declaration for javax.swing.JLabel class.

```
public class JLabel extends JComponent implements SwingConstants, Accessible
```

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.
JLabel(String s, Icon i, int horizontalAlignment)	Creates a JLabel instance with the specified text, image, and horizontal alignment.

Commonly used Methods:

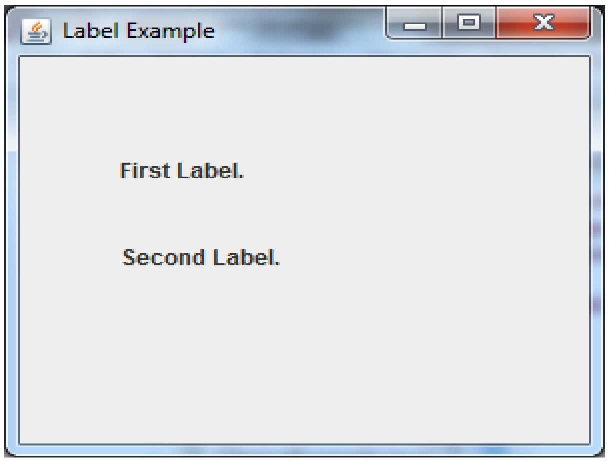
Methods	Description
String getText()	It returns the text string that a label displays.
void setText(String text)	It defines the single line of text this component will display.
void setHorizontalAlignment(int alignment)	It sets the alignment of the label's contents along the X axis.
Icon getIcon()	It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

Java JLabel 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:



```
First Label.

Second Label.
```

Java JLabel Example with ActionListener

Try Following program to know more about JLabel with ActionListener

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class LabelExample extends Frame implements ActionListener{
    JTextField tf; JLabel l; JButton b;
    LabelExample(){
        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 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);}
    }
    public static void main(String[] args) {
        new LabelExample();
    } }
```

- JFrame:

The javax.swing.JFrame class is a type of container which inherits the java.awt.Frame class. JFrame works like the main window where components like labels, buttons, textfields are added to create a GUI.

Unlike Frame, JFrame has the option to hide or close the window with the help of setDefaultCloseOperation(int) method.

Nested Class

Modifier and Type	Class	Description
protected class	JFrame.AccessibleJFrame	This class implements accessibility support for the JFrame class.

Fields

Modifier and Type	Field	Description
protected AccessibleContext	accessibleContext	The accessible context property.
static int	EXIT_ON_CLOSE	The exit application default window close operation.
protected JRootPane	rootPane	The JRootPane instance that manages the contentPane and optional menuBar for this frame, as well as the glassPane.
protected boolean	rootPaneCheckingEnabled	If true then calls to add and setLayout will be forwarded to the contentPane.

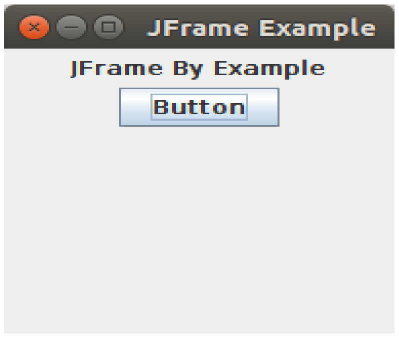
Constructors

Constructor	Description
JFrame()	It constructs a new frame that is initially invisible.
JFrame(GraphicsConfiguration gc)	It creates a Frame in the specified GraphicsConfiguration of a screen device and a blank title.
JFrame(String title)	It creates a new, initially invisible Frame with the specified title.
JFrame(String title, GraphicsConfiguration gc)	It creates a JFrame with the specified title and the specified GraphicsConfiguration of a screen device.

JFrame Example

```
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.Jpanel;
public class JFrameExample {
   public static void main(String s[]) {
      JFrame frame = new JFrame("JFrame Example");
      JPanel panel = new JPanel();
      panel.setLayout(new FlowLayout());
      JLabel label = new JLabel("JFrame By Example");
      JButton button = new JButton();
      button.setText("Button");
      panel.add(label);
      panel.add(button);
      frame.add(panel);
      frame.setSize(200, 300);
      frame.setLocationRelativeTo(null);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true);
   }
}
```

Output:



- <u>JCheckBox:</u>

The JCheckBox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a CheckBox changes its state from "on" to "off" or from "off" to "on ".It inherits <u>JToggleButton</u> class.

JCheckBox class declaration

Let's see the declaration for javax.swing.JCheckBox class.

UNIT 3. GUI- Components- 2 (Swings):

1. **public class** JCheckBox **extends** JToggleButton **implements** Accessible

Commonly used Constructors:

Constructor	Description
JJCheckBox()	Creates an initially unselected check box button with no text, no icon.
JChechBox(String s)	Creates an initially unselected check box with text.
JCheckBox(String text, boolean selected)	Creates a check box with text and specifies whether or not it is initially selected.
JCheckBox(Action a)	Creates a check box where properties are taken from the Action supplied.

Commonly used Methods:

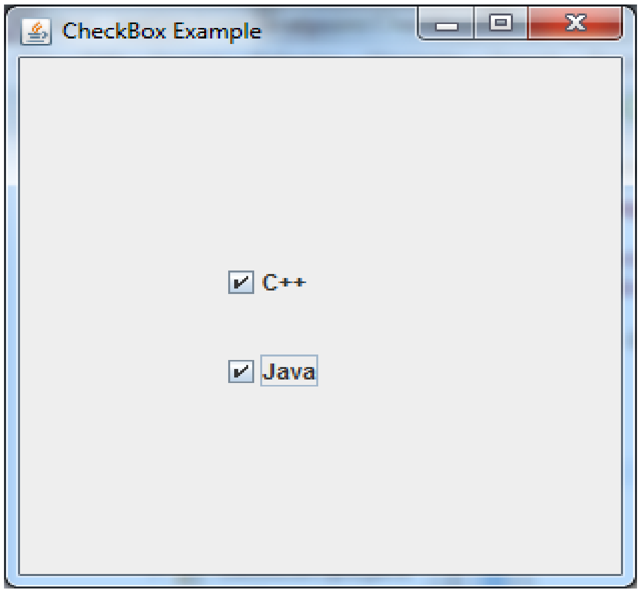
Methods	Description
AccessibleContext getAccessibleContext()	It is used to get the AccessibleContext associated with this JCheckBox.
protected String paramString()	It returns a string representation of this JCheckBox.

Java JCheckBox Example

```
import javax.swing.*;
public class CheckBoxExample
{
   CheckBoxExample(){
     JFrame f= new JFrame("CheckBox Example");
```

```
        JCheckBox checkBox1 = new JCheckBox("C++");
        checkBox1.setBounds(100,100, 50,50);
        JCheckBox checkBox2 = new JCheckBox("Java", true);
        checkBox2.setBounds(100,150, 50,50);
        f.add(checkBox1);
        f.add(checkBox2);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
public static void main(String args[])
    {
    new CheckBoxExample();
    }}
```

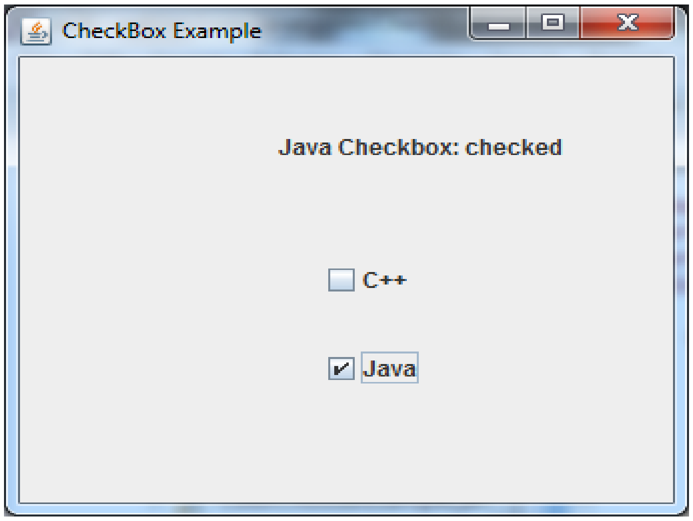
Output:



Java JCheckBox Example with ItemListener

```
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);
        JCheckBox checkbox1 = new JCheckBox("C++");
        checkbox1.setBounds(150,100, 50,50);
        JCheckBox checkbox2 = new JCheckBox("Java");
        checkbox2.setBounds(150,150, 50,50);
        f.add(checkbox1); f.add(checkbox2); f.add(label);
        checkbox1.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {
              label.setText("C++ Checkbox: "
              + (e.getStateChange()==1?"checked":"unchecked"));
            }
         });
        checkbox2.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {
              label.setText("Java Checkbox: "
              + (e.getStateChange()==1?"checked":"unchecked"));
            }
         });
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
public static void main(String args[])
{
    new CheckBoxExample();
}
}
```

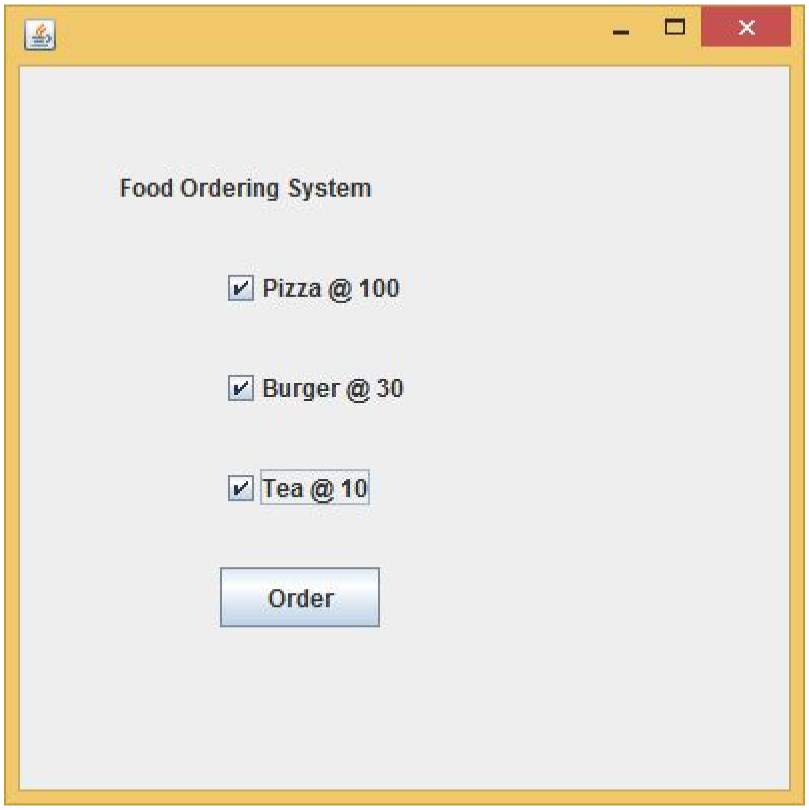
Output:

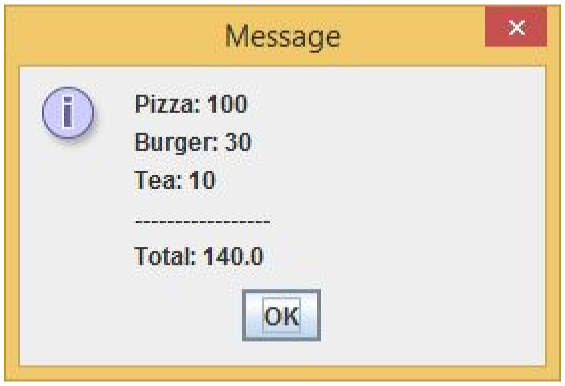


Java JCheckBox Example: Food Order

```
import javax.swing.*;
import java.awt.event.*;
public class CheckBoxExample extends JFrame implements ActionListener{
   JLabel l;
   JCheckBox cb1,cb2,cb3;
   JButton b;
   CheckBoxExample(){
      l=new JLabel("Food Ordering System");
      l.setBounds(50,50,300,20);
      cb1=new JCheckBox("Pizza @ 100");
      cb1.setBounds(100,100,150,20);
      cb2=new JCheckBox("Burger @ 30");
      cb2.setBounds(100,150,150,20);
      cb3=new JCheckBox("Tea @ 10");
      cb3.setBounds(100,200,150,20);
      b=new JButton("Order");
      b.setBounds(100,250,80,30);
      b.addActionListener(this);
      add(l);add(cb1);add(cb2);add(cb3);add(b);
      setSize(400,400);
      setLayout(null);
      setVisible(true);
      setDefaultCloseOperation(EXIT_ON_CLOSE);
   }
   public void actionPerformed(ActionEvent e){
      float amount=0;
      String msg="";
      if(cb1.isSelected()){
          amount+=100;
          msg="Pizza: 100\n";
      }
      if(cb2.isSelected()){
          amount+=30;
          msg+="Burger: 30\n";
      }
      if(cb3.isSelected()){
          amount+=10;
          msg+="Tea: 10\n";
      }
      msg+="-----------------\n";
      JOptionPane.showMessageDialog(this,msg+"Total: "+amount);
   }
   public static void main(String[] args) {
      new CheckBoxExample();
   }
}
```

Output:





```
Pizza: 100
Burger: 30
Tea: 10
-----------------
Total: 140.0
```

- <u>JRadioButton:</u>

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.

JRadioButton class declaration

Let's see the declaration for javax.swing.JRadioButton class.

UNIT 3. GUI- Components- 2 (Swings):

1. **public class** JRadioButton **extends** JToggleButton **implements** Accessible

   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.

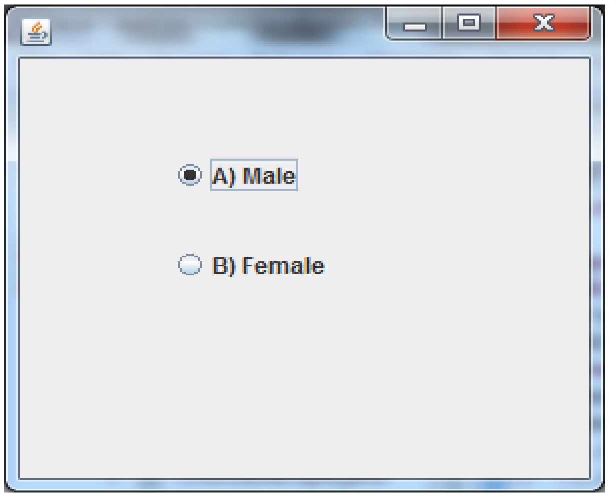
Commonly used Methods:

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.

Java JRadioButton 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:

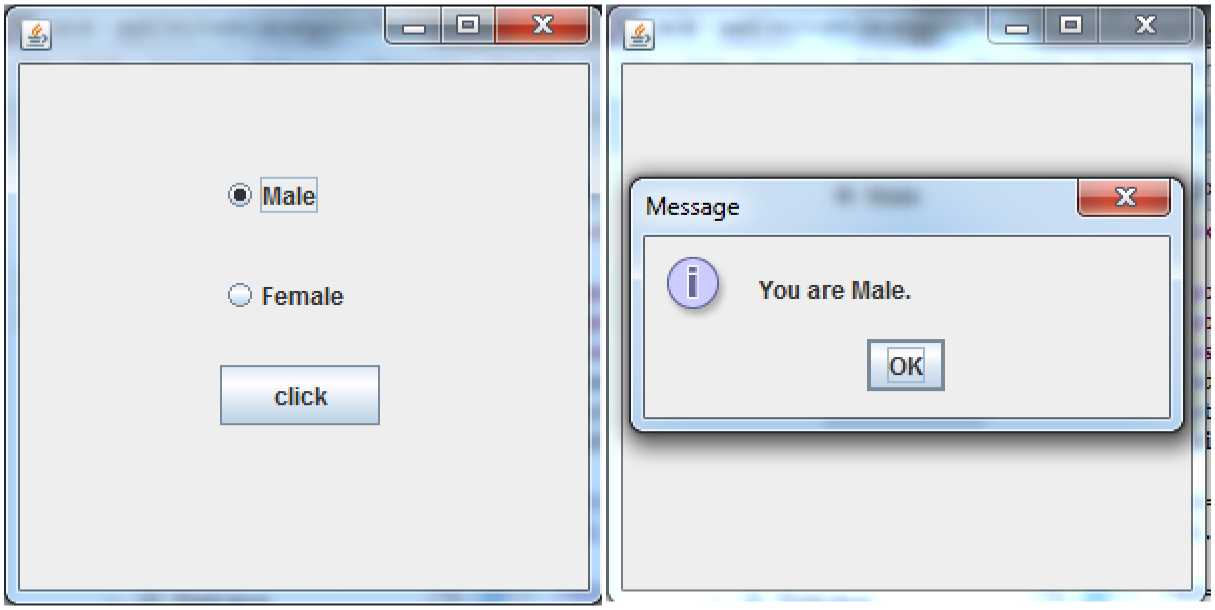


Java JRadioButton Example with ActionListener

```
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 void actionPerformed(ActionEvent e){
if(rb1.isSelected()){
JOptionPane.showMessageDialog(this,"You are Male.");
}
if(rb2.isSelected()){
JOptionPane.showMessageDialog(this,"You are Female.");
}
}
public static void main(String args[]){
new RadioButtonExample();
}}
```

Output:



- <u>JComboBox:</u>

The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a menu. It inherits JComponent class.

JComboBox class declaration

Let's see the declaration for javax.swing.JComboBox class.

1. **public class** JComboBox **extends** JComponent **implements** ItemSelectable, ListDataListener, ActionListener, Accessible

Commonly used Constructors:

Constructor	Description
JComboBox()	Creates a JComboBox with a default data model.
JComboBox(Object[] items)	Creates a JComboBox that contains the elements in the specified array.

JComboBox(Vector<?> items)	Creates a JComboBox that contains the elements in the specified Vector.

Commonly used Methods:

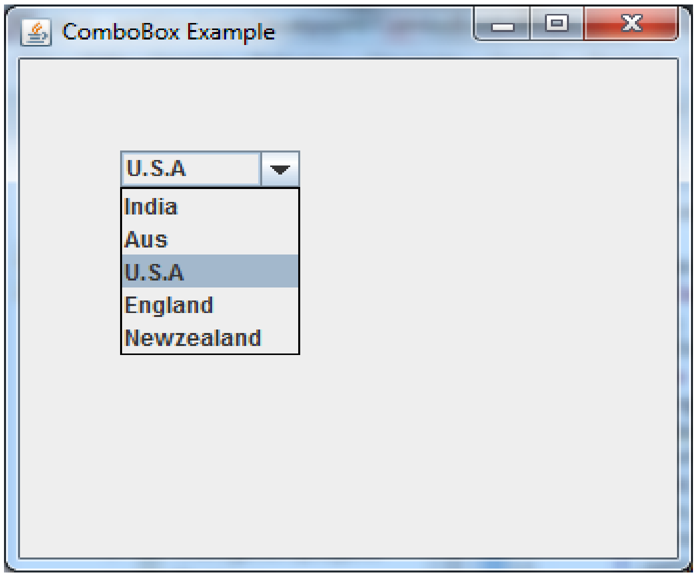
Methods	Description
void addItem(Object anObject)	It is used to add an item to the item list.
void removeItem(Object anObject)	It is used to delete an item to the item list.
void removeAllItems()	It is used to remove all the items from the list.
void setEditable(boolean b)	It is used to determine whether the JComboBox is editable.
void addActionListener(ActionListener a)	It is used to add the ActionListener.
void addItemListener(ItemListener i)	It is used to add the ItemListener.

Java JComboBox Example

```
import javax.swing.*;
public class ComboBoxExample {
JFrame f;
ComboBoxExample(){
   f=new JFrame("ComboBox Example");
   String country[]={"India","Aus","U.S.A","England","Newzealand"};
   JComboBox cb=new JComboBox(country);
```

```
    cb.setBounds(50, 50,90,20);
    f.add(cb);
    f.setLayout(null);
    f.setSize(400,500);
    f.setVisible(true);
}
public static void main(String[] args) {
    new ComboBoxExample();
}
}
```

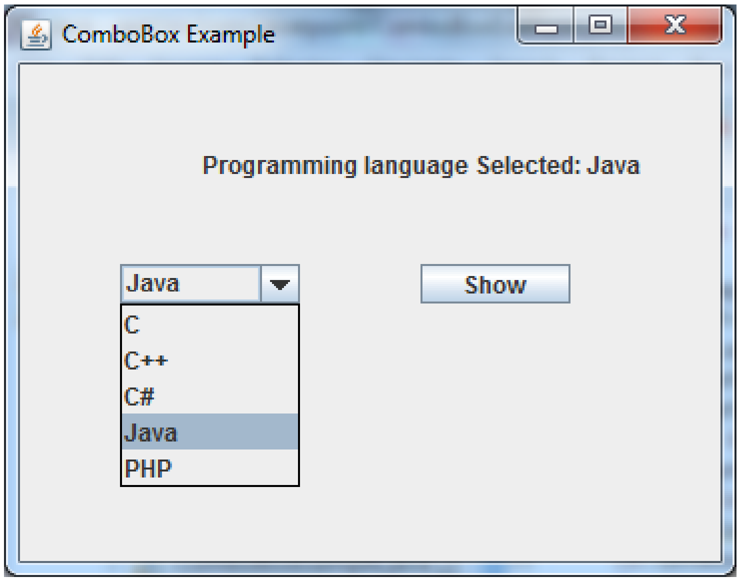
Output:



Java JComboBox Example with ActionListener

```
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);
    JButton b=new JButton("Show");
    b.setBounds(200,100,75,20);
    String languages[]={"C","C++","C#","Java","PHP"};
    final JComboBox cb=new JComboBox(languages);
    cb.setBounds(50, 100,90,20);
    f.add(cb); f.add(label); f.add(b);
    f.setLayout(null);
    f.setSize(350,350);
    f.setVisible(true);
    b.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
String data = "Programming language Selected: "
   + cb.getItemAt(cb.getSelectedIndex());
label.setText(data);
}
});
}
public static void main(String[] args) {
    new ComboBoxExample();
}
}
```

Output:



```
Programming language Selected: Java
```

- <u>JList:</u>

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.

JList class declaration

Let's see the declaration for javax.swing.JList class.

1. **public class** JList **extends** JComponent **implements** Scrollable, Accessible

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.
JList(ListModel<ary> dataModel)	Creates a JList that displays elements from the specified, non-null, model.

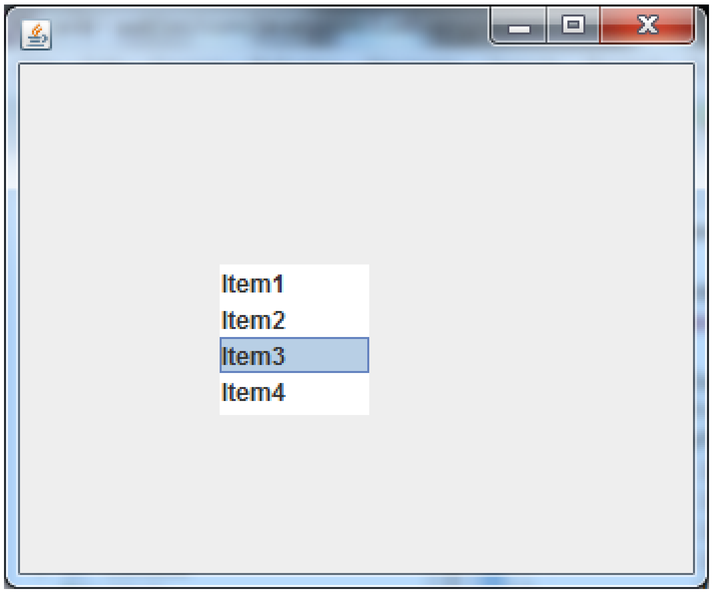
Commonly used Methods:

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.

Java JList Example

```
import javax.swing.*;
public class ListExample
{
    ListExample(){
      JFrame f= new JFrame();
      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);
        f.add(list);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
public static void main(String args[])
  {
  new ListExample();
  }}
```

Output:



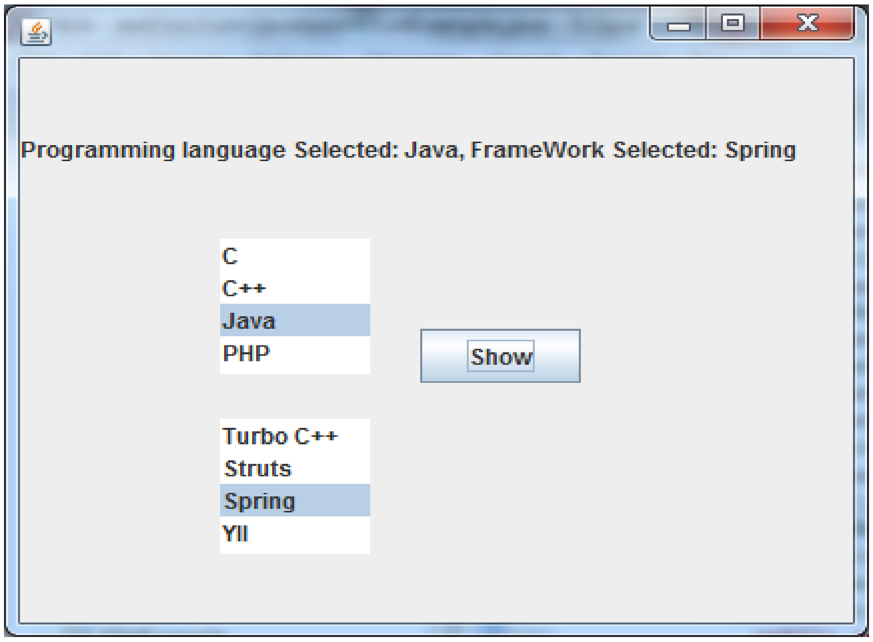
```
Item1
Item2
Item3
Item4
```

Java JList Example with ActionListener

```
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);
     final DefaultListModel<String> l1 = new DefaultListModel<>();
      l1.addElement("C");
      l1.addElement("C++");
      l1.addElement("Java");
      l1.addElement("PHP");
      final JList<String> list1 = new JList<>(l1);
      list1.setBounds(100,100, 75,75);
      DefaultListModel<String> l2 = new DefaultListModel<>();
      l2.addElement("Turbo C++");
      l2.addElement("Struts");
      l2.addElement("Spring");
      l2.addElement("YII");
      final JList<String> list2 = new JList<>(l2);
      list2.setBounds(100,200, 75,75);
      f.add(list1); f.add(list2); f.add(b); f.add(label);
      f.setSize(450,450);
      f.setLayout(null);
      f.setVisible(true);
      b.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
           String data = "";
           if (list1.getSelectedIndex() != -1) {
             data = "Programming language Selected: " + list1.getSelectedValue();
             label.setText(data);
           }
           if(list2.getSelectedIndex() != -1){
             data += ", FrameWork Selected: ";
             for(Object frame :list2.getSelectedValues()){
               data += frame + " ";
             }
           }
```

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

Output:



Programming language Selected: Java, FrameWork Selected: Spring

- <u>Multiple-Selection List:</u>

Java JList Multiple Selection Example

In the previous tutorials, we have discussed about <u>JList with single item selection</u>. In this tutorial, I am going to show a most useful example of how to use Java **JList Multiple Selection** mode.

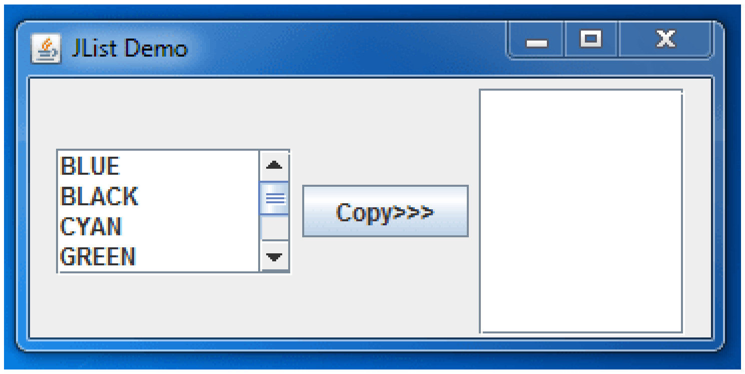
Java JList Multiple Selection :

I am going to reuse the previous example here and enable multiple selection modes and copy the selected items to another JList.

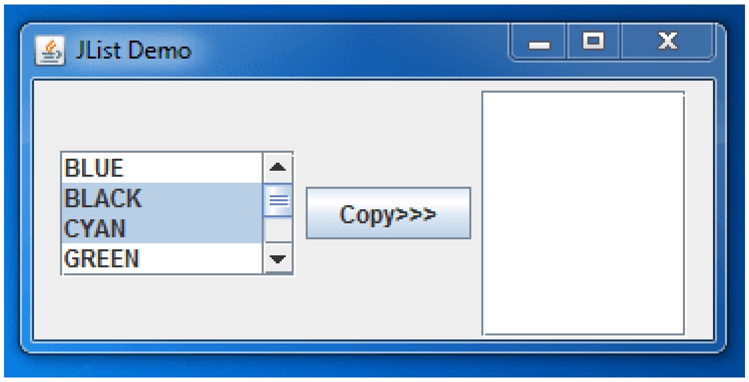
JListCopyDemo.java

```
import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JList;
import javax.swing.JScrollPane;
import javax.swing.ListSelectionModel;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;

import com.onlinetutorialspoint.swing.JListDemo;


public class JListCopyDemo extends JFrame {
    private JList jList;
    private JList jListForCopy;
    private JButton copyButton;
    private static final String[] listItems = { "BLUE", "BLACK", "CYAN",
            "GREEN", "GRAY", "RED", "WHITE" };
    private static final Color[] colors = { Color.BLUE, Color.BLACK,
            Color.CYAN, Color.GREEN, Color.GRAY, Color.RED, Color.WHITE };

    public JListCopyDemo() {
        super("JList Demo");
        setLayout(new FlowLayout());

        jList = new JList(listItems);
        jList.setFixedCellHeight(15);
        jList.setFixedCellWidth(100);
        jList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        jList.setVisibleRowCount(4);
        add(new JScrollPane(jList));

        copyButton = new JButton("Copy>>>");

        copyButton.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent e) {
                jListForCopy.setListData(jList.getSelectedValues());
            }
        });

        add(copyButton);
        jListForCopy = new JList();
        jListForCopy.setFixedCellHeight(15);
        jListForCopy.setFixedCellWidth(100);
        jList.setVisibleRowCount(4);
        jList.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
        add(new JScrollPane(jListForCopy));
    }
    public static void main(String[] args) {
        JListCopyDemo jListDemo = new JListCopyDemo();
        jListDemo.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jListDemo.setSize(350, 150);
        jListDemo.setVisible(true);
    }
}
```

Output :

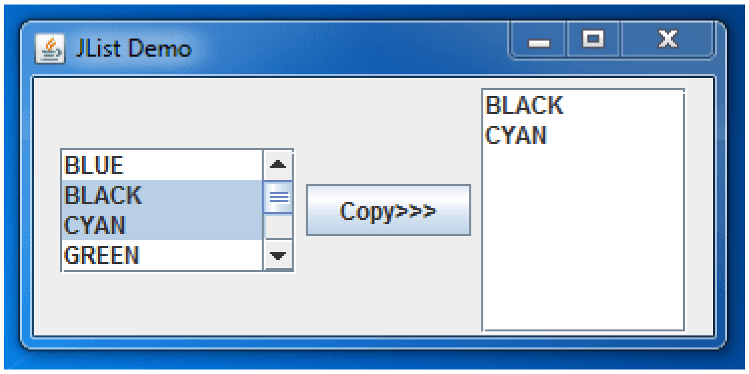


JList Multiple Selection :



UNIT 3. GUI- Components- 2 (Swings):

Copy JList Multiple Selection :



- <u>Mouse Event Handling:</u>

Java MouseListener Interface

The Java MouseListener is notified whenever you change the state of mouse. It is notified against MouseEvent. The MouseListener interface is found in java.awt.event package. It has five methods.

Methods of MouseListener interface

The signature of 5 methods found in MouseListener interface are given below:

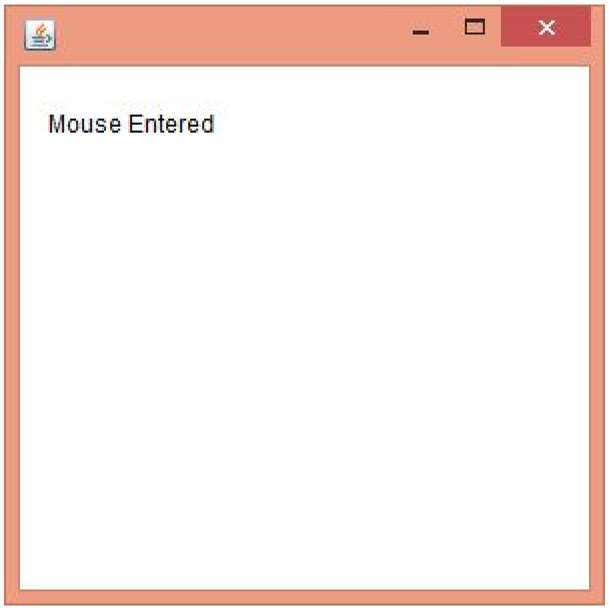
1. **public abstract void** mouseClicked(MouseEvent e);
2. **public abstract void** mouseEntered(MouseEvent e);
3. **public abstract void** mouseExited(MouseEvent e);
4. **public abstract void** mousePressed(MouseEvent e);
5. **public abstract void** mouseReleased(MouseEvent e);

Java MouseListener Example

```
import java.awt.*;
import java.awt.event.*;
public class MouseListenerExample extends Frame implements MouseListener{
   Label l;
   MouseListenerExample(){
      addMouseListener(this);

      l=new Label();
      l.setBounds(20,50,100,20);
```

```
        add(l);
        setSize(300,300);
        setLayout(null);
        setVisible(true);
    }
    public void mouseClicked(MouseEvent e) {
        l.setText("Mouse Clicked");
    }
    public void mouseEntered(MouseEvent e) {
        l.setText("Mouse Entered");
    }
    public void mouseExited(MouseEvent e) {
        l.setText("Mouse Exited");
    }
    public void mousePressed(MouseEvent e) {
        l.setText("Mouse Pressed");
    }
    public void mouseReleased(MouseEvent e) {
        l.setText("Mouse Released");
    }
public static void main(String[] args) {
    new MouseListenerExample();
}
}
```

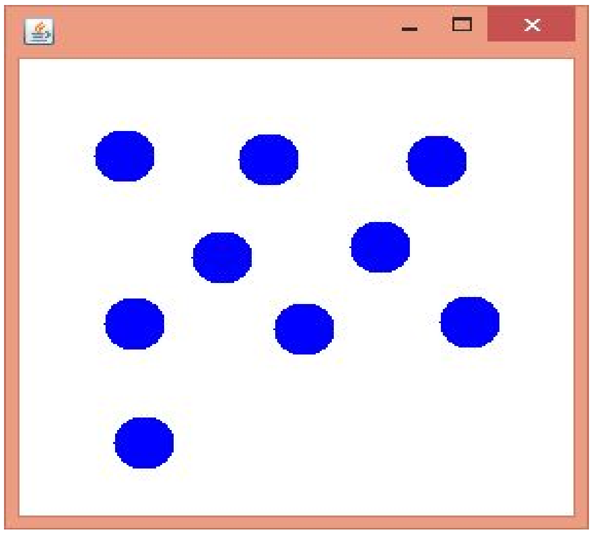
Output:


Mouse Entered

Java MouseListener Example 2

```
import java.awt.*;
import java.awt.event.*;
public class MouseListenerExample2 extends Frame implements MouseListener{
   MouseListenerExample2(){
      addMouseListener(this);

      setSize(300,300);
      setLayout(null);
      setVisible(true);
   }
   public void mouseClicked(MouseEvent e) {
      Graphics g=getGraphics();
      g.setColor(Color.BLUE);
      g.fillOval(e.getX(),e.getY(),30,30);
   }
   public void mouseEntered(MouseEvent e) {}
   public void mouseExited(MouseEvent e) {}
   public void mousePressed(MouseEvent e) {}
   public void mouseReleased(MouseEvent e) {}

public static void main(String[] args) {
   new MouseListenerExample2();
}
}
```

Output:



Java MouseMotionListener Interface

The Java MouseMotionListener is notified whenever you move or drag mouse. It is notified against MouseEvent. The MouseMotionListener interface is found in java.awt.event package. It has two methods.

Methods of MouseMotionListener interface

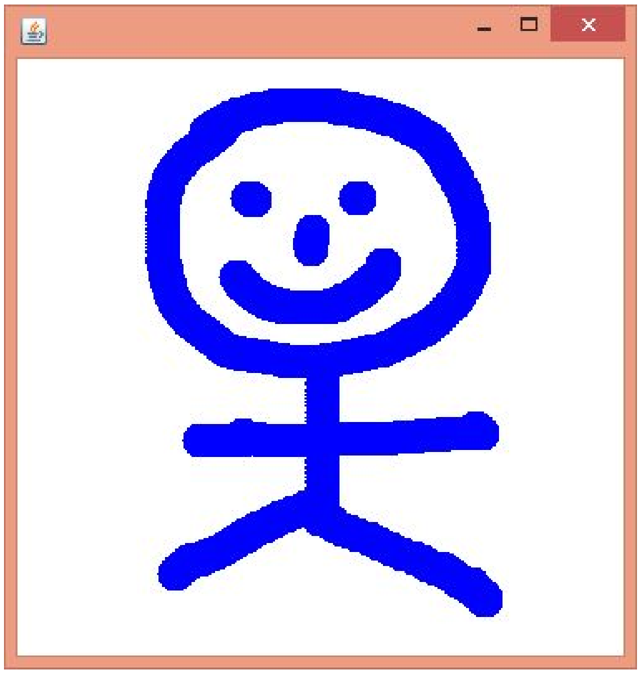
The signature of 2 methods found in MouseMotionListener interface are given below:

1. **public abstract void** mouseDragged(MouseEvent e);
2. **public abstract void** mouseMoved(MouseEvent e);

Java MouseMotionListener Example

```
import java.awt.*;
import java.awt.event.*;
public class MouseMotionListenerExample extends Frame implements MouseMotionListener{
  MouseMotionListenerExample(){
    addMouseMotionListener(this);

    setSize(300,300);
    setLayout(null);
    setVisible(true);
    }
public void mouseDragged(MouseEvent e) {
    Graphics g=getGraphics();
    g.setColor(Color.BLUE);
    g.fillOval(e.getX(),e.getY(),20,20);
}
public void mouseMoved(MouseEvent e) {}

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

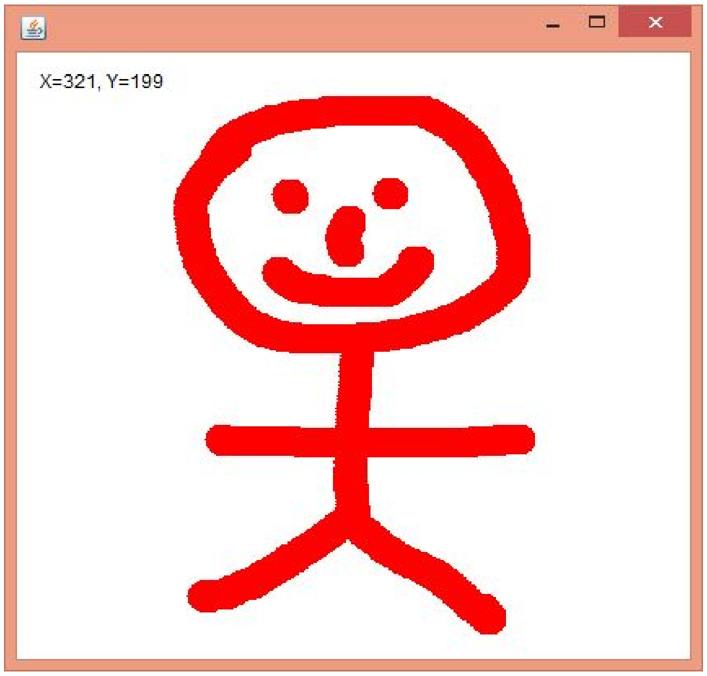
Output:



Java MouseMotionListener Example 2

```
import java.awt.*;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionListener;
public class Paint extends Frame implements MouseMotionListener{
    Label l;
    Color c=Color.BLUE;
    Paint(){
    l=new Label();
    l.setBounds(20,40,100,20);
    add(l);

    addMouseMotionListener(this);

    setSize(400,400);
    setLayout(null);
    setVisible(true);
}
public void mouseDragged(MouseEvent e) {
    l.setText("X="+e.getX()+", Y="+e.getY());
    Graphics g=getGraphics();
    g.setColor(Color.RED);
    g.fillOval(e.getX(),e.getY(),20,20);
}
public void mouseMoved(MouseEvent e) {
    l.setText("X="+e.getX()+", Y="+e.getY());
}
public static void main(String[] args) {
    new Paint();
}
}
```

Output:



- <u>Adapter Classes:</u>

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*.

The adapter classes are found in **java.awt.event, java.awt.dnd** 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
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
HierarchyBoundsAdapter	HierarchyBoundsListener

java.awt.dnd Adapter classes

Adapter class	Listener interface
DragSourceAdapter	DragSourceListener
DragTargetAdapter	DragTargetListener

javax.swing.event Adapter classes

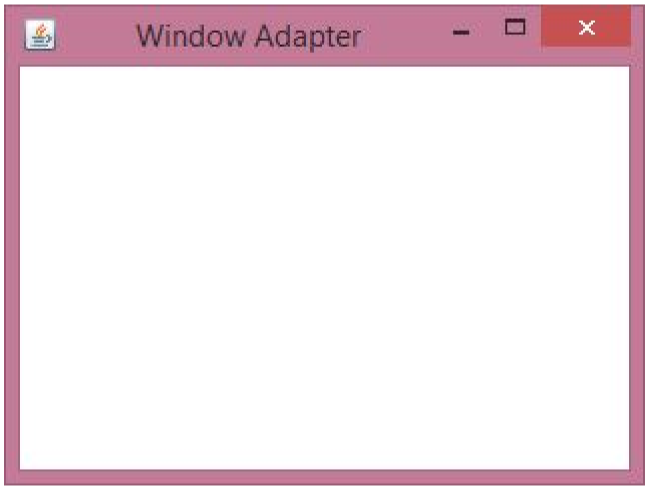
Adapter class	Listener interface
MouseInputAdapter	MouseInputListener
InternalFrameAdapter	InternalFrameListener

Java WindowAdapter Example

1. **import** java.awt.*;

```
import java.awt.event.*;
public class AdapterExample{
  Frame f;
  AdapterExample(){
    f=new Frame("Window Adapter");
    f.addWindowListener(new WindowAdapter(){
      public void windowClosing(WindowEvent e) {
        f.dispose();
      }
    });

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

Output:



Java 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();
}
}
```

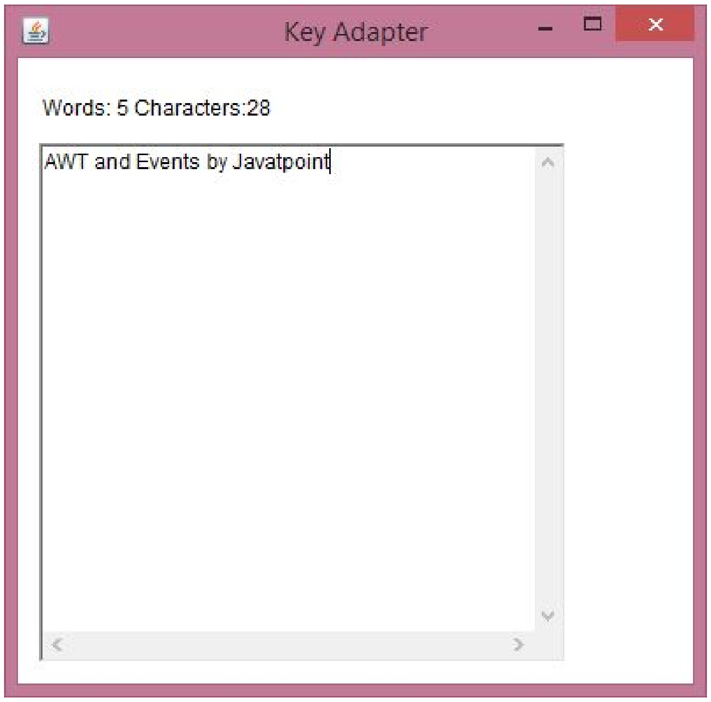
Java MouseMotionAdapter Example

```
import java.awt.*;
import java.awt.event.*;
public class MouseMotionAdapterExample extends MouseMotionAdapter{
    Frame f;
    MouseMotionAdapterExample(){
        f=new Frame("Mouse Motion Adapter");
        f.addMouseMotionListener(this);

        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
public void mouseDragged(MouseEvent e) {
    Graphics g=f.getGraphics();
    g.setColor(Color.ORANGE);
    g.fillOval(e.getX(),e.getY(),20,20);
}
public static void main(String[] args) {
    new MouseMotionAdapterExample();
}
}
```

Java KeyAdapter Example

```
import java.awt.*;
import java.awt.event.*;
public class KeyAdapterExample extends KeyAdapter{
   Label l;
   TextArea area;
   Frame f;
   KeyAdapterExample(){
      f=new Frame("Key Adapter");
      l=new Label();
      l.setBounds(20,50,200,20);
      area=new TextArea();
      area.setBounds(20,80,300, 300);
      area.addKeyListener(this);

      f.add(l);f.add(area);
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
   }
   public void keyReleased(KeyEvent e) {
      String text=area.getText();
      String words[]=text.split("\\s");
      l.setText("Words: "+words.length+" Characters:"+text.length());
   }

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

Output:



Words: 5 Characters:28

AWT and Events by Javatpoint

- <u>Key Event Handling:</u>

Java KeyListener Interface

The Java KeyListener is notified whenever you change the state of key. It is notified against KeyEvent. The KeyListener interface is found in java.awt.event package. It has three methods.

Methods of KeyListener interface

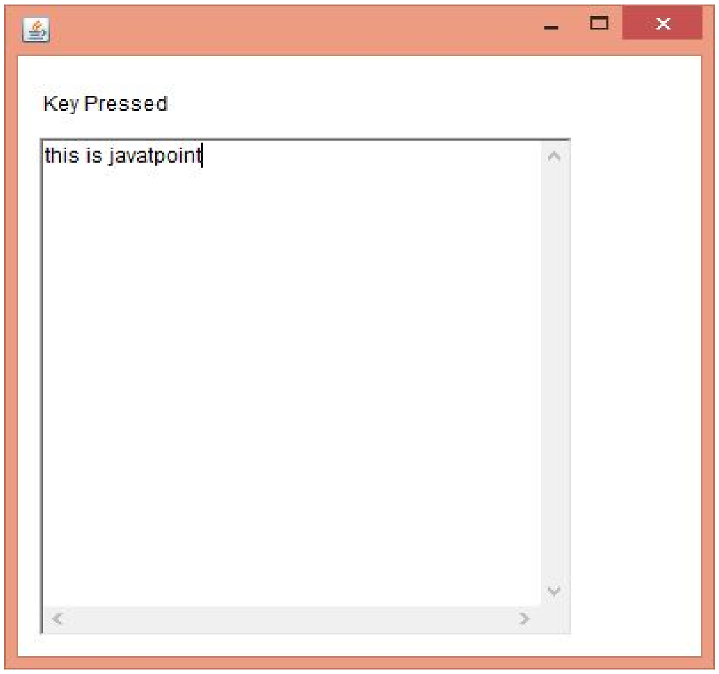
The signature of 3 methods found in KeyListener interface are given below:

1. **public abstract void** keyPressed(KeyEvent e);
2. **public abstract void** keyReleased(KeyEvent e);
3. **public abstract void** keyTyped(KeyEvent e);

Java KeyListener Example

```
import java.awt.*;
import java.awt.event.*;
public class KeyListenerExample extends Frame implements KeyListener{
   Label l;
   TextArea area;
KeyListenerExample(){

   l=new Label();
   l.setBounds(20,50,100,20);
   area=new TextArea();
   area.setBounds(20,80,300, 300);
   area.addKeyListener(this);

   add(l);add(area);
   setSize(400,400);
   setLayout(null);
   setVisible(true);
}
public void keyPressed(KeyEvent e) {
   l.setText("Key Pressed");
}
public void keyReleased(KeyEvent e) {
   l.setText("Key Released");
}
public void keyTyped(KeyEvent e) {
   l.setText("Key Typed");
}

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

Output:



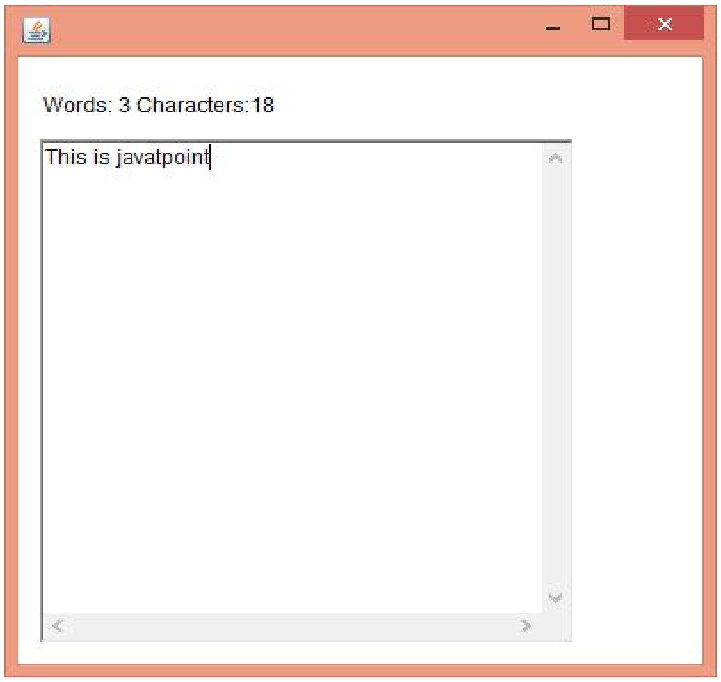
```
this is javatpoint
```

Java KeyListener Example 2: Count Words & Characters

```java
import java.awt.*;
import java.awt.event.*;
public class KeyListenerExample extends Frame implements KeyListener{
  Label l;
  TextArea area;
  KeyListenerExample(){

    l=new Label();
    l.setBounds(20,50,200,20);
    area=new TextArea();
    area.setBounds(20,80,300, 300);
    area.addKeyListener(this);

    add(l);add(area);
    setSize(400,400);
    setLayout(null);
    setVisible(true);
  }
  public void keyPressed(KeyEvent e) {}
  public void keyReleased(KeyEvent e) {
    String text=area.getText();
    String words[]=text.split("\\s");
```

```
        l.setText("Words: "+words.length+" Characters:"+text.length());
    }
    public void keyTyped(KeyEvent e) {}

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

Output:



```
Words: 3 Characters:18
This is javatpoint
```

- <u>Layout Managers:</u>

BorderLayout (LayoutManagers)

Java LayoutManager

The LayoutManagers are used to arrange components in a particular manner. LayoutManager is an interface that is implemented by all the classes of layout managers. There are following classes that represents the layout managers:

1. java.awt.BorderLayout
2. java.awt.FlowLayout

3. java.awt.GridLayout
4. java.awt.CardLayout
5. java.awt.GridBagLayout
6. javax.swing.BoxLayout
7. javax.swing.GroupLayout
8. javax.swing.ScrollPaneLayout
9. javax.swing.SpringLayout etc.

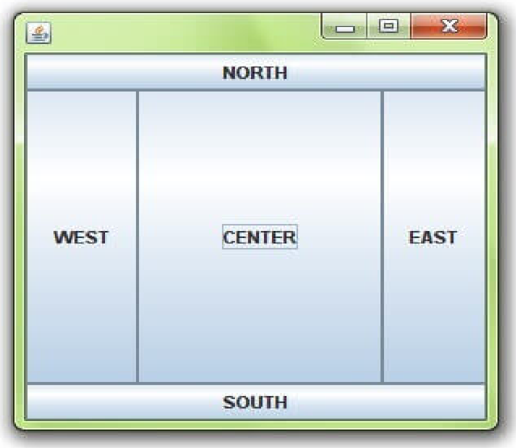
Java BorderLayout

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 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**

Constructors of BorderLayout class:

- **BorderLayout():** creates a border layout but with no gaps between the components.
- **JBorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components.
- Example of BorderLayout class



```
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();
}
}
```

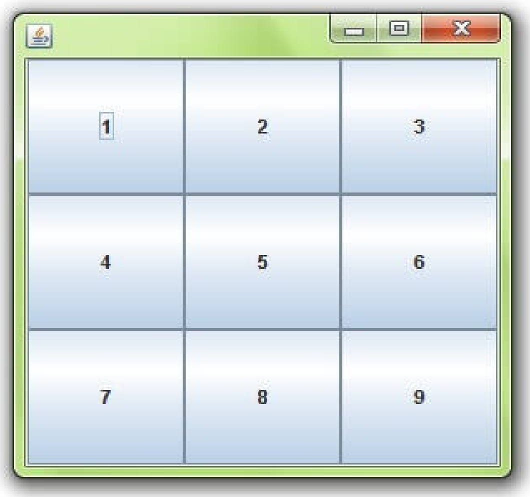
Java GridLayout

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

Constructors of GridLayout class

1. **GridLayout():** creates a grid layout with one column per component in a row.
2. **GridLayout(int rows, int columns):** creates a grid layout with the given rows and columns but no gaps between the components.
3. **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 of GridLayout class



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

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);

    f.setLayout(new GridLayout(3,3));
    //setting grid layout of 3 rows and 3 columns

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

Java 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

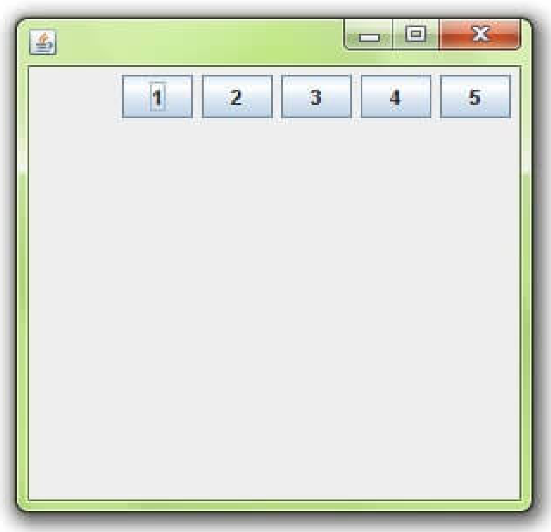
1. **public static final int LEFT**
2. **public static final int RIGHT**
3. **public static final int CENTER**
4. **public static final int LEADING**
5. **public static final int TRAILING**

Constructors of FlowLayout class

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

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

Example of FlowLayout class



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

public class MyFlowLayout{
JFrame f;
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);

    f.setLayout(new FlowLayout(FlowLayout.RIGHT));
    //setting flow layout of right alignment

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

Java BoxLayout

The BoxLayout is used to arrange the components either vertically or horizontally. For this purpose, BoxLayout provides four constants. They are as follows:

Note: BoxLayout class is found in javax.swing package.

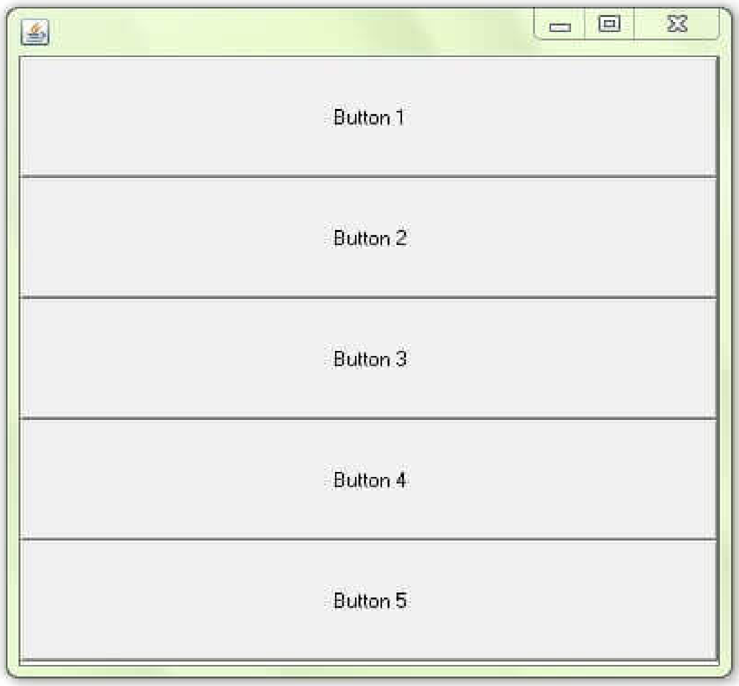
Fields of BoxLayout class

1. **public static final int X_AXIS**
2. **public static final int Y_AXIS**
3. **public static final int LINE_AXIS**
4. **public static final int PAGE_AXIS**

Constructor of BoxLayout class

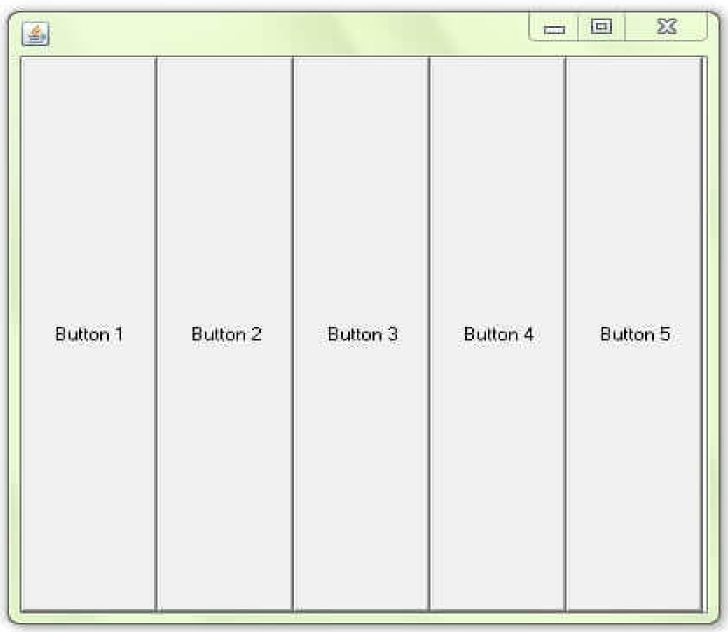
1. **BoxLayout(Container c, int axis):** creates a box layout that arranges the components with the given axis.

Example of BoxLayout clss with Y-AXIS:



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

public class BoxLayoutExample1 extends Frame {
 Button buttons[];

 public BoxLayoutExample1 () {
   buttons = new Button [5];

  for (int i = 0;i<5;i++) {
     buttons[i] = new Button ("Button " + (i + 1));
     add (buttons[i]);
  }

setLayout (new BoxLayout (this, BoxLayout.Y_AXIS));
setSize(400,400);
setVisible(true);
}

public static void main(String args[]){
BoxLayoutExample1 b=new BoxLayoutExample1();
}
}
```

Example of BoxLayout class with X-AXIS



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

public class BoxLayoutExample2 extends Frame {
 Button buttons[];

 public BoxLayoutExample2() {
   buttons = new Button [5];

   for (int i = 0;i<5;i++) {
      buttons[i] = new Button ("Button " + (i + 1));
      add (buttons[i]);
   }

setLayout (new BoxLayout(this, BoxLayout.X_AXIS));
setSize(400,400);
setVisible(true);
}

public static void main(String args[]){
```

```
BoxLayoutExample2 b=new BoxLayoutExample2();
}
}
```

Java CardLayout

The CardLayout class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout.

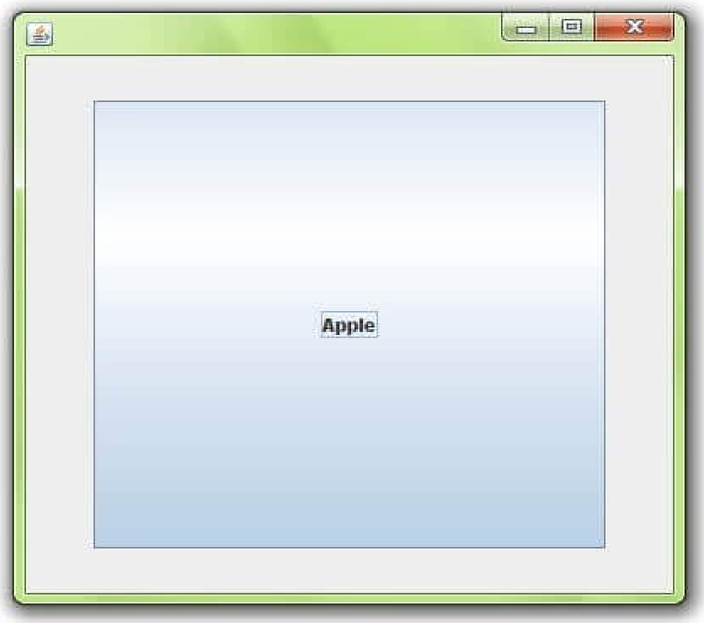
Constructors of CardLayout class

1. **CardLayout():** creates a card layout with zero horizontal and vertical gap.
2. **CardLayout(int hgap, int vgap):** creates a card layout with the given horizontal and vertical gap.

Commonly used methods of CardLayout class

- **public void next(Container parent):** is used to flip to the next card of the given container.
- **public void previous(Container parent):** is used to flip to the previous card of the given container.
- **public void first(Container parent):** is used to flip to the first card of the given container.
- **public void last(Container parent):** is used to flip to the last card of the given container.
- **public void show(Container parent, String name):** is used to flip to the specified card with the given name.

Example of CardLayout class



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

import javax.swing.*;

public class CardLayoutExample extends JFrame implements ActionListener{
CardLayout card;
JButton b1,b2,b3;
Container c;
    CardLayoutExample(){

        c=getContentPane();
        card=new CardLayout(40,30);
//create CardLayout object with 40 hor space and 30 ver space
        c.setLayout(card);

        b1=new JButton("Apple");
        b2=new JButton("Boy");
        b3=new JButton("Cat");
        b1.addActionListener(this);
        b2.addActionListener(this);
        b3.addActionListener(this);

        c.add("a",b1);c.add("b",b2);c.add("c",b3);

    }
    public void actionPerformed(ActionEvent e) {
    card.next(c);
    }

    public static void main(String[] args) {
        CardLayoutExample cl=new CardLayoutExample();
        cl.setSize(400,400);
        cl.setVisible(true);
        cl.setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}
```

Java GridBagLayout

The Java GridBagLayout class is used to align components vertically, horizontally or along their baseline.

The components may not be of same size. Each GridBagLayout object maintains a dynamic, rectangular grid of cells. Each component occupies one or more cells known as its display area. Each component associates an instance of GridBagConstraints. With the help of constraints object we arrange component's display area on the grid. The GridBagLayout manages each component's minimum and preferred sizes in order to determine component's size.

Fields

Modifier and Type	Field	Description
double[]	columnWeights	It is used to hold the overrides to the column weights.
int[]	columnWidths	It is used to hold the overrides to the column minimum width.
protected Hashtable<Component,GridBag	comptable	It is used to maintains the association between a component and its gridbag constraints.

Constraints>		
protected GridBagConstraints	defaultConstraints	It is used to hold a gridbag constraints instance containing the default values.
protected GridBagLayoutInfo	layoutInfo	It is used to hold the layout information for the gridbag.
protected static int	MAXGRIDSIZE	No longer in use just for backward compatibility
protected static int	MINSIZE	It is smallest grid that can be laid out by the grid bag layout.
protected static int	PREFERREDSIZE	It is preferred grid size that can be laid out by the grid bag layout.
int[]	rowHeights	It is used to hold the overrides to the row minimum heights.
double[]	rowWeights	It is used to hold the overrides to the row weights.

Useful Methods

Modifier and Type	Method	Description
void	addLayoutComponent(Component comp, Object constraints)	It adds specified component to the layout, using the specified constraints object.
void	addLayoutComponent(String name, Component comp)	It has no effect, since this layout manager does not use a per-component string.

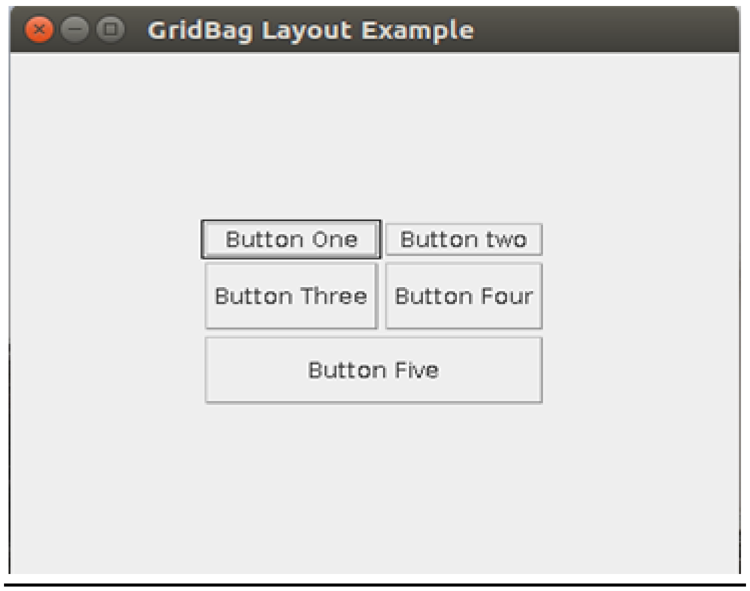
protected void	adjustForGravity(GridBagConstraints constraints, Rectangle r)	It adjusts the x, y, width, and height fields to the correct values depending on the constraint geometry and pads.
protected void	AdjustForGravity(GridBagConstraints constraints, Rectangle r)	This method is for backwards compatibility only
protected void	arrangeGrid(Container parent)	Lays out the grid.
protected void	ArrangeGrid(Container parent)	This method is obsolete and supplied for backwards compatibility
GridBagConstraints	getConstraints(Component comp)	It is for getting the constraints for the specified component.
float	getLayoutAlignmentX(Container parent)	It returns the alignment along the x axis.
float	getLayoutAlignmentY(Container parent)	It returns the alignment along the y axis.
int[][]	getLayoutDimensions()	It determines column widths and row heights for the layout grid.
protected GridBagLayoutInfo	getLayoutInfo(Container parent, int sizeflag)	This method is obsolete and supplied for backwards compatibility.
protected GridBagLayoutInfo	GetLayoutInfo(Container parent, int sizeflag)	This method is obsolete and supplied for backwards compatibility.
Point	getLayoutOrigin()	It determines the origin of the layout area, in the graphics coordinate space of the target container.

double[][]	getLayoutWeights()	It determines the weights of the layout grid's columns and rows.
protected Dimension	getMinSize(Container parent, GridBagLayoutInfo info)	It figures out the minimum size of the master based on the information from getLayoutInfo.
protected Dimension	GetMinSize(Container parent, GridBagLayoutInfo info)	This method is obsolete and supplied for backwards compatibility only

Example

```
import java.awt.Button;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;

import javax.swing.*;
public class GridBagLayoutExample extends JFrame{
  public static void main(String[] args) {
        GridBagLayoutExample a = new GridBagLayoutExample();
    }
    public GridBagLayoutExample() {
  GridBagLayoutgrid = new GridBagLayout();
        GridBagConstraints gbc = new GridBagConstraints();
        setLayout(grid);
        setTitle("GridBag Layout Example");
        GridBagLayout layout = new GridBagLayout();
  this.setLayout(layout);
  gbc.fill = GridBagConstraints.HORIZONTAL;
  gbc.gridx = 0;
  gbc.gridy = 0;
  this.add(new Button("Button One"), gbc);
  gbc.gridx = 1;
  gbc.gridy = 0;
  this.add(new Button("Button two"), gbc);
gbc.fill = GridBagConstraints.HORIZONTAL;
gbc.ipady = 20;
gbc.gridx = 0;
gbc.gridy = 1;
this.add(new Button("Button Three"), gbc);
gbc.gridx = 1;
gbc.gridy = 1;
this.add(new Button("Button Four"), gbc);
gbc.gridx = 0;
gbc.gridy = 2;
gbc.fill = GridBagConstraints.HORIZONTAL;
gbc.gridwidth = 2;
this.add(new Button("Button Five"), gbc);
        setSize(300, 300);
        setPreferredSize(getSize());
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);

    }

}
```

Output:



Example 2

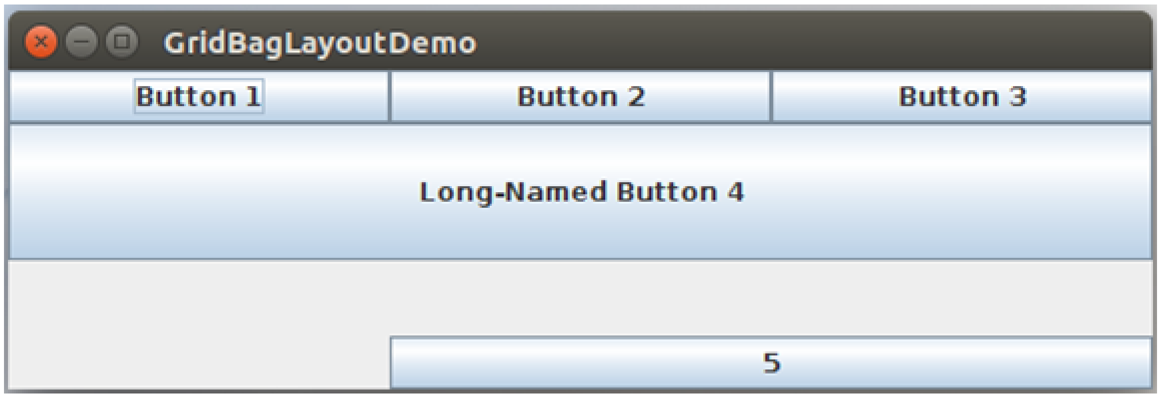
UNIT 3. GUI- Components- 2 (Swings):

```
public class GridBagLayoutDemo {
final static boolean shouldFill = true;
final static boolean shouldWeightX = true;
final static boolean RIGHT_TO_LEFT = false;

public static void addComponentsToPane(Container pane) {
if (RIGHT_TO_LEFT) {
pane.setComponentOrientation(ComponentOrientation.RIGHT_TO_LEFT);
}

JButton button;
pane.setLayout(new GridBagLayout());
GridBagConstraints c = new GridBagConstraints();
if (shouldFill) {
//natural height, maximum width
c.fill = GridBagConstraints.HORIZONTAL;
}

button = new JButton("Button 1");
if (shouldWeightX) {
c.weightx = 0.5;
}
c.fill = GridBagConstraints.HORIZONTAL;
c.gridx = 0;
c.gridy = 0;
pane.add(button, c);

button = new JButton("Button 2");
c.fill = GridBagConstraints.HORIZONTAL;
c.weightx = 0.5;
c.gridx = 1;
c.gridy = 0;
pane.add(button, c);

button = new JButton("Button 3");
c.fill = GridBagConstraints.HORIZONTAL;
c.weightx = 0.5;
c.gridx = 2;
c.gridy = 0;
pane.add(button, c);

```

```
button = new JButton("Long-Named Button 4");
c.fill = GridBagConstraints.HORIZONTAL;
c.ipady = 40;      //make this component tall
c.weightx = 0.0;
c.gridwidth = 3;
c.gridx = 0;
c.gridy = 1;
pane.add(button, c);

button = new JButton("5");
c.fill = GridBagConstraints.HORIZONTAL;
c.ipady = 0;       //reset to default
c.weighty = 1.0;   //request any extra vertical space
c.anchor = GridBagConstraints.PAGE_END; //bottom of space
c.insets = new Insets(10,0,0,0);  //top padding
c.gridx = 1;       //aligned with button 2
c.gridwidth = 2;   //2 columns wide
c.gridy = 2;       //third row
pane.add(button, c);
}


private static void createAndShowGUI() {
//Create and set up the window.
JFrame frame = new JFrame("GridBagLayoutDemo");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

//Set up the content pane.
addComponentsToPane(frame.getContentPane());

//Display the window.
frame.pack();
frame.setVisible(true);
}

public static void main(String[] args) {
javax.swing.SwingUtilities.invokeLater(new Runnable() {
public void run() {
createAndShowGUI();
}
});
```

```
}
}
```

Output:



ScrollPaneLayout

The layout manager used by JScrollPane. JScrollPaneLayout is responsible for nine components: a viewport, two scrollbars, a row header, a column header, and four "corner" components.

Nested Class

Modifier and Type	Class	Description
static class	ScrollPaneLayout.UIResource	It is UI resource version of ScrollPaneLayout.

Field

Modifier and Type	Field	Description
protected JViewport	colHead	It is column header child.
protected JScrollBar	hsb	It is scrollpane's horizontal scrollbar child.

protected int	hsbPolicy	It displays policy for the horizontal scrollbar.
protected Component	lowerLeft	This displays the lower left corner.
protected Component	lowerRight	This displays in the lower right corner.
protected JViewport	rowHead	It is row header child.
protected Component	upperLeft	This component displays in the upper left corner.
protected Component	upperRight	This component displays in the upper right corner.
protected JViewport	viewport	It is scrollpane's viewport child.
protected JScrollBar	vsb	It is scrollpane's vertical scrollbar child.
protected int	vsbPolicy	It is the display policy for the vertical scrollbar.

Useful methods

Modifier and Type	Method	Description
void	addLayoutComponent(String s, Component c)	It adds the specified component to the layout.
protected Component	addSingletonComponent(Component oldC, Component newC)	It removes an existing component.

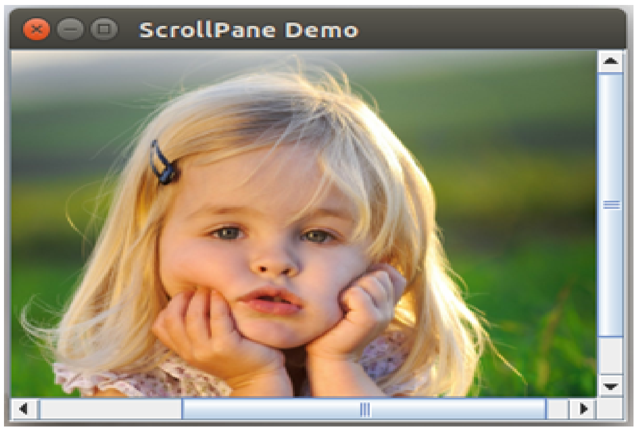
JViewport	getColumnHeader()	It returns the JViewport object that is the column header.
Component	getCorner(String key)	It returns the Component at the specified corner.
JScrollBar	getHorizontalScrollBar()	It returns the JScrollBar object that handles horizontal scrolling.
int	getHorizontalScrollBarPolicy()	It returns the horizontal scrollbar-display policy.
JViewport	getRowHeader()	It returns the JViewport object that is the row header.
JScrollBar	getVerticalScrollBar()	It returns the JScrollBar object that handles vertical scrolling.
int	getVerticalScrollBarPolicy()	It returns the vertical scrollbar-display policy.
JViewport	getViewport()	It returns the JViewport object that displays the scrollable contents.

Example:

```
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JScrollPane;
public class ScrollPaneDemo extends JFrame
{
```

```
public ScrollPaneDemo() {
super("ScrollPane Demo");
ImageIcon img = new ImageIcon("child.png");

JScrollPane png = new JScrollPane(new JLabel(img));

getContentPane().add(png);
setSize(300,250);
setVisible(true);
}

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

Output:



- <u>JTextArea:</u>

Java 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

JTextArea class declaration

Let's see the declaration for javax.swing.JTextArea class.

```
public class JTextArea extends JTextComponent
```

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.

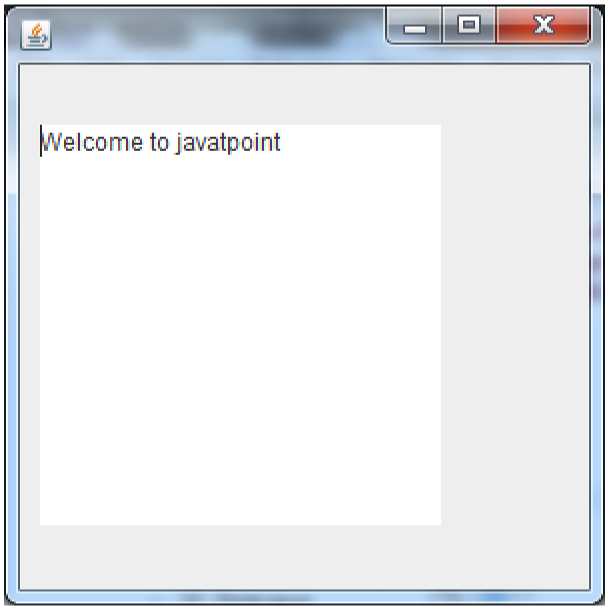
Commonly used Methods:

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.

Java JTextArea Example

```
import javax.swing.*;
public class TextAreaExample
{
     TextAreaExample(){
        JFrame f= new JFrame();
        JTextArea 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:

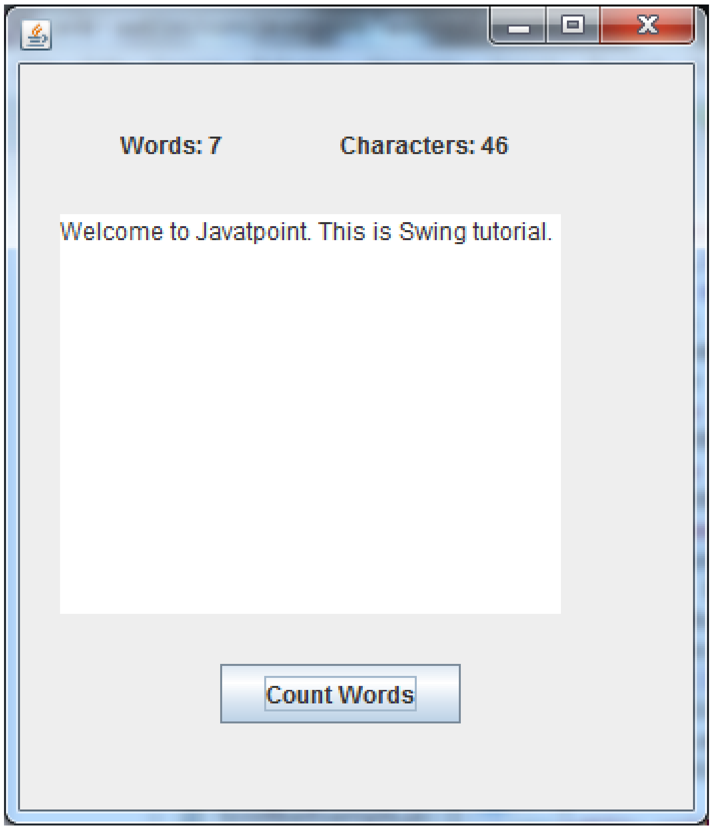


```
Welcome to javatpoint
```

Java JTextArea Example with ActionListener

```
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();
}
}
```

Output:



Welcome to Javatpoint. This is Swing tutorial.