MODULE-5.1
SWING FUNDAMENTALS

Introduction:

Java is one of the most in-demand programming languages for developing a variety of applications. The popularity of Java can be attributed to its versatility as it can be used to design customized applications that are light and fast and serve a variety of purposes ranging from web services to android applications. Java is fast, reliable and secure. There are multiple ways to develop GUI based applications in java, out of which the most popular ones are AWT and Swing.

Java AWT:

AWT stands for Abstract Window Toolkit. It is a platform-dependent API to develop GUI (Graphical User Interface) or window-based applications in Java. It was developed by Sun Microsystems In 1995. It is heavy-weight in use because it is generated by the system's host operating system. It contains a large number of classes and methods, which are used for creating and managing GUI.

JAVA Swing

Java Swing l 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.

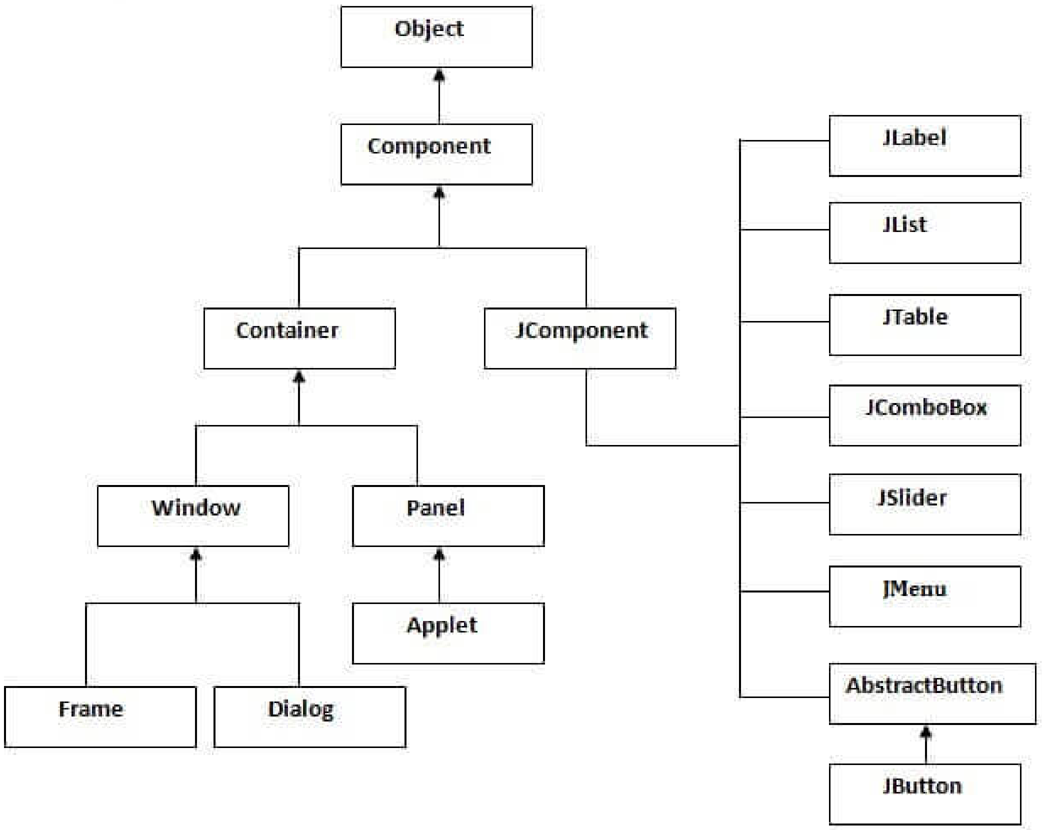
Unlike AWT, Java Swing provides platform-independent and lightweight components.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

What is JFC

The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.

The hierarchy of java swing API is given below.



Commonly used Methods of Component class

The methods of Component class are widely used in java swing that are given below.

Method	Description
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.
public void setVisible(boolean b)	sets the visibility of the component. It is by default false.

Java Swing Examples

There are two ways to create a frame:

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

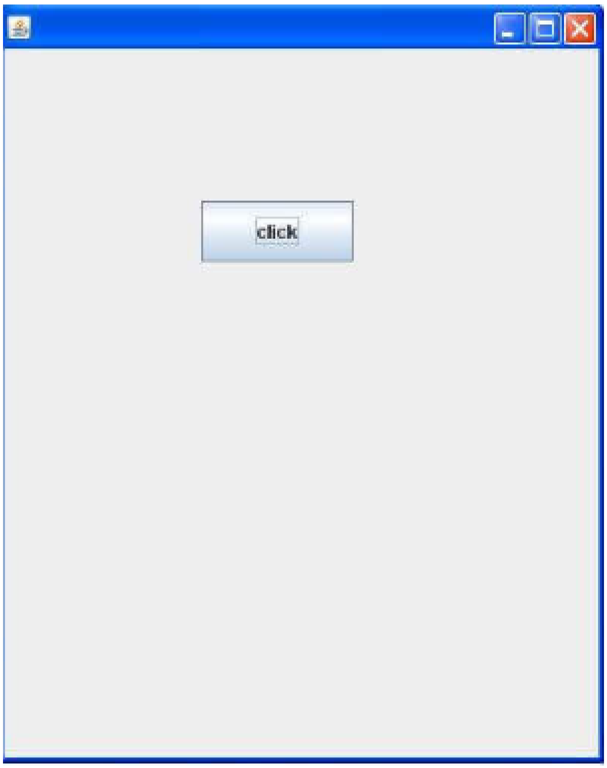
We can write the code of swing inside the main(), constructor or any other method.

Simple Java Swing Example

Let's see a simple swing example where we are creating one button and adding it on the JFrame object inside the main() method.

File: FirstSwingExample.java

```java
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
}
}
```



Example of Swing by Association inside constructor

We can also write all the codes of creating JFrame, JButton and method call inside the java constructor.

File: Simple.java

```java
import javax.swing.*;
public class Simple {
JFrame f;
Simple(){
f=new JFrame();//creating instance of JFrame

JButton b=new JButton("click");//creating instance of JButton
b.setBounds(130,100,100, 40);

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
}

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

The setBounds(int xaxis, int yaxis, int width, int height)is used in the above example that sets the position of the button.

Simple example of Swing by inheritance

We can also inherit the JFrame class, so there is no need to create the instance of JFrame class explicitly.

File: Simple2.java

```
import javax.swing.*;
public class Simple2 extends JFrame{//inheriting JFrame
JFrame f;
Simple2(){
JButton b=new JButton("click");//create button
b.setBounds(130,100,100, 40);

add(b);//adding button on frame
setSize(400,500);
setLayout(null);
setVisible(true);
}
public static void main(String[] args) {
new Simple2();
}}
```

Difference between AWT and Swing

No.	Java AWT	Java Swing

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

1.Java 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.

JLabel class declaration

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

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

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 Constructors:

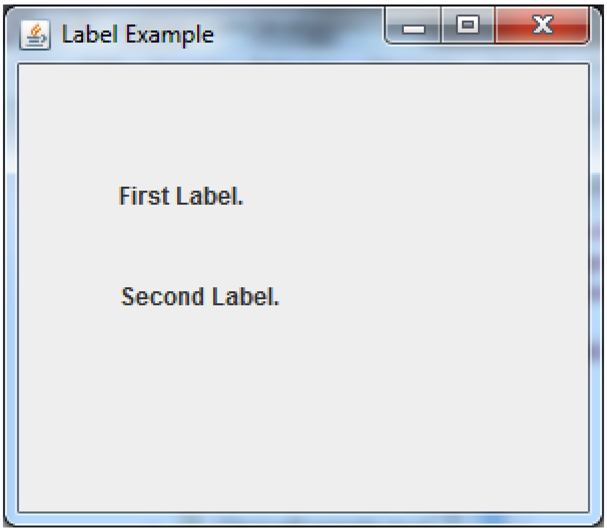
Commonly used Methods:

Methods	Description
String getText()	t 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:



Java 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.

```
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.

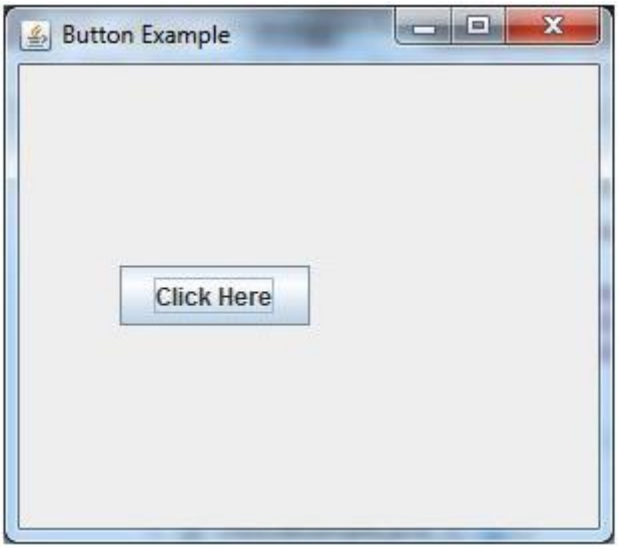
Commonly used Methods of AbstractButton class:

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);
}
}
```

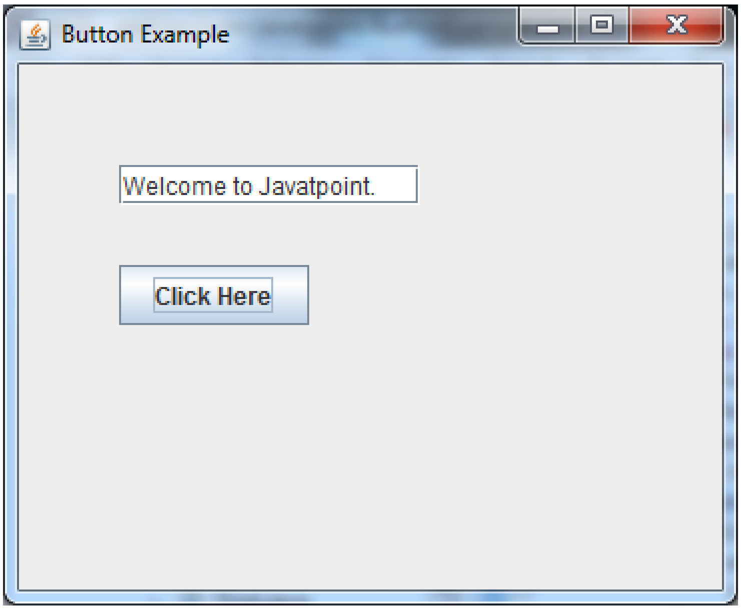
Output:



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:

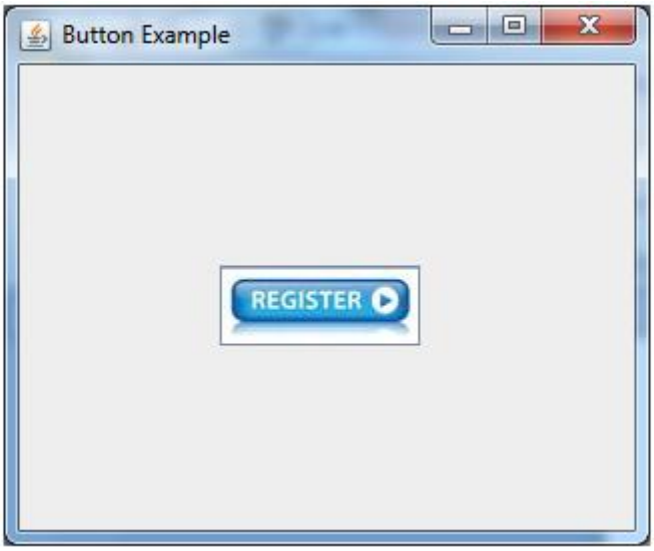


Welcome to Javatpoint.

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:



Java JTextField

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

JTextField class declaration

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

```
public class JTextField extends JTextComponent implements SwingConstants
```

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)	It is used to 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)	It is used to set the current font.
void removeActionListener(ActionListener l)	It is used to remove the specified action listener so that it no longer receives action events from this textfield.

Java JTextField Example

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

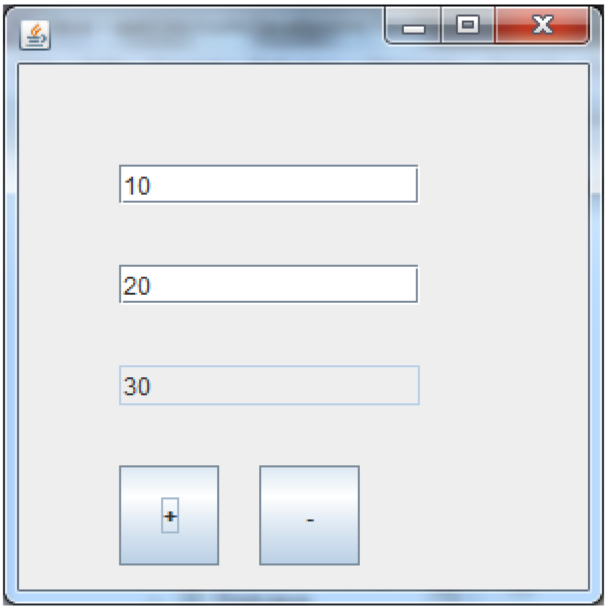
Output:

Output:

Java JTextField Example with ActionListener

```java
import javax.swing.*;
import java.awt.event.*;
public class TextFieldExample implements ActionListener{
    JTextField tf1,tf2,tf3;
    JButton b1,b2;
    TextFieldExample(){
        JFrame f= new JFrame();
        tf1=new JTextField();
        tf1.setBounds(50,50,150,20);
        tf2=new JTextField();
        tf2.setBounds(50,100,150,20);
        tf3=new JTextField();
        tf3.setBounds(50,150,150,20);
        tf3.setEditable(false);
        b1=new JButton("+");
        b1.setBounds(50,200,50,50);
        b2=new JButton("-");
        b2.setBounds(120,200,50,50);
        b1.addActionListener(this);
        b2.addActionListener(this);
        f.add(tf1);f.add(tf2);f.add(tf3);f.add(b1);f.add(b2);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
    public void actionPerformed(ActionEvent e) {
        String s1=tf1.getText();
        String s2=tf2.getText();
        int a=Integer.parseInt(s1);
        int b=Integer.parseInt(s2);
        int c=0;
        if(e.getSource()==b1){
            c=a+b;
        }
        else if(e.getSource()==b2){
            c=a-b;
        }
        String result=String.valueOf(c);
        tf3.setText(result);
    }
public static void main(String[] args) {
    new TextFieldExample();
} }
```

Output:



10

20

30

ava 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.

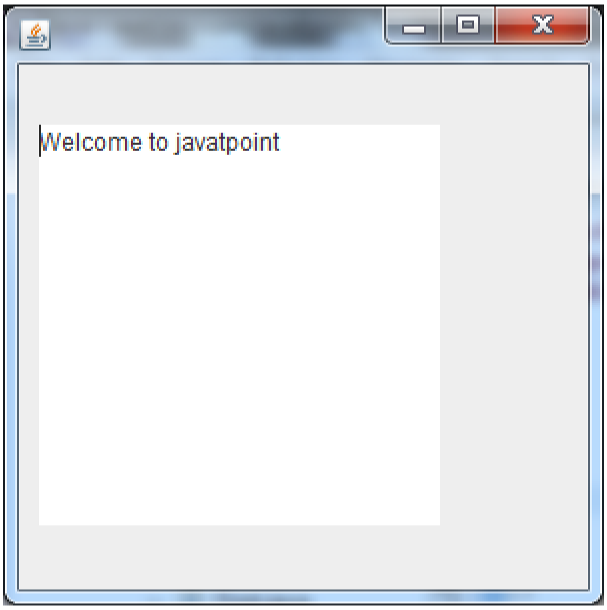
Commonly used Methods:

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:

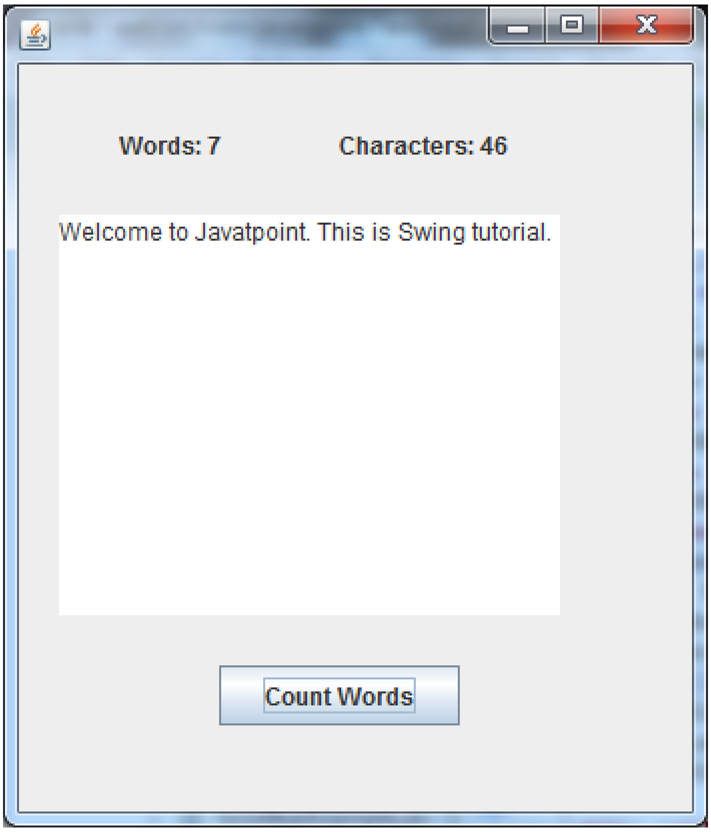


Welcome to javatpoint

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:



Words: 7

Characters: 46

Welcome to Javatpoint. This is Swing tutorial.

Java 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.

JRadioButton class declaration

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

```
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:

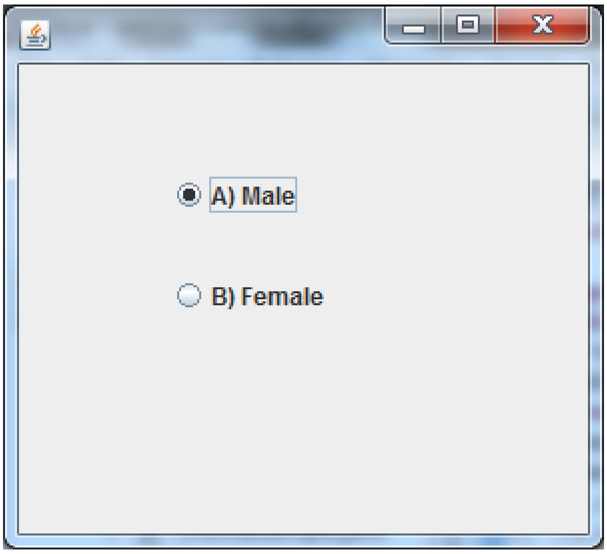
Methods	Description
void setText(String s)	It is used to set specified text on button.
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()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



● A) Male

○ B) Female

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);
```

```java
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:

Java JCheckBox

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 JToggleButton class.

JCheckBox class declaration

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

```java
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:

Java JCheckBox

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 JToggleButton class.

JCheckBox class declaration

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

```
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();
    }}
```

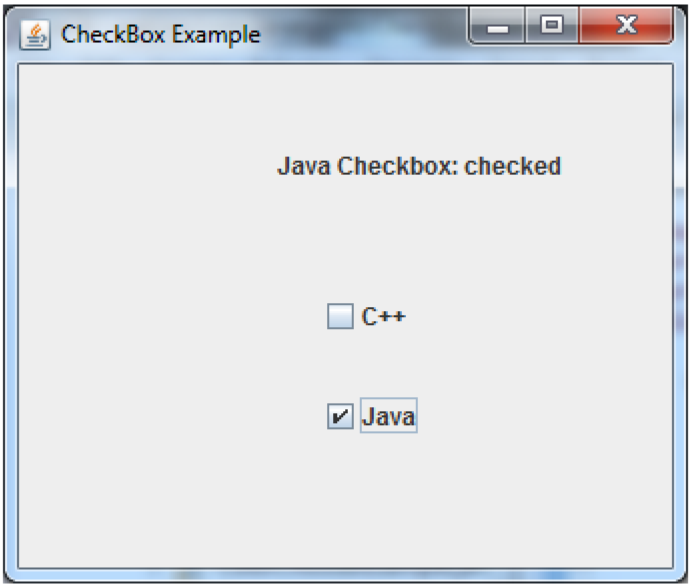
Output:

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:

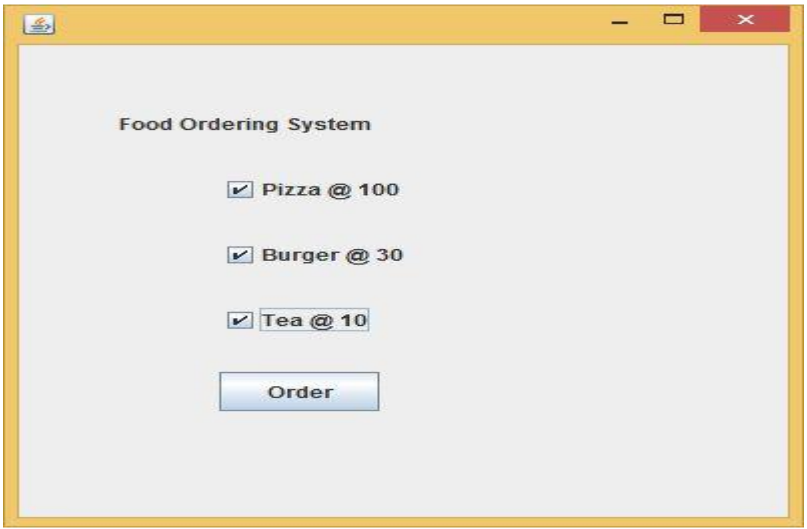


Java Checkbox: checked

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

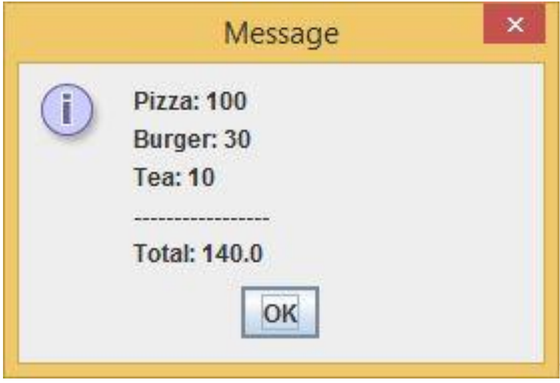


Food Ordering System

☑ Pizza @ 100

☑ Burger @ 30

☑ Tea @ 10



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

t:

Java 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.

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.

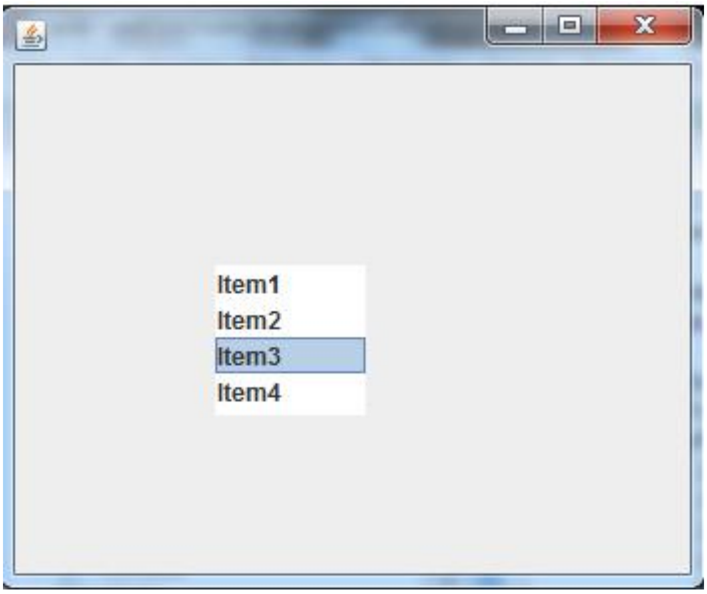
Commonly used Methods:

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:

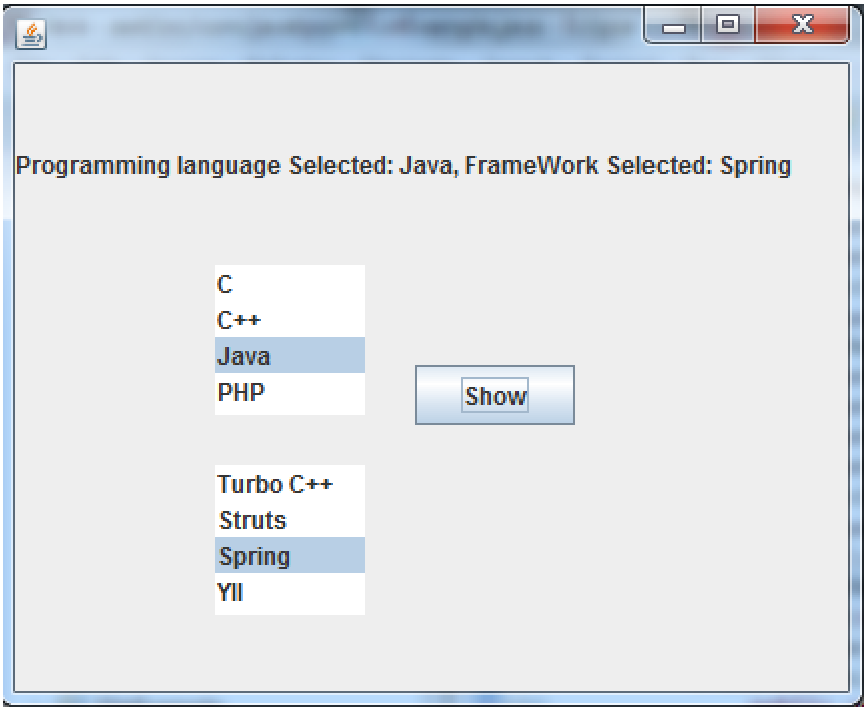


Java JList Example with ActionListener

```
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();
   }}
```



Programming language Selected: Java, FrameWork Selected: Spring

Java JScrollPane

A JscrollPane is used to make scrollable view of a component. When screen size is limited, we use a scroll pane to display a large component or a component whose size can change dynamically.

Constructors

Constructor	Purpose
JScrollPane()	It creates a scroll pane. The Component parameter, when present, sets the scroll pane's client. The two int parameters, when present, set the vertical and horizontal scroll bar policies (respectively).
JScrollPane(Component)	
JScrollPane(int, int)	

JScrollPane(Component, int, int)	

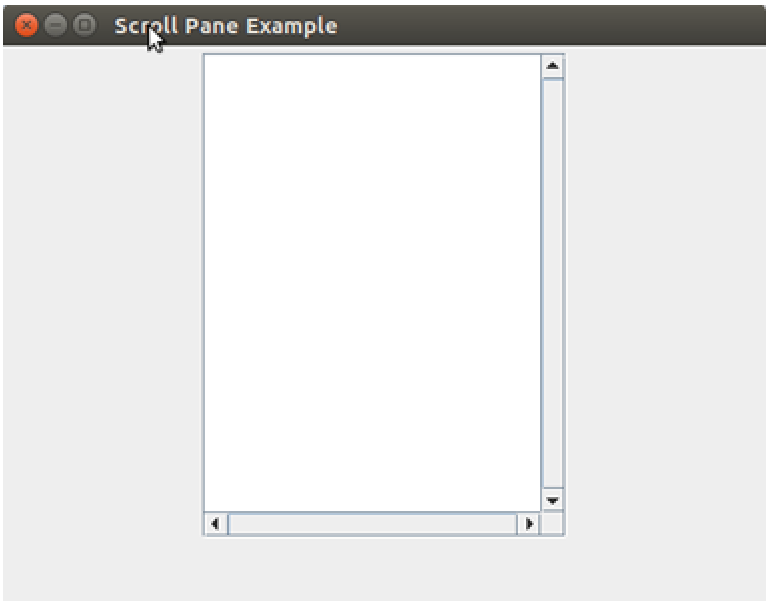
Useful Methods

Modifier	Method	Description
void	setColumnHeaderView(Component)	It sets the column header for the scroll pane.
void	setRowHeaderView(Component)	It sets the row header for the scroll pane.
void	setCorner(String, Component)	It sets or gets the specified corner. The int parameter specifies which corner and must be one of the following constants defined in ScrollPaneConstants: UPPER_LEFT_CORNER, UPPER_RIGHT_CORNER, LOWER_LEFT_CORNER, LOWER_RIGHT_CORNER, LOWER_LEADING_CORNER, LOWER_TRAILING_CORNER, UPPER_LEADING_CORNER, UPPER_TRAILING_CORNER.
Component	getCorner(String)	
void	setViewportView(Component)	Set the scroll pane's client.

JScrollPane Example

```
import java.awt.FlowLayout;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JtextArea;

public class JScrollPaneExample {
    private static final long serialVersionUID = 1L;

    private static void createAndShowGUI() {

        // Create and set up the window.
        final JFrame frame = new JFrame("Scroll Pane Example");

        // Display the window.
        frame.setSize(500, 500);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // set flow layout for the frame
        frame.getContentPane().setLayout(new FlowLayout());

        JTextArea textArea = new JTextArea(20, 20);
        JScrollPane scrollableTextArea = new JScrollPane(textArea);

        scrollableTextArea.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
        scrollableTextArea.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);

        frame.getContentPane().add(scrollableTextArea);
    }
    public static void main(String[] args) {


        javax.swing.SwingUtilities.invokeLater(new Runnable() {

            public void run() {
                createAndShowGUI();
            }
        });
    }
}
```

Output:



Java JTable

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

JTable class declaration

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

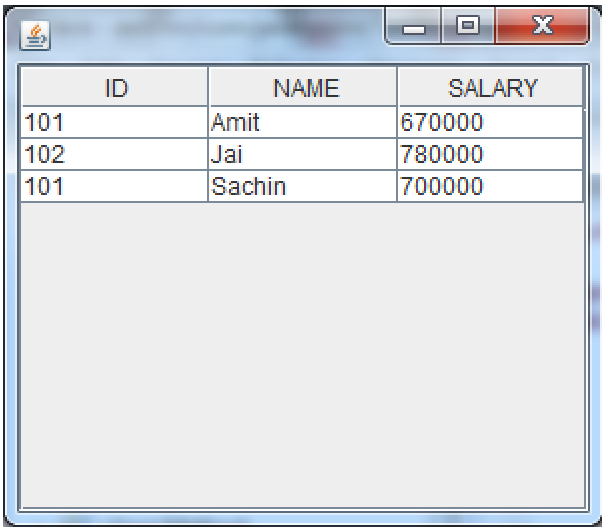
Commonly used Constructors:

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

Java JTable Example

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

Output:

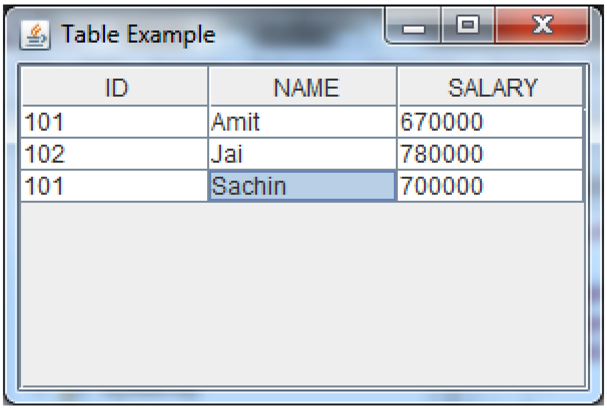


ID	NAME	SALARY
101	Amit	670000
102	Jai	780000
101	Sachin	700000

Java JTable Example with ListSelectionListener

```
import javax.swing.*;
import javax.swing.event.*;
public class TableExample {
    public static void main(String[] a) {
       JFrame f = new JFrame("Table Example");
       String data[][]={ {"101","Amit","670000"},
                                      {"102","Jai","780000"},
                                      {"101","Sachin","700000"}};
               String column[]={"ID","NAME","SALARY"};
               final JTable jt=new JTable(data,column);
       jt.setCellSelectionEnabled(true);
       ListSelectionModel select= jt.getSelectionModel();
       select.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
       select.addListSelectionListener(new ListSelectionListener() {
        public void valueChanged(ListSelectionEvent e) {
         String Data = null;
         int[] row = jt.getSelectedRows();
         int[] columns = jt.getSelectedColumns();
         for (int i = 0; i < row.length; i++) {
          for (int j = 0; j < columns.length; j++) {
           Data = (String) jt.getValueAt(row[i], columns[j]);
          } }
         System.out.println("Table element selected is: " + Data);
        }
       });
       JScrollPane sp=new JScrollPane(jt);
       f.add(sp);
       f.setSize(300, 200);
       f.setVisible(true);
      }
     }
```

Output:



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

If you select an element in column **NAME**, name of the element will be displayed on the console:

```
1. Table element selected is: Sachin
```

IMAGE ICON

Introduction

The class **ImageIcon** is an implementation of the Icon interface that paints Icons from Images.

Class Declaration

Following is the declaration for **javax.swing.ImageIcon** class −

```
public class ImageIcon   extends Object implements Icon,
Serializable, Accessible
```

Field

Following are the fields for **javax.swing.ImageIcon** class −

- protected static Component component
- protected static MediaTracker tracker

Class Constructors

Sr.No.	Constructor & Description
1	**ImageIcon()** Creates an uninitialized image icon.
2	**ImageIcon(byte[] imageData)** Creates an ImageIcon from an array of bytes which were read from an image file containing a supported image format, such as GIF, JPEG, or (as of 1.3) PNG.

3	**ImageIcon(byte[] imageData, String description)** Creates an ImageIcon from an array of bytes which were read from an image file containing a supported image format, such as GIF, JPEG, or (as of 1.3) PNG.
4	**ImageIcon(Image image)** Creates an ImageIcon from an image object.
5	**ImageIcon(Image image, String description)** Creates an ImageIcon from the image.
6	**ImageIcon(String filename)** Creates an ImageIcon from the specified file.
7	**ImageIcon(String filename, String description)** Creates an ImageIcon from the specified file.
8	**ImageIcon(URL location)** Creates an ImageIcon from the specified URL.
9	**ImageIcon(URL location, String description)** Creates an ImageIcon from the specified URL.

Class Methods

Sr.No.	Method & Description
1	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this ImageIcon.
2	**String getDescription()** Gets the description of the image.
3	**int getIconHeight()** Gets the height of the icon.

4	**int getIconWidth()** Gets the width of the icon.
5	**Image getImage()** Returns this icon's Image.
6	**int getImageLoadStatus()** Returns the status of the image loading operation.
7	**ImageObserver getImageObserver()** Returns the image observer for the image.
8	**protected void loadImage(Image image)** Loads the image, returning only when the image is loaded.
9	**void paintIcon(Component c, Graphics g, int x, int y)** Paints the icon.
10	**void setDescription(String description)** Sets the description of the image.
11	**void setImage(Image image)** Sets the image displayed by this icon.
12	**void setImageObserver(ImageObserver observer)** Sets the image observer for the image.
13	**String toString()** Returns a string representation of this image.

Methods Inherited

This class inherits methods from the following classes −

- java.lang.Object

ImageIcon Example

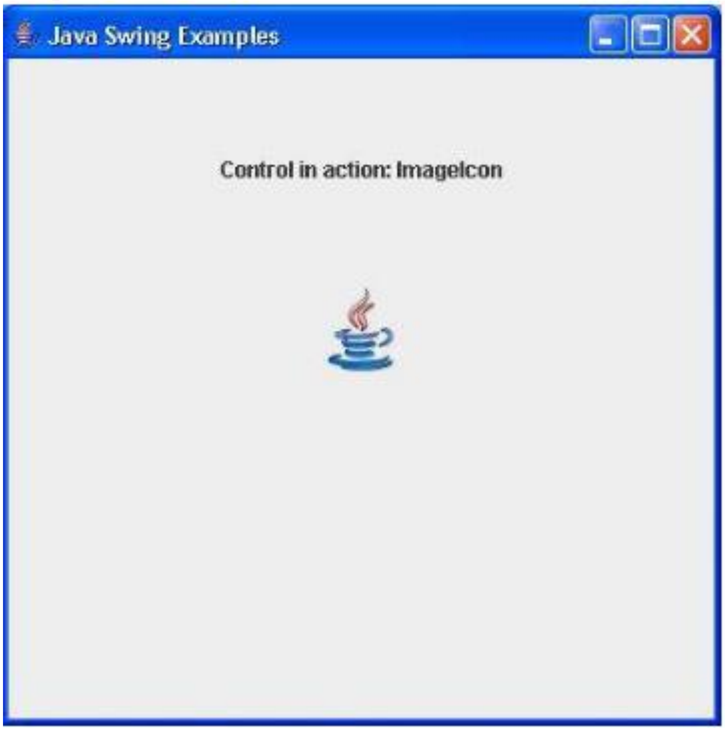
Create the following Java program using any editor of your choice in say **D:/ > SWING > com > tutorialspoint > gui >**

SwingControlDemo.java

```java
package com.tutorialspoint.gui;

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

public class SwingControlDemo {
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingControlDemo(){
      prepareGUI();
   }
   public static void main(String[] args){
      SwingControlDemo  swingControlDemo = new 
SwingControlDemo();
      swingControlDemo.showImageIconDemo();
   }
   private void prepareGUI(){
      mainFrame = new JFrame("Java Swing Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));

      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
      headerLabel = new JLabel("", JLabel.CENTER);
      statusLabel = new JLabel("",JLabel.CENTER);
      statusLabel.setSize(350,100);

      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }
   // Returns an ImageIcon, or null if the path was invalid.
   private static ImageIcon createImageIcon(String path,
      String description) {
      java.net.URL imgURL = 
SwingControlDemo.class.getResource(path);
```

```
      if (imgURL != null) {
         return new ImageIcon(imgURL, description);
      } else {
         System.err.println("Couldn't find file: " + path);
         return null;
      }
   }
   private void showImageIconDemo(){
      headerLabel.setText("Control in action: ImageIcon");
      ImageIcon icon =
createImageIcon("/resources/java_icon.png","Java");

      JLabel commentlabel = new JLabel("", icon,JLabel.CENTER);
      controlPanel.add(commentlabel);
      mainFrame.setVisible(true);
   }
}
```

Output:



Control in action: ImageIcon

EVENT HANDLING

What is an Event?

Change in the state of an object is known as event i.e. event describes the change in state of source. Events are generated as result of user interaction with the graphical user interface components. For example, clicking on a button, moving the mouse, entering a character through keyboard,selecting an item from list, scrolling the page are the activities that causes an event to happen.

Types of Event

The events can be broadly classified into two categories:

- **Foreground Events** - Those events which require the direct interaction of user.They are generated as consequences of a person interacting with the graphical components in Graphical User Interface. For example, clicking on a button, moving the mouse, entering a character through keyboard,selecting an item from list, scrolling the page etc.
- **Background Events** - Those events that require the interaction of end user are known as background events. Operating system interrupts, hardware or software failure, timer expires, an operation completion are the example of background events.

What is Event Handling?

Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism have the code which is known as event handler that is executed when an event occurs. Java Uses the Delegation Event Model to handle the events. This model defines the standard mechanism to generate and handle the events.Let's have a brief introduction to this model.

The Delegation Event Model has the following key participants namely:

- **Source** - The source is an object on which event occurs. Source is responsible for providing information of the occurred event to it's handler. Java provide as with classes for source object.
- **Listener** - It is also known as event handler.Listener is responsible for generating response to an event. From java implementation point of view the listener is also an object. Listener waits until it receives an event. Once the event is received , the listener process the event an then returns.

The benefit of this approach is that the user interface logic is completely separated from the logic that generates the event. The user interface element is able to delegate the processing of an event to the separate piece of code. In this model ,Listener needs to be registered with the source object so that the listener can receive the event notification. This is an efficient way of handling the event because the event notifications are sent only to those listener that want to receive them.

Steps involved in event handling

- The User clicks the button and the event is generated.
- Now the object of concerned event class is created automatically and information about the source and the event get populated with in same object.

- Event object is forwarded to the method of registered listener class.
- the method is now get executed and returns.

Points to remember about listener

- In order to design a listener class we have to develop some listener interfaces.These Listener interfaces forecast some public abstract callback methods which must be implemented by the listener class.
- If you do not implement the any if the predefined interfaces then your class can not act as a listener class for a source object.

Callback Methods

These are the methods that are provided by API provider and are defined by the application programmer and invoked by the application developer. Here the callback methods represents an event method. In response to an event java jre will fire callback method. All such callback methods are provided in listener interfaces.

If a component wants some listener will listen to it's events the the source must register itself to the listener.

Event and Listener (Java Event Handling)

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

Java Event classes and Listener interfaces

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener

ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

Steps to perform Event Handling

Following steps are required to perform event handling:

1. Register the component with the Listener

Registration Methods

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

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

Java Event Handling Code

We can put the event handling code into one of the following places:

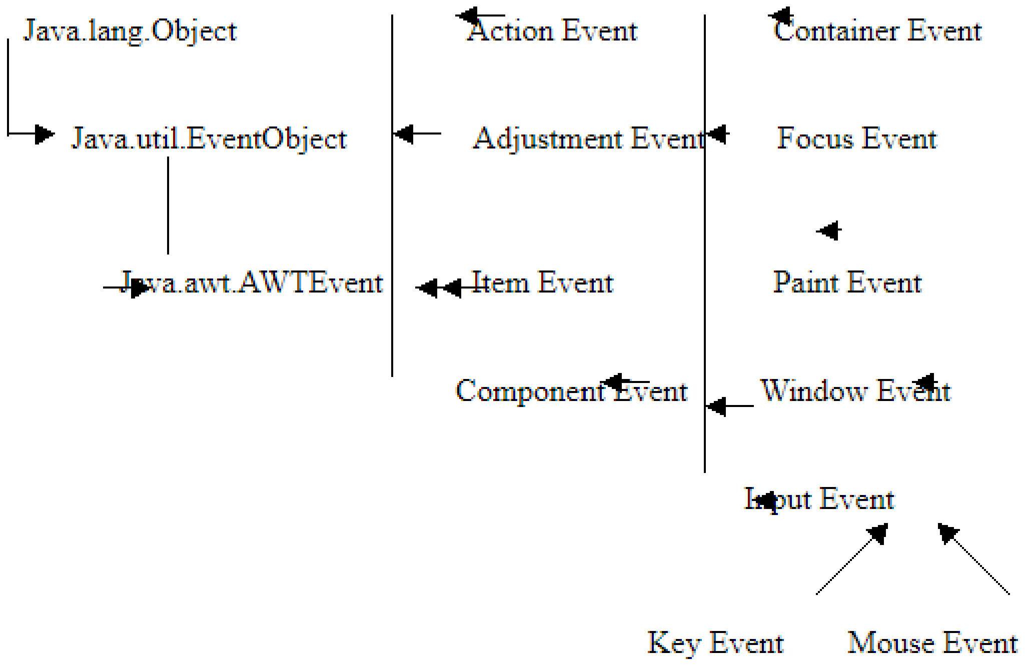
1. Within class
2. Other class
3. Anonymous class

EVENT HANDLING MODEL

- GUIs are event driven (i.e. they generate events when the user of the program interacts with the GUI). Some common interactions are moving the mouse, clicking the mouse, clicking a button, typing in a text field, selecting an item from a menu, closing a window etc.

- When a user interaction occurs, an event is sent to the program. GUI event information is stored in an object of a class that extends AWTEvent.

- AWTEvents are used with both AWT components and Swing components.



-

EVENT HANDLING MECHANISM:
There are three parts to the event handling mechanism

- **Event Source**
- **Event Object**
- **Event Listener**

- **The event source** is the particular GUI component with which the user interacts. The event object encapsulates information about the event that occurred. This information includes a reference to the event source and any event-specific information that may be required by the event listener to handle the event.

- **The event listener** is an object that is notified by the event source when and event occurs. The event listener receives an event object when it is notified of the event, then uses the object to respond to the event. The event source is required to provide methods that enable listeners to be registered and unregistered. The event source also is required to maintain a list of its registered listeners and be able to notify its listeners when an event occurs.

- The programmer must perform **two key tasks** to process a graphical user interface event in a program. 1) Register an event listener for the GUI component that is expected to generate the event. 2) Implement and event handling method (or set of event-handling methods), called **event handlers.**

- An event listener for a GUI event is an object of a class that implements one or more of the event-listener interfaces from package java.awt.evetn and package javax.swing.event. Many of the event-listener are common to both packages. The main event that all listeners extend from is the **java.util.EventListener** .
  Additional event listeners can be found in the **javax.swing.event** package.

- An event listener object listens for specific types of events generated by event sources in a program. An event handler is a method that is called in response to a particular type of event. Each event interface specifies one or more event-handling methods that **must be** defined in the class that implements the event-listener interface. REMEMBER that interfaces define **abstract** methods. Any class that implements an interface must define all the methods of that interface.

- When an event occurs, the GUI component with which the user interacted notifies its registered listeners by calling each listeners appropriate event handling method. For example, when the user presses the Enter Key in a JTextField, the registered listener’s actionPerformed method is called.

```
// Fig. 12.7: TextFieldTest.java
// Demonstrating the JTextField class.

// Java core packages
import java.awt.*;
import java.awt.event.*;

// Java extension packages
import javax.swing.*;

public class TextFieldTest extends JFrame {
   private JTextField textField1, textField2, textField3;
   private JPasswordField passwordField;

   // set up GUI
   public TextFieldTest()
   {
      super( "Testing JTextField and JPasswordField" );

      Container container = getContentPane();
      container.setLayout( new FlowLayout() );

      // construct textfield with default sizing
      textField1 = new JTextField( 10 );
      container.add( textField1 );

      // construct textfield with default text
      textField2 = new JTextField( "Enter text here" );
      container.add( textField2 );

      // construct textfield with default text and
      // 20 visible elements and no event handler
      textField3 = new JTextField( "Uneditable text field", 20 );
      textField3.setEditable( false );
      container.add( textField3 );

      // construct textfield with default text
      passwordField = new JPasswordField( "Hidden text" );
      container.add( passwordField );

      // register event handlers
      TextFieldHandler handler = new TextFieldHandler();
      textField1.addActionListener( handler );
      textField2.addActionListener( handler );
      textField3.addActionListener( handler );
      passwordField.addActionListener( handler );

      setSize( 325, 100 );
      setVisible( true );
   }

   // execute application
   public static void main( String args[] )
   {
      TextFieldTest application = new TextFieldTest();

      application.addWindowListener(new WindowAdapter()
                                { public void windowClosing( WindowEvent
event )
                                    {
                                          System.exit( 0 );
                                    }

                              }
                  );
   }

   // private inner class for event handling
   private class TextFieldHandler implements ActionListener {

      // process text field events
      public void actionPerformed( ActionEvent event )
      {
         String string = "";

         // user pressed Enter in JTextField textField1
         if ( event.getSource() == textField1 )
            string = "textField1: " + event.getActionCommand();

         // user pressed Enter in JTextField textField2
         else if ( event.getSource() == textField2 )
            string = "textField2: " + event.getActionCommand();

         // user pressed Enter in JTextField textField3
         else if ( event.getSource() == textField3 )
            string = "textField3: " + event.getActionCommand();

         // user pressed Enter in JTextField passwordField
         else if ( event.getSource() == passwordField ) {
            JPasswordField pwd =
               ( JPasswordField ) event.getSource();
            string = "passwordField: " +
                new String( passwordField.getPassword() );
         }

         JOptionPane.showMessageDialog( null, string );
      }

   }  // end private inner class TextFieldHandler

}  // end class TextFieldTest
```

Module 5.2

Java Database Connectivity

Java Database Connectivity (JDBC) is an application programming interface (API) which allows the programmer to connect and interact with databases. It provides methods to query and update data in the database through update statements like SQL's CREATE, UPDATE, DELETE and INSERT and query statements such as SELECT. Additionally, JDBC can run stored procedures.

Why Should We Use JDBC

Before JDBC, ODBC API was the database API to connect and execute the query with the database. But, ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

We can use JDBC API to handle database using Java program and can perform the following activities:

1. Connect to the database
2. Execute queries and update statements to the database
3. Retrieve the result received from the database.

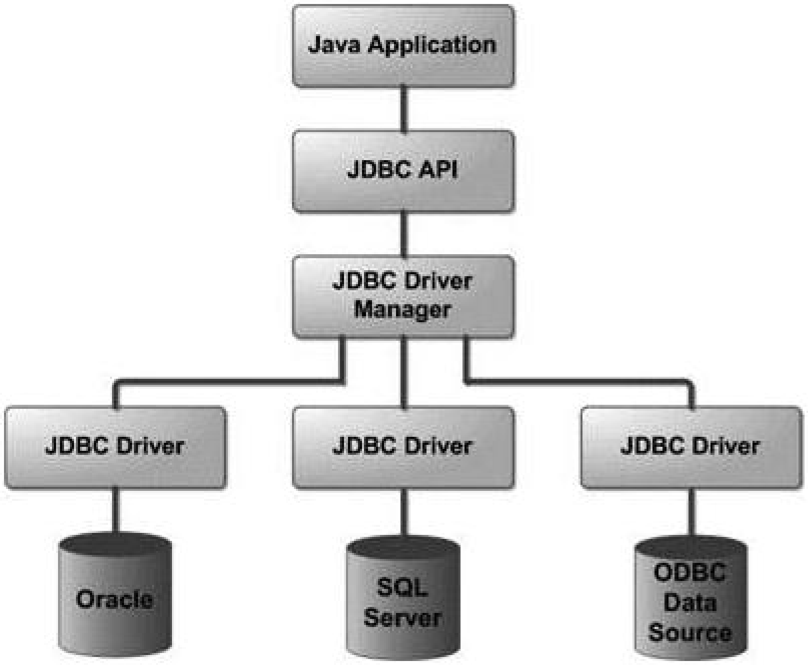
JDBC Architecture

The JDBC API supports both two-tier and three-tier processing models for database access but in general, JDBC Architecture consists of two layers −

- **JDBC API:** This provides the application-to-JDBC Manager connection.
- **JDBC Driver API:** This supports the JDBC Manager-to-Driver Connection.

The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases.

The JDBC driver manager ensures that the correct driver is used to access each data source. Following is the architectural diagram, which shows the location of the driver manager with respect to the JDBC drivers and the Java application –



Common JDBC Components

The JDBC API provides the following interfaces and classes −

- **DriverManager:** This class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol. The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.

- **Driver:** This interface handles the communications with the database server. You will interact directly with Driver objects very rarely. Instead, you use DriverManager objects, which manages objects of this type.

- **Connection:** This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.

- **Statement:** We use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.

- **ResultSet:** These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.

- **SQLException:** This class handles any errors that occur in a database application.

JDBC Packages

The JDBC API is contained in two packages. The first package is called java.sql and contains core JDBC interfaces of the JDBC API. These include the JDBC interfaces that provide the basics for connecting to the DBMS and interacting with data stored in the DBMS. java.sql is part of the J2SE. The other package that contains the JDBC API is javax.sql, which extends java.sql.

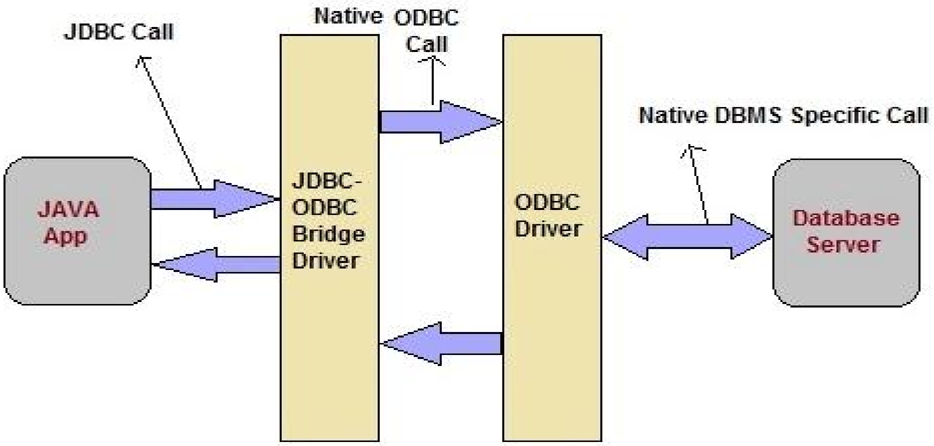
JDBC Drivers

JDBC Driver is required to process SQL requests and generate result. The following are the different types of driver available in JDBC.

- **Type-1 Driver** or **JDBC-ODBC bridge**
- **Type-2 Driver** or **Native API Partly Java Driver**
- **Type-3 Driver** or **Network Protocol Driver**
- **Type-4 Driver** or **Thin Driver**

Type-1 Driver or JDBC-ODBC bridge

Type-1 Driver act as a bridge between JDBC and other database connectivity mechanism(ODBC). This driver converts JDBC calls into ODBC calls and redirects the request to the ODBC driver.



Advantages

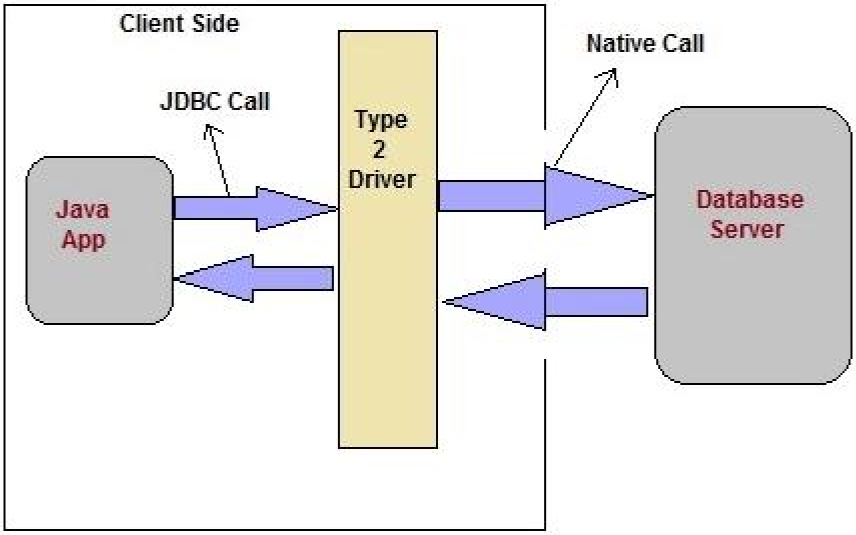
- Easy to use
- Allow easy connectivity to all database supported by the ODBC Driver.

Disadvantages

- Since the Bridge driver is not written fully in Java, Type 1 drivers are not portable
- A performance issue is seen as a JDBC call goes through the bridge to the ODBC driver, then to the database, and this applies even in the reverse process. They are the slowest of all driver types.
- The client system requires the ODBC Installation to use the driver.
- Not good for the Web.

Type-2 Driver or Native API Partly Java Driver

This type of driver make use of Java Native Interface(JNI) call on database specific native client API. These native client API are usually written in C and C++.



Advantages

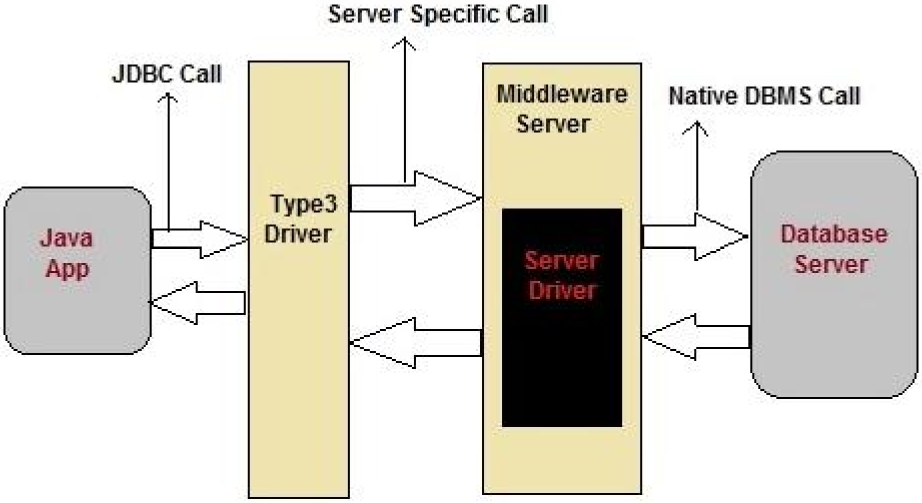
- faster as compared to **Type-1 Driver**
- Contains additional features.

Disadvantages

- Native API must be installed in the Client System and hence type 2 drivers cannot be used for the Internet.
- Like Type 1 drivers, it’s not written in Java Language which forms a portability issue.
- If we change the Database we have to change the native api as it is specific to a database

Type-3 Driver or Network Protocol Driver

This driver translate the JDBC calls into a database server independent and Middleware server-specific calls. Middleware server further translate JDBC calls into database specific calls.



Advantages

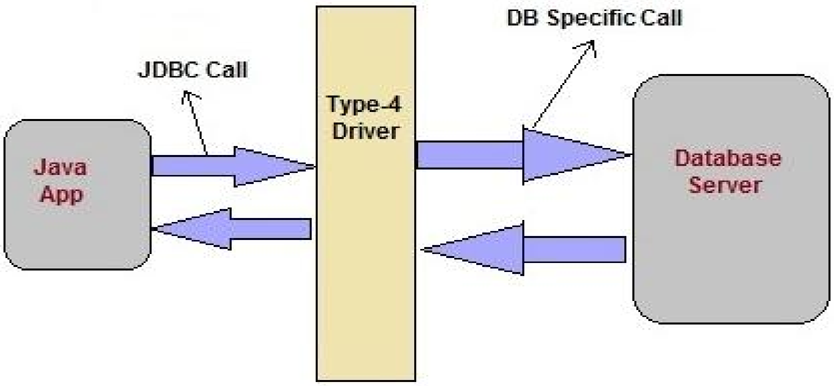
- This driver is server-based, so there is no need for any vendor database library to be present on client machines.
- This driver is fully written in Java and hence Portable. It is suitable for the web.
- This driver is very flexible allows access to multiple databases using one driver.
- They are the most efficient amongst all driver types

Disadvantage

- It requires another server application to install and maintain.

Type-4 Driver or Thin Driver

This is Driver called Pure Java Driver because. This driver interact directly with database. It does not require any native database library, that is why it is also known as Thin Driver.

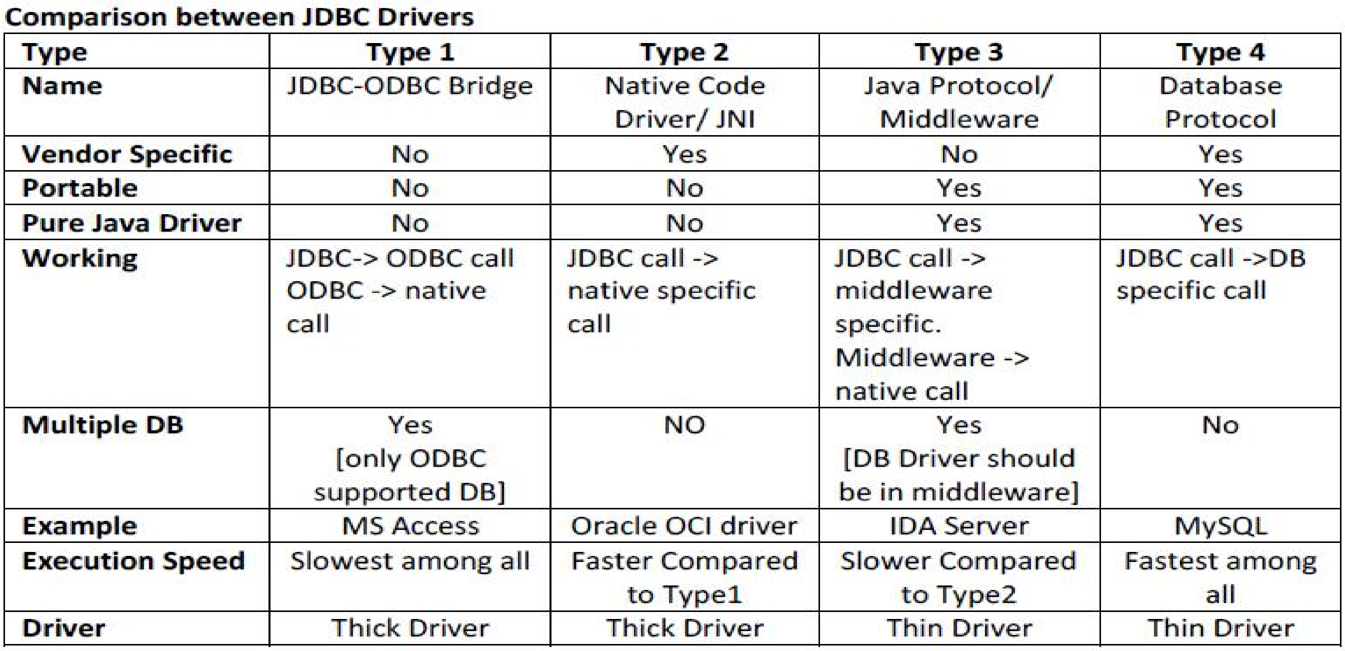


Advantages

- The major benefit of using a type 4 jdbc drivers are that they are completely written in Java to achieve platform independence
- It is most suitable for the web.
- You don’t need to install special software on the client or server.
- Further, these drivers can be downloaded dynamically

Disadvantage

- Slow due to increase number of network call.

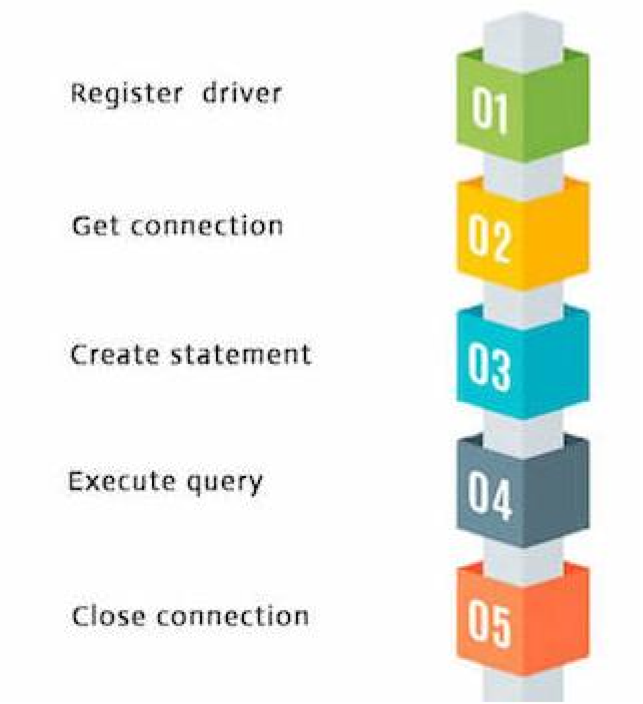


Comparison between JDBC Drivers

Type	Type 1	Type 2	Type 3	Type 4
Name	JDBC-ODBC Bridge	Native Code Driver/ JNI	Java Protocol/ Middleware	Database Protocol
Vendor Specific	No	Yes	No	Yes
Portable	No	No	Yes	Yes
Pure Java Driver	No	No	Yes	Yes
Working	JDBC-> ODBC call ODBC -> native call	JDBC call -> native specific call	JDBC call -> middleware specific. Middleware -> native call	JDBC call ->DB specific call
Multiple DB	Yes [only ODBC supported DB]	NO	Yes [DB Driver should be in middleware]	No
Example	MS Access	Oracle OCI driver	IDA Server	MySQL
Execution Speed	Slowest among all	Faster Compared to Type1	Slower Compared to Type2	Fastest among all
Driver	Thick Driver	Thick Driver	Thin Driver	Thin Driver

Java Database Connectivity with 5 Steps / Overview of the JDBC Process

There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:



1) Register the driver class

The **forName()** method of Class class is used to register the driver class. This method is used to dynamically load the driver class

Syntax of forName() method

```
public static void forName(String className)throws ClassNotFoundException
```

Example to register the MySqlDriver class

```
Class.forName("com.mysql.jdbc.Driver");
```

2) Create the connection object

The **getConnection()** method of DriverManager class is used to establish connection with the database.

Syntax of getConnection() method

```
public static Connection getConnection(String url,String name,String password)
```

```
throws SQLException
```

Example to establish connection with the MYSql database

```
        Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3308/java","root","");
```

3) Create the Statement object

The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

Syntax of createStatement() method

```
        public Statement createStatement()throws SQLException
```

Example to create the statement object

```
        Statement stmt=con.createStatement();
```

4) Execute the query

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

Syntax of executeQuery() method

```
        public ResultSet executeQuery(String sql)throws SQLException
```

Example to execute query

```
        ResultSet rs=stmt.executeQuery("select * from author");

        while(rs.next()){
        System.out.println(rs.getString(1)+" "+rs.getString(2) +" "+rs.getInt(3));
        }
```

5) Close the connection object

By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

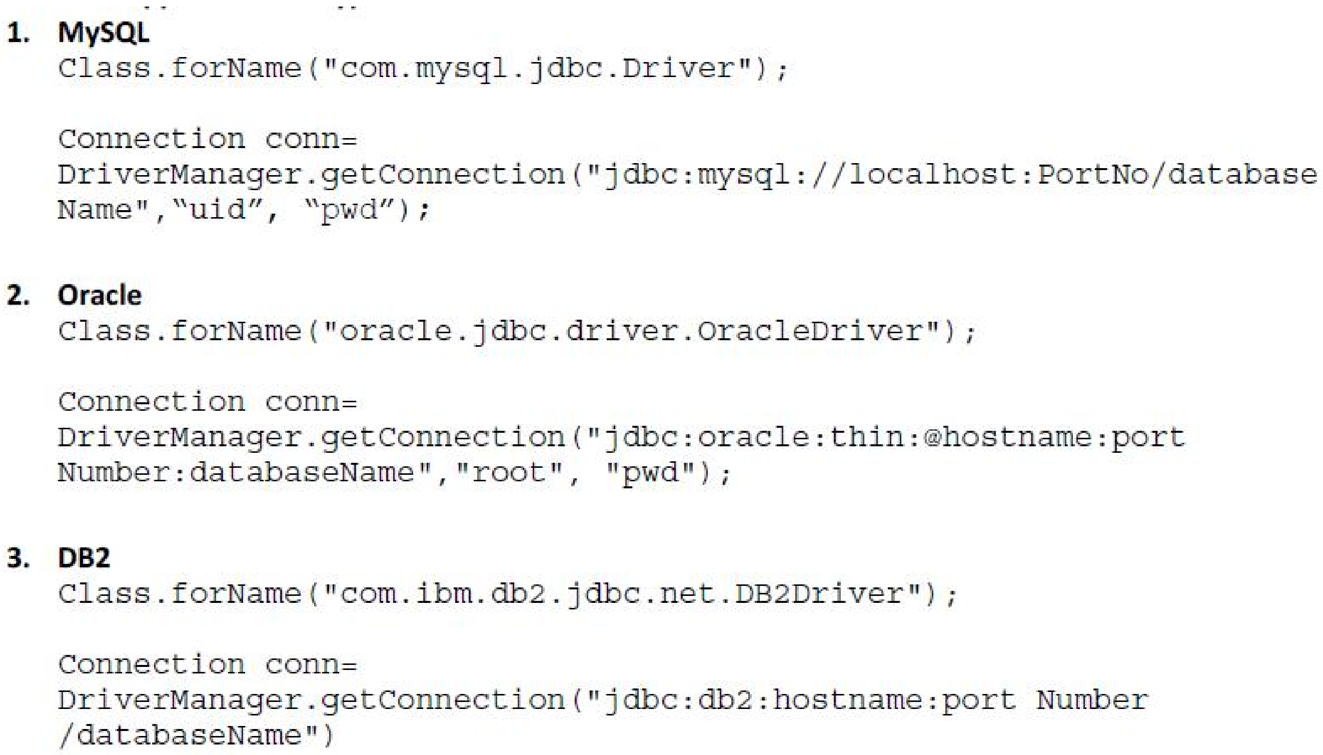
Syntax of close() method

```
public void close()throws SQLException
```

Example to close connection

```
        con.close();
```

Code snippets for each type of JDBC connection



1. **MySQL**

```
Class.forName("com.mysql.jdbc.Driver");

Connection conn=
DriverManager.getConnection("jdbc:mysql://localhost:PortNo/database
Name","uid", "pwd");
```

2. **Oracle**

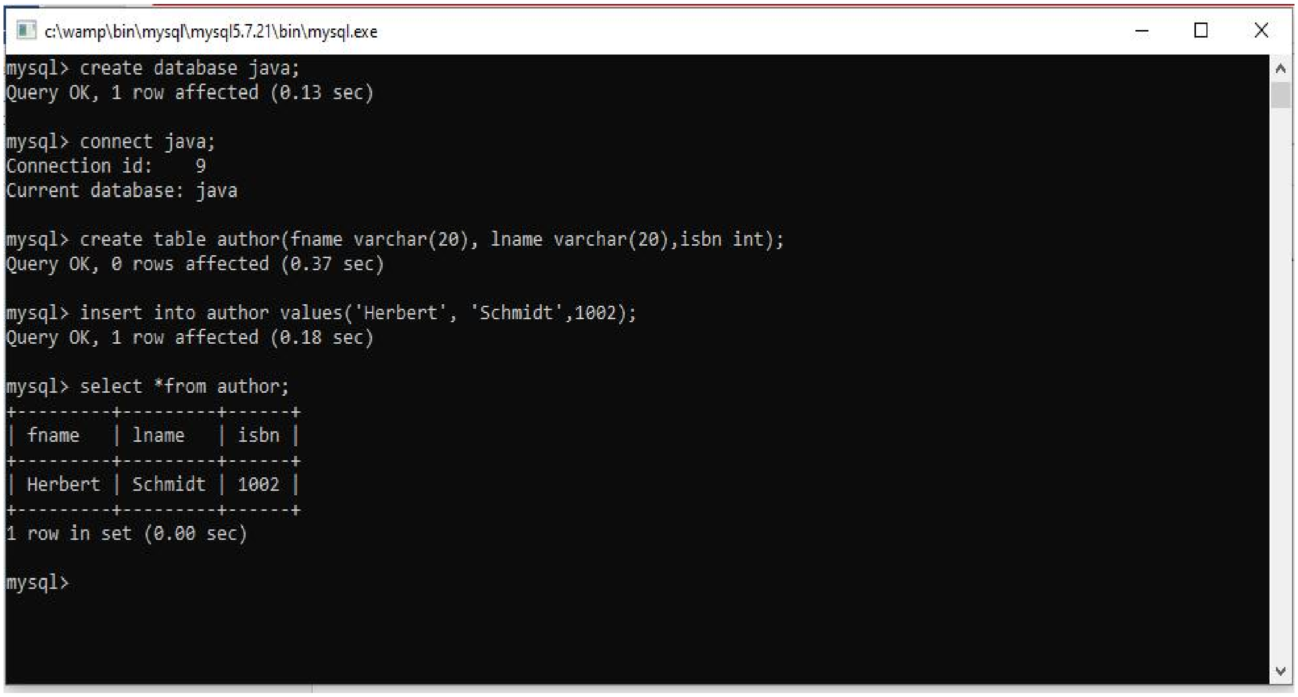
```
Class.forName("oracle.jdbc.driver.OracleDriver");

Connection conn=
DriverManager.getConnection("jdbc:oracle:thin:@hostname:port
Number:databaseName","root", "pwd");
```

3. **DB2**

```
Class.forName("com.ibm.db2.jdbc.net.DB2Driver");

Connection conn=
DriverManager.getConnection("jdbc:db2:hostname:port Number
/databaseName")
```

Working with MYSql

To create a table author under the database named java. The author table has 3 attributes namely first name, last name of the author and isbn of the book and table is populated with the data.



```
mysql> create database java;
Query OK, 1 row affected (0.13 sec)

mysql> connect java;
Connection id:    9
Current database: java

mysql> create table author(fname varchar(20), lname varchar(20),isbn int);
Query OK, 0 rows affected (0.37 sec)

mysql> insert into author values('Herbert', 'Schmidt',1002);
Query OK, 1 row affected (0.18 sec)

mysql> select *from author;
+---------+---------+------+
| fname   | lname   | isbn |
+---------+---------+------+
| Herbert | Schmidt | 1002 |
+---------+---------+------+
1 row in set (0.00 sec)

mysql>
```

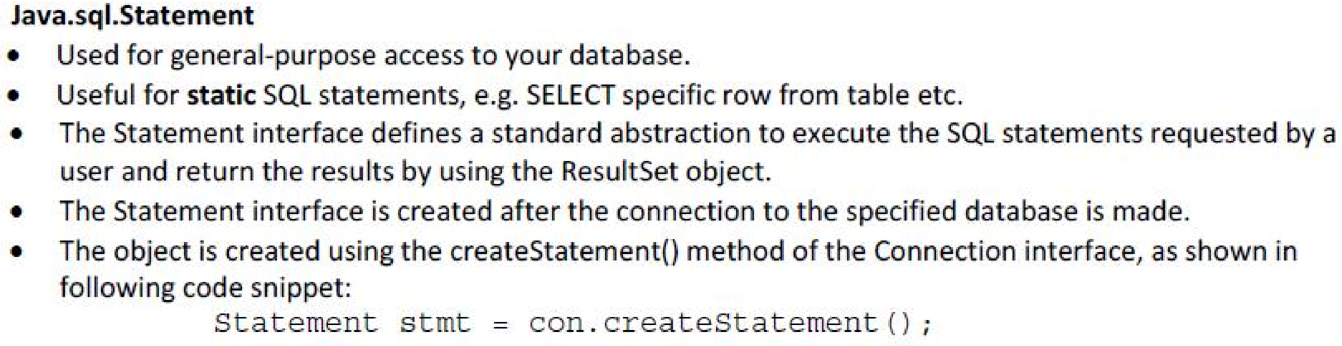
Different types of statements in JDBC

- **Statement**: Use this for general-purpose access to your database. Useful when you are using static SQL statements at runtime. The Statement interface cannot accept parameters.
- **PreparedStatement**: Use this when you plan to use the SQL statements many times. The PreparedStatement interface accepts input parameters at runtime.
- **CallableStatement:** Use this when you want to access the database stored procedures. The CallableStatement interface can also accept runtime input parameters.

Executing the Statement object

Statement

The Statement interface represents the static SQL statement. It helps you to create a general purpose SQL statements using Java.



Java.sql.Statement

- Used for general-purpose access to your database.
- Useful for **static** SQL statements, e.g. SELECT specific row from table etc.
- The Statement interface defines a standard abstraction to execute the SQL statements requested by a user and return the results by using the ResultSet object.
- The Statement interface is created after the connection to the specified database is made.
- The object is created using the createStatement() method of the Connection interface, as shown in following code snippet:

```
Statement stmt = con.createStatement();
```

Once you have created the statement object you can execute it using one of the execute methods namely, execute(), executeUpdate() and, executeQuery().

- **boolean execute (String SQL)**: Returns a boolean value of true if a ResultSet object can be retrieved; otherwise, it returns false. Use this method to execute SQL DDL statements or when you need to use truly dynamic SQL.
- **int executeUpdate (String SQL)**: Returns the number of rows affected by the execution of the SQL statement. Use this method to execute SQL statements for which you expect to get a number of rows affected - for example, an INSERT, UPDATE, or DELETE statement.
- **ResultSet executeQuery (String SQL)**: Returns a ResultSet object. Use this method when you expect to get a result set, as you would with a SELECT statement.

// A JDBC Application to create a student table

```
import java.sql.*;

public class JDBC1
{
   public static void main(String[] args) throws Exception
   {
Class.forName("com.mysql.jdbc.Driver");
Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/java","root","");
Statement stmt = con.createStatement();
      String query ="create table STUDENT(ST_NAME varchar(20),ST_USN int, SEM int)";
      boolean b=stmt.execute(query);
       System.out.println("TABLE Created  "+b);
       con.close();


   }
  }
```

OUTPUT

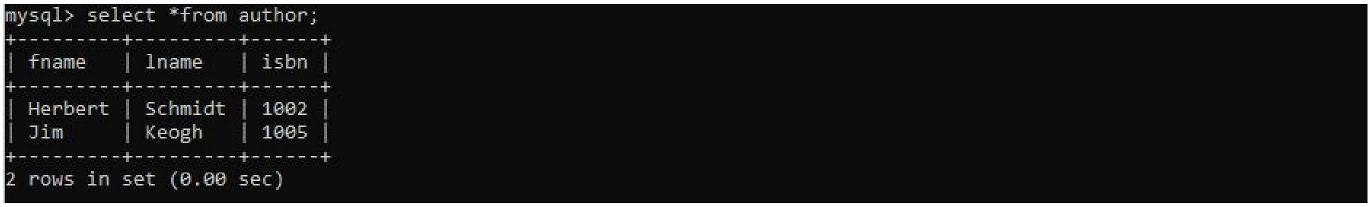
```
TABLE Created  false
```

// A JDBC Application to insert a record into the table

```
import java.sql.*;

public class JDBC1
{
   public static void main(String[] args) throws Exception
   {
      Class.forName("com.mysql.jdbc.Driver");
      Connection con =
      DriverManager.getConnection("jdbc:mysql://localhost:3306/student","root","");
      Statement stmt = con.createStatement();
      String query ="insert into fourth values(101,’RAM’)";
      int a=stmt.executeUpdate(query);
        System.out.println("No of Record Inserted   "+a);
      con.close();

   }
  }
```

OUTPUT

```
No of Record Inserted   1
```



```
mysql> select *from author;
+---------+---------+------+
| fname   | lname   | isbn |
+---------+---------+------+
| Herbert | Schmidt | 1002 |
| Jim     | Keogh   | 1005 |
+---------+---------+------+
2 rows in set (0.00 sec)
```

// A JDBC Application to update records of the table

```
import java.sql.*;

public class JDBC1
{
   public static void main(String[] args) throws Exception
   {
Class.forName("com.mysql.jdbc.Driver");
Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/java","root","");
Statement stmt = con.createStatement();
      String query ="update author set isbn=2000 where fname='Jim'";
      int a=stmt.executeUpdate(query);
        System.out.println("No of Record Inserted   "+a);
       con.close();

   }
  }
```

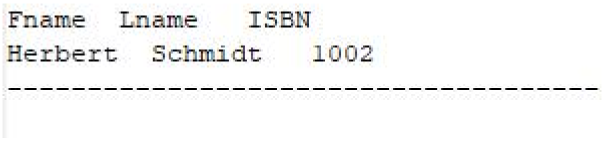
// A JDBC Application to delete records from the table

```
import java.sql.*;

public class JDBC1
{
   public static void main(String[] args) throws Exception
   {
Class.forName("com.mysql.jdbc.Driver");
Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3308/java","root","");
Statement stmt = con.createStatement();
      String query ="delete from author where fname='Jim'";
      int a=stmt.executeUpdate(query);
        System.out.println("No of Record Inserted   "+a);
       con.close();

   }
  }
```

// A JDBC Application to display the contents of the table

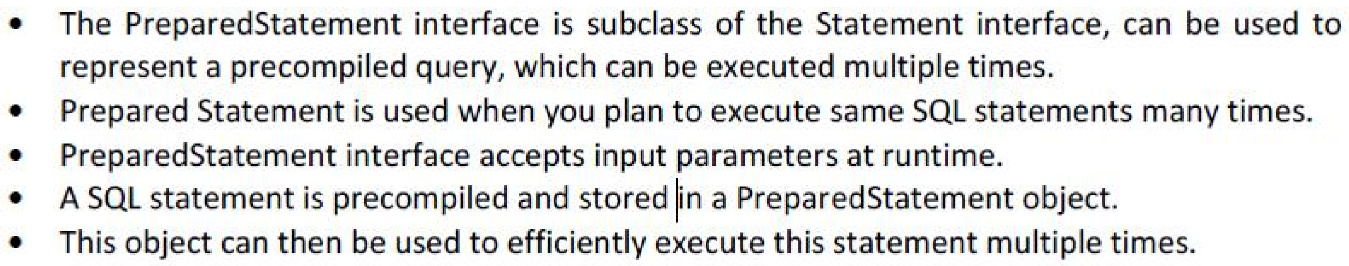
```
import java.sql.*;

public class JDBC1
{
   public static void main(String[] args) throws Exception
   {
Class.forName("com.mysql.jdbc.Driver");
Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3308/java","root","");
Statement stmt = con.createStatement();
      String query ="SELECT fname,lname,isbn from author";
      ResultSet rs = stmt.executeQuery(query);
      System.out.println("Fname  Lname   ISBN");
      while (rs.next())
     {
         String fname = rs.getString("fname");
         String lname = rs.getString("lname");
         int isbn = rs.getInt("isbn");
         System.out.println(fname + "  " + lname+"   "+isbn);
         System.out.println("----------------------------------------------");
      }
      con.close();
      System.out.println();
      System.out.println();
   }
  }
```

OUTPUT



```
Fname  Lname   ISBN
Herbert  Schmidt   1002
-------------------------------------
```

Java PreparedStatement Interface



- The PreparedStatement interface is subclass of the Statement interface, can be used to represent a precompiled query, which can be executed multiple times.
- Prepared Statement is used when you plan to execute same SQL statements many times.
- PreparedStatement interface accepts input parameters at runtime.
- A SQL statement is precompiled and stored in a PreparedStatement object.
- This object can then be used to efficiently execute this statement multiple times.

Prepared statements offer better performance, as they are pre-compiled. Advantages of Prepared Statement in Java JDBC. benefit of using Prepared Statement is it prevents from SQL Injection. PreparedStatement is fast and gives better performance.

Prepared statements are more secure because they use bind variables, which can prevent SQL injection attack.

The most common type of SQL injection attack is SQL manipulation. The attacker attempts to modify the SQL statement by adding elements to the WHERE clause or extending the SQL with the set operators like UNION, INTERSECT etc.

SQL Injection is code injection technique where SQL is injected by user (as part of user input) into the back end query. Injected SQL data alters the purpose of original query and upon execution can gives harmful result.

A SQL injection attack is very dangerous and attacker can,

1. Read sensitive data from the database.

2. Update database data (Insert/Update/Delete).

3. Slowdown the complete Database system etc

Let us look at the following SQL

```
SELECT * FROM employee where ename=’john’ AND password=’xyfdsw’;
```

The attacker can manipulate the SQL as follows

```
SELECT * FROM employee where ename=’john’ AND password=’xyfdsw’ or 1 = 1;
```

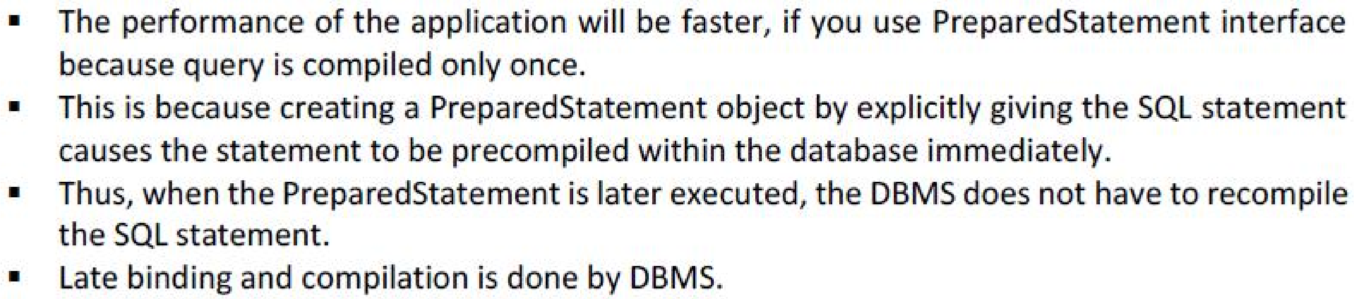
The above “WHERE” clause is always true because of the operator precedence. The PreparedStatement can prevent this by using bind variables:

```
String strSQL = SELECT * FROM employee where ename=? AND password=?);

PreparedStatement pstmt = myConnection.prepareStatement(strSQL);

pstmt.setString(1,”john”);

pstmt.setString(2, “xyfdsw”);

pstmt.execute();
```

// A JDBC Application to insert record into the table using PreparedStatement

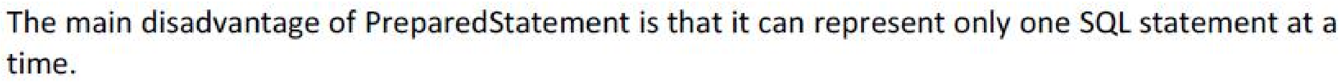
```
import java.sql.*;

public class JDBC1
{
   public static void main(String[] args) throws Exception
   {
Class.forName("com.mysql.jdbc.Driver");
Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/java","root","");

//Creating a Prepared Statement
String query="INSERT INTO student(ST_NAME, ST_USN, SEM)VALUES(?, ?, ?)";
PreparedStatement pstmt = con.prepareStatement(query);
pstmt.setString(1, "Amit");
pstmt.setInt(2, 300);
pstmt.setInt(3, 4);
pstmt.execute();
System.out.println("Record Inserted");

       con.close();

   }
  }
```

Advantages of PreparedStatement Interface



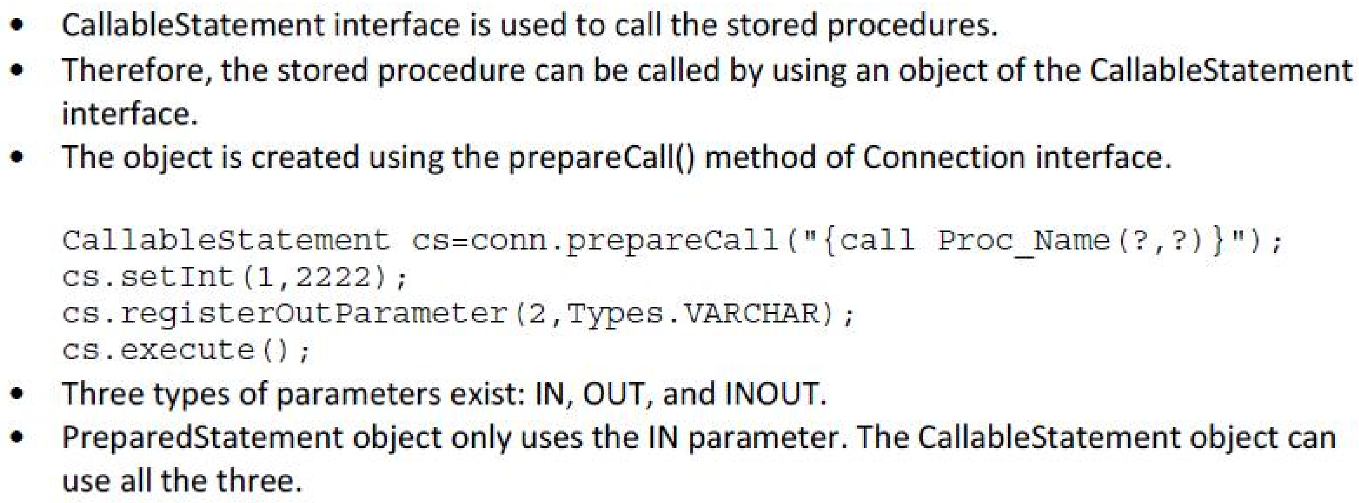
- The performance of the application will be faster, if you use PreparedStatement interface because query is compiled only once.
- This is because creating a PreparedStatement object by explicitly giving the SQL statement causes the statement to be precompiled within the database immediately.
- Thus, when the PreparedStatement is later executed, the DBMS does not have to recompile the SQL statement.
- Late binding and compilation is done by DBMS.

Disdvantage of PreparedStatement Interface



The main disadvantage of PreparedStatement is that it can represent only one SQL statement at a time.

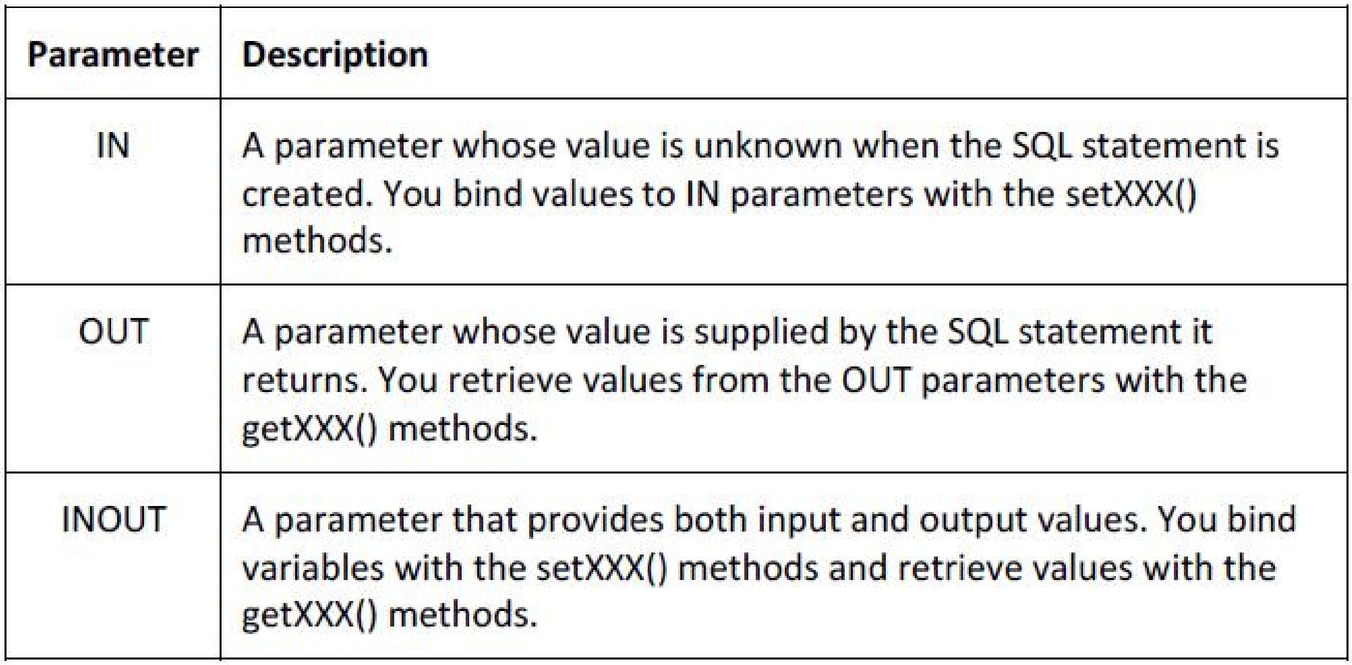
Java CallableStatement Interface



- CallableStatement interface is used to call the stored procedures.
- Therefore, the stored procedure can be called by using an object of the CallableStatement interface.
- The object is created using the prepareCall() method of Connection interface.

```
CallableStatement cs=conn.prepareCall("{call Proc_Name(?,?)}");
cs.setInt(1,2222);
cs.registerOutParameter(2,Types.VARCHAR);
cs.execute();
```

- Three types of parameters exist: IN, OUT, and INOUT.
- PreparedStatement object only uses the IN parameter. The CallableStatement object can use all the three.



Parameter	Description
IN	A parameter whose value is unknown when the SQL statement is created. You bind values to IN parameters with the setXXX() methods.
OUT	A parameter whose value is supplied by the SQL statement it returns. You retrieve values from the OUT parameters with the getXXX() methods.
INOUT	A parameter that provides both input and output values. You bind variables with the setXXX() methods and retrieve values with the getXXX() methods.

Example of CallableStatement

Writa a Callable Statement program to retrieve branch of the student using {getBranch() procedure} from given enrollment number. Also write code for Stored Procedure

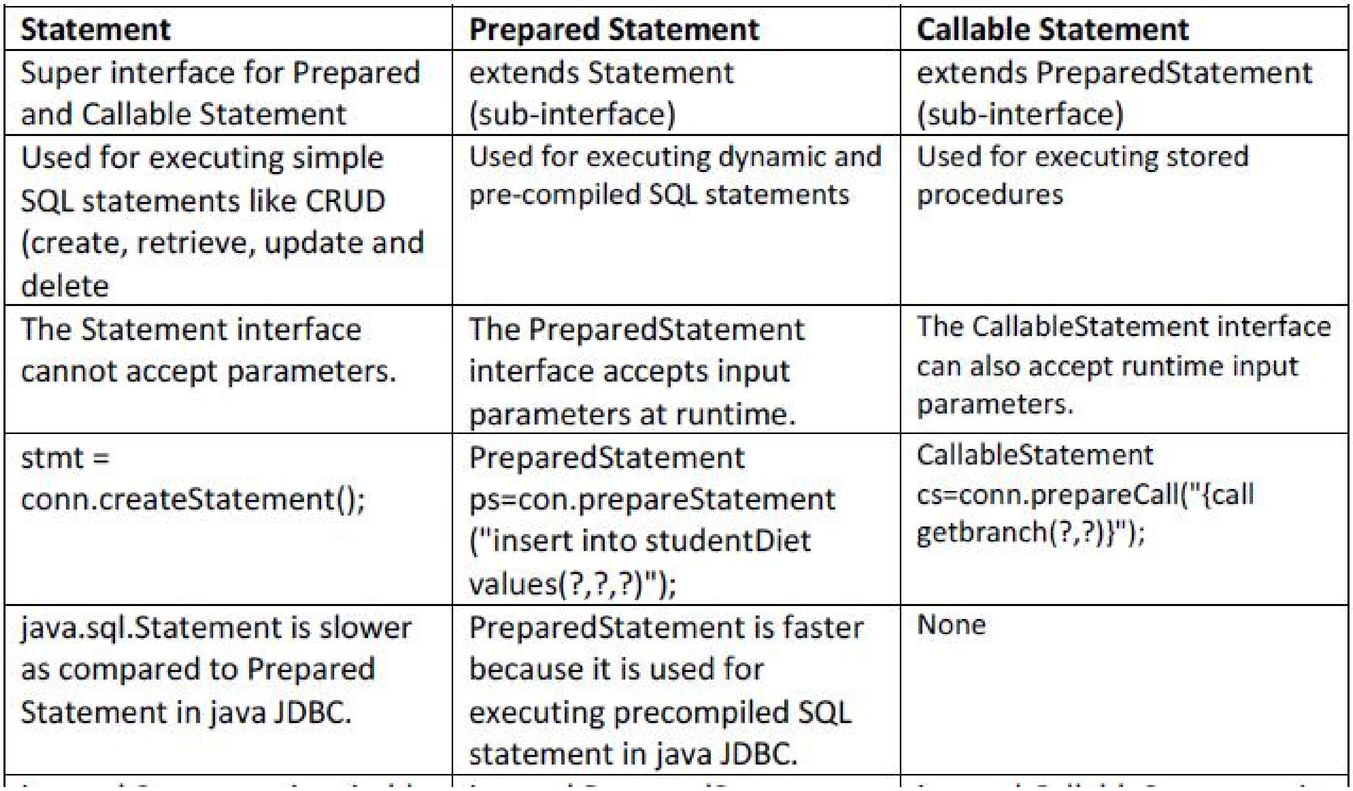
Stored Procedure: getbranch()

```
DELIMITER @@
DROP PROCEDURE getbranch @@
CREATE PROCEDURE databaseName.getbranch
(IN enr_no INT, OUT my_branch VARCHAR(10))
BEGIN
SELECT branch INTO my_branch
FROM dietStudent
WHERE enr_no=enrno;
END @@
DELIMITER ;
```

Callable Statement program

```
import java.sql.*;
public class CallableDemo {
public static void main(String[] args) {
try {
  Class.forName("com.mysql.jdbc.Driver");
  Connection conn= DriverManager.getConnection
          ("jdbc:mysql://localhost:3306/Diet", "root","pwd");

  CallableStatement cs=conn.prepareCall("{call getbranch(?,?)}");
                  cs.setInt(1,2222);
                  cs.registerOutParameter(2,Types.VARCHAR);
                  cs.execute();
                  System.out.println("branch="+cs.getString(2));
                  cs.close();
                  conn.close();
              }catch(Exceptione){System.out.println(e.toString());}
             }//PSVM
}//class
```

Differentiate Statement, Prepared Statement and Callable Statement.



Statement	Prepared Statement	Callable Statement
Super interface for Prepared and Callable Statement	extends Statement (sub-interface)	extends PreparedStatement (sub-interface)
Used for executing simple SQL statements like CRUD (create, retrieve, update and delete	Used for executing dynamic and pre-compiled SQL statements	Used for executing stored procedures
The Statement interface cannot accept parameters.	The PreparedStatement interface accepts input parameters at runtime.	The CallableStatement interface can also accept runtime input parameters.
stmt = conn.createStatement();	PreparedStatement ps=con.prepareStatement ("insert into studentDiet values(?,?,?)");	CallableStatement cs=conn.prepareCall("{call getbranch(?,?)}");
java.sql.Statement is slower as compared to Prepared Statement in java JDBC.	PreparedStatement is faster because it is used for executing precompiled SQL statement in java JDBC.	None

Triggers

```
CREATE TRIGGER minAtt BEFORE INSERT ON student
   FOR EACH ROW
BEGIN
     IF (:new.att < 75)
          THEN RAISE_APPLICATION_ERROR (-20004,
     'Attendance is less than 75%');
     END IF;
END;
```