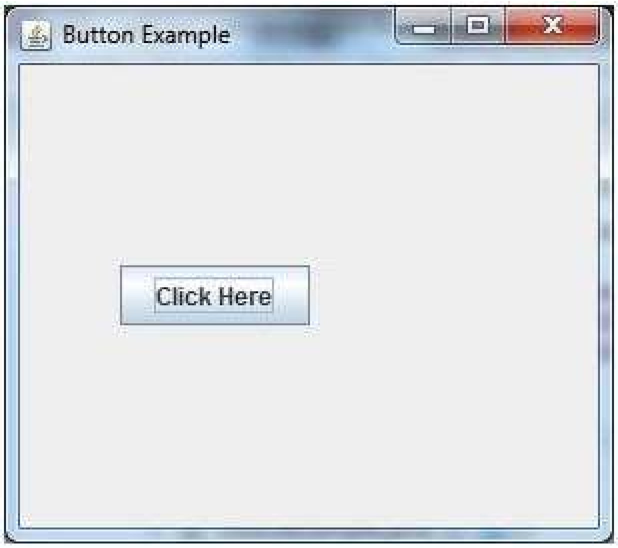
```
                    f.setVisible(true);
}
}
```

Output:



1.3.2. 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, A ccessible

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:

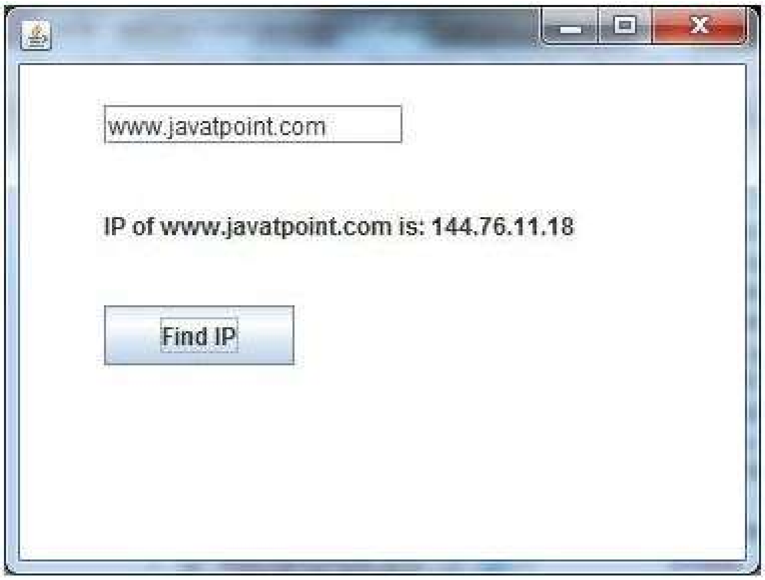
Methods	**Description**
String getText()	t returns the text string that a label displays.
void setText(String text)	It defines the single line of text this component will display.
void setHorizontalAlignment (int alignment)	It sets the alignment of the label's contents along the X axis.
Icon getIcon()	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 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();
   } }
```

Output:



1.3.3.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** SwingCon stants

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:

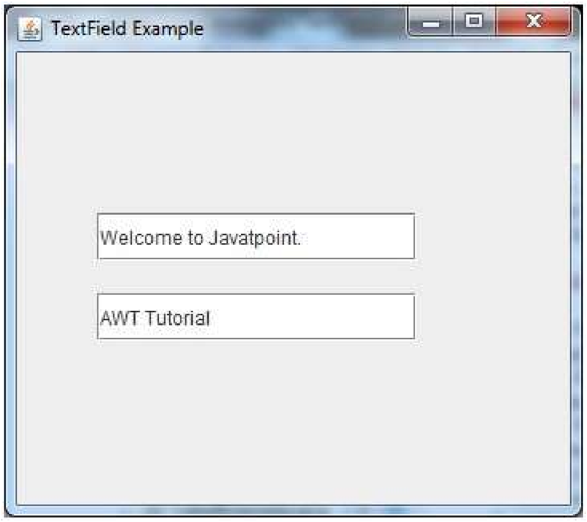
Methods	**Description**
void addActionListener(ActionListener l)	It is used to add the specified action listener to receive action events from this textfield.
Action getAction()	It returns the currently set Action for this ActionEvent source, or null if no Action is set.
void setFont(Font f)	It is used 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 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 Java");
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:



Welcome to Javatpoint.

AWT Tutorial

1.3.4. 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.

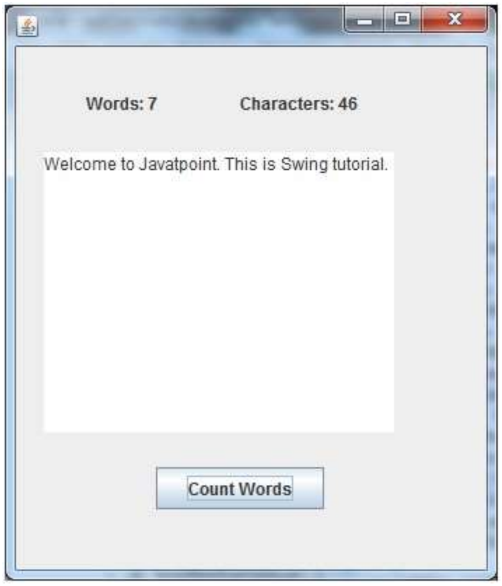
void insert(String s, int position)	It is used to insert the specified text on the specified position.
void append(String s)	It is used to append the given text to the end of the document.

Java JTextArea Example 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.

1.3.5.Java JPasswordField

The object of a JPasswordField class is a text component specialized for password entry. It allows the editing of a single line of text. It inherits JTextField class.

JPasswordField class declaration

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

public class JPasswordField **extends** JTextField Commonly used Constructors:

Constructor	**Description**
JPasswordField()	Constructs a new JPasswordField, with a default document, null starting text string, and 0 column width.
JPasswordField(int columns)	Constructs a new empty JPasswordField with the specified number of columns.
JPasswordField(String text)	Constructs a new JPasswordField initialized with the specified text.
JPasswordField(String text, int columns)	Construct a new JPasswordField initialized with the specified text and columns.

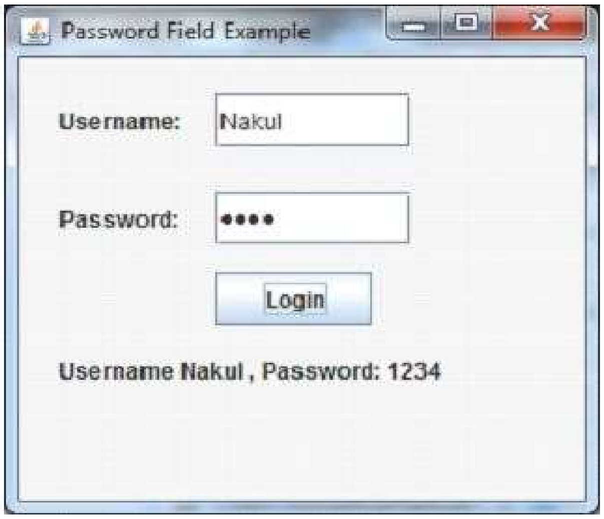
Java JPasswordField Example with ActionListener

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

public class PasswordFieldExample

```
{         public static void main(String[]
args)
{
   JFrame f=new JFrame("Password Field Example");
    final JLabel label = new JLabel();
label.setBounds(20,150, 200,50);        final
JPasswordField value = new JPasswordField();
    value.setBounds(100,75,100,30);
JLabel l1=new JLabel("Username:");
l1.setBounds(20,20, 80,30);             JLabel
l2=new JLabel("Password:");
l2.setBounds(20,75, 80,30);             JButton b
= new JButton("Login");
b.setBounds(100,120, 80,30);             final
JTextField text = new JTextField();
text.setBounds(100,20, 100,30);
             f.add(value);    f.add(l1);    f.add(label);    f.add(l2);    f.add(b);
f.add(text);
             f.setSize(300,300);
             f.setLayout(null);
             f.setVisible(true);
             b.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
String data = "Username " + text.getText();                    data
+= ", Password: "
               +                new                String(value.getPassword());
label.setText(data);
             }
          });
}
}
```

Output:



Username Nakul , Password: 1234

1.3.6. 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.

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:

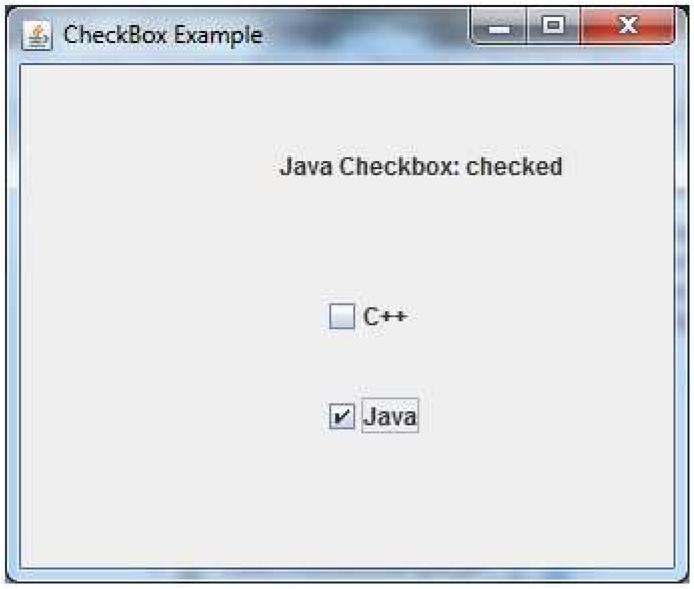
Methods	**Description**
AccessibleContext getAccessibleContext()	It is used to get the AccessibleContext associated with this JCheckBox.

protected String paramString()	It returns a string representation of this JCheckBox.

Java JCheckBox Example 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:



1.3.7.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.

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.

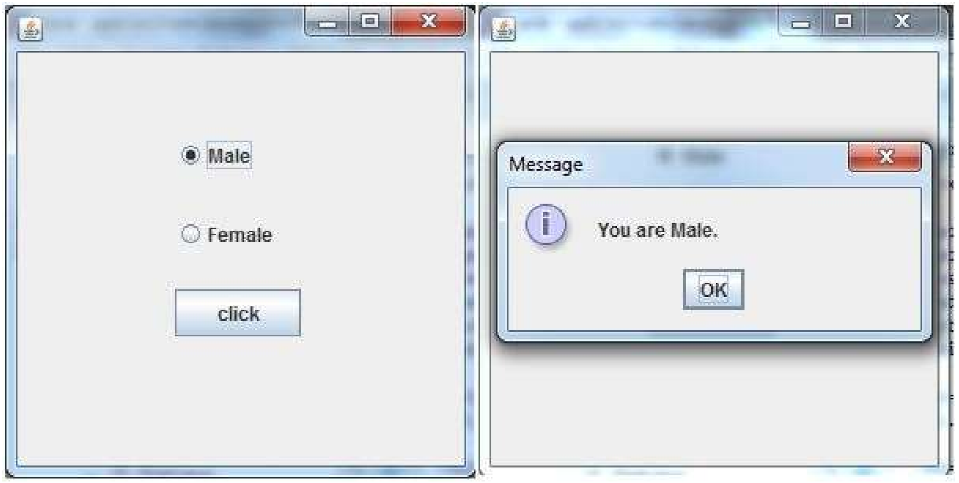
String getText()	It is used to return the text of the button.
Methods	**Description**
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int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



You are Male.

1.3.8. Java JComboBox

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.

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:

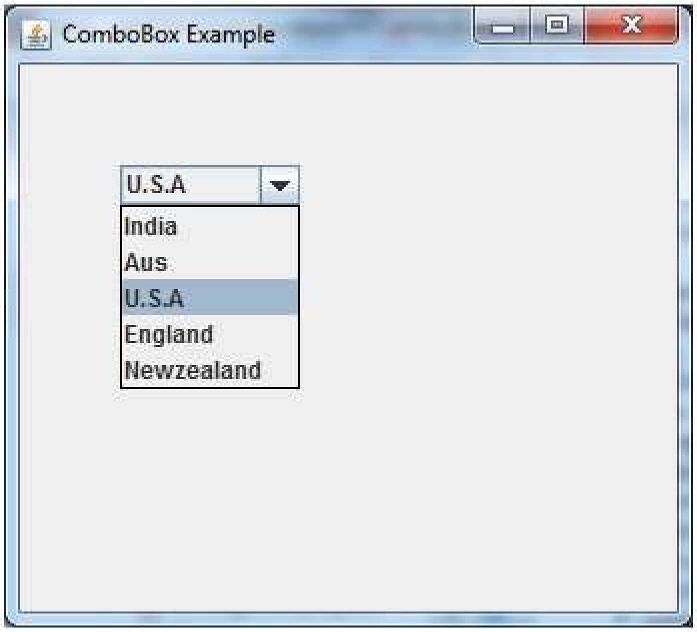
Methods	**Description**
void addItem(Object anObject)	It is used to add an item to the item list.
void removeItem(Object anObject)	It is used to delete an item to the item list.
void removeAllItems()	It is used to remove all the items from the list.
void setEditable(boolean b)	It is used to determine 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();
}    }
```

Outp ut:



1.3.9.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.

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.

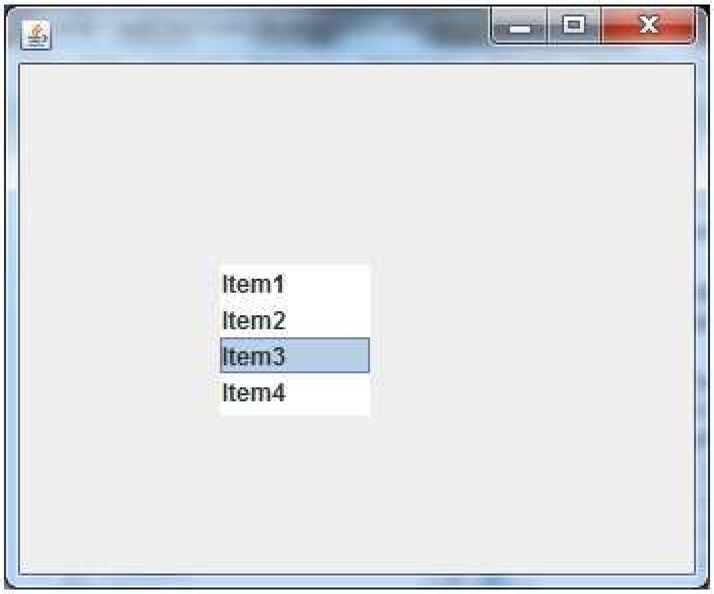
Commonly used Methods:

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.
Methods	**Description**
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

1.4 SUMMARY

In this chapter we learn about the swing to develop the graphical use interface and difference between AWT and Swing. Also we learn about the Swing component like Jtextfield, Jtextarea, Jpasswordfield ,JLabel, etc…

1.5 QUESTIONS

1. Write a short note on swing.
2. What is the difference between AWT and Swing.
3. Explain JTextarea with suitable example.
4. Explain JRadioButton with suitable example.
5. Explain JList with suitable example.
6. Explain JPasswordField with suitable example.

❄❄❄❄❄❄❄

2 JDBC

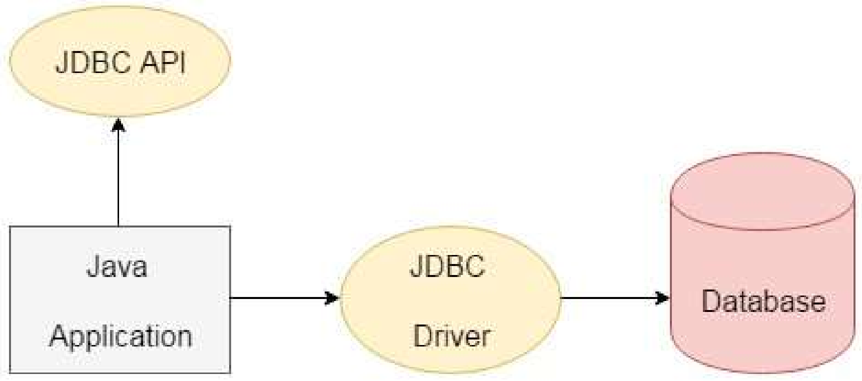
Unit Structure :

2.1 JDBC

JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers:

- JDBC-ODBC Bridge Driver,
- Native Driver,
- Network Protocol Driver, and
- Thin Driver

We can use JDBC API to access tabular data stored in any relational database. By the help of JDBC API, we can save, update, delete and fetch data from the database. It is like Open Database Connectivity (ODBC) provided by Microsoft.



The current version of JDBC is 4.3. It is the stable release since 21st September, 2017. It is based on the X/Open SQL Call Level Interface. The **java.sql** package contains classes and interfaces for JDBC API. A list of popular interfaces of JDBC API are given below:

- Driver interface
- Connection interface
- Statement interface
- PreparedStatement interface
- CallableStatement interface
- ResultSet interface
- ResultSetMetaData interface
- DatabaseMetaData interface
- RowSet interface

A list of popular *classes* of JDBC API are given below:

- DriverManager class
- Blob class
- Clob class
- Types class

2.1.1. 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.

What is API?

API (Application programming interface) is a document that contains a description of all the features of a product or software. It represents classes and interfaces that software programs can follow to communicate with each other. An API can be created for applications, libraries, operating systems, etc.

2.2 JDBC DRIVER

JDBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

1) **JDBC-ODBC bridge driver:** The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.

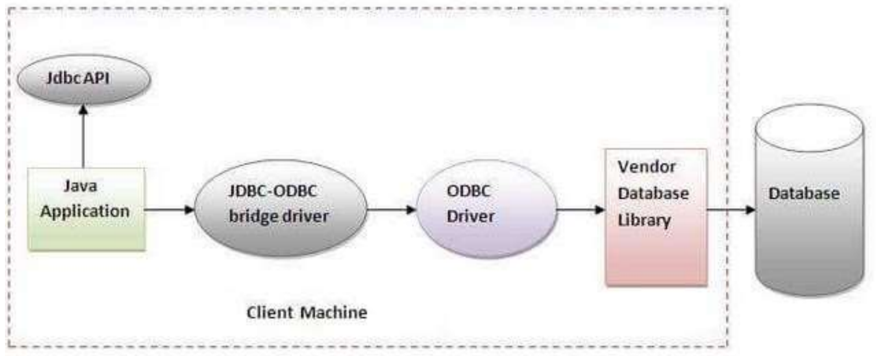


Figure- JDBC-ODBC Bridge Driver

Oracle does not support the JDBC-ODBC Bridge from Java 8. Oracle recommends that you use JDBC drivers provided by the vendor of your database instead of the JDBC-ODBC Bridge.

Advantages:

Easy to use.

Can be easily connected to any database.

Disadvantages:

Performance degraded because JDBC method call is converted into the ODBC function calls.

The ODBC driver needs to be installed on the client machine.

2) **Native-API driver:** The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

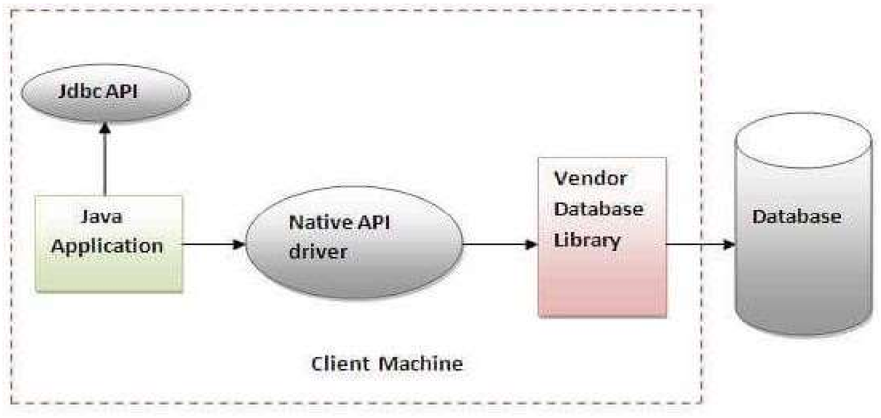


Figure- Native API Driver

Advantage:

Performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

The Native driver needs to be installed on the each client machine.

The Vendor client library needs to be installed on client machine.

3) **Network Protocol driver:** The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

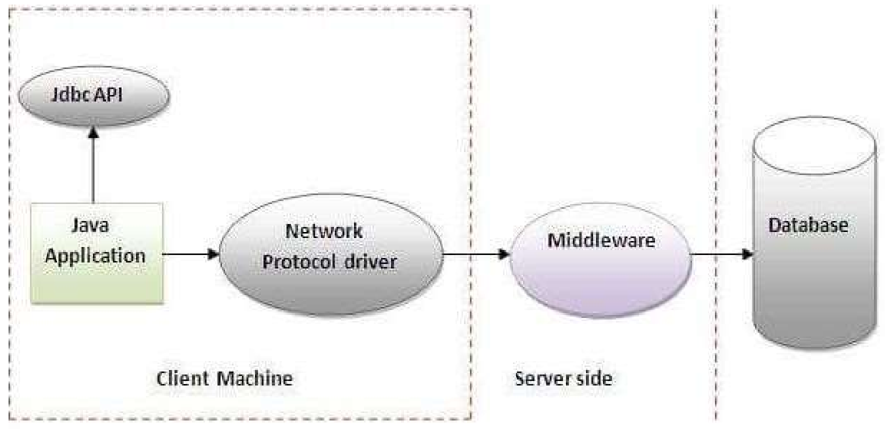


Figure- Network Protocol Driver

Advantage:

No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

Network support is required on client machine.

Requires database-specific coding to be done in the middle tier.

Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

4) **Thin driver:** The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

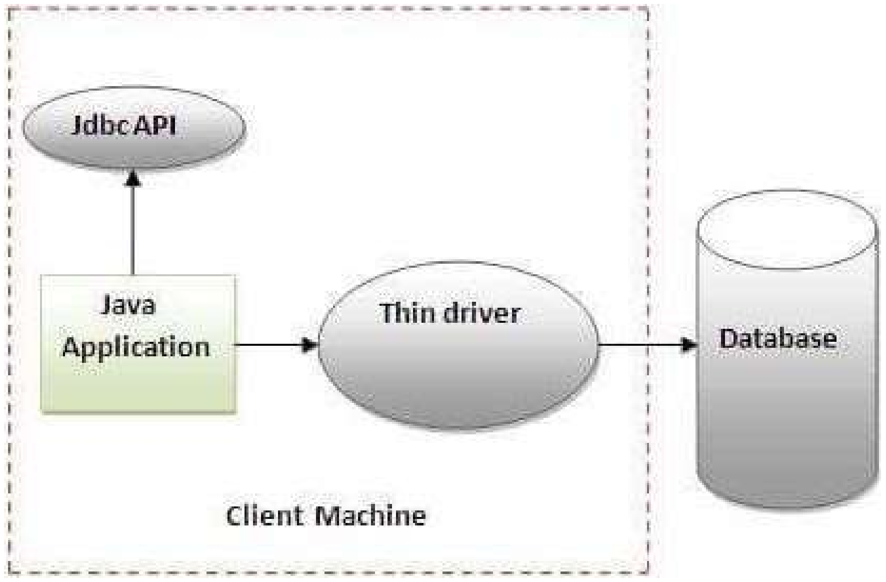


Figure- Thin Driver Advantage:

Better performance than all other drivers.

No software is required at client side or server side.

Disadvantage:

Drivers depend on the Database.

Java Database Connectivity with 5 Steps

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

1. Register the Driver class
2. Create connection
3. Create statement
4. Execute queries
5. Close connection

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 ClassNotFound Exception

Example to register the OracleDriver class

Here, Java program is loading oracle driver to esteblish database connection.

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

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

Syntax of getConnection() method

1) public static Connection getConnection(String url)throws SQLEx ception

2) public static Connection getConnection(String url,String name,Str ing password) throws SQLException

Example to establish connection with the Oracle database

Connection con=DriverManager.getConnection(

"jdbc:oracle:thin:@localhost:1521:xe","system","password");

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

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();

2.3 JAVA DATABASE CONNECTIVITY WITH ORACLE

To connect java application with the oracle database, we need to follow 5 following steps. In this example, we are using Oracle 10g as the database. So we need to know following information for the oracle database:

Driver class: The driver class for the oracle database is **oracle.jdbc.driver.OracleDriver**.

Connection URL: The connection URL for the oracle10G database is **jdbc:oracle:thin:@localhost:1521:xe** where jdbc is the API, oracle is the database, thin is the driver, localhost is the server name on which oracle is running, we may also use IP address, 1521 is the port number and XE is the Oracle service name. You may get all these information from the tnsnames.ora file.

Username: The default username for the oracle database is **system**.

Password: It is the password given by the user at the time of installing the oracle database.

Create a Table

Before establishing connection, let's first create a table in oracle database. Following is the SQL query to create a table.

create table emp(id number(10),name varchar2(40),age number(3));

2.3.1. Example to Connect Java Application with Oracle database

In this example, we are connecting to an Oracle database and getting data from **emp** table. Here, **system** and **oracle** are the username and password of the Oracle database. import java.sql.*;   class OracleCon{

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

try{
//step1 load the driver class
Class.forName("oracle.jdbc.driver.OracleDriver");

//step2 create  the connection object
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

//step3 create the statement object
Statement stmt=con.createStatement();

//step4 execute query
ResultSet rs=stmt.executeQuery("select * from emp");   while(rs.next())
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
//step5 close the connection object   con.close();
}
catch(Exception e){ System.out.println(e);}
}
}
```

The above example will fetch all the records of emp table.

To connect java application with the Oracle database ojdbc14.jar file is required to be loaded. **Two ways to load the jar file:**

paste the ojdbc14.jar file in jre/lib/ext folder set

classpath

1) **paste the ojdbc14.jar file in JRE/lib/ext folder:** Firstly, search the ojdbc14.jar file then go to JRE/lib/ext folder and paste the jar file here.

2) **set classpath:** There are two ways to set the classpath:
   - temporary
   - permanent

How to set the temporary classpath:

Firstly, search the ojdbc14.jar file then open command prompt and write: C:>set classpath=c:\folder\ojdbc14.jar;.;

How to set the permanent classpath:

Go to environment variable then click on new tab. In variable name write **classpath** and in variable value paste the path to ojdbc14.jar by appending ojdbc14.jar;.; as
C:\oraclexe\app\oracle\product\10.2.0\server\jdbc\lib\ojdbc14.jar;.;

2.4 DRIVERMANAGER CLASS

The DriverManager class acts as an interface between user and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver. The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method DriverManager.registerDriver().

Useful methods of DriverManager class

Method	**Description**
1) public static void registerDriver(Driver driver):	is used to register the given driver with DriverManager.
2) public static void deregisterDriver(Driver driver):	is used to deregister the given driver (drop the driver from the list) with DriverManager.

3) public static Connection getConnection(String url):	is used to establish the connection with the specified url.
4) public static Connection getConnection(String url,String userName,String password):	is used to establish the connection with the specified url, username and password.

2.4.1. Connection interface

A Connection is the session between java application and database. The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData i.e. object of Connection can be used to get the object of Statement and DatabaseMetaData. The Connection interface provide many methods for transaction management like commit(), rollback() etc.

By default, connection commits the changes after executing queries.

Commonly used methods of Connection interface:

1) **public Statement createStatement():** creates a statement object that can be used to execute SQL queries.

2) **public Statement createStatement(int resultSetType,int resultSetConcurrency):** Creates a Statement object that will generate ResultSet objects with the given type and concurrency.

3) **public void setAutoCommit(boolean status):** is used to set the commit status.By default it is true.

4) **public void commit():** saves the changes made since the previous commit/rollback permanent.

5) **public void rollback():** Drops all changes made since the previous commit/rollback.

6) public void close(): **closes the connection and Releases a JDBC resources immediately.**

2.5 STATEMENT INTERFACE

The **Statement interface** provides methods to execute queries with the database. The statement interface is a factory of ResultSet i.e. it provides factory method to get the object of ResultSet.

Commonly used methods of Statement interface:

The important methods of Statement interface are as follows:

1) **public ResultSet executeQuery(String sql):** is used to execute SELECT query. It returns the object of ResultSet.

2) **public int executeUpdate(String sql):** is used to execute specified query, it may be create, drop, insert, update, delete etc.

3) **public boolean execute(String sql):** is used to execute queries that may return multiple results.

4) **public int[] executeBatch():** is used to execute batch of commands

2.5.1. Example of Statement interface

Let's see the simple example of Statement interface to insert, update and delete the record. import java.sql.*; class FetchRecord{

```
public static void main(String args[])throws Exception{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localh
ost:1521:xe","system","oracle");
Statement stmt=con.createStatement();
//stmt.executeUpdate("insert into emp765 values(33,'Irfan',50000)");
//int result=stmt.executeUpdate("update emp765 set name='Vimal',salary=
10000 where id=33");
int result=stmt.executeUpdate("delete from emp765 where id=33");
System.out.println(result+" records affected"); con.close();
}
}
```

2.6 RESULTSET INTERFACE

The object of ResultSet maintains a cursor pointing to a row of a table. Initially, cursor points to before the first row.

By default, ResultSet object can be moved forward only and it is not updatable.

But we can make this object to move forward and backward direction by passing either TYPE_SCROLL_INSENSITIVE or TYPE_SCROLL_SENSITIVE in createStatment(int,int) method as well as
we can make this object as updatable by:

```
Statement stmt =
con.createStatement(ResultSet.TYPE_SCROLL_INSEN SITIVE,
        ResultSet.CONCUR_UPDATABLE);
```

Commonly used methods of ResultSet interface

1) public boolean next():	is used to move the cursor to the one row next from the current position.
2) public boolean previous():	is used to move the cursor to the one row previous from the current position.
3) public boolean first():	is used to move the cursor to the first row in result set object.
4) public boolean last():	is used to move the cursor to the last row in result set object.
5) public boolean absolute(int row):	is used to move the cursor to the specified row number in the ResultSet object.
6) public boolean relative(int row):	is used to move the cursor to the relative row number in the ResultSet object, it may be positive or negative.
7) public int getInt(int columnIndex):	is used to return the data of specified column index of the current row as int.
8) public int getInt(String columnName):	is used to return the data of specified column name of the current row as int.
9) public String getString(int columnIndex):	is used to return the data of specified column index of the current row as String.
10) public String getString(String columnName):	is used to return the data of specified column name of the current row as String.

2.6.1. Example of Scrollable ResultSet

Let's see the simple example of ResultSet interface to retrieve the data of 3rd row.

```
import java.sql.*;   class FetchRecord{   public static
void main(String args[])throws Exception{
  Class.forName("oracle.jdbc.driver.OracleDriver");
Connection  con=DriverManager.getConnection("jdbc:oracle:thin:@localh
ost:1521:xe","system","oracle");
Statement stmt=con.createStatement(ResultSet.TYPE_SCROLL_SENSIT
IVE,ResultSet.CONCUR_UPDATABLE);
ResultSet rs=stmt.executeQuery("select * from emp765");
```

```
  //getting the record of 3rd row
rs.absolute(3);
System.out.println(rs.getString(1)+" "+rs.getString(2)+" "+rs.getString(3))
;
  con.close();
}
}
```

2.7 PREPAREDSTATEMENT INTERFACE

The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query.

Let's see the example of parameterized query:

String sql="insert into emp values(?,?,?)";

As you can see, we are passing parameter (?) for the values. Its value will be set by calling the setter methods of PreparedStatement.

Why use PreparedStatement?

Improves performance: The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

How to get the instance of PreparedStatement?

The prepareStatement() method of Connection interface is used to return the object of PreparedStatement. Syntax:

public PreparedStatement prepareStatement(String query)throws SQLExc eption{}

Methods of PreparedStatement interface

The important methods of PreparedStatement interface are given below:

Method	**Description**
public void setInt(int paramIndex, int value)	sets the integer value to the given parameter index.
public void setString(int paramIndex, String value)	sets the String value to the given parameter index.

public void setFloat(int paramIndex, float value)	sets the float value to the given parameter index.
public void setDouble(int paramIndex, double value)	sets the double value to the given parameter index.
public int executeUpdate()	executes the query. It is used for create, drop, insert, update, delete etc.
public ResultSet executeQuery()	executes the select query. It returns an instance of ResultSet.

2.7.1 Example of PreparedStatement interface that inserts the record

First of all create table as given below:

create table emp(id number(10),name varchar2(50)); Now
insert records in this table by the code given below:

```
import java.sql.*;   class
InsertPrepared{   public static void
main(String args[]){   try{
Class.forName("oracle.jdbc.driver.OracleDriver");

Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localh
ost:1521:xe","system","oracle");

PreparedStatement stmt=con.prepareStatement("insert into Emp values(?,?
)");
stmt.setInt(1,101);//1 specifies the first parameter in the query
stmt.setString(2,"Ratan");

int i=stmt.executeUpdate();
System.out.println(i+" records inserted");

con.close();
  }catch(Exception e){ System.out.println(e);}
}
}
```

2.7.2 Example of PreparedStatement interface that updates the record

```
PreparedStatement stmt=con.prepareStatement("update emp set name=? w
here id=?");   stmt.setString(1,"Sonoo");//1 specifies the first parameter in
the query i.e. name   stmt.setInt(2,101);     int i=stmt.executeUpdate();
System.out.println(i+" records updated");
```

2.7.3 Example of PreparedStatement interface that deletes the record

```
PreparedStatement stmt=con.prepareStatement("delete from emp where id
=?");   stmt.setInt(1,101);
int i=stmt.executeUpdate();
System.out.println(i+" records deleted");
```

2.7.4 Example of PreparedStatement interface that retrieve the records of a table

```
PreparedStatement stmt=con.prepareStatement("select * from emp");
ResultSet rs=stmt.executeQuery();
while(rs.next()){
System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

2.7.5. Example of PreparedStatement to insert records until user press n

```
import java.sql.*;   import java.io.*;   class RS{   public static void
main(String args[])throws Exception{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection   con=DriverManager.getConnection("jdbc:oracle:thin:@localh
ost:1521:xe","system","oracle");
 PreparedStatement ps=con.prepareStatement("insert into emp130 values(
?,?,?)");
BufferedReader br=new BufferedReader(new InputStreamReader(System.
in));   do{
System.out.println("enter  id:");      int
id=Integer.parseInt(br.readLine());
System.out.println("enter name:");
String name=br.readLine();
System.out.println("enter salary:");   float
salary=Float.parseFloat(br.readLine());
ps.setInt(1,id);   ps.setString(2,name);
ps.setFloat(3,salary);   int
i=ps.executeUpdate();
System.out.println(i+" records affected");

System.out.println("Do you want to continue: y/n");
String s=br.readLine();
if(s.startsWith("n")){   break;
}
}while(true);   con.close();
}
}
```

2.8 SUMMARY

In this chapter we learn the concept of Data Base concept in java using jdbc and its different types of Drivers and how to connect the database like oracle, SQL, etc.. with java program. Also we will discuss about the interfaces like statement, resultset, preparedstatement etc..

2.9 QUESTIONS

1. Write a short note on JDBC.
2. Why Should We Use JDBC.
3. Explain Native-API driver.
4. Write a short note on Resultset.
5. Explain PreparedStatement in detail.
6. Explain JDBC APT components.

❄❄❄❄❄❄❄

3 SERVLETS

Unit Structure:

3.0 INTRODUCTION TO JAVA SERVLETS

Today we all are aware of the need of creating dynamic web pages i.e the ones which have the capability to change the site contents according to the time or are able to generate the contents according to the request received by the client. If you like coding in Java, then you know that using Java there also exists a way to generate dynamic web pages and that way is Java Servlet.

3.1 WEB APPLICATION ARCHITECTURE IN JAVA

Web application architecture is a mechanism that gives us a clarification that how the connection is established between the client and the server. It determines how the components in an application communicate with each other. It doesn't matter what's is the size and the complexity level of the application is, they all follow the same principle only the details may differ.

In technical terms, when a user makes a request on a website, various components of the applications, user interfaces, middleware systems, databases, servers, and the browser interact with each other. Web Application Architecture is a framework that ties up this relation together and maintains the interaction between these components.

Http protocol and http method

For HTTP/1.1, the set of common methods are defined below. This set can be expanded based on the requirements. The name of these methods is case sensitive, and they must be used in uppercase.

i) **GET :** This method retrieves information from the given server using a given URI. GET request can retrieve the data. It can not apply other effects on the data.

ii) HEAD : This method is the same as the GET method. It is used to transfer the status line and header section only.

iii) POST : The POST request sends the data to the server. For example, file upload, customer information, etc. using the HTML forms.

iv) PUT : The PUT method is used to replace all the current representations of the target resource with the uploaded content.

v) DELETE : The DELETE method is used to remove all the current representations of the target resource, which is given by URI.

vi) CONNECT : This method establishes a tunnel to the server, which is identified by a given URI.

vii) OPTIONS : This method describes the options of communication for the target resource.

3.2 WEB SERVER AND WEB CONTAINER

Web Server

A web server can be characterized as software that receives HTTP requests, processes them, and sends back responses.

A web server is a software program that, as its name implies, serves websites to users. It does this by responding to HTTP requests from the user's computer. The response includes HTML content that is sent over the internet and displayed in the user's browser.

Examples of web servers include Apache, Nginx, Microsoft Internet Information Server (IIS). Web Container

A web container, on the other hand, is an application that includes a web server as well as additional components like a servlet container, Enterprise JavaBean (EJB) container, and so forth.

Examples of web containers include Tomcat, Glassfish, JBoss Application Server, or Wildfly.

The benefit of using a web container is that there are fewer applications to maintain and configure. For instance, if your application requires an EJB, you can use the JBoss Application Server instead of Tomcat. If you are using the Spring framework, you can choose between Spring Source to Server or JBoss Application Server to host your Spring apps.

Servlet Interface

Servlet interface provides common behavior to all the servlets. Servlet interface defines methods that all servlets must implement.

Servlet interface needs to be implemented for creating any servlet (either directly or indirectly). It provides 3 life cycle methods that are used to initialize the servlet, to service the requests, and to destroy the servlet and 2 non-life cycle methods.

Methods of Servlet interface

There are 5 methods in Servlet interface. The init, service and destroy are the life cycle methods of servlet. These are invoked by the web container.

- Genericservlet
- GenericServlet class

- GenericServlet class implements Servlet, ServletConfig and Serializable interfaces. It provides the implementation of all the methods of these interfaces except the service method.
- GenericServlet class can handle any type of request so it is protocol-independent.

You may create a generic servlet by inheriting the GenericServlet class and providing the implementation of the service method.

3.3 METHODS OF GENERICSERVLET CLASS

There are many methods in GenericServlet class. They are as follows:

1. Public void init(ServletConfig config) is used to initialize the servlet.
2. Public abstract void service(ServletRequest request, ServletResponse response) provides service for the incoming request. It is invoked at each time when user requests for a servlet.
3. Public void destroy() is invoked only once throughout the life cycle and indicates that servlet is being destroyed.
4. Public ServletConfig getServletConfig() returns the object of ServletConfig. Public String getServletInfo() returns information about servlet such as writer, copyright, version etc.
5. Public void init() it is a convenient method for the servlet programmers, now there is no need to call super.init(config)
6. Public ServletContext getServletContext() returns the object of ServletContext.
7. Public String getInitParameter(String name) returns the parameter value for the given parameter name.

8. Public Enumeration getInitParameterNames() returns all the parameters defined in the web.xml file.

9. Public String getServletName() returns the name of the servlet object.

10. Public void log(String msg) writes the given message in the servlet log file.

11. Public void log(String msg,Throwable t) writes the explanatory message in the servlet log file and a stack trace.

HttpServlet

Public abstract class HttpServlet extends GenericServlet. Provides an abstract class to be subclassed to create an HTTP servlet suitable for a Web site. A subclass of HttpServlet must override at least one method, usually one of these: doGet , if the servlet supports HTTP GET requests. doPost , for HTTP POST requests.

Servlet Life Cycle

The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet:

1) **Servlet class is loaded :** The classloader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.

2) **Servlet instance is created :** The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.

3) **Init method is invoked :** The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface. Syntax of the init method is given below:

Public void init(ServletConfig config) throws ServletException

4) **Service method is invoked :** The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method. If servlet is initialized, it calls the service method. Notice that servlet is initialized only once. The syntax of the service method of the Servlet interface is given below:

Public void service(ServletRequest request, ServletResponse response)

Throws ServletException, IOException

5) Destroy method is invoked : The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc. The syntax of the destroy method of the Servlet interface is given below: Public void destroy()

3.4 SERVLETCONFIG

javax.servlet.ServletConfig is an interface as a part of servlet API. For every Servlet class in our application, the web container will create one ServletConfig object and the web container will pass this object as an argument to the public void init(ServletConfig config) method of our Servlet class object. Some of the important points on ServletConfig are:

ServletConfig is an object containing some initial parameters or configuration information created by Servlet Container and passed to the servlet during initialization.

ServletConfig is for a particular servlet, which means one should store servlet-specific information in web.xml and retrieve them using this object.

ServletContext

Javax.servlet.ServletConfig is an interface as a part of servlet API. For every Servlet class in our application, the web container will create one ServletConfig object and the web container will pass this object as an argument to the public void init(ServletConfig config) method of our Servlet class object. Some of the important points on ServletConfig are:

ServletConfig is an object containing some initial parameters or configuration information created by Servlet Container and passed to the servlet during initialization.

ServletConfig is for a particular servlet, which means one should store servlet-specific information in web.xml and retrieve them using this object.

Differentiate between ServletContext and ServletConfig.

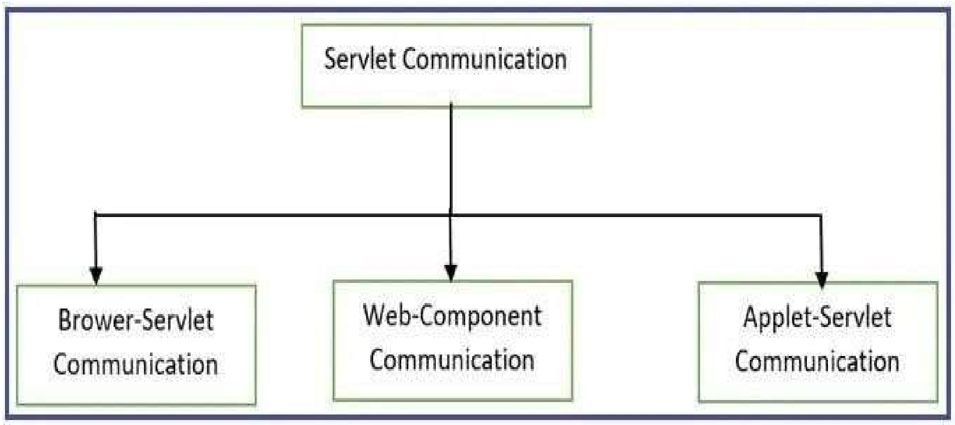
Sr. No.	ServletConfig	ServletContext
1	ServletConfig is one per Servlet	ServletContext is one per web application

2	It can be used to pass the deployment time parameters to the servlet using during the servlet initialization like database name, file name, etc	It can be used to access the Web application parameters configured in the deployment descriptor file (WEB.xml)
3	It can be used to access the ServletContext object.	It can be used for the inter application communication between servlets, JSPs and other components of a web application.
4	Can access the initialization parameters for a servlet instance.	Can be used to access the server information about the container and the version of the API it supports.

Servlet Communication

In general, web application deployment is not at all suggestible to provide the complete application logic within a single web resource, it is suggestible to distribute the complete application logic over multiple web resources.

In the above context, to execute the application we must require communication between all the web resources, for this, we have to use Servlet Communication. In the web application, we are able to provide servlet communication in the following three ways:


Servlet Communication
Brower-Servlet Communication
Web-Component Communication
Applet-Servlet Communication

Browser-Servlet Communication