|  |  |  |
| --- | --- | --- |
| Sno | Topic Name | Page Numbers |
| 1. | JDBC | 2 |
| 2. | Statement | 7 |
| 3. | PreparedStatement | 16 |
| 4. | CallableStatement | 21 |
| 5. | Bi-DirectionalResultSet | 25 |
| 6. | Bi-directional ResultSet using PreparedStatement | 31 |
| 7. | Transactions | 37 |
| 8. | BatchUpdates | 40 |
| 9. | Remove Hard Coding | 42 |
| 10. | MetaData  10.1.DataBaseMetaData  10.2.ResultSetMetaData | 44 |
| 11. | JDBC Drivers | 48 |

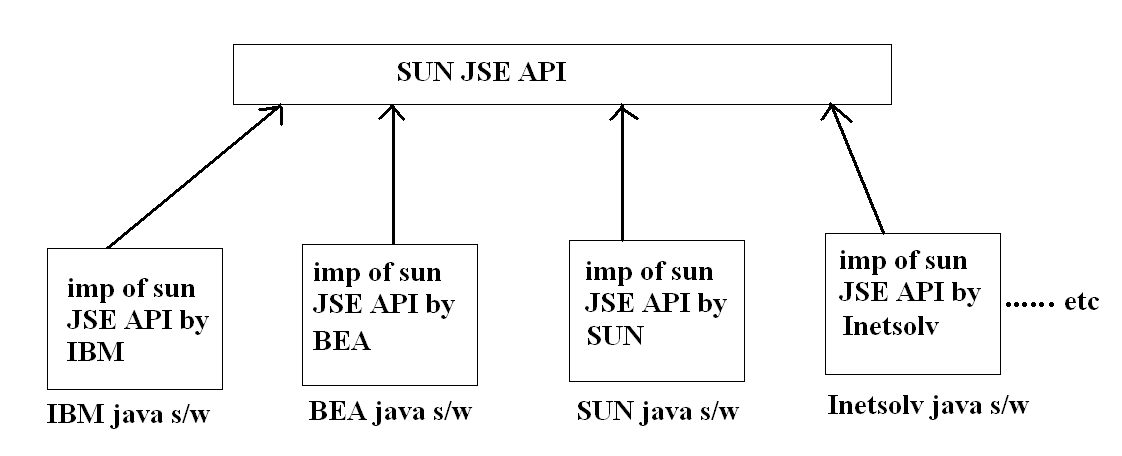
Contents

**1. JDBC**

**What is java software?**

* Java is not software it is an API. Released by sun micro system. The sun micro system has released the java software has open specification.
* As the sun micro system has released open specification API anyone in this world can provide the implementation.
* If anyone provide the implementation based on the specification the implementation is called as software.

**Diagram:**



* From the above diagram we understood that after sun micro system released the JSE API so many companies have provided the implementation.
* Some of the company names are IBM, BEA, SUN micro system, Inetsolv and etc.
* The implementation is called as software.

**What is JEE?**

* JEE java enterprise edition is an API which is composed of other API’s (Servlet API, JSP API, JNDI API, JAVAMAIL API and etc).

**Note:** JDBC is not part of JEE API. It is part of JSE API.

**JDBC: Java Database Connectivity (or) Java to Database Connectivity**.

* JDBC API helps the java developer in establishing a connection with database and performs the operations.
* There are so many popular database server vendors are available. They are Oracle, mysql, DB2, Sybase and etc.
* Whenever we buy (or) downloaded any database software they will provide two different software’s. They are:

1. Server Software
2. Client Software

* If we buy Oracle Server from Oracle Corporation they give Oracle Server Software, Oracle client software.
* Similarly mysql, Sybase and etc will also provide server software as well as the client software.
* Oracle server software will installed on a server computer and the server computer will be having the following details.

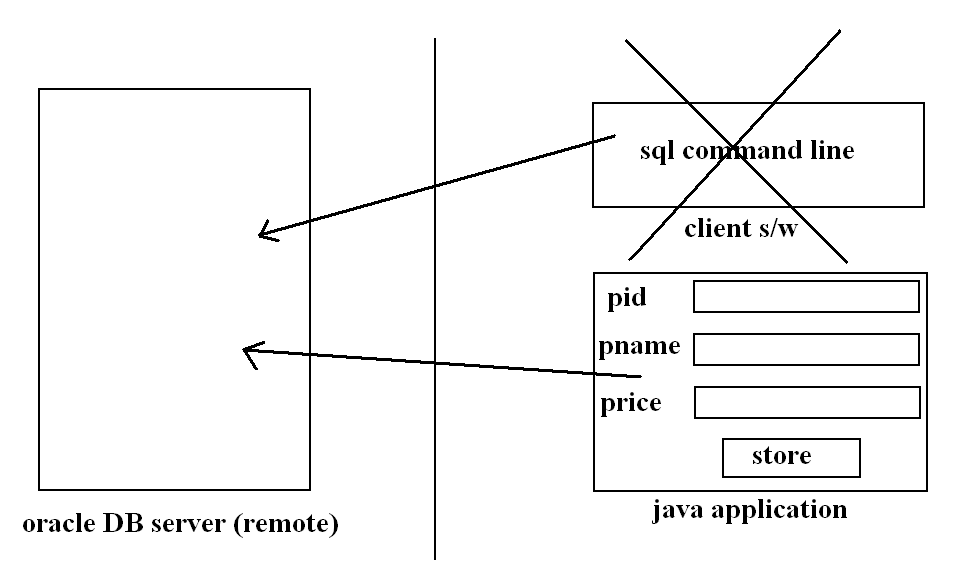
IP address-------------------------127.0.0.1

Service Name------------------------asdf

Port no --------------------------------1521

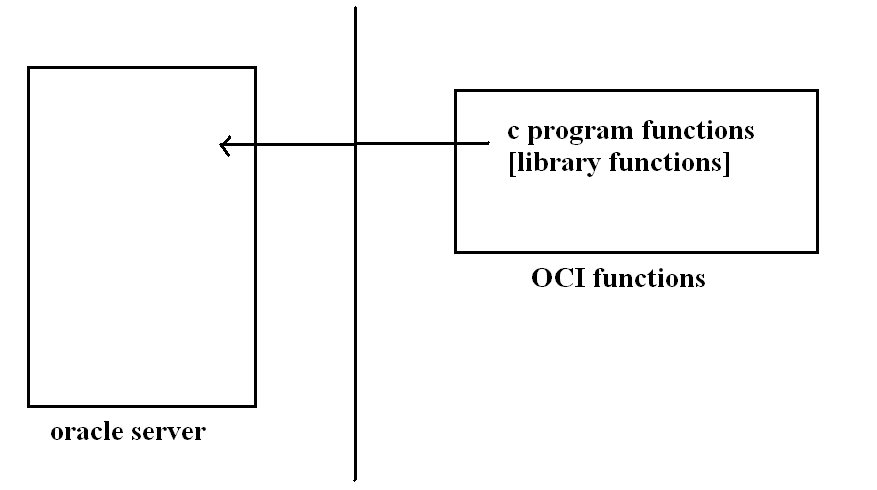
* **The Oracle administrator provides all the above details.**

**Diagram:**

****

* To communicate with oracle database server we can use oracle client program (or) java program.
* By using oracle client program to interact with database the end user must be aware of Sql commands.
* We develop GUI application for the client to interact with database.
* We can develop the GUI applications using Servlets, jsp, Struts, spring, awt, swing.
* The GUI application interacts with database.
* Every database server vendor releases call interface functions.
* In case of oracle they have released OCI functions [oracle call interface] these functions will be provided in the form of library functions. These functions are developed in ‘c’ language.

**Diagram:**

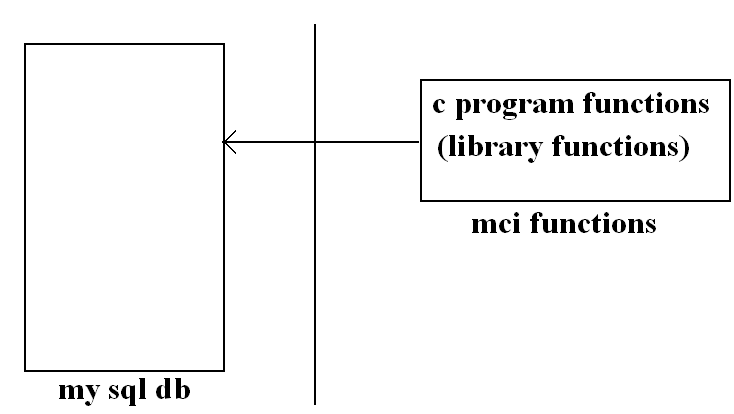


* From the above diagram we understood that if any one interacts with oracle database server they can interact through OCI function only.

**Note:** We can install OCI functions by installing the client software.

* Similarly mysql database server has released MCI functions to interact with mysql database.
* With the help of MCI functions we can interact with mysql database server.

**Diagram:**

****

* Every database vendors releases their own CI functions [call interface].
* The following are the examples of OCI functions.

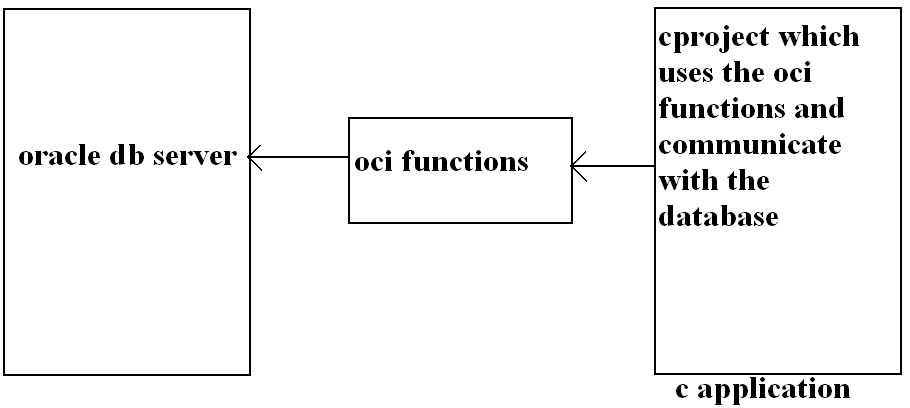
Ologon(username,password)

Odisconnect();

Oexecute(query);

* We can use all the above functions to interact with oracle database server.
* With the help of database server and CI functions we can develop C program which communicate with database server.

**Diagram:**

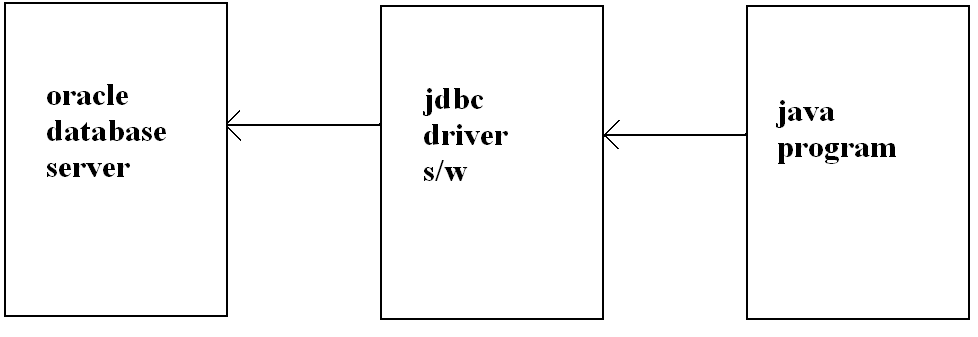


* From the above diagram we have understood that the ‘C’ program we have developed OCI functions. Whenever we execute the ‘C’ program which uses OCI it executes the code available in OCI functions and the code available in OCI functions talk to database server.
* The problem with above project is it we change the database server the ‘c’ program will not work. The ‘c’ program is developed based on the OCI functions given by Oracle Corporation.
* [Every database server will have their own CI functions].
* If we want to work with ‘C’ program which talk to new database we need to change ever C application with the latest database server CI functions.

**NOTE:** Every time which change the database we need change the ‘C’ applications.

* The above problem is resolved with the help of ODBC API [open database connectivity].
* ODBC API is released by Microsoft.
* By using ODBC API we can develop a ‘C’ application which talks to (or) connects to any database in this world without changing ‘C’ code.
* By using JDBC API we can write a java program to connect to any database in this world without changing the java code.

**Diagram:**

****

* From the above diagram we have understood that java program talks to JDBC driver s/w. JDBC driver talk to the database server.
* The JDBC driver talks to the database server.
* SUN Microsystems has released JDBC API this API consists of set of classes and interfaces.
* The JDBC related classes and interfaces are available in the following packages. They are:

1. Java.sql
2. Javax.sql

* The most important interface names of JDBC is

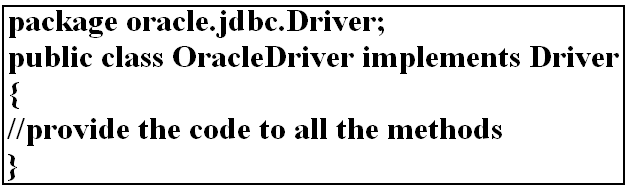
1. **Driver**
2. **Connection**
3. **Statement**
4. **PreparedStatement**
5. **CallableStatement**
6. **ResultSet**
7. **DatabaseMetaData**
8. **ResultSetMetaData…….etc**

* The most important classes are in JDBC they are

1. **DriverManager**
2. **Types**

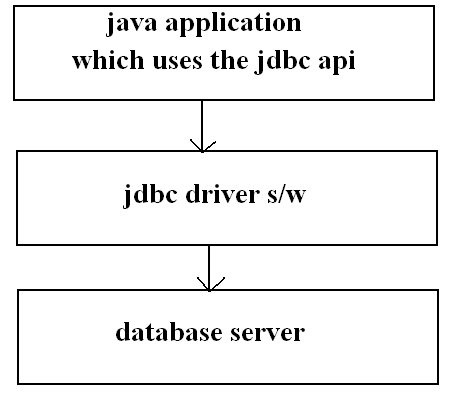
* Every driver implementer must provide implementation to driver interface.

**Example:** Oracle has provided implementation as shown below.



* If we want to develop the java application which talks to database we need to have JDBC driver.
* JDBC driver is a software. JDBC driver contains the code which really talks to the database server.
* We can find the JDBC driver software in the following location.
* **C:\oraclexe\app\oracle\product\10.2.0\server\jdbc\lib**
* In the above folder we will find the jar file whose name is **ojdbc14.jar.**
* This jar file consists of class files. This jar file is called as JDBC driver software.

**Diagram:**



* In case of oracle9i the name of the JDBC driver is classes12.jar.
* In case of oracle 10g the name of the JDBC driver is ojdbc14.jar.

**2. Statement**

**Procedure to use Statement**

**Step1:** Register the driver.

**Step2:** Get the connection from database.

**Step3:** Create the statement object. To create the statement object we use a method CreateStatement().

**Step4:** Execute the query using executeUpdate() method (or) executeQuery() method.

**Step5:** Close the connection.

**Procedure to develop a java application which establish connection with database and execute the queries:**

**Program:**

|  |
| --- |
| public class DBconnect{  public static void main(String args[])throws sqlException{  //step1:Register the driver  Driver d= new oracle.jdbc.driver.OracleDriver();  DriverManager.registerDriver(d);  System.out.println("driver is registered");  }  } |

**Errors:**

**D:\jdbc>javac DBconnect.java**

DBconnect.java:2: cannot find symbol

Symbol: class SQLException

Location: class DBconnect

public static void main(String args[])throws SQLException

DBconnect.java:5: cannot find symbol

Symbol: class Driver

Location: class DBconnect

Driver d= new oracle.jdbc.driver.OracleDriver();

DBconnect.java:6: cannot find symbol

Symbol: variable DriverManager

Location: class DBconnect

DriverManager.registerDriver(d);

**3 errors**

* To resolve the above errors we have to import the import statements

**After import the import statements:**

|  |
| --- |
| import java.sql.\*;  public class DBconnect{  public static void main(String args[])throws SQLException{  //step1:Register the driver  Driver d= new oracle.jdbc.driver.OracleDriver();  DriverManager.registerDriver(d);  System.out.println("driver is registered");  }  } |

**Error:**

D:\jdbc>javac DBconnect.java

DBconnect.java:3: cannot find symbol

Symbol: class SQLException

Location: class DBconnect

public static void main(String args[])throws SQLException

**1 error**

* To resolve the above error we have to write the SQLException in capital letters.

**Program:**

|  |
| --- |
| import java.sql.\*;  public class dbconnect1{  public static void main(String args[])throws SQLException{  //step1:register the driver  //Driver d=new oracle.jdbc.driver.OracleDriver();  //DriverManager.registerDriver(d);  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  System.out.println("driver is registered");  //step2:get the connection from database  Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","manager");  System.out.println("i have got the connection and object is:"+con);  }  }  **Output:**  D:\jdbc>javac dbconnect1.java  D:\jdbc>java dbconnect1  Driver is registered  I have got the connection and object is:oracle.jdbc.driver.T4CConnection@198dfafut: |

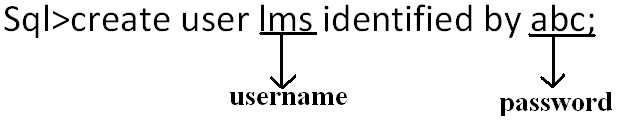
**The following is the procedure to create the user in the database:**

**Step 1:** Open Sql command line program.

**Step 2:** Type the command

**Connect system/manager;**

**Step 3:** Issue the following query to create the user.



**Step 4:** Issue the following command to grant the privileges.



**Requirement:** Develop a JDBC program which will able to establish a connection with vbr user.

**Program:**

|  |
| --- |
| import java.sql.\*;  public class JdbcconnectionWithVbrUser{  public static void main(String args[])throws SQLException{  //step1:registered the driver  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  System.out.println("driver is registered");  //step2:get the connection from database  Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","vbr","bhaskar");  System.out.println("i got the connection : "+con);  }  }  **Output:**  E:\jdbc>java JdbcconnectionWithVbrUser  Driver is registered  I got the connection: oracle.jdbc.driver.T4CConnection@1b90b39 |

* In java we have two types of queries are available. They are:

1. select query
2. non select query

**What is a select Query?**

* The query which starts with select keyword is called as select query.

**What is a non select Query?**

* The query which will not start with select is called as non select query.
* To execute the “select queries” we use “executeQuery() method”.
* To execute the “nonselect queries**”** we use “executeUpdate() method”.

**Requirement:** Develop a JDBC program which will be able to create the table EMP with columns empno, name, address, salary, date of joining.

**Note:** The table must be created in user vbr.

**Program:**

|  |
| --- |
| import java.sql.\*;  public class CreateTableUsingStatement{  public static void main(String args[])throws SQLException{  //step1:registered the driver  DriverManager.registerDriver (new oracle.jdbc.Driver.OracleDriver());  System.out.println ("driver is registered");  //step2:get the connection from database  Connection objcon=DriverManager.getConnection ("jdbc:oracle:thin:@localhost:1521:xe","vbr","bhaskar");  System.out.println ("I got the connection: "+objcon);  //step3:create statement object  Statement objstmt=objcon.createStatement();  System.out.println ("created the statement object: "+objstmt);  //step4:execute query (sending query to the database through executeUpdate () method)  objstmt.executeUpdate("create table emp(eno number(10),ename varchar2(50),salary number(10),address varchar2(50))");  System.out.println ("table is created");  }  }  **Output:**  D:\jdbc>javac CreateTableUsingStatement.java  D:\jdbc>java CreateTableUsingStatement  Driver is registered  I got the connection: oracle.jdbc.driver.T4CConnection@198dfaf  Created the statement object: oracle.jdbc.driver.T4CStatement@6b97fd  Table is created |

**Requirement:** Develop a JDBC program to insert a record into the database in the vbr user we have a table whose name is EMP.

**Program:**

|  |
| --- |
| import java.sql.\*;  public class InsertEmpValuesUsingStatement{  public static void main(String args[])throws SQLException{  //step1:registered the driver  DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver());  System.out.println ("driver is registered");  //step2:get the connection from database  Connection objcon=DriverManager.getConnection ("jdbc:oracle:thin:@localhost:1521:xe"," vbr ","Bhaskar");  System.out.println ("i got the connection: "+objcon);  //step3:create statement object  Statement objstmt=objcon.createStatement();  System.out.println ("created the statement object: "+objstmt);  //step4:execute query (sending query to the database through executeUpdate () method)  objstmt.executeUpdate ("insert into emp values (1,'bhaskar', 30000,'ongole')");  System.out.println ("values are inserted");  }  }  **Output:**  D:\jdbc>javacInsertEmpValuesUsingStatement.java  D:\jdbc>java InsertEmpValuesUsingStatement  Driver is registered  I got the connection: oracle.jdbc.driver.T4CConnection@198dfaf  Created the statement object: oracle.jdbc.driver.T4CStatement@6b97fd  Values are inserted |

* executeUpdate() method returns an integer value. The integer value indicates how many records are inserted by the given Sql query.

**Requirement:** Develop a JDBC program which updates a record in the EMP table we need to update the employee name to Krishna whose empno is 1.

**Program:**

|  |
| --- |
| import java.sql.\*;  public class UpdateRecordUsingStatement{  public static void main(String args[])throws SQLException{  //step1:registered the driver  DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver());  System.out.println ("driver is registered");  //step2:get the connection from database  Connection objcon=DriverManager.getConnection ("jdbc:oracle:thin:@localhost:1521:xe","system","manager");  System.out.println ("I got the connection: "+objcon);  //step3:create statement object  Statement objstmt=objcon.createStatement();  System.out.println ("created the statement object: "+objstmt);  //step4:execute query (sending query to the database through executeUpdate () method)  //here n indicates how many records are modified at the run time  int n=objstmt.executeUpdate("update emp set ename='krishna' where eno=3");  System.out.println (n+"record is updated");  }  }  **Output:**  D:\jdbc>javac UpdateRecordUsingStatement.java  D:\jdbc>java UpdateRecordUsingStatement  Driver is registered  I got the connection: oracle.jdbc.driver.T4CConnection@198dfaf  The statement object is: oracle.jdbc.driver.T4CStatement@6b97fd  1record is updated |

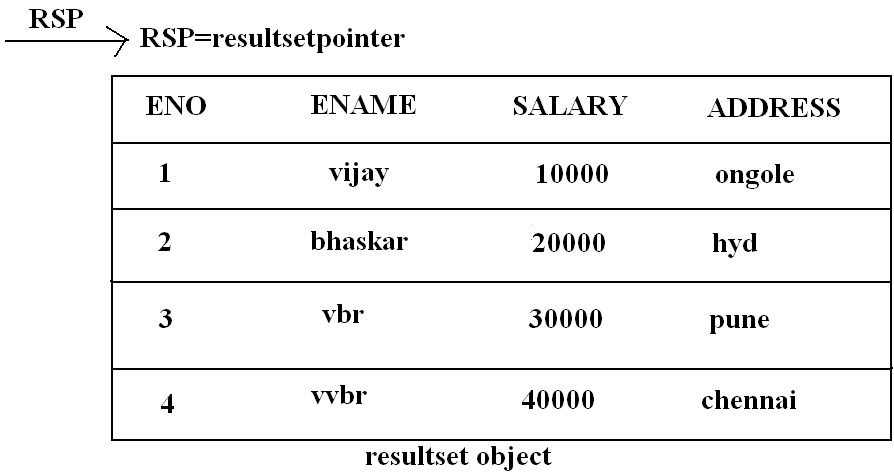
**How do you close the connection object? And who is the responsible to close the connection object?**

* Every programmer must make sure that he writes the code to close the connection.
* To close the connection we use a method “close()”on the connection object.

**Example:** con.close();

**Requirement:** Develop a JDBC program which retrieves the record which are available in EMP table [get all the records which are available in EMP table].

**Diagram:**

****

* Whenever we got the ResultSet object the ResultSet object contain the relational data.
* As part of ResultSet object we will find a pointer this pointer is called as ResultSet pointer.
* When we get the ResultSet object by default the pointer is pointing to an invisible record before the first row.
* We can move the pointer from one record to another record. To move the pointer from one record to another record we use next () method.
* The next () method returns a Boolean value either true (or) false. If it returns true it indicates that the next record is available. If it returns false it indicates that next record is not available.

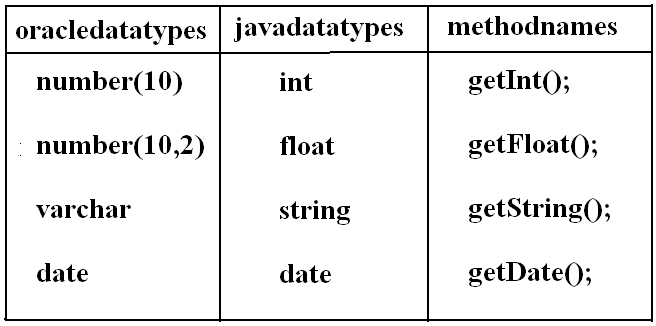
**Example of using next() method**

ResultSet rs = stmt.executeQuery(“select \*from emp”);

rs.next ();

* If the ResultSet pointer is pointing to first row we can read all the columns values of the first row.
* To read data from the columns we use getxxx() method.
* The xxx is dependent on the datatype of column.

**Example of getter methods:**

****

**Program:**

|  |
| --- |
| import java.sql.\*;  public class RetrieveRecordsUsingStatement{  public static void main(String args[])throws SQLException{  DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver());  Connection objcon=DriverManager.getConnection ("jdbc:oracle:thin:@localhost:1521:xe","system","manager");  Statement objstmt=objcon.createStatement();  String query="select \*from emp9";  ResultSet objrs=objstmt.executeQuery(query);  System.out.println ("query is executed");  System.out.println ("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  System.out.println ("eno"+"\t"+"ename"+"\t"+"salary");  System.out.println ("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  While (objrs.next()){  System.out.print(objrs.getInt("eno")+"\t");  System.out.print(objrs.getString("ename")+"\t");  System.out.println(objrs.getFloat("salary")+"\t");  }  System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  }  }  **Output:**  D:\jdbc>javac RetrieveRecordsUsingStatement.java  D:\jdbc>java RetrieveRecordsUsingStatement  Query is executed  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  eno ename salary  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  1 bhaskar 30000.0  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |

* To retrieve records from the columns we have two types of “getter methods” are available they are:

1. getxxx(String ColumnName);
2. getxxx(int ColumnIndex);

* The method which takes column as argument checks for whether the column name is available in the ResultSet object or not. If it is available it picks the value of that column and sends to the user.

**Requirement:** Write a JDBC program to retrieve the records from product table the table contains 3 columns they are PID, pname, price. To develop this JDBC program we can use any of the following queries.

Select \*from product

Select pid, pname, dop from product

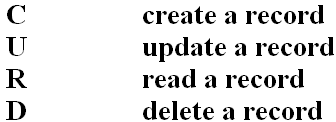
Select pid as productid, name, dop from product

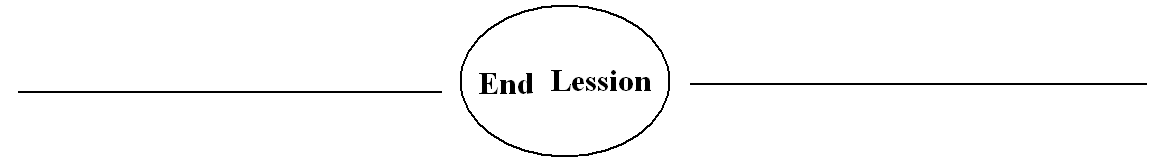
Select dop, pid, pname from product

**Program:**

|  |
| --- |
| import java.sql.\*;  public class RetrieveProductRecordsStatement{  public static void main(String args[])throws SQLException{  //register the driver  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  System.out.println("driver is registered");  //get the connection from database  Connection objcon=DriverManager.getConnection ("jdbc:oracle:thin:@localhost:1521:xe","system","manager");  System.out.println("connection class : "+objcon);  //create statement object  Statement objstmt=objcon.createStatement();  System.out.println("statement object is created");  String query="select \*from product";  //create resultset object  ResultSet objrs=objstmt.executeQuery(query);  System.out.println("ResultSet object is created");  System.out.println("query is executed");  while(objrs.next()){  System.out.println(objrs.getInt("pid"));  System.out.println(objrs.getString("pname"));  System.out.println(objrs.getFloat("price"));  }  System.out.println("records are completed in ResultSet object");  }  }  **Output:**  D:\jdbc>javac RetrieveProductRecords.java  D:\jdbc>java RetrieveProductRecords  Driver is registered  Connection class: oracle.jdbc.driver.T4CConnection@198dfaf  Statement object is created  ResultSet object is created  Query is executed  Records are completed in ResultSet object |

* Most of the developers tries to use getxxx(columnname) this is because if there is change in the order of SQL query we know need to change the remaining java code.
* If we use getxxx(index) if there is any change in the query we need to change the remaining java code also.
* Instead of using getxxx() methods we can use a common method that is **getString()** to retrieve records from any column type.
* Generally we perform **CURD** operations on the database table.





**3.** **Prepared statement**

* Prepared statement improves the performance of an application.

**Procedure to use PreparedStatement:**

**Step 1:** Register the driver

**Step 2:** Get the connection object

**Step 3:** Create the prepared statement object. To create the prepared statement object we use a method prepareStatement() this method takes the SQL query as input. The SQL query will contains the question mark.

**Step 4:** Supply the values to the question mark using setxxx() method. This method takes two arguments they are:

1. question mark index
2. value

**Step 5:** Execute the query using “executeUpdate()” method (or) “executeQuery()” method.

**Step 6:** Close the connection.

**Requirement:** Develop a JDBC program which inserts a record into **EMP** table. [Develop using prepared statement].

**Program:**

|  |
| --- |
| import java.sql.\*;  public class InsertRecordUsingPreparedStatement{  public static void main(String args[])throws SQLException{  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  Connection objcon=DriverManager.getConnection ("jdbc:oracle:thin:@localhost:1521:xe","system","manager");  //step3:create preparedstatement object  PreparedStatement objpstmt=objcon.prepareStatement("insert into emp values(?,?,?)");  objpstmt.setInt(1,2);  objpstmt.setString(2,"vijayabhaskar");  objpstmt.setFloat(3,50000);  int noofmodifiedrecords=objpstmt.executeUpdate();  System.out.println("the modified records are/is: "+noofmodifiedrecords);  objcon.close();  }  }  **Output:**  D:\jdbc>javac InsertRecordUsingPreparedStatement.java  D:\jdbc>java InsertRecordUsingPreparedStatement  The modified records are/is: 1 |

**Requirement:** Develop a JDBC program which updates the record in the EMP table we need to update the name to Anjireddy whose EMP no is 1. [Use prepared statement].

**Program:**

|  |
| --- |
| import java.sql.\*;  public class UpdateRecordUsingPreparedStatement{  public static void main(String args[])throws SQLException{  //step1:register the driver  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  System.out.println("driver is registered");  //step2:get the connection from database  Connection objcon=DriverManager.getConnection  ("jdbc:oracle:thin:@localhost:1521:xe","system","manager");  System.out.println("i got the connection");  //step3:create the preparedstatement object  PreparedStatement objpstmt=objcon.prepareStatement("update emp set ename=? where eno=?");  System.out.println("preparedstatement object is created");  //step4:supply values to the questionmarks  objpstmt.setString(1,"anjireddy");  objpstmt.setInt(2,1);  System.out.println("values are supplied");  //step5:execute the query  int noofrecordsmodified=objpstmt.executeUpdate();  System.out.println("query is executed with the help of executeUpdate() method");  System.out.println("the modified records are/is: " +noofrecordsmodified);  //step6:close the connection  objcon.close();  System.out.println("connection closed");  }  }  **Note:** Before update the record the table should be committing.  **Output:**  D:\jdbc>java UpdateRecordUsingPreparedStatement  Driver is registered  I got the connection  PreparedStatement object is created  The values are supplied  Query is executed with the help of executeUpdate() method  The modified records are/is: 1  Connection closed |

**Requirement**: Develop a JDBC program which deletes the record from EMP table whose eno is 2 to implement this we have to use PreparedStatement.

**Program:**

|  |
| --- |
| import java.sql.\*;  public class DeleteRecordUsingPreparedStatement{  public static void main(String args[])throws SQLException{  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  Connection objcon=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","manager");  PreparedStatement objpstmt=objcon.prepareStatement("delete emp where eno=?");  objpstmt.setInt(1,2);  int noofrecordsmodified=objpstmt.executeUpdate();  System.out.println("row is delected");  System.out.println("the modified records are/is: " +noofrecordsmodified);  objcon.close();  }  }  **Output:**  D:\jdbc>javac DeleteRecordUsingPreparedStatement.java  D:\jdbc>java DeleteRecordUsingPreparedStatement  Row is deleted  The modified records are/is: 1 |

**Requirement:** Develop a JDBC program which can retrieve the records from the EMP table using PreparedStatement.

**Program:**

|  |
| --- |
| import java.sql.\*;  public class SelectRecordUsingPreparedStatement{  public static void main(String args[])throws SQLException{  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  Connection objcon=DriverManager.getConnection  ("jdbc:oracle:thin:@localhost:1521:xe","system","manager");  PreparedStatement objpstmt=objcon.prepareStatement("select \*from emp");  ResultSet objrs=objpstmt.executeQuery();  System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  System.out.println("eno"+"\t\t"+"ename"+"\t\t"+"salary");  System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  while(objrs.next()){  System.out.print(objrs.getInt("eno")+"\t\t");  System.out.print(objrs.getString("ename")+"\t\t");  System.out.println(objrs.getFloat("salary")+"\t\t");  }  System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  objcon.close();  }  }  **Output:**  D:\jdbc>javac SelectRecordUsingPreparedStatement.java  D:\jdbc>java SelectRecordUsingPreparedStatement  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  eno ename salary  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  2 vbr 40000.0  1 vijayabhaskar 40000.0  3 Vijay 50000.0  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |

**Requirement:** Develop a JDBC program which can retrieve the one record from the EMP table using PreparedStatement.

**Program:**

|  |
| --- |
| import java.sql.\*;  public class SelectOneRecordUsingPreparedStatement{  public static void main(String args[])throws SQLException{  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  Connection objcon=DriverManager.getConnection  ("jdbc:oracle:thin:@localhost:1521:xe","system","manager");  PreparedStatement objpstmt=objcon.prepareStatement("select \*from emp where eno=?");  objpstmt.setInt(1,1);  ResultSet objrs=objpstmt.executeQuery();  System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  System.out.println("eno"+"\t\t"+"ename"+"\t\t"+"salary");  System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  while(objrs.next()){  System.out.print(objrs.getInt("eno")+"\t\t");  System.out.print(objrs.getString("ename")+"\t\t");  System.out.println(objrs.getFloat("salary")+"\t\t");  }  System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  objcon.close();  }  }  **Output:**  D:\jdbc>javac SelectOneRecordUsingPreparedStatement.java  D:\jdbc>java SelectOneRecordUsingPreparedStatement  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  eno ename salary  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  1 vijayabhaskar 40000.0  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |

* Whenever we send a query to the database it carries out the following steps in the database.

**Step 1:** After the database server has received the query the database server checks are there any syntax errors are available. If the syntax errors are available the database server throws an error message.

**Step 2:** If there are no syntax errors the database server checks whether all the required objects are available or not. [All the table names or column names etc].

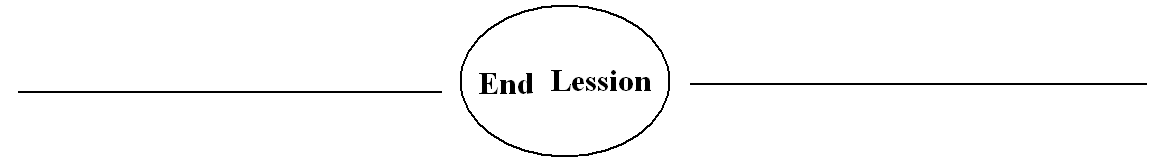
**Step 3:** The database server passes the query in machine understandable language.

**Step 4:** Oracle database server generates multiple algorithms to retrieve the data from the database.

**Step 5:** The oracle database server chooses (picks) the best suitable algorithm based on the conditions.

**Step 6:** Oracle database server executes the algorithm and gets the records from the **dbf** **file** and return the records to the client.

* Whenever we use a prepared statement in java the JDBC driver converts prepared statement into bind variables.
* In oracle we use the following query as bind variable.
  + - **Insert into product values(:vpid, :vpname, :vprice);**
* The values of the bind variables replaced dynamically by oracle database server.



**4. Callable Statement**

**Why CallableStatement are used?**

**Reply:** CallableStatement is used to execute the procedures.

**What is the advantage of using procedures?**

**Reply:** As part of procedure we provide the business logic. Procedure improves the performance of the application.

**Requirement:** Develop a JDBC program which calls the procedure which is available in the database. The name of procedure is myproc.

**Note:** The database developer develops a procedure whose name is myproc. This procedure doesn’t take any parameter.

**Create Procedure without parameters:**

|  |
| --- |
| SQL> create or replace procedure myproc  2 as  3 begin  4 insert into product values(1,'pone',1000);  5 end myproc;  6 /  Procedure created. |

* The procedure starts with “begin” and ends with “end” and also “slash”.
* To execute the procedures from oracle we use the following command.

**Execute the procedure:**

SQL> exec myproc;

PL/SQL procedure successfully completed.

**Procedure to use callable statement:**

**Step 1:** Register the driver.

**Step 2:** Get the connection from database.

**Step 3:** Create the callable statement object and supply the name of the procedure as input.

To call the procedure we have to follow a specific syntax.

{call procedurename}

(Or)

{call procedurename(?’?)}

Here “call” is keyword.

**Step 4:** If the procedure is having parameters supply the values to parameters.

**Step 5:** Ask JDBC driver to execute the procedure.

**Step 6:** Close the connection.

**Java Program:**

|  |
| --- |
| import java.sql.\*;  public class CallProcedureWithOutArgumentsUsingCallableStatement{  public static void main(String args[])throws SQLException{  //Step1:register the driver  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  System.out.println("driver is register");  //step2:get the connection from database  Connection objcon=DriverManager.getConnection  ("jdbc:oracle:thin:@localhost:1521:xe","system","manager");  System.out.println("i got the connectio from database : "+objcon);  //step3:create CallableStatement object with the help of prepareCall() method.  //To calls the procedure using “call” keyword.  CallableStatement objcstmt=objcon.prepareCall("{call myproc}");  System.out.println("procedure successfully called");  //step4:execute the procedure with the help of execute() method.  //execute() method is used to execute the procedures.  objcstmt.execute();  System.out.println("procedure successfully executed");  //step5:close the connection  objcon.close();  System.out.println("connection closed");  }  }  **Output:**  D:\jdbc>javac CallProcedureWithOutArgumentsUsingCallableStatement.java  D:\jdbc>java CallProcedureWithOutArgumentsUsingCallableStatement  Driver is register  I got the connection from database : oracle.jdbc.driver.T4CConnection@198dfaf  Procedure successfully called  Procedure successfully executed  Connection closed |

**Procedures which takes parameters:**

|  |
| --- |
| SQL> create or replace procedure myproc(vpid in number,vpname in varchar2,vprice  in number)  2 as  3 begin  4 insert into product values(vpid,vpname,vprice);  5 end myproc;  6 /  Procedure created. |

* To execute the procedure with parameters from oracle we use the following command.

**Execute the procedure with parameters:**

SQL> exec myproc(2,'ptwo',2000);

PL/SQL procedure successfully completed.

**Note:** Question marks indicate the parameters of the procedure.

**Program:** The following is an example of JDBC program which calls the procedure with parameters.

|  |
| --- |
| import java.sql.\*;  public class CallProcedureWithParametersUsingCallableStatement{  public static void main(String args[])throws SQLException{  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  Connection objcon=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","manager");  CallableStatement objcstmt=objcon.prepareCall("{call myproc(?,?,?)}");  objcstmt.setInt(1,3);  objcstmt.setString(2,"pthree");  objcstmt.setFloat(3,30000);  //objcstmt.setInt(3,30000);  objcstmt.execute();  objcon.close();  }  } |

* A procedure can take two different types of inputs they are **in** parameters and **out** parameters. In parameters are represented with a keyword **in** output parameters are represented with a keyword **out**.

**Requirement**: Develop a procedure with takes three parameters. Two parameters are of type **input** one parameter is of type **output**. The procedure should be able to add two input parameters and place the result in output parameter.

|  |
| --- |
| SQL> create or replace procedure myproc(vno1 in number,vno2 in number,result out  number)  2 as  3 begin  4 result := vno1+vno2;  5 end myproc;  6 /  Procedure created. |

**Procedure to execute the above procedure:**

**Step 1:** To handle the output variables we have to create a variable in oracle.

**SQL> variable vresult number;**

**Step 2:** Call the procedure by using **exec** command.

**SQL> exec myproc(10,20,:vresult);**

**Step 3:** After the procedure is successfully executed it places the data in vresult variable.

**Step 4:** To see the value of vresult we use the following command.

**SQL> print vresult;**

* The following shows how to use a procedure which is having **in** and **out** parameters.
* In a procedure if we are having out parameter we have to register all the out parameter. To register the out parameters we use a method **registerOutParameter().**

**Requirement:** Develop a procedure with takes three parameters. Two parameters are of type **input** one parameter is of type **output**. The procedure should be able to add two input parameters and place the result in output parameter.

**Program:**

|  |
| --- |
| import java.sql.\*;  public class CallProcedureWithInAndOutParametersUsingCallableStatement{  public static void main(String args[])throws SQLException{  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  Connection objcon=DriverManager.getConnection  ("jdbc:oracle:thin:@localhost:1521:xe","system","manager");  CallableStatement objcstmt=objcon.prepareCall("{call myproc(?,?,?)}");  objcstmt.setInt(1,10);  objcstmt.setInt(2,20);  //step4:register the output variable  objcstmt.registerOutParameter(3,Types.NUMERIC);  System.out.println("output variable is registered");  objcstmt.execute();  System.out.println("sum of vno1 and vno2 is :"+objcstmt.getInt(3));  objcon.close();  }  } |

**5. Bi-Directional ResultSet**

* There are two types of ResultSet objects are available. They are:

1. Forward only ResultSet
2. Bi-directional ResultSet

**What is forward only ResultSet?**

* If the ResultSet object is able to move only in one direction that ResultSet is called as forward only ResultSet. (**or**)
* A ResultSet which will be able to go forward is called as forward only ResultSet.

**What is Bi-directional ResultSet?**

* If the ResultSet object is able to move in both the directions that is forward as well as backward then the ResultSet object is called as Bi-directional ResultSet. (**or**)
* A ResultSet which will be able to go forward as well as backward is called as bi-directional ResultSet.
* By using **rs.getRow ()** method I can find where the ResultSet pointer is pointing. [To which the record the ResultSet is pointing].
* **Previous ()** method is used to go the previous record.
* By default the ResultSet object is of type **forward only ResultSet**.
* If it is forward only ResultSet we can use next (**)** method only. If we try to use previous () method we get an exception saying **invalid operation for forward only ResultSet.**
* If we want to use previous (), absolute () methods we have to create Bi-directional ResultSet.
* To create the Bi-directional ResultSet at the time of statement object creation we have to supply some arguments.
* The first argument to the create statement method is ResultSet type. We can use the following three ResultSet types as the first argument.

**TYPE\_FORWARD\_ONLY**

### TYPE\_SCROLL\_SENSITIVE

### TYPE\_SCROLL\_INSENSITIVE

**Note:** These instance variables are available in ResultSet interface.

* The second argument for create statement method is ResultSet concurrency. We can supply the following variables as ResultSet concurrency values.

### CONCUR\_READ\_ONLY

### CONCUR\_UPDATABLE

**NOTE:** These variables are available in ResultSet interface.

### By default the ResultSet object takes TYPE\_FORWARD\_ONLY ResultSet and CONCUR\_READ\_ONLY.

**What is sensitive ResultSet?**

* Whenever there is change in the database immediately if the change is being reflected in our java application then the ResultSet is called sensitive ResultSet.

**What is insensitive ResultSet?**

* After the java program has acquired the data if there are any changes in the database those changes will not be reflected in our java application these ResultSet are called as insensitive ResultSet.

**What is CONCUR\_UPDATABLE?**

* In the java application we can update the records through ResultSet object. If we are able to update the records through ResultSet object is called as concurrent updatable.

**What is CONCUR\_READ\_ONLY?**

* If we are not able to update the records using ResultSet object then it is called CONCUR\_READ\_ONLY.
* We can find all these variables in the ResultSet API.

**Requirement:** Develop a JDBC program to retrieve the records from ResultSet object. Use previous() method without passing parameters to the createStatement() method.

**Program:**

|  |
| --- |
| import java.sql.\*;  public class retrieverecords4{  public static void main(String args[])throws SQLException{  //step1:register the driver  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  System.out.println("driver is registered");  //step2:get the connection from database  Connection objcon=DriverManager.getConnection  ("jdbc:oracle:thin:@localhost:1521:xe","lms","abc");  System.out.println("i got connection from database :"+objcon);  //step3:create statement object  Statement objstmt=objcon.createStatement();  //step4:execute the query  ResultSet objrs=objstmt.executeQuery("select \*from product");  System.out.println("----------------------------------------------------");  System.out.println("pid"+"\t"+"pname"+"\t\t"+"price"+"\t");  System.out.println("----------------------------------------------------");  objrs.next();  objrs.next();  System.out.println(objrs.getRow());  objrs.previous();  }  }  **Output:**  E:\jdbc>java retrieverecords4  Driver is registered  I got connection from database: oracle.jdbc.driver.T4CConnection@1b90b39  ----------------------------------------------------  pid pname price  ----------------------------------------------------  2  Exception in thread "main" java.sql.SQLException: **Invalid operation for forward only** **ResultSet: previous.** |

* To resolve the above error we have to pass the parameters while creating the statement object (or) PreparedStatement object. Then only the records which are available in the ResultSet object can access both directions. The following program shown below.

**Program:**

|  |
| --- |
| import java.sql.\*;  public class retrieverecords9{  public static void main(String args[])throws SQLException{  //step1:register the driver  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  System.out.println("driver is registered");  //step2:get the connection from database  Connection objcon=DriverManager.getConnection  ("jdbc:oracle:thin:@localhost:1521:xe","lms","abc");  System.out.println("i got connection from database :"+objcon);  //step3:create statement object  Statement objstmt=objcon.createStatement  (ResultSet.TYPE\_SCROLL\_INSENSITIVE,ResultSet.CONCUR\_READ\_ONLY);  //step4:execute the query  ResultSet objrs=objstmt.executeQuery("select \*from product");  System.out.println("----------------------------------------------------");  System.out.println("pid"+"\t"+"pname"+"\t\t"+"price"+"\t");  System.out.println("----------------------------------------------------");  objrs.next();  objrs.next();  objrs.next();  System.out.println("cursor pointing position is : "+objrs.getRow());  objrs.previous();  objrs.previous();  System.out.println("cursor pointing position is : "+objrs.getRow());  }  }  **Output:**  E:\jdbc>java retrieverecords9  Driver is registered  I got connection from database: oracle.jdbc.driver.T4CConnection@18fe7c3  ----------------------------------------------------  pid pname price  ----------------------------------------------------  Cursor pointing position is: 3  Cursor pointing position is: 1 |

* As part of JDBC API the absolute() method helps us to move to a specific record in the ResultSet object. For example rs.absolute(5); will move the ResultSet pointer to 5th record.

**Sensitive program:** The following java program is a sensitive java program. Whenever there are changes in database it will be reflected in java application also.

|  |
| --- |
| import java.sql.\*;  public class Sensitive{  public static void main(String args[])throws Exception{  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  Connection con=DriverManager.getConnection  ("jdbc:oracle:thin:@localhost:1521:xe","lms","abc");  Statement stmt=con.createStatement  (ResultSet.TYPE\_SCROLL\_SENSITIVE,ResultSet.CONCUR\_READ\_ONLY);  ResultSet rs=stmt.executeQuery("select pid,pname,price from product");  while(rs.next()){  System.out.println("press any key to continue");  System.in.read();  System.in.read();  rs.refreshRow();  System.out.print(rs.getString(1)+"\t");  System.out.print(rs.getString(2)+"\t");  System.out.println(rs.getString(3)+"\t");  System.out.println("\n");  }  }  } |

**Execution process:**

* While running the java application the application will ask the press any key in this time we go to the database and modify the data in the database and commit data. After commit the data in the database come back to the java application press any key then the java application will displays the data and also the modify data.

**Note:** In sensitive application only “update” command is working. “Select” and “delete” commands will not working. (That means the insert records and delete records will not reflect in the java application only update records will be reflected).

**Output:**

E:\jdbc>java Sensitive

Press any key to continue

1 pone 1000

Press any key to continue

2 ptwo 2000

Press any key to continue

5 pfive 5000

**Note:** refreshRow() method is not supported for insensitive application. If we use the refreshRow() in the java application the compiler will throws the following error.

Exception in thread "main" java.sql.SQLException: Unsupported feature: refreshRow

**In sensitive program:**

|  |
| --- |
| import java.sql.\*;  public class InSensitive{  public static void main(String args[])throws Exception{  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  Connection con=DriverManager.getConnection  ("jdbc:oracle:thin:@localhost:1521:xe","lms","abc");  Statement stmt=con.createStatement  (ResultSet.TYPE\_SCROLL\_INSENSITIVE,ResultSet.CONCUR\_READ\_ONLY);  ResultSet rs=stmt.executeQuery("select \*from product");  while(rs.next()){  System.out.println("press any key to continue");  System.in.read();  System.in.read();  System.out.print(rs.getString(1)+"\t");  System.out.print(rs.getString(2)+"\t");  System.out.println(rs.getString(3)+"\t");  System.out.println("\n");  }  }  } |

* Execution process is same like sensitive application.

**Output:**

E:\jdbc>java InSensitive

Press any key to continue

1 pone 1000

Press any key to continue

2 ptwo 2000

Press any key to continue

5 five 3000

**6. Bi-directional ResultSet using PreparedStatement**

**Sensitive program:**

|  |
| --- |
| import java.sql.\*;  public class SensitiveUsingPrepareStatement{  public static void main(String args[])throws Exception{  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  Connection con=DriverManager.getConnection  ("jdbc:oracle:thin:@localhost:1521:xe","lms","abc");  PreparedStatement pstmt=con.prepareStatement("select pid,pname,price from product",ResultSet.TYPE\_SCROLL\_SENSITIVE,ResultSet.CONCUR\_READ\_ONLY);  ResultSet rs=pstmt.executeQuery();  while(rs.next()){  System.out.println("press any key to continue");  System.in.read();  System.in.read();  rs.refreshRow();  System.out.print(rs.getString(1)+"\t");  System.out.print(rs.getString(2)+"\t");  System.out.println(rs.getString(3)+"\t");  System.out.println("\n");  }  }  } |

* The above program executions like same as sensitive program.

**Output:**

**Before execution:**

PID PNAME PRICE

---------- ---------- ----------

1 pone 1000

2 ptwo 2000

3 five 3000

* In the middle of execution we update the following record.

**After execution:**

SQL> select \*from product;

PID PNAME PRICE

---------- ---------- ----------

1 pone 1000

2 ptwo 2000

5 five 3000

**Insensitive using PreparedStatement:**

|  |
| --- |
| import java.sql.\*;  public class InSensitiveUsingPrepareStatement{  public static void main(String args[])throws Exception{  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  Connection con=DriverManager.getConnection  ("jdbc:oracle:thin:@localhost:1521:xe","lms","abc");  PreparedStatement pstmt=con.prepareStatement("select \*from product",ResultSet.TYPE\_SCROLL\_INSENSITIVE,ResultSet.CONCUR\_READ\_ONLY);  ResultSet rs=pstmt.executeQuery();  while(rs.next()){  System.out.println("press any key to continue");  System.in.read();  System.in.read();  System.out.print(rs.getString(1)+"\t");  System.out.print(rs.getString(2)+"\t");  System.out.println(rs.getString(3)+"\t");  System.out.println("\n");  }  }  } |

**Output:**

SQL> select \*from product;

PID PNAME PRICE

---------- ---------- ----------

1 pone 1000

2 ptwo 2000

5 five 3000

* While running the java application we can **update** the record in the database, insert record into the database, and delete the record into the database. The following three programs are shown below.

**Update record while running the java program in the database:**

**Program:**

|  |
| --- |
| //update the record using CONCUR\_UPDATABLE(bi-directional)  import java.sql.\*;  public class update1{  public static void main(String args[])throws Exception{  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","lms","abc");  Statement stmt=con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE,ResultSet.CONCUR\_UPDATABLE);  ResultSet rs=stmt.executeQuery("select pid,pname,price from product");  rs.next();  rs.next();  rs.updateString(1,"2");  rs.updateString(2,"vijaya");  rs.updateString(3,"40000");  rs.updateRow();  System.out.println("second record is successfully updated in the database");  rs.next();  rs.updateString("pid","3");  rs.updateString("pname","bhaskar");  rs.updateString("price","50000");  rs.updateRow();  System.out.println("third record is successfully updated in the database");  }  }  **Output:**  E:\jdbc>java update1  Second record is successfully updated in the database  Third record is successfully updated in the database  **Before update the product table:**  SQL> select \*from product;  PID PNAME PRICE  ---------- ---------- ----------  1 pone 1000  2 ptwo 2000  3 pthree 3000  **After update the product table:**  SQL> select \*from product;  PID PNAME PRICE  ---------- ---------- ----------  1 pone 1000  2 vijaya 40000  3 Bhaskar 50000 |

**Insert record into the database while running the java application:**

**Program:**

|  |
| --- |
| //insert the record using CONCUR\_UPDATABLE(bi-directional)  import java.sql.\*;  public class Insert1{  public static void main(String args[])throws Exception{  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","lms","abc");  Statement stmt=con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE,ResultSet.CONCUR\_UPDATABLE);  ResultSet rs=stmt.executeQuery("select pid,pname,price from product");  //moveToInsertRow() method is used to Moves the cursor to the insert row.  rs.moveToInsertRow();  //Updates the designated column with a String value. The updater methods are  //used to update column values in the current row or the insert row  rs.updateString(1, "2");  rs.updateString(2, "ptwo");  rs.updateString(3, "2000");  //Inserts the contents of the insert row into this ResultSet object and into  //the database. The cursor must be on the insert row when this method is called.  rs.insertRow();  System.out.println(" record is successfully inserted in the database");  }  }  **Output:**  E:\jdbc>java Insert1  Record is successfully inserted in the database  **Before insert record into the database:**  SQL> select \*from product;  PID PNAME PRICE  ---------- ---------- ----------  1 pone 1000  2 vijaya 40000  3 Bhaskar 50000  **After insert the record into the database:**  SQL> select \*from product;  PID PNAME PRICE  ---------- ---------- ----------  1 pone 1000  2 vijaya 40000  3 Bhaskar 50000  2 ptwo 2000 |

**Delete the record while running the java application:**

**Program:**

|  |
| --- |
| //delete the record using CONCUR\_UPDATABLE(bi-directional)  import java.sql.\*;  public class Delete{  public static void main(String args[])throws Exception{  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","lms","abc");  Statement stmt=con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE,ResultSet.CONCUR\_UPDATABLE);  ResultSet rs=stmt.executeQuery("select pid,pname,price from product");  rs.next();  rs.next();  rs.deleteRow();  System.out.println(" record is successfully deleted in the database");  }  }  **Output**:  E:\jdbc\bi>java delete  Record is successfully deleted in the database  **Before delete record from the database:**  SQL> select \*from product;  PID PNAME PRICE  ---------- ---------- ----------  1 pone 1000  2 vijaya 40000  3 Bhaskar 50000  2 ptwo 2000  **After delete record from database:**  SQL> select \*from product;  PID PNAME PRICE  ---------- ---------- ----------  1 pone 1000  3 Bhaskar 50000  2 ptwo 2000 |

**7. TRANSACTIONS**

* Executing sequence of steps is called transactions.
* As part of oracle data base server the database server can perform row level locking. If someone is updating a row in the database at the same time through another session if someone is updating the same record, Oracle will not allow us until the first user’s session is ended.
* Till now in all the examples the JDBC driver is taken care of the transaction. In JDBC applications by default the JDBC driver starts the transactions as well as ends the transactions.

**The default behavior of JDBC driver transactions is:**

* In case of JDBC applications JDBC driver is responsible to start the transactions. By default the JDBC driver starts the transaction when we establish the connection with database.
* By default JDBC driver is responsible to end the transactions. In case of JDBC, the JDBC driver commits the transactions after executing every SQL query.
* If required we can override the default behavior of JDBC driver transaction management.
* The following JDBC program shows how to manage the transactions by the programmer.

|  |
| --- |
| * **In oracle we should maintain the transaction**. **In java application by default JDBC driver manage the transactions.** * Whenever we establish connection with database the JDBC driver starts the transaction as well as ends the transaction after every sql statement is executed. * JDBC driver starts the transaction once again once if ended the transaction. * In java if required user can control the transaction if the user is able to control the transaction those transactions are called as user defined transactions. * By using setAutoCommit(false) we can start the transaction in our own. If the user wants to start the transaction we use a code setAutoCommit (false) if we use setAutoCommit(false) JDBC driver will not start the transaction. |

**Program:** The following JDBC program shows how to manage the transactions by the programmer (This is successfully program).

|  |
| --- |
| import java.sql.\*;  public class TransferFundsSuccessfully{  public static void main(String args[])throws Exception{  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  Connection con=DriverManager.getConnection ("jdbc:oracle:thin:@localhost:1521:xe","system","manager");  //whenever we establish the connection with database by default the JDBC driver starts //the transaction as well as ends the transaction. In java if required we can control the //transaction. By using setAutoCommit(false) we can start the transaction on our own.  con.setAutoCommit(false);  try{  Statement stmt=con.createStatement();  stmt.executeUpdate("update acctable set balance=90000 where accno=123");  int i=10/10;  stmt.executeUpdate ("update acctable set balance=10010 where accno=456");  //if the user starts the transaction **user is responsible to end the transaction**. User can //end the transaction by **commit (or) rollback** that means con.commit() (or) con.rollback().  con.commit();  System.out.println ("funds successfully transferred");  }catch(Exception e){  e.printStackTrace();  System.out.println ("problem is occurred while transferring the balance");  con.rollback();  }  }  }  **OUTPUT:**  D:\jdbc>java TransferFundsSuccessfully  Funds successfully transferred |

**PROGRAM:** The following JDBC program shows how to manage the transactions by the programmer (This is failure program).

|  |
| --- |
| import java.sql.\*;  public class TransferFundsFailure{  public static void main(String args[])throws Exception{  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","manager");  con.setAutoCommit(false);  try{  Statement stmt=con.createStatement();  stmt.executeUpdate("update acctable set balance=90000 where accno=123");  int i=10/0;  stmt.executeUpdate("update acctable set balance=10010 where accno=456");  con.commit();  System.out.println("funds successfully transfered");  }catch(Exception e){  e.printStackTrace();  System.out.println("problem is occured while transfering the balance");  con.rollback();  }  }  }  **OUTPUT:**  D:\jdbc>java TransferFundsFailure  java.lang.ArithmeticException: / by zero  at TransferFundsFailure.main(TransferFundsFailure.java:10)  Problem is occurred while transferring the balance |

**8. Batch Updates**

* Most of java developers prefer writing the business logic in java.
* If we write the business logic in java if we need to perform the updating of records in the database every time we have to interact with database server.
* If we need to update 10,000 thousand records in the database instead of sending 10,000 queries independently we group all the queries and send it to the database.
* **With this approach we will be able improve the performance of the application**.

**Procedure to use batch updates:**

**Step 1:** Whatever queries we want to send it to the database add all the queries to batch.

* A batch can contain any number of queries after we make a sufficient batch with queries ask the JDBC driver to execute the batch.

**Step 2:** To execute the batch we need to use a method executebatch().

* The following JDBC program is an example of the batch updates.

**Program:**

|  |
| --- |
| import java.sql.\*;  public class BatchUpdates{  public static void main(String args[])throws Exception{  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","ma nager");  Statement stmt=con.createStatement();  String q1="create table emp(eid number(10),ename varchar2(20),esalary number(10,2))";  String q2="insert into emp values(1,'vijayabhaskar',30000)";  String q3="insert into emp values(2,'bhaskarareddy',40000)";  String q4="insert into emp values(3,'ANJIREDDY',50000)";  String q5="insert into emp values(4,'ramireddy',60000)";  String q6="update emp set esalary=35000 where eid=1";  String q7="delete emp where eid=1";  //String q8="select \*from emp";  stmt.addBatch(q1);  stmt.addBatch(q2);  stmt.addBatch(q3);  stmt.addBatch(q4);  stmt.addBatch(q5);  stmt.addBatch(q6);  stmt.addBatch(q7);  //stmt.addBatch(q8);  System.out.println("all commands is added to the batch successfully");  stmt.executeBatch();  System.out.println ("all commands successfully executed");  }  }  **OUTPUT:**  D:\jdbc>java BatchUpdates  All commands is added to the batch successfully  All commands successfully executed  **ERROR**  D:\jdbc>java BatchUpdates  All commands is added to the batch successfully  Exception in thread "main" java.sql.BatchUpdateException: invalid batch command:  **Invalid SELECT batch command 7** |

**9. Remove hard coding**

* Till now whatever JDBC programs we have to written we have hard coded the DRIVER NAME, URL, USERNAME, and PASSWORD.
* If we hardcode the java application/program if the client has ask to connect to another database user by using the same program we cannot do that.
* If we want to develop an application which connects to any database without changing the java code we have to remove the hardcoded values.
* Class.forname () is used to register the JDBC driver. According to the JDBC API whenever we use class.forName () it should be able to create the object for the driver class and register the driver.
* Every driver vendor will provide a static block as part of the driver class.
* The following is the same code for the same package oracle.jdbc.driver.

|  |
| --- |
| **public class OracleDriver implements Driver{**  **static{**  **oracleDriver o=new Oracledriver();**  **DriverManager.registerDriver(o);**  **}**  **}** |

**The following program is able to remove the hardcoded values of Driver name, Url, Username, Password.**

**Program:**

|  |
| --- |
| import java.sql.\*;  public class RemoveHardCode{  public static void main(String args[])throws Exception{  //get property is a static method of system class. It takes input as string and it returns a string**.**  String drivername =System.getProperty("drivername");  String url =System.getProperty("url");  String username =System.getProperty("username");  String pwd=System.getProperty("pwd");  Class.forName(drivername);  Connection objcon=DriverManager.getConnection(url,username,pwd);  System.out.println("connection object is :"+objcon);  }  }  **OUTPUT:**  D:\jdbc>javac RemoveHardCode.java  D:\jdbc>java -Ddrivername=oracle.jdbc.driver.OracleDriver -Durl=jdbc:oracle:thin:@localhost:1521:xe -Dusername=system -Dpwd=manager RemoveHardCode  Connection object is: oracle.jdbc.driver.T4CConnection@e48e1b |

**Another way of registering the driver by removing the hard coding:**

|  |
| --- |
| import java.sql.\*;  public class Dbconnect{  public static void main(String args[])throws Exception{  String drivername =System.getProperty("drivername");  String url =System.getProperty("url");  String username =System.getProperty("username");  String pwd=System.getProperty("pwd");  Class c=Class.forName(drivername);  Object o=c.newInstance();  Driver d=(Driver)o;  DriverManager.registerDriver(d);  Connection con=DriverManager.getConnection(url,username,pwd);  System.out.println("connection object is : "+con);  }  }  **Output:**  D:\>java -Ddrivername=oracle.jdbc.driver.OracleDriver -Durl=jdbc:oracle:thin:@lo  calhost:1521:xe -Dusername=vbr -Dpwd=123 Dbconnect  connection object is : oracle.jdbc.driver.T4CConnection@f38798 |

**10. Metadata**

**Metadata**: Data about data is called as metadata. As part of JDBC API sun micro system has provided two interfaces for Meta data. They are:

1. **DatabaseMetaData**
2. **ResultSetMetaData**

* Giving more information about data is called as metadata.

<marks>90</marks>

<fathername>ramesh</fathername>

<houseno>97</houseno>

<qualification>mca</qualification>

**What is database metadata?**

* Database metadata is an object. Which provides information about the database. [Database Meta data gives more information about to which database we have to connect].

**Program:**

|  |
| --- |
| import java.sql.\*;  public class DatabaseMetaDataApplication{  public static void main(String args[])throws Exception{  //step1:register the driver  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  System.out.println("driver is registered");  //step2:get the connection from database  Connection con=DriverManager.getConnection("jdbc:oracl:thin:@localhost:1521:xe","system","man ager");  System.out.println("i got the connection from database");  System.out.println("connection object is :"+con);  //step3:create database metadata object  DatabaseMetaData dbmd=con.getMetaData();  System.out.println("metadata object is :"+dbmd);  System.out.println("----------------------------------------------------------");  System.out.println("the driver details are");  System.out.println("----------------------------------------------------------");  System.out.println("driver name is : "+dbmd.getDriverName());  System.out.println("driver version is : "+dbmd.getDriverVersion());  System.out.println("driver major version is :"+dbmd.getDriverMajorVersion());  System.out.println("driver minor version is :"+dbmd.getDriverMinorVersion());  System.out.println("----------------------------------------------------------");  System.out.println("the database details are");  System.out.println("----------------------------------------------------------");  System.out.println("product name of the database : "+dbmd.getDatabaseProductName());  System.out.println("product version of the database : "+dbmd.getDatabaseProductVersion());  System.out.println("major version of the database :"+dbmd.getDatabaseMajorVersion());  System.out.println("minor version of the database :"+dbmd.getDatabaseMinorVersion());  System.out.println("----------------------------------------------------------");  System.out.println("the url details are");  System.out.println("----------------------------------------------------------");  System.out.println("the url is : "+dbmd.getURL());  System.out.println("----------------------------------------------------------");  System.out.println("the user name details are");  System.out.println("----------------------------------------------------------");  System.out.println("the user name is :"+dbmd.getUserName());  }  }  **OUTPUT:**  D:\jdbc>java DatabaseMetaDataApplication  Driver is registered  I got the connection from database  Connection object is: oracle.jdbc.driver.T4CConnection@c3c749  Metadata object is: oracle.jdbc.driver.OracleDatabaseMetaData@150bd4d  **----------------------------------------------------------**  **The driver details are**  **----------------------------------------------------------**  Driver name is: Oracle JDBC driver  Drive version is: 10.2.0.1.0XE  Drive major version is: 10  Drive minor version is: 2  **----------------------------------------------------------**  **The database details are**  **----------------------------------------------------------**  Product name of the database: Oracle  Product version of the database: Oracle Database 10g Express Edition Release 1  0.2.0.1.0 - Production  Major version of the database: 10  Minor version of the database: 2  **----------------------------------------------------------**  **The URL details are**  **----------------------------------------------------------**  The URL is: jdbc:oracl:thin:@localhost:1521:xe  **----------------------------------------------------------**  **The user name details are**  **----------------------------------------------------------**  The user name is: SYSTEM |

**What is ResultSetMetaData?**

* ResultSetMetaData gives the information about the no of columns retrieved by the query. By using ResultSetMetaData we can find the names of the columns.
* By using ResultSetMetaData we can find the data types of columns retrieved by the user.
* As part of java.sql package sun micro system has provided “Types class”.
* “As part of this class set of variables are defined”. All the variables are type static and final. We cannot change the final variable values. All the variables are initialized with their constant values.

**Requirement:** Develop a JDBC program which retrieves all the records from product table. Our jdbc application must tell about the names of the column ResultSet, datatypes of the ResultSet for the user.

**Program:**

|  |
| --- |
| import java.sql.\*;  public class ExampleResultSetMetaData{  public static void main(String args[])throws Exception{  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  Connection con=DriverManager.getConnection("jdbc:oracl:thin:@localhost:1521:xe","vbr","123 ");  Statement stmt=con.createStatement();  ResultSet rs=stmt.executeQuery("select \*from product");  ResultSetMetaData rsmd=rs.getMetaData();  int noofcolumns=rsmd.getColumnCount();  System.out.println("no of columns is/are : "+noofcolumns);  for(int i=1;i<=noofcolumns;i++){  System.out.println(i+" column name is :"+rsmd.getColumnName(i));  int ctype=rsmd.getColumnType(i);  if(ctype==Types.NUMERIC){  System.out.println(i+" column type is:INTEGER");  }  if(ctype==Types.VARCHAR){  System.out.println(i+" column type is:VARCHAR");  }  }  }  }  **OUTPUT:**  D:\>java ExampleResultSetMetaData  no of columns is/are : 3  1 column name is :PID  1 column type is: INTEGER  2 column name is :PNAME  2 column type is:VARCHAR  3 column name is :PRICE  3 column type is: INTEGER |

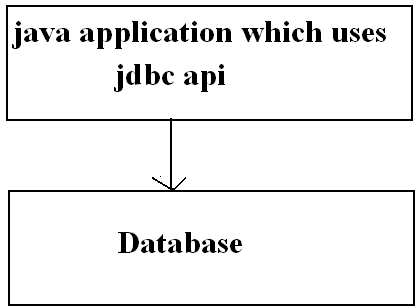
**11. JDBC Drivers**

* There are four types of drivers are available to connect to database. They are:
* **Type one** driver (jdbc-odbc bridge)
* **Type two** driver (java native API)
* **Type three** driver (java network protocol driver)
* **Type four** driver (thin driver (or) pure java driver)

**Type 1 driver**: Type one driver is called as **jdbc-odbc Bridge**.

**Requirement:** Develop a JDBC application which can communicate with database by using type one driver.

**Diagram:**

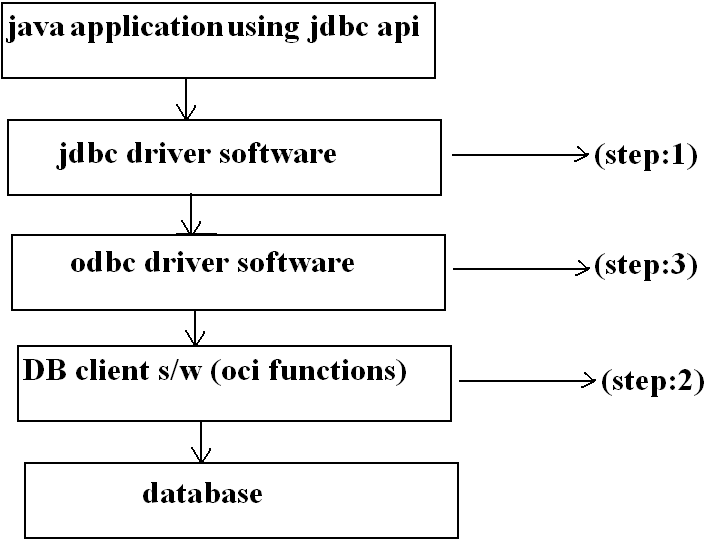


* If we want to develop a java application which communicates with database **by using** **type one driver** we have to install the following software’s in the client computer.
* **Install the database client software** this will give us the specific **CI** (call interface) **functions**.
* We need to install the **ODBC driver software**.
* We need to install the **JDBC driver software**.

**Problems with type1 driver:**

* If we develop the project using type1 driver the end user need to install all the required software in his pc. They are:
* Client software
* Odbc driver software
* JDBC driver software

**Installation Diagram:**

****

* Here java application internally communicates with JDBC driver software. JDBC driver software internally communicates with odbc driver software. ODBC driver software internally communicates with database client software. Database client software internally communicates with database.

**Procedure to develop java application which communicates database using type one driver:**

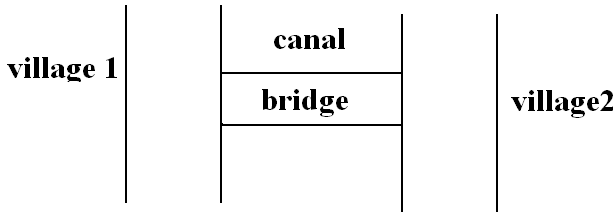
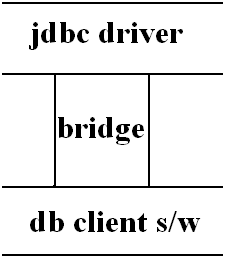
**Step 1:** Sun Microsystems has developed type one driver. We get this driver software by default in the java software.

**Step 2:** Install the oracle client software if it is not installed.

**Step 3:** In windows operating system we no need to install Odbc driver because by default we get the Odbc driver. In Linux operating system we have to install the Odbc driver software.

**Step 4:** After install the Odbc driver software configures the Odbc driver to communicate with **JDBC driver** and **database client software**. Here the driver acting as a bridge because the driver is called **jdbc-Odbc bridge**.

**Diagram:**

**What is the use of the odbc driver?**

* By using odbc driver software we can communicate any database. (Think like that if ODBC driver software is not used I wants to change my database whenever we change the database automatically the client software will be changed. Now we have to modify our java application also for this purpose we use the odbc driver software. By using odbc driver software we can communicate any database without modify our java application).

**Note:** In the interview we should never claim that we have used type one driver in the project.

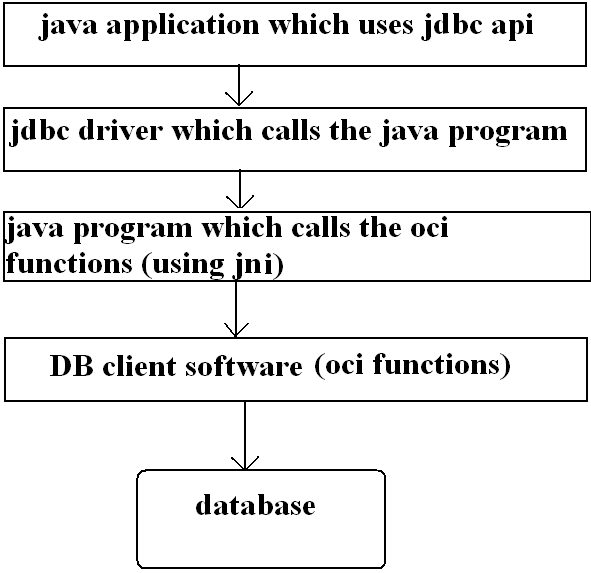
**Type 2 driver:** The JNI API helps us in calling (or) executing the c program functions from our java applications. The following is an example programs how a java application calls the functions available in **hello.dll**.

**Program:**

|  |
| --- |
| //do not practices this program in the lab.  //we understands how the java program will call the c language functions into the java application.  puclic class HelloWorld{  //native indicates this is c program function  public native void sayHello();  static{  System.loadLibrary("hello");  }  public static void main(String args[]){  //calling the sayHello() method  new HelloWorld().sayHello();  }  } |

* In java program if we want to call the native functions we need to declare functions with a keyword native.
* By using System.loadLibrary we load the **dll file** (function) into the jvm’s memory .when we execute (or) call the native functions the function will be executed from the dll file.

**Diagram:**



* In type 2 driver we need to install the client software.
* The type 2 driver uses JNI programs to call the oci functions from java applications. The JDBC driver calls the java program which is using the JNI functions.

**Note:** No company has provided the implementation of type 2 driver this is because this driver is going to change the behavior of java applications.

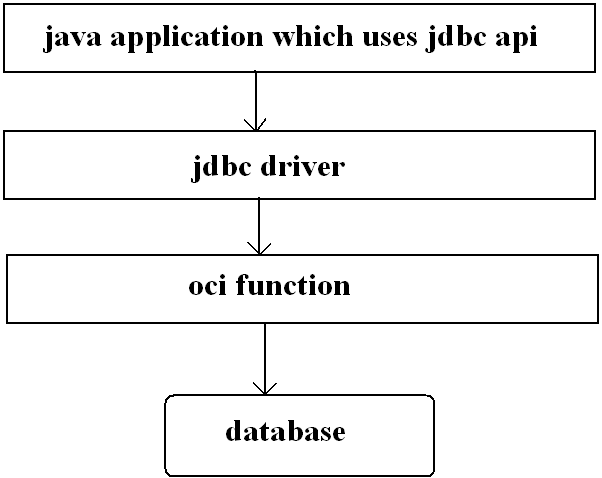
**Advantages of using type 2driver:**

* Performance of this driver better than any of the driver.

**Disadvantages of using type 2 driver:**

* This driver is platform dependent. We will not able to execute program in all the platforms.
* We have to install the client software to get the CI functions

**Type2 driver:**

****

* In type 2 driver we will be developing java application. The java application talks to JDBC driver. As part of JDBC driver JNI code is provided to call oci functions directly.

**Advantages of using type 2driver:**

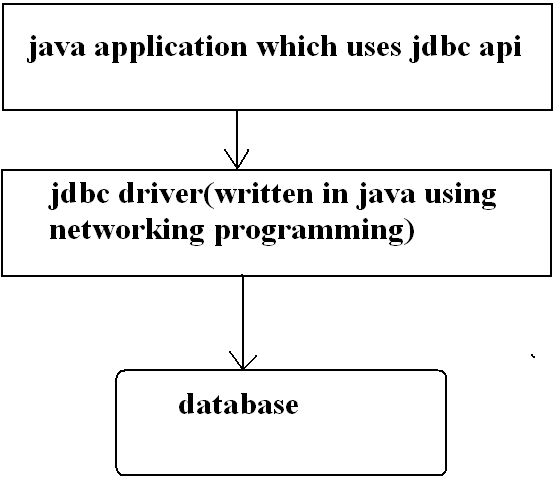
* Performance of this driver better than any of the driver.

**Disadvantages of using type 2 driver:**

* This driver is platform dependent. We will not able to execute program in all the platforms.
* We have to install the client software to get the CI functions

**Note:** It is very difficult to use type1 driver, type2 driver in web based applications.

**Type 4 driver**:



* As part of type 4 driver the java application directly communicates with JDBC driver. JDBC driver directly communicates with database server.
* In most of the projects we use type 4 driver only.

**Advantages of using type 4 driver:**

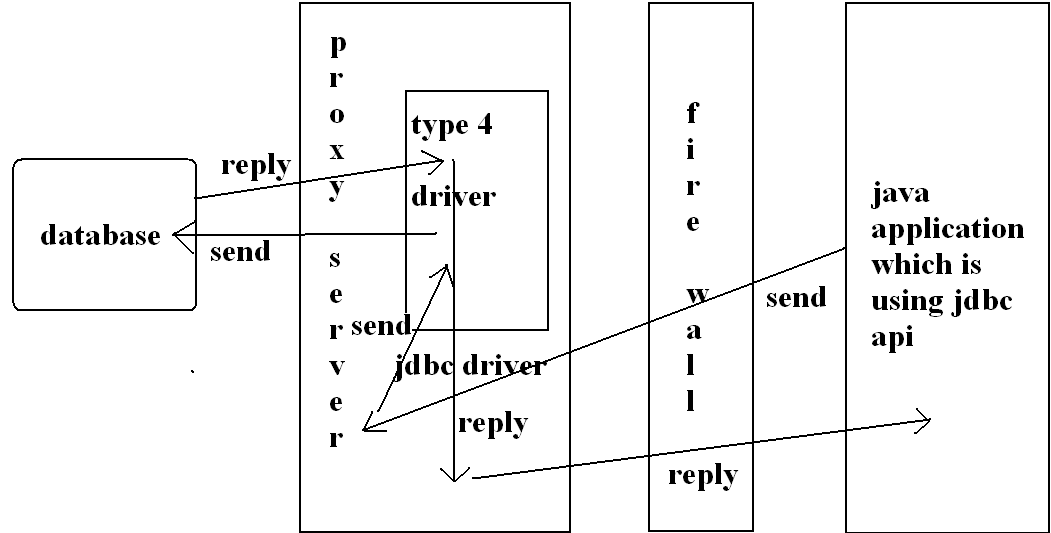
* We no need to install client software.
* This driver is platform independent (can run on any pc).
* We can use different company’s drivers.

**Disadvantages of using type 4 driver:**

* Performance of this driver is little bit slowly when compared with type1/type2 driver.
* Whenever there is change in the database we need to look for the appropriate JDBC driver to communicate with database.

**Type 3 driver:**

**Diagram:**



* In type 3 driver java application will not directly communicate with type 3 driver.
* The java application talks to proxy server and proxy server talks to type 4 driver and type 4 driver talks to database

**Advantages of using type 3 driver:**

* Pure java driver (plat form independent).
* More secured driver.
* We no need to install the client software.

**Disadvantages of using type 3 driver:**

* We need to purchase the proxy server. The cost of the server is too high.
* Performance is little bit slower when compared with type1 and type2.

Contents

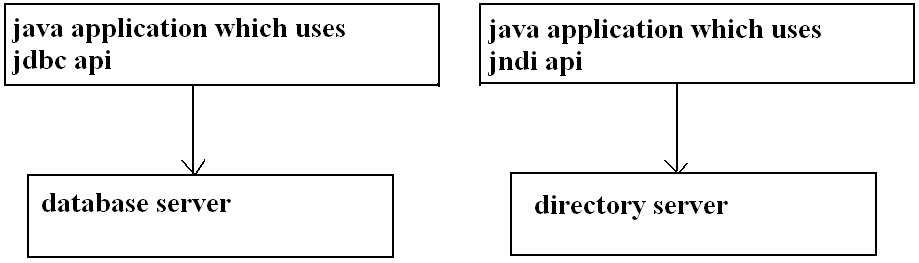
|  |  |  |
| --- | --- | --- |
| Sno | Topic Name | Page Numbers |
| 1. | JNDI | 2 |
| 2. | Connection Pool Technique | 16 |

**12. JNDI**

**What are the difference between JDBC and JNDI?**

|  |  |
| --- | --- |
| **JDBC(java Database Connectivity)** | **Jndi(java Naming Directory Interface)** |
| 1. JDBC is an API. The API consist set of classes and interfaces. The API is released by sun Microsystems. Once the API is released any one can provide the implementation. This implementation is nothing but software. This software is really communicated with database server. | 1. JNDI is an API. The API consist set of classes and interfaces. The API is released by sun Microsystems. Once the API is released any one can provide the implementation. This implementation is nothing but software. This software is really communicated with directory server. |
| 1. Database server is used to store the data. As part of the database server the data will be stored in the form of relational data. | 1. Directory server is also to store the data. As part of the directory server the data will be stored in the form of objects. |
| 1. In database server we can insertion the data more no of times and retrieve the data from database more no of times. | 1. In directory server we can insertion the data less no of times and retrieve the data from directory server more no of times. |
| 1. As part of the database server we can store the huge amount of data | 1. As part of the directory server we can store the small amount of data. |
| 1. In database server sql queries can be applied | 1. In directory server sql queries cannot be applied. |
| 1. To communicate with database server the JDBC driver class is required. The class which provides the implementation of **driver** interface is called driver class. | 1. To communicate with directory server the Jndi driver class is required. The class which provides the implementation of **context** interface is called jndi driver class. |
| 1. The JDBC driver class varies from server to server. for example oracle driver class is: **oracle.jdbc.driver.OracleDriver** | 1. The JNDI driver class also varies from server to server. For example weblogic JNDI driver class is: weblogic.jndi.WLInitialContextFactory |
| 1. The JDBC related classes and interfaces are available the following packages  * **Java.sql** * **Javax.sql** | 1. The JNDI related classes and interfaces are available the following package. **javax.naming** |

**DIAGRAM:**

****

**JNDI:** Java Naming and Directory Interface.

* JNDI API is used to communicate with directory server.
* Similar to the database server directory server is used to store the data.
* As part of the directory server the data will be stored in the form of objects.
* We can choose directory server in insertion operations very less and select operations is for more no of times.
* **We can store the data into directory server for fast retrieval rather than database server.**
* As part of directory server we can store very small amount of data.
* The SQL queries cannot be applied on directory server.
* As a java programmer we can write a java application which can talk to directory server. If a java program needs to talk directory server we have to use JNDI API.
* A java application which uses JNDI API can communicate with directory server.
* There are different directory servers are available in the market. They are:
* LDAP (Light Weight Directory Access Protocol)
* ADS (Active Directory Server from Microsoft)
* NDS (Navel Directory Server from Navel)
* Our java application which uses JNDI API can communicate with any directory server.
* Generally directory servers are part of J2EE servers. If we buy the J2EE server we get the directory server.
* The following are the most popular J2EE servers
  + - * WEBLOGIC
      * WEBSPHERE
      * JBOSE
      * RESIN
      * PRAGATHI J2EE Server (This is Hyderabad Company)……………….etc.
* LDAP directory server is supplied with weblogic server. If we start the weblogic server by default the directory server will be started.

**Procedure to configure weblogic 10.3 version:**

**Step1:** 

**Step 2:** The above step will launch a dialog box whose name is **oracle weblogic configuration** **wizard**. In this dialog box **choose create a new** **weblogic domain radio button** and click on next.

**Step 3:**  Choose **generate a domain configured** **automatically to support the following products** **radio button** and click on next.

**Step 4:** Provide the **username** and **password**.

**Step 5:** Choose the available **SDK** and click on next button.

**Step 6:** Provide the **domain name** (mydomain) and click on create button.

**What is creating domain?**

* Copying the all required files into a folder is called create domain.
* The above steps will create a folder with name mydomain in **user\_project**.
* Once if we configure the weblogic server to start the weblogic server we need to double click on startweblogic.cmd (windows) or startweblogic.sh (Linux).

**To access the weblogic server perform the following steps:**

**Step 1:** Open the Internet Explorer (any browser) go to the URL and type the following URL.

<http://localhost:7001/console/>

* To login to the server provide the **username** and **password** which is given at the time of configuration wizard.

**Note:** If I want to write a java application which interacts with directory server we need to use JNDI API.

* All the classes and interfaces of JNDI API are available as part of **javax.naming** package.
* The most important interface of JNDI is **Context**.
* The most important class name of JNDI is **InitialContext**.
* In case of weblogic server LDAP directory server is integrated with our server. If we start the weblogic server internally it goes to the directory server also.
* We can develop a java application which interacts with directory server.
* **Weblogic people have provided implementation of JNDI API. This implementation is called as JNDI driver. This driver is available with weblogic server in the form of jar files.**

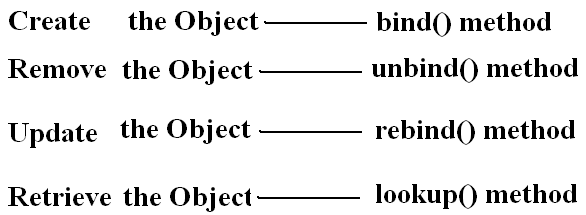
**Procedure to develop the java application which interacts with the directory server:**

**Step 1:** To establish the connection with directory server we need the JNDI driver class name which provides the implementation of context interface, URL, USERNAME of weblogic server and PASSWORD of weblogic server.

* We need to place all these details inside a Hashtable object.

**Step 2:** We need to supply the Hashtable object has input to **InitialContext ()** constructor.

**Step 3:** Once if we got the InitialContext object we can perform the following operations.



* We have to use the following **keys** in the Hashtable.
* **Context.INITIAL\_CONTEXT\_FACTORY**
* **Context.PROVIDER\_URL**
* **Context. SECURITY\_PRINCIPAL**
* **Context. SECURITY\_CREDENTIALS**

**Requirement:** Develop a Jndi application which can get the InitialContext object.

**Program:**

|  |
| --- |
| public class JNDIApp{  public static void main(String args[])throws Exception{  //step1:create Hashtable object  Hashtable objht= new Hashtable();  ht.put(Context.INITIAL\_CONTEXT\_FACTORY,"weblogic.jndi.WLInitialContextFactory");  objht.put(Context.PROVIDER\_URL,"t3://localhost:7001");  objht.put(Context.SECURITY\_PRINCIPAL,"weblogic");  objht.put(Context.SECURITY\_CREDENTIALS,"12345678");  //step2:create initialcontext object with the help of initialcontext() constructor  //and supply the Hashtable object  InitialContext objic=new InitialContext(objht);  //Whenever we create the initialcontext object the jndi driver establish  the connection with directory server. After establish the connection with  directory server we can perform the operations. (bind(),rebind().lookup(),rebind()).  //Context objic=new InitialContext(objht);  System.out.println("initialcontext object is : "+objic);  }  } |

**ERRORS:**

D:\jndi>javac JNDIApp.java

1. JNDIApp.java:4: cannot find symbol

Symbol: class Hashtable

Location: class JNDIApp

Hashtable objht= new Hashtable();

1. JNDIApp.java:4: cannot find symbol

Symbol: class Hashtable

Location: class JNDIApp

Hashtable objht= new Hashtable();

1. JNDIApp.java:5: cannot find symbol

Symbol: variable Context

Location: class JNDIApp

objht.put(Context.INITIAL\_CONTEXT\_FACTORY, "weblogic.jndi.WLInitialContextFacto

ry");

1. JNDIApp.java:6: cannot find symbol

Symbol: variable Context

Location: class JNDIApp

objht.put(Context.PROVIDER\_URL,"t3://localhost:7001");

1. JNDIApp.java:7: cannot find symbol

Symbol: variable Context

Location: class JNDIApp

objht.put(Context.SECURITY\_PRINCIPAL,"weblogic");

1. JNDIApp.java:8: cannot find symbol

Symbol: variable Context

Location: class JNDIApp

objht.put(Context.SECURITY\_CREDENTIALS,"12345678");

1. JNDIApp.java:11: cannot find symbol

Symbol: class InitialContext

Location: class JNDIApp

InitialContext objic=new InitialContext(objht);

1. JNDIApp.java:11: cannot find symbol

Symbol: class InitialContext

Location: class JNDIApp

* InitialContext objic=new InitialContext(objht);
* To resolve the above errors we have to import the import statements before compilation the java program then only the errors will be removed.

**After import the statements:**

**Program:**

|  |
| --- |
| Import java.util.\*;  import javax.naming.\*;  public class JNDIApp{  public static void main(String args[])throws Exception{  //step1:create Hashtable object  Hashtable objht= new Hashtable();  ht.put(Context.INITIAL\_CONTEXT\_FACTORY,"weblogic.jndi.WLInitialContextFactory");  objht.put(Context.PROVIDER\_URL,"t3://localhost:7001");  objht.put(Context.SECURITY\_PRINCIPAL,"weblogic");  objht.put(Context.SECURITY\_CREDENTIALS,"12345678");  //step2:create initialcontext object with the help of initialcontext() constructor  //and supply the Hashtable object  InitialContext objic=new InitialContext(objht);  System.out.println("initialcontext object is : "+objic);  }  }  **Output:**  D:\jndi>javac JNDIApp.java  Note: JNDIApp.java uses unchecked or unsafe operations.  Note: Recompile with -Xlint:unchecked for details. |

**AFTER COMPILATION RUN THE ABOVE PROGRAM:**

**ERROR:**

D:\jndi>java JNDIApp

Exception in thread "main" javax.naming.NoInitialContextException: Cannot instan

tiate class: weblogic.jndi.WLInitialContextFactory [Root exception is java.lang.

ClassNotFoundException: weblogic.jndi.WLInitialContextFactory]

* To resolve the above error we have to set the class path (**D:\jndi>set classpath=wlclient.jar;.;)**In command prompt or in environment variable.

**After setting the class path:**

D:\jndi>java JNDIApp

Exception in thread "main" org.omg.CORBA.COMM\_FAILURE: vmcid: SUN minor code:

203 completed: No

* To resolve the above error we have **start the weblogic server** before running the java application then only the error will be removed.

**After set the class path and started the weblogic server:**

D:\jndi>set classpath=wlclient.jar;.;

D:\jndi>java JNDIApp

Initialcontext object is: [javax.naming.InitialContext@10da5eb](mailto:javax.naming.InitialContext@10da5eb)

**Requirement:** Develop a JNDI application which adds a **name** **object** to the directory server. To store the name object use the key name as **sname**.

**Program:**

|  |
| --- |
| import java.util.\*;  import javax.naming.\*;  public class AddNameObjectToDirectoryServer{  public static void main(String args[])throws Exception{  //step1:create Hashtable object  Hashtable ht=new Hashtable();  ht.put(Context.INITIAL\_CONTEXT\_FACTORY,"weblogic.JNDI.WLInitialContextFactory");  ht.put(Context.PROVIDER\_URL,"t3://localhost:7001");  ht.put(Context.SECURITY\_PRINCIPAL,"weblogic");  ht.put(Context.SECURITY\_CREDENTIALS,"12345678");  //step2:create InitialContext object and supply the Hashtable object  InitialContext ic= new InitialContext(ht);  //step3:create string object  String name="vijayabhaskarareddy";  //step4:add string object to InitialCintext object  ic.bind("sname",name);  System.out.println("object is added to directory server");  }  }  **Output:**  D:\jndi>javac AddNameObjectToDirectoryServer.java  Note: AddNameObjectToDirectoryServer.java uses unchecked or unsafe operations.  Note: Recompile with -Xlint:unchecked for details.  D:\jndi>java AddNameObjectToDirectoryServer  Object is added to directory server |

**The following path is needed to see the object in the directory server:**

**Step 1:** Type the following URL in any browser <http://localhost:7001/console/>

**Step 2:** Enter the username and password.

**Step 3:** Go to environment and select the servers.

**Step 4:** Select the admin server.

**Step 5:** Select the JNDI tree.

**Requirement:** Develop a JNDI application which adds an **ArrayList** object to the directory server. To store the **ArrayList object** use thekey name as **emplist.**

**Program:**

|  |
| --- |
| import java.util.\*;  import javax.naming.\*;  public class CreateArrayListObjectToTheDirectoryServer{  public static void main(String args[])throws Exception{  //step1:create Hashtable object  Hashtable ht=new Hashtable();  ht.put(Context.INITIAL\_CONTEXT\_FACTORY,"weblogic.jndi.WLInitialContextFactory");  ht.put(Context.PROVIDER\_URL,"t3://localhost:7001");  ht.put(Context.SECURITY\_PRINCIPAL,"weblogic");  ht.put(Context.SECURITY\_CREDENTIALS,"12345678");  //step2:create InitialContext object and supply the Hashtable object  InitialContext ic= new InitialContext(ht);  //step3:create ArrayList object  ArrayList a=new ArrayList();  a.add("empone");  a.add("empTwo");  a.add("empThree");  a.add("empFour");  a.add("empFive");  //step4:add string object to InitialCintext object  ic.bind("emplist",a);  System.out.println("object is added to directory server");  }  }  **OUTPUT:**  D:\jndi>javac CreateArrayListObjectToTheDirectoryServer.java  Note: CreateArrayListObjectToTheDirectoryServer.java uses unchecked or unsafe operations.  Note: Recompile with -Xlint:unchecked for details.  D:\jndi>java CreateArrayListObjectToTheDirectoryServer  Object is added to directory server |

**Note** (very important point):when we use bind () method it adds the object to the directory server. The bind () method converts the **object into supper class reference object**.

Example: ArrayList a = new ArrayList ();

Object o=a;

* In the above example ArrayList is type casted to object class.

**Requirement:** Develop a JNDI application which deletes an object which store in the directory server.

**Note:** Delete the object which is associated to **sname key**.

**Program:**

|  |
| --- |
| import java.util.\*;  import javax.naming.\*;  public class DeleteTheObjectFromDirectoryServer{  public static void main(String args[])throws Exception{  //step1:create Hashtable object  Hashtable ht=new Hashtable();  ht.put(Context.INITIAL\_CONTEXT\_FACTORY,"weblogic.jndi.WLInitialContextFactory");  ht.put(Context.PROVIDER\_URL,"t3://localhost:7001");  ht.put(Context.SECURITY\_PRINCIPAL,"weblogic");  ht.put(Context.SECURITY\_CREDENTIALS,"12345678");  //step2:create InitialContext object and supply the Hashtable object  InitialContext ic= new InitialContext(ht);  //step4:delete object from directory server which is associated with sname key  ic.unbind("sname");  System.out.println("object is deleted from directory server");  }  }  **Output:**  D:\jndi>javac DeleteTheObjectFromDirectoryServer.java  Note: DeleteTheObjectFromDirectoryServer.java uses unchecked or unsafe operations.  Note: Recompile with -Xlint:unchecked for details.  D:\jndi>java DeleteTheObjectFromDirectoryServer  Object is deleted from directory server |

* If we try to add different objects with the same key the directory server will not allow. It will give an exception **java.naming.NameAlreadyBoundException**.

**Requirement:** Develop a JNDI application which updates the object stored in the directory server the key name of the object is **sname**.

**Program:**

|  |
| --- |
| import java.util.\*;  import javax.naming.\*;  public class UpdateNameObjectFromDirectoryServer{  public static void main(String args[])throws Exception{  //step1:create Hashtable object  Hashtable objht=new Hashtable();  ht.put(Context.INITIAL\_CONTEXT\_FACTORY,"weblogic.jndi.WLInitialContextFactory");  objht.put(Context.PROVIDER\_URL,"t3://localhost:7001");  objht.put(Context.SECURITY\_PRINCIPAL,"weblogic");  objht.put(Context.SECURITY\_CREDENTIALS,"12345678");  //step2:create InitialContext object and supply the Hashtable object  InitialContext objic= new InitialContext(objht);  //step3:update the string object  objic.rebind("sname","Vaka.VijayaBhaskaraReddy ");  System.out.println("object is updated from directory server");  }  }  **OUTPUT:**  D:\jndi>javac UpdateNameObjectFromDirectoryServer.java  Note: UpdateNameObjectFromDirectoryServer.java uses unchecked or unsafe operations.  Note: Recompile with -Xlint:unchecked for details.  D:\jndi>java UpdateNameObjectFromDirectoryServer  Object is updated from directory server |

**Requirement:** Develop a JNDI application which retrieves Object stored in the directory server the key name of the object is **sname**.

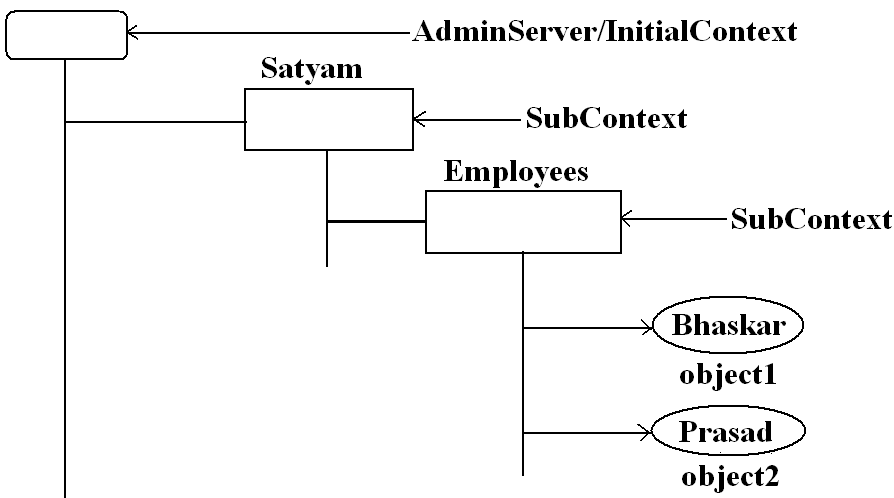
**Program:**

|  |
| --- |
| import java.util.\*;  import javax.naming.\*;  public class RetrieveTheObjectFromDirectoryServer{  public static void main(String args[])throws Exception{  //step1:create Hashtable object  Hashtable objht=new Hashtable();  ht.put(Context.INITIAL\_CONTEXT\_FACTORY,"weblogic.jndi.WLInitialContextFactory");  objht.put(Context.PROVIDER\_URL,"t3://localhost:7001");  objht.put(Context.SECURITY\_PRINCIPAL,"weblogic");  objht.put(Context.SECURITY\_CREDENTIALS,"12345678");  //step2:create InitialContext object and supply the Hashtable object  InitialContext objic= new InitialContext(objht);  //step3:retrieve the object from the key sname  Object o=objic.lookup("sname");  String s=(String)o;  System.out.println("retrieved object from directory server : "+s);  }  }  **OUTPUT:**  D:\jndi>javac RetrieveTheObjectFromDirectoryServer.java  **Note:** RetrieveTheObjectFromDirectoryServer.java uses unchecked or unsafe operations.  **Note:** Recompile with -Xlint: unchecked for details.  D:\jndi>java RetrieveTheObjectFromDirectoryServer  Retrieved object from directory server: Vaka.VijayaBhaskaraReddy |

* When we add an object to the directory server the object will be added to InitialContext directly.
* If I want to place all the related objects in one group I can create a subcontext.

The following diagram shows below.

**Diagram:**



**What is subcontext?**

* We can store all the related objects in one group under the InitialContext is called subcontext.

**Note:** for a java program subcontext is something similar to package.

* In package all the related classes grouped into a package

To create subcontext under the initialcontext we use the method as follows.

**Objic.createSubcontext(contextname);**

**Requirement:** Develop a JNDI application which creates a subcontext under the initialcontext.

The subcontext name is **college**.

**Program:**

|  |
| --- |
| import java.util.\*;  import javax.naming.\*;  public class CtrateSubContextInTheDirectoryServer{  public static void main(String args[])throws Exception{  //step1:create Hashtable object  Hashtable ht=new Hashtable();  ht.put(Context.INITIAL\_CONTEXT\_FACTORY,"weblogic.jndi.WLInitialContextFactory");  ht.put(Context.PROVIDER\_URL,"t3:localhost:7001");  ht.put(Context.SECURITY\_PRINCIPAL,"weblogic");  ht.put(Context.SECURITY\_CREDENTIALS,"12345678");  //step2:create initialcontext object with the help of initialcontext() constructor  InitialContext ic=new InitialContext(ht);  //step3:create subcontext under the initialContext  ic.createSubcontext("college");  }  }  **ERROR:**  D:\jndi>javac CtrateSubContextInTheDirectoryServer.java  Note: CtrateSubContextInTheDirectoryServer.java uses unchecked or unsafe operati ons.  Note: Recompile with -Xlint:unchecked for details.  D:\jndi>java CtrateSubContextInTheDirectoryServer  Exception in thread "main" javax.naming.InvalidNameException: **url does not contain //** |

* To resolve the above error define the // in URLthen only the error will be removed**.**

**Program:**

|  |
| --- |
| import java.util.\*;  import javax.naming.\*;  public class CtrateSubContextInTheDirectoryServer{  public static void main(String args[])throws Exception{  //step1:create Hashtable object  Hashtable ht=new Hashtable();  ht.put(Context.INITIAL\_CONTEXT\_FACTORY,"weblogic.jndi.WLInitialContextFactory");  ht.put(Context.PROVIDER\_URL,"t3://localhost:7001");  ht.put(Context.SECURITY\_PRINCIPAL,"weblogic");  ht.put(Context.SECURITY\_CREDENTIALS,"12345678");  //step2:create initialcontext object with the help of initialcontext() constructor  InitialContext ic=new InitialContext(ht);  //step3:create subcontext under the initialContext  ic.createSubcontext("colleget");  System.out.println("subcontext is created under the initialcontext");  }  }  **OUTPUT:**  D:\jndi>java CtrateSubContextInTheDirectoryServer  Subcontext is created under the initialcontext |

**Requirement:** Develop a JNDI application which creates a subcontext with the name M.C.A under the subcontext name is student.

**Program:**

|  |
| --- |
| import java.util.\*;  import javax.naming.\*;  public class CtrateSubContextUnderTheSubcontextInTheDirectoryServer{  public static void main(String args[])throws Exception{  //step1:create Hashtable object  Hashtable ht=new Hashtable();  ht.put(Context.INITIAL\_CONTEXT\_FACTORY,"weblogic.jndi.WLInitialContextFactory");  ht.put(Context.PROVIDER\_URL,"t3://localhost:7001");  ht.put(Context.SECURITY\_PRINCIPAL,"weblogic");  ht.put(Context.SECURITY\_CREDENTIALS,"12345678");  //step2:create initialcontext object with the help of initialcontext() constructor  InitialContext ic=new InitialContext(ht);  //step3:create subcontext under the subcontext  ic.createSubcontext("college.mca");  System.out.println("subcontext is created under the subcontext student");  }  }  **OUTPUT:**  D:\jndi>java CtrateSubContextUnderTheSubcontextInTheDirectoryServer  subcontext is created under the subcontext student |

**Requirement:** Develop a jndi application which adds an object with the name sthree under the subcontext (college) of subcontext (m.c.a).

**Program:**

|  |
| --- |
| import java.util.\*;  import javax.naming.\*;  public class AddObjectUnderSubContextOfSubcontextInTheDirectoryServer{  public static void main(String args[])throws Exception{  //step1:create Hashtable object  Hashtable ht=new Hashtable();  ht.put(Context.INITIAL\_CONTEXT\_FACTORY,"weblogic.jndi.WLInitialContextFactory");  ht.put(Context.PROVIDER\_URL,"t3://localhost:7001");  ht.put(Context.SECURITY\_PRINCIPAL,"weblogic");  ht.put(Context.SECURITY\_CREDENTIALS,"12345678");  //step2:create initialcontext object with the help of initialcontext() constructor  InitialContext ic=new InitialContext(ht);  //step3:add object under the subcontext of subcontext  String name="VijayaVhaskaraReddy";  ic.bind("college.mca.sthree",name);  System.out.println("sthree object is added under the subcontext(college) of subcontext(m.c.a) ");  }  }  **OUTPUT:**  D:\jndi>java AddObjectUnderSubContextOfSubcontextInTheDirectoryServer  Sthree object is added under the subcontext(college) of subcontext(m.c.a). |

* Create subcontext() method will create a subcontext inside initialcontext. To create the subcontext we use the code as shown below.

Context ic=new InitialContext(h);

Ic.createSubContext(“abcEngcoll”);

Ic.createSubContext(“abcEngcoll/btech”);

Ic.createSubContext(“abcEngcoll/mca”);

* To destroy the subcontext we use a method destroySubContext().

Context ic=new InitialContext(h);

Ic.destroySubContext(“abcEngColl/btech”);

Ic.destroySubContext(“abcEngColl/mca”);

**13. Connection pool Technique**

**Connection pool:**

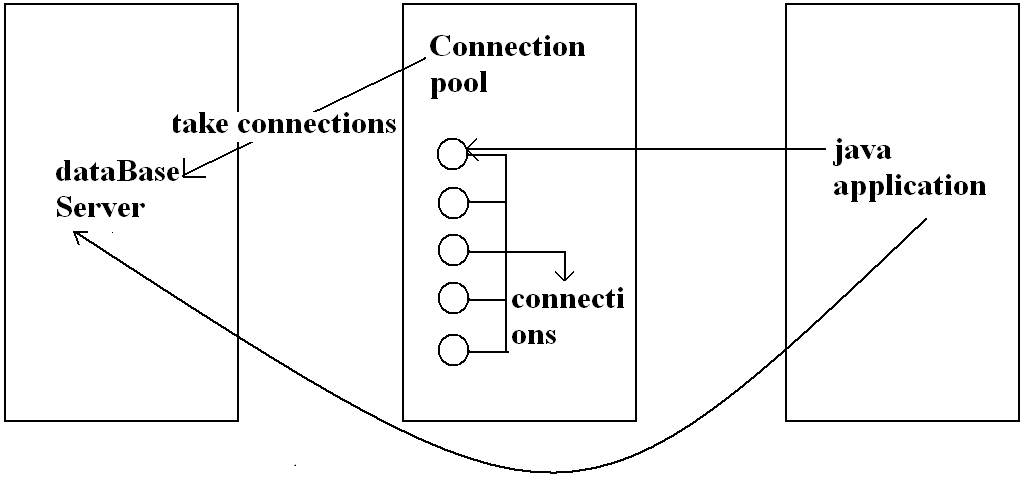
**Note:** **Without this technique we will not develop any projects.**

* Every database administrator will configure how many maximum numbers of connections can be given by database server.
* When the administrator configure that it store inside a file. The name of the file is **init.ora** (initialization of oracle is called init.ora). The path of the file is **C:\oraclexe\app\oracle\product\10.2.0\server\config\scripts**
* Every database server will be having a limit on issuing the connections to the applications.

**What is a connection pool?**

* Connection pool is a java program. Which can maintain set of connections. The connection pool program acquires (getting) the connections from database and holds those connections.
* **As part of every J2EE server we will be having a connection pool program**.

**Diagram:**

****

**Procedure to configure the connection pool in the weblogic server:**

**Step 1:** To configure the connection pool we have to **start the JEE server** (start weblogic server).

**Note:** Connection pool is a program which is available as part of J2EE servers.

**Step 2:** Access weblogic server admin console using the following **URL** <http://localhost:7001/console/>

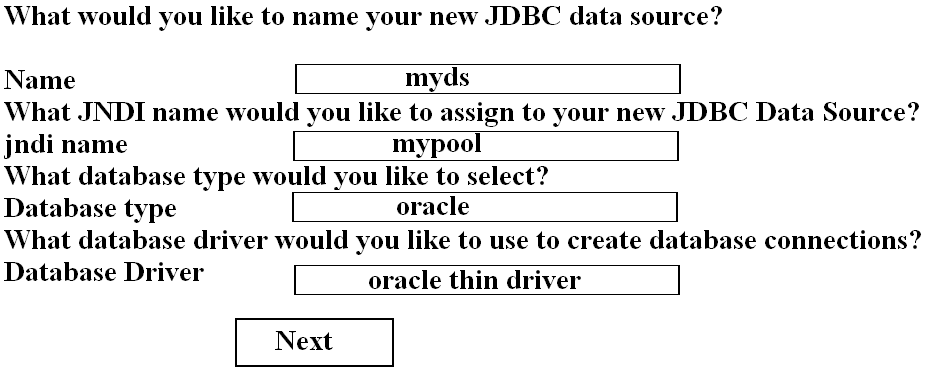
**Step 3:** We need to configure DataSource. To configure the DataSource from the **domain structure** menu go to **services option**. From the services option chooses **JDBC service**. From the JDBC service click on **Data Source**.

**Note:** Once if we click on **Data Source** we get a webpage which gives the **Summary of JDBC Data** **Sources**. If any **Data Sources** is already configured we can see those **data Sources** also.

**Step 4:** To create the new DataSource we have to click on **new button** from summary of JDBC data Sources web page.

**Step 5:** We have to provide the following details in **Create a New JDBC Data Source page**.

**Diagram:**



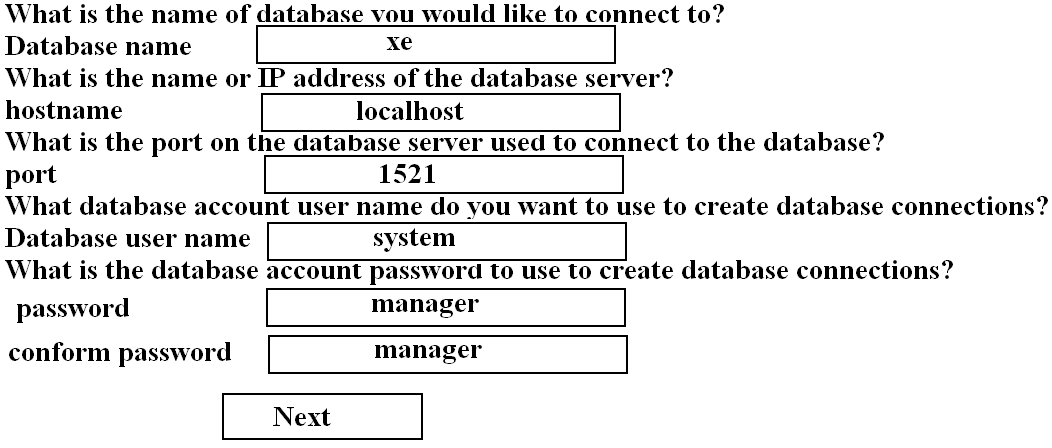
* After we fill the above forms click on **next** button.

**Step 6:** Once if we click the next button it displays **Transactions options**.

**Note:** Do not change any options and click on next button.

**Step 7:** We need to fill/provide the following **connection properties**.

**Diagram:**



* After filling the above form click on **next** button.

**Step 8:** Once if I click the next button we get a screen called as **Test Database Connection** in this screen we can check whether the connection pool program is able to communicate with database server or not.

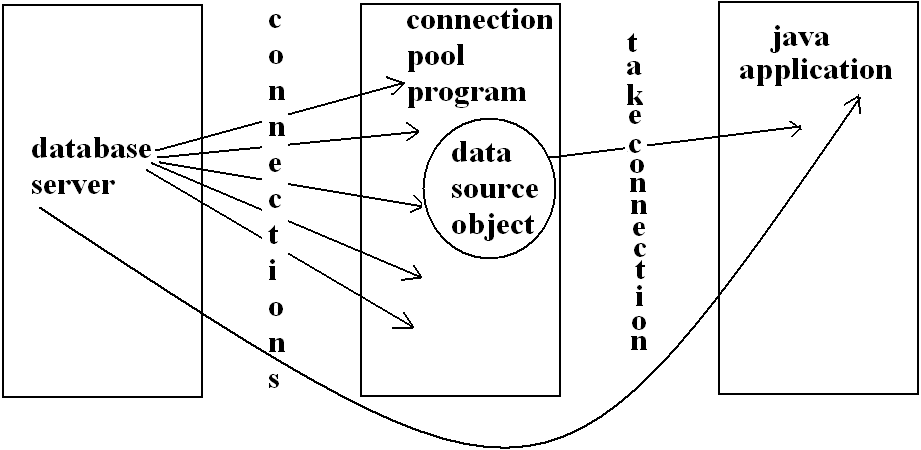
* To test this click on a button **Test configuration** otherwise click on **next button**.

**Step 9:** Once if we click on the next button we get a screen to associate the connection pool with the server. Select the name of the server to which we want to associate the connection pool and click on **finish**.

**Note:** In weblogic server after we perform all the above steps it creates the connection pool with initial capacity as 1(one).To modify this go to the **services** and go to **Data Source** option and click on **inetds** and then go to **connection pool tab**.

**Procedure to get a connection from connection pool:**

**Diagram:**

****

* The connection pool program acquires the connections from database server. After getting the connections from database server it stores the all connections into DataSource object. After stores the connections into DataSource object the connection pool program bound (attach) the DataSource object to directory server (initialcontext). Our java application communicates with database server first we get the DataSource object from directory server.

**Step 1:** Write the JNDI code to get the Data Source object from directory server.

**Step 2:** From the Data Source object get the connection.

**Note:** If we try to get the connection object from Data Source that connection object of type logical. We get the connection from weblogic server.

**Step 3:** Create the Statement object.

**Step 4:** Execute the query.

**Step 5:** Close the connection.

**Note:** DataSource interface is a part of **javax.sql** **package**.

**Requirement:** Develop a JNDI application which gets the connection from DataSource object. The DataSource object is available in the directory server.

**Program:**

|  |
| --- |
| import java.util.\*;  import java.sql.\*;  import javax.sql.\*;  import javax.naming.\*;  public class DBconnect1{  public static void main(String args[])throws Exception{  //step1:create Hashtable object  Hashtable ht=new Hashtable();  ht.put(Context.INITIAL\_CONTEXT\_FACTORY,"weblogic.jndi.WLInitialContextFactory");  ht.put(Context.PROVIDER\_URL,"t3://localhost:7001");  ht.put(Context.SECURITY\_PRINCIPAL,"weblogic");  ht.put(Context.SECURITY\_CREDENTIALS,"12345678");  //step2:create InitialContext object  InitialContext ic=new InitialContext(ht);  System.out.println("InitialContext Object is : "+ic);  //step3:get the datasource object from directory server  Object o=ic.lookup("mypool");  **//A JDBC data source is an object bound to the JNDI tree that provides**  **//database connectivity through a pool of JDBC connections. Applications can**  **//look up a data source on the JNDI tree and then borrow a database connection**  **//from a data source.**  DataSource ds=(DataSource)o;  System.out.println("DataSource object is : "+ds);  //step4:get the connection from DataSource object  Connection con=ds.getConnection();  System.out.println("connection object is : "+con);  //step5:create statement object  Statement stmt=con.createStatement();  System.out.println("statement object is : "+stmt);  //step6:execute the query with the help of executeQuery() or executeUpdate()  ResultSet rs=stmt.executeQuery("select \*from product");  System.out.println("query is executed");  System.out.println("----------------------------------------------");  System.out.println("product details are");  System.out.println("----------------------------------------------");  while(rs.next()){  System.out.print(rs.getInt("pid")+"\t");  System.out.print(rs.getString("pname")+"\t");  System.out.println(rs.getString("price")+"\t");  }  System.out.println("----------------------------------------------");  }  }  **ERROR:**  D:\jndi>java DBconnect1  Exception in thread "main" javax.naming.NoInitialContextException: Cannot instan  tiate class: weblogic.jndi.WLInitialContextFactory [Root exception is java.lang.  ClassNotFoundException: weblogic.jndi.WLInitialContextFactory] |

* To resolve the above error set the class path for jar file in the command prompt. The jar file name is **wlclient.jar** .After set the class path run the above program. The following command is used to set the class path **set classpath=wlclient.jar;.;**

**ERROR:**

D:\jndi>java DBconnect1

InitialContext Object is: javax.naming.InitialContext@e1d5ea

Exception in thread "main" javax.naming.NamingException: Unhandled exception in

Lookup [Root exception is org.omg.CORBA.**NO\_PERMISSION**: vmcid: 0x0 minor code:

1. completed: No]

* To resolve the above error once check the following constants and values INITIAL\_CONTEXT\_FACTORY, PROVIDER\_URL, SECURITY\_PRINCIPAL, SECURITY\_CREDENTIALS.

**ERROR**

D:\jndi>java DBconnect1

* Exception in thread "main" java.lang.ClassCastException: weblogic.jdbc.common.internal.ConnectionEnv cannot be cast to java.io.**Serializable**
* To resolve the above error execute the **setDomainEnv.cmd** in the command prompt .The **setDomainEnv.cmd** is available in the following path **C:\bea\user\_projects\domains\mydomain\bin>setDomainEnv.cmd;.;**
* **After set the above command run the above program (DBconnect1.java)**

**OUTPUT:**

D:\jndi>java DBconnect1

**InitialContext Object is**: javax.naming.InitialContext@dc67e

**Data Source object is** **:** ClusterableRemoteRef(-2529061471423375098S:127.0.0.1:

[7001, 7001,-1,-1,-1,-1,-1]:mydomain:AdminServer [-2529061471423375098S:127.0.0.1

:[7001,7001,-1,-1,-1,-1,-1]:mydomain:AdminServer/285])/285

**Connection object is :** weblogic.jdbc.rmi.SerialConnection\_weblogic\_jdbc\_rmi\_

internal\_ConnectionImpl\_weblogic\_jdbc\_wrapper\_PoolConnection\_oracle\_jdbc\_driver\_

T4CConnection\_1030\_WLStub@1

**Statement object is :** weblogic.jdbc.rmi.SerialStatement\_weblogic\_jdbc\_rmi\_i

nternal\_StatementStub\_weblogic\_jdbc\_rmi\_internal\_StatementImpl\_weblogic\_jdbc\_wra

pper\_Statement\_oracle\_jdbc\_driver\_OracleStatementWrapper\_1030\_WLStub@4

**Query is executed**

----------------------------------------------

The Product details are

----------------------------------------------

6 seven 7000

7 eight 8000

1 pone 10000

2 ptwo 20000

----------------------------------------------

**Requirement**: Write a JNDI program to get connections from data source object.

**Program:**

|  |
| --- |
| import java.util.\*;  import javax.naming.\*;  import javax.sql.\*;  import java.sql.\*;  public class Dbconnect{  public static void main(String args[])throws Exception{  Hashtable objht= new Hashtable();  ht.put(Context.INITIAL\_CONTEXT\_FACTORY,"weblogic.jndi.WLInitialContextFactory");  objht.put(Context.PROVIDER\_URL,"t3://localhost:7001");  objht.put(Context.SECURITY\_PRINCIPAL,"weblogic");  objht.put(Context.SECURITY\_CREDENTIALS,"12345678");  InitialContext objic=new InitialContext(objht);  Object o=objic.lookup("mypool");  DataSource ds=(DataSource)o;  Connection con1=ds.getConnection();  System.out.println("connection object one is : "+con1);  System.out.println("press any key");  System.in.read();  System.in.read();  Connection con2=ds.getConnection();  System.out.println("connection object twois : "+con2);  System.out.println("press any key");  System.in.read();  System.in.read();  Connection con3=ds.getConnection();  System.out.println("connection object three is : "+con3);  System.out.println("press any key");  System.in.read();  System.in.read();  Connection con4=ds.getConnection();  System.out.println("connection object four is : "+con4);  System.out.println("press any key");  System.in.read();  System.in.read();  Connection con5=ds.getConnection();  System.out.println("connection object five is : "+con5);  System.out.println("press any key");  System.in.read();  System.in.read();  Connection con6=ds.getConnection();  System.out.println("connection object six is : "+con6);  //Statement stmt=con.createStatement();  //ResultSet rs=stmt.executeQuery("select \*from product");  //while(rs.next()){  //System.out.print(rs.getInt("pid")+"\t");  //System.out.print(rs.getString("pname")+"\t");  //System.out.println(rs.getString("price"));  }} |

**OUTPUT:**

D:\jndi>java Dbconnect

Connection object one is: weblogic.jdbc.rmi.SerialConnection\_weblogic\_jdbc\_rmi\_

internal\_ConnectionImpl\_weblogic\_jdbc\_wrapper\_PoolConnection\_oracle\_jdbc\_driver\_

T4CConnection\_1030\_WLStub@1

Press any key

Connection object two is: weblogic.jdbc.rmi.SerialConnection\_weblogic\_jdbc\_rmi\_i

nternal\_ConnectionImpl\_weblogic\_jdbc\_wrapper\_PoolConnection\_oracle\_jdbc\_driver\_T

4CConnection\_1030\_WLStub@3

Press any key

Connection object three is: weblogic.jdbc.rmi.SerialConnection\_weblogic\_jdbc\_rm

i\_internal\_ConnectionImpl\_weblogic\_jdbc\_wrapper\_PoolConnection\_oracle\_jdbc\_drive

r\_T4CConnection\_1030\_WLStub@5

Press any key

Connection object four is: weblogic.jdbc.rmi.SerialConnection\_weblogic\_jdbc\_rmi

\_internal\_ConnectionImpl\_weblogic\_jdbc\_wrapper\_PoolConnection\_oracle\_jdbc\_driver

\_T4CConnection\_1030\_WLStub@7

Press any key

Connection objectfive is: weblogic.jdbc.rmi.SerialConnection\_weblogic\_jdbc\_rmi

\_internal\_ConnectionImpl\_weblogic\_jdbc\_wrapper\_PoolConnection\_oracle\_jdbc\_driver

\_T4CConnection\_1030\_WLStub@9

Press any key

Connection object six is: weblogic.jdbc.rmi.SerialConnection\_weblogic\_jdbc\_rmi\_

internal\_ConnectionImpl\_weblogic\_jdbc\_wrapper\_PoolConnection\_oracle\_jdbc\_driver\_

T4CConnection\_1030\_WLStub@b

**ERROR**

D:\jndi>javac Dbconnect.java

Dbconnect.java:51: **class, interface, or enum expected**

**}**

* To resolve the above error remove the extra brace (**}**) from the program (Dbconnect).Then compile it.

**ERROR**

D:\jndi>javac Dbconnect.java

Dbconnect.java:49: reached end of file while parsing

}

* To resolve the above error add the one brace (**{**) to the program after add the program compile it.
* The connection pool program works based on three parameters. They are:

1. Initial capacity
2. Maximum capacity
3. Capacity increment

* When we stop the weblogic server it acquires the initial connections based on **initial capacity parameter**. If the initial capacity is 5 when the connection pool program starts it acquires 5 connections and holds the 5 connections.
* If there are no free connections available with connection pool program the connection pool program is responsible to get more no of connections from database server to get more no of connections the connection pool program uses **capacity increment parameter**. If the capacity increment parameter value 3 the connection pool program acquires 3 more connections from database.
* The connection pool program uses **maximum capacity** **parameter** to set the maximum limit of connection pool program. If the maximum limit is 15 the connection pool program cannot acquires more than 15 connections.

**Procedure to get connection from database:**

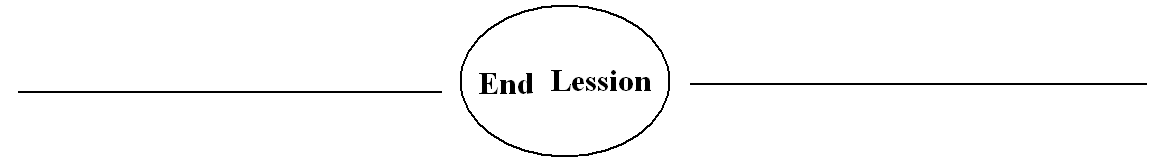
**Step 1:** Get the DataSource object from directory server.

**Step 2:** From the DataSource object get the connection using a method **getConnection()**.

**Step 3:** Create the required statement object.

**Step 4:** Execute the query.

**Step 5:** Return the connection back to connection pool using a method **con.close()**.



**HIBERNATE**

* There are so many frame works are available in the market **to simplify the project** **development**.
* The most popular frameworks are hibernate, struts, spring, JPA and etc.

**What is a framework?**

* A framework is a piece of software which contains a solution for a problem which occurs in multiple projects.
* As an alternative technique to JDBC API there are so many API’S are released into to the market. Some of the API’S are:
* Top link API (oracle)
* JPA (java persistence API used in EJB(3.0))
* JDO (java data object)
* Hibernate
* SDO (service data object) and etc.
* All the above tools are called as **ORM tools**.
* Once if we work with ORM API we can work with any other ORM tools.

**Note:** ORM stands for **object relational mapping**.

**What is ORM?**

* ORM (Object relational mapping) representing the relational data in the form of objects is called ORM.

**Can’t we represent the relational data in the form of objects by using JDBC API?**

* Yes. We can represent the relational data in the form of objects by using JDBC API but we can write a lot of code to represent the relational data in the form of objects.
* By using the ORM tools very quickly and with less amount of code we can represent the relational data in the form of objects.

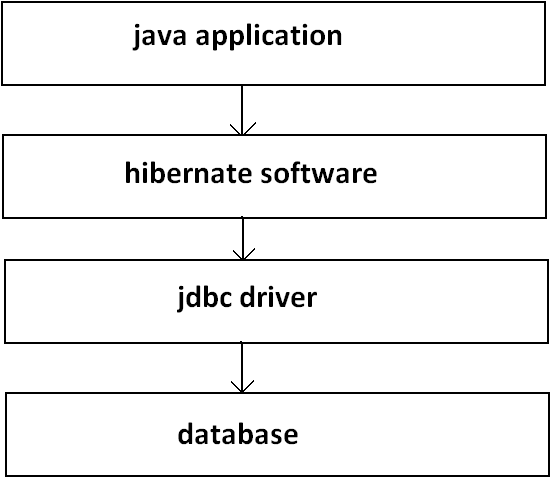
**What is Hibernate?**

* Hibernate is a java software. Hibernate is a framework. Hibernate is an ORM tool.ORM stands for object relational mapping. Object relational mapping representing the relational data in the form of objects is called ORM.

**What is the use of hibernate?**

* Hibernate is a software which is used to communicate with database.
* In java program we write the hibernate code to interact with database.
* To work with hibernate we use MyEclipse IDE.
* IDE stands for integrated development environment.
* To develop the java projects we can use any of the following three IDE’S. They are:
* Eclipse
* MyEclipse
* Netbeans
* As part of IDE we will be having different perspectives. Some of the perspective names are:
* Java perspective
* Debug perspective
* MyEclipse hibernate perspective
* MyEclipse database explorer perspective
* Java enterprise perspective …………….etc
* The following diagram shows the flow of Hibernate application.

**Diagram:**

****

**Note:** If a java application want to interact with hibernate software we need to develop the following 3 types of files. They are:

1. **Pojo classes**
2. **Hibernate configuration file**
3. **Hibernate mapping file**

**Note:** If hibernate software interact with database hibernate need to have a special privileges to the database user.

**Note:** If we want to develop an application which communicates with database by using hibernate compulsory the database tables must contain the primary key.

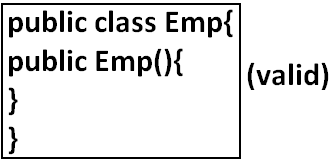
**Pojo classes:**

**Note:** In hibernate we call java beans as Pojo classes (**plain old java object**).

* A java bean is a simple java program we can call every java program as java bean if it follows the java bean rules.

**Rule 1:** Every java bean program **must contain a default constructor**.

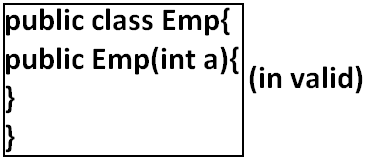
**Example 1:**



**Example 2:**



**Example 3:**



**Rule 2:** Every java bean program **must contain a** **package name**.

**Rule 3:** Java bean can contain **set of properties and events**.

**Note:** We should not call instance variables as property variables.

**Note:** A property internally uses instance variable to store the data.

* If we want to develop Hibernate application which can deal with EMP, Product we need to develop two java bean programs. They are:

1. Employee.java
2. Product.java

* As part of the hibernate documentation we see a name call as POJO classes (plain old java object).
* Hibernate guys are chooses this because the developers should not think that this is equivalent to EJB.
* If a project contains 100 tables if we want to develop java application which deals with database using hibernate we need to provide 100 POJO classes.
* **To use ORM we use POJO classes in hibernate. (or)**
* **We take the help of POJO classes to represent the relational data in the form of objects**.

Employee e=new Employee();

e.setEno(1);

e.setName("eone");

e.setAddress("hyd");

e.setSalary(1000);

**What is a POJO class?**

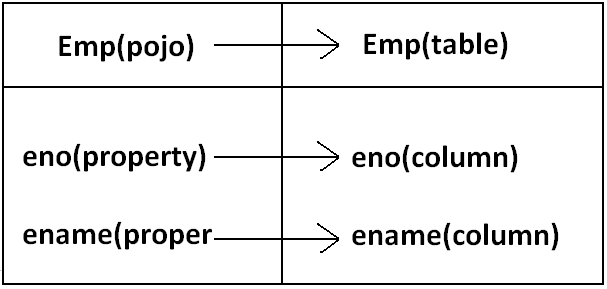
* Pojo class is a java bean**.**

**Hibernate mapping file:**

* We required **HBM** (Hibernate mapping file) **files** to represent which POJO class is mapped to which table**.**

**Note:** The HBM files contain mapping information between POJO classes and tables.

**Diagram:**

****

**Note:** Here arrow indicates which POJO class is mapped to which table and arrow is reading **“is mapped** **to”**.

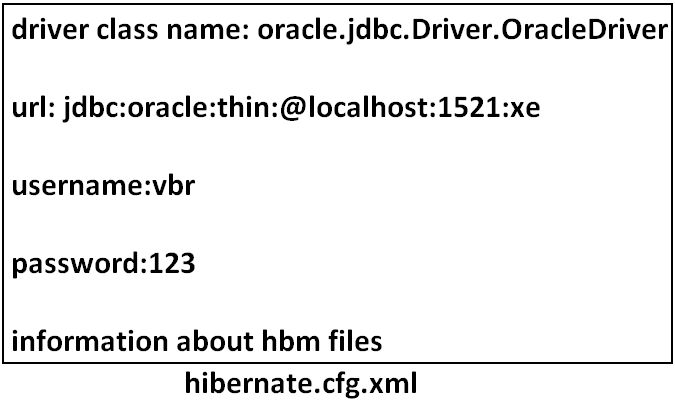
**What is hibernate mapping file?**

* A file which contains information about HBM files is called hibernate mapping file. To represent the relationship between the POJO classes and tables we use tags.

**Hibernate configuration file:**

* As part of hibernate configuration file we provide the information about driver class name, URL, username, password, dialect and information about hbm files.

**Diagram:**



**Note:** If we use MyEclipse IDE we no need to generate POJO classes and hbm files and hibernate configuration file. We can generate all this files by using IDE.

* If hibernate software interact with database hibernate need to have a special privileges to the database user.

**Procedure to provide privileges to the database user:**

**Step 1:** Oracle has released one web based application whose name is apex (application express). This application runs inside tomcat server of oracle. It uses a port number 8080.

**Step 2:** To open the application start the browser and type the following URL.

**http://localhost:8080/apex/**

**Step 3:** login to the database by supplying database administrator username.[we have to use system user].

**Step 4:** To provide the privileges go to **administration menu option**. From the administration choose **database users** and select **manage users**.

**Step 5:** Select the user to which we would like to provide the privileges.

**Step 6:** From the **user** **privileges** select all the check boxes and click on a button **alter user**.

* If we want to develop an application which communicates with database by using hibernate compulsory the database tables must contain the primary key.

**Query 1:** Create table emp(eno number(5)primary key,

Name varchar2(20),

Address varchar2(20),

Salary number(10,2));

**Query 2:** Create table product(pid number(5)primary key,

Pname varchar2(20),

Price number(10,2));

**Procedure to setup a database connection from MyEclipse IDE to database**:

**Step 1:** Create a folder and provide a name to the folder (hibernate).

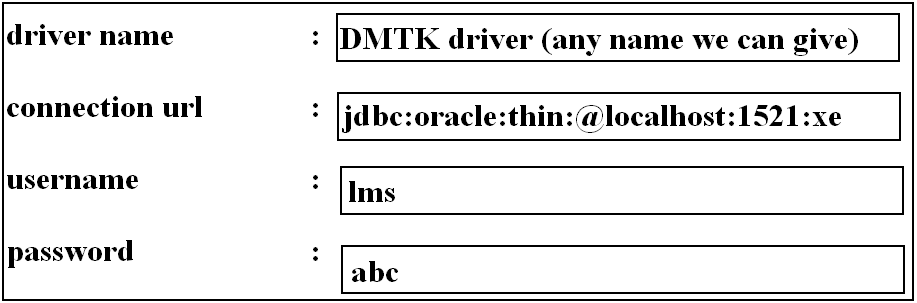
**Step 2:** Copy ojdbc14.jar in the above created folder.

**Step 3:** Start the MyEclipse IDE and it has to point to a folder which is created in step1.

**Step 4:** Open MyEclipse database explorer perspective.

**Step 5:** In the **DB browser view** right clicks and selects an option **new.** Itwilllaunch a dialog box with a name **database driver**.

**Step 6:** Choose the appropriate driver template from the list box (select oracle thin driver).



Add the ojdbc14.jar by clicking on **add Jars** button G:\hibernate\ojdbc14.jar(path) and click on finish button.

**Procedure to add hibernate capabilities and hibernate configuration file:**

**Note:** We need to make sure we configure MyEclipse IDE to interact with database.

**Step 1:** To work with hibernate in MyEclipse IDE we have to create new java project.



**Step 2:** The above step wills launch a dialog box with a name **new java project**. Provide the name of the java project and click on finish (hibernate project).

**Step 3:** To the above created java project we need to add hibernate capabilities.

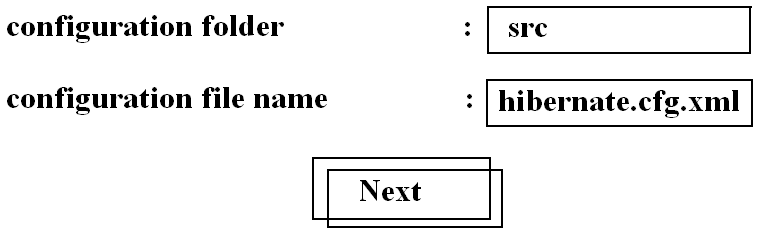
**Note:** Add hibernate capabilities means add all the jar files and it will create hibernate.cfg.xml file.

**Note:** Create a package with the name info.inetsolv.

**Step 4:** Select the project and go to **MyEclipse menu** item and choose **project capabilities** and select **add** **Hibernate** **capabilities.**

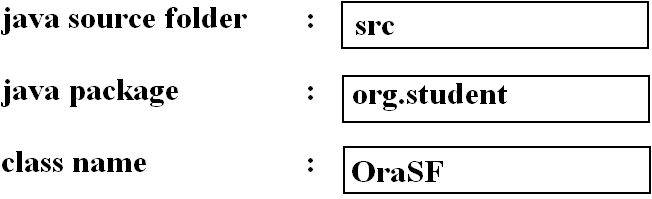
**Step 5:** The above step will launch **add hibernate capabilities dialog box**. In this screen we can find the version of hibernate software as well as the jar files which are been added to the project and click on **next button**.

**Step 6:** We need to specify the information about configuration folder name and configuration file name.



**Step 7:** Select DB driver which we have created when we start MyEclipse IDE (VBR driver). This will automatically populates driver name, username, password and dialect information and click on next.

**Step 8:** Choose java package name and class name.



* Click on finish to add the jar files and to create the configuration file.

**Procedure to generate hbm files and POJO classes:**

**Step 1:** Open MyEclipse database explorer prospective.

**Step 2:** Right click on db browser and click on open connection.

**Step 3:** Select the tables to which we would like to generate hbm files and POJO classes. Right click on this tables it will launch a popup menu from the popup menu choose **hibernate reverse** **engineering**.

**Step 4:** Select the src folder and java package.

**Step 5:** Select the first **3 checkboxes** and **uncheck create abstract class** and click on **finish button**.

**Procedure to develop a java program which interacts with hibernate api:**

* The following are the most important **class** and i**nterfaces** of hibernate. They are:

org.hibernate.cfg.Configuration (class).

org.hibernate.SessionFactory(interface)

org.hibernate..Session(interface).

org.hibernate..Transaction(interface).

* **Every hibernate application contains the following steps:**

**Step 1:** Create the configuration object with the help of org.hibernate.cfg.Configuration class. Call the configure() method on the configuration class object.

**Step 2:** Get the SessionFactory object with the help of configuration object. (Or)

**Step 2:** Get the SessionFactory object by calling the buildSessionFactory() method on the above created configuration object.

**Step 3:** Get the session object from SessionFactory object. (Or)

**Step 3:** Get the session object with the help of SessionFactory object.

**Step 4:** Start the transaction (get the transaction object).

\***Step 5:** Create the pojo class object and represent the data in the form of object.

**Note:** To which table we want to interact to that table we have to create pojo class object.

\***Step 6:** Ask hibernate to perform the required operation (CURD operations).

**Step 7:** End the transaction.

**Step 8:** Close the session.

**Note:** \*(star) indicates Depend upon the requirement **five and six steps** varies from application to application remaining steps will same to any application.

**Requirement:** Insert a record into the database by using hibernate software.

**Program:**

|  |
| --- |
| package info.inetsolv;  import org.hibernate.Session;  import org.hibernate.SessionFactory;  import org.hibernate.Transaction;  import org.hibernate.cfg.Configuration;  **public** **class** InsertRecord {  **public** **static** **void** main(String[] args) {  //step1: create the configuration object and call the configure method.  Configuration cfg=**new** Configuration();  cfg.configure();  //step2: get the SessionFactory object with the help of configuration object.  SessionFactory sf=cfg.buildSessionFactory();  //step3: get the session object with the help of SessionFactory object.  Session hsession=sf.openSession();  //step4: start the transaction. To start the transaction creates the Transaction object with the help of session object.  Transaction tx=hsession.beginTransaction();  //step5: to which table we want to interact to that table create the pojo class object.  info.inetsolv.Product p=**new** info.inetsolv.Product();  //Represent the data in the form of object.  p.setPid(2);  p.setPname("ptwo");  p.setPrice(2000d);  //step6: Ask hibernate to save the data in the database.  hsession.save(p);  //step7: End the transaction.  tx.commit();  //step8: Close the session.  System.*out*.println("record successfully inserted");  hsession.close();  }  } |

**Output:**

Record successfully inserted.

**How the program will be executed in jvm’s memory:**

**What will happen when we call the configure () method in the project?**

**Step 1:** **Configuration cfg=new configuration();**

**cfg.configure();**

* When we create the configuration object by using new operator just we are creating the object for configuration class.
* When we call the configure() method by default it search for **hibernate.cfg.xml file**.

**Note:** If we change the hibernate.cfg.xml file name we have to pass while calling the configure() method.

**Example:** cfg.configure("one.cfg.xml");

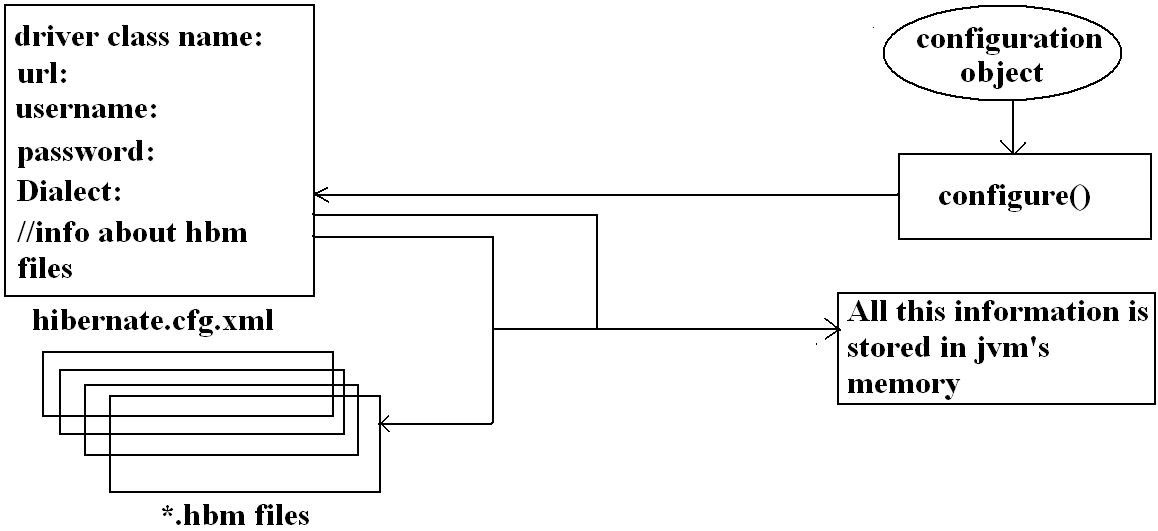
**Note:** While reading the any file we have to give name of the file within double quotes.

**Note:** If we didn’t pass the modified configuration file name while calling the configure() method hibernate throws the following exception.

**Error:** Exception in thread "main" org.hibernate.HibernateException: **/hibernate.cfg.xml not** **found.**

* If the method finds the configuration file it opens the configuration file and read driver class name, URL, user name, pwd, dialect and it finds the path to all the hbm files.
* Now the configure() method search for all the hbm files and start the reading contents from hbm files. Once if configure() method reads the contents from hbm files it stores all the information inside the jvm’s memory.

**Diagram:**



**Note:** Calling the configure() method repeatedly in the project is not recommended(encouraged) this is because in a project we will be having more no of hbm files. Reading from those many no of hbm files will have the performance issues.

**What will happen when we call the buildSessionFactory() method in the project?**

**Step 2:** **SessionFactory sf=cfg.buildSessionFactory();**

* When we call the buildSessionFactory() method it will get driver class name, url, username, password from jvm’s memory and hibernate internally create the connection pool with these details.

**Note:** Hibernate internally uses Apache DBCP (database connection pool program).

* The hibernate program gets a connection from the connection pool and establish the connection with database.
* After establish the connection with the database the buildSessionFactory() method checks whether all the required tables are available inside the database or not.
* **If the required tables are not available and if required buildSessionFactory() method creates the tables in the database** andbuildSessionFactory() method prepares the **CURD** **operations** **(or) quires** for all the database tables and store the queries inside the jvm’s memory.

**Note:** If hibernate need to create the tables automatically we have to use additional property “hbm2ddl.auto” property.

**Note:** It is not at all recommended to call buildSessionFactory() method repeatedly in the project.

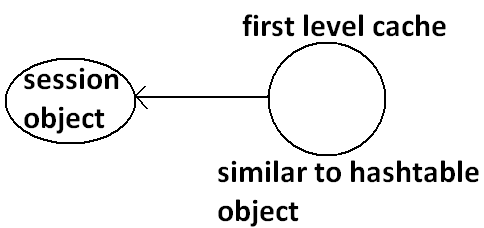
**Step 3:** **Session hsession=sf.openSession();**

* Get the session object by using openSession() method. This method is not an expensive operation. We can call this method repeatedly in the project.

**Note:** Opening session is equivalent to getting the connection object.

* When we got the session object internally hibernate will create a cache object and it will be associated to session object. This is called as 1st level cache.

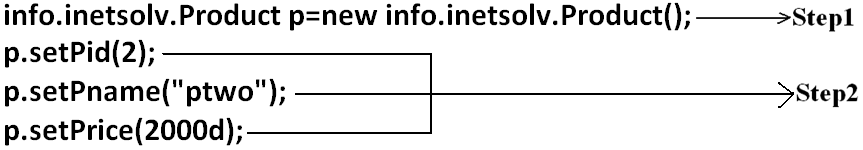
**Diagram:**



**Step 4:** **Transaction tx=hsession.beginTransaction();**

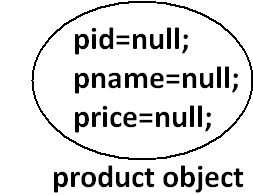
* As part of hibernate compulsory the user must start the user defined transaction. To start the transaction we use hsession.beginTransaction().

**Step5 and step6:**



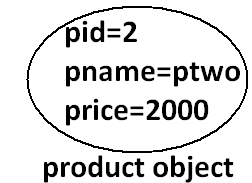
* In the above code when first step is executed it will create the product object and the instance variables will be initialized to the default values.

**Diagram:**



* When the second step is executed it modifies the instance variables data and the object becomes as shown below.

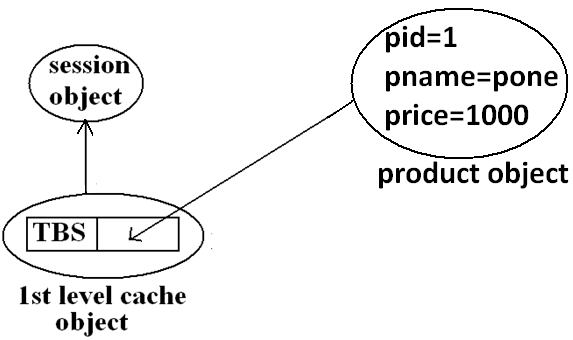
**Diagram:**



**What will happen when we call the save() method [hsession. Save (p)]?**

* When we call the save() method hibernate will not save the record into the database. When we call the save() method hibernate registers the object as part of first level cache object. To register the object in 1st level cache the save() method uses a registration code. That is **TBS** (to be saved). The following diagram shows how it works.

**Diagram:**



**What will happen when we end the transaction that is tx.commit?**

* The hibernate checks are there any register objects are available who are waiting for the appropriate action.
* If it is available it checks to which class the pojo class object is created. Now the hibernate finds to which table this pojo class is mapped.
* Now hibernate finds the registration code of the pojo class object. Based on the registration code hibernate picks the appropriate query from jvm’s memory. Now the hibernate gets the data from pojo class object and replaces the question marks with values. Now hibernate sends the query to the database.

**Requirement:** Develop one hibernate application which retrieves the record from product table. We want to retrieve a record whose product number is 2.

**To retrieve record from database by using hibernates we have two approaches:**

They are:

* 1. by using load() method
  2. By using get() method
* To retrieve the records from database table if we use load() method. The load() method excepts two parameters. They are:

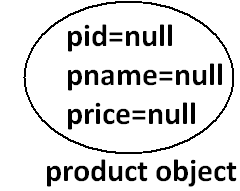
1. **Pojo class object**
2. **Primary key value**

**Program (by using load() method):**

|  |
| --- |
| package info.inetsolv;  import org.hibernate.Session;  import org.hibernate.SessionFactory;  import org.hibernate.Transaction;  import org.hibernate.cfg.Configuration;  **public** **class** RetrieveRecordsUsingLoad {  **public** **static** **void** main(String[] args) {  Configuration cfg=**new** Configuration();  cfg.configure();  SessionFactory sf=cfg.buildSessionFactory();  Session hsession=sf.openSession();  //Transaction tx=hsession.beginTransaction();  info.inetsolv.Product p=**new** info.inetsolv.Product(); step1  //step6: Ask hibernate load the record from database  **hsession.load(p, 2); (or) hsession.load(p,new Integer(2));** step2  System.*out*.println(p.getPid());  System.*out*.println(p.getPname());  System.*out*.println(p.getPrice());  //step7: End the transaction.  //tx.commit();  //step8: Close the session.  System.out.println("record successfully retrieved");  hsession.close();  }  } |

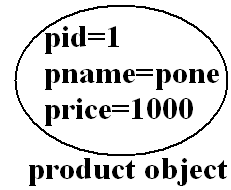
* When the step1 is executed the pojo class object will be created and all the instance variables will be initialized to null.

**Diagram:**



* When the hibernate calls the load() method it finds the pojo class name of the object and hibernate finds to which corresponding table it is mapped. Now the hibernate gets the appropriate select query from the jvm’s memory and hibernate replaces the question marks with supplied value. Now hibernate will send query to the database. The database will execute query and send the ResultSet object to the hibernate software.
* Now the hibernate software gets the data from ResultSet object and store it inside pojo class object. To store inside pojo class object internally it will call setter methods.
* After the load() method is executed successfully now the data is available in pojo class object.

**Diagram:**

****

**Note:** As part of the product table we are having four records only. We are try to develop one hibernate application to retrieve records from product table whose record is not available in the database.

**hsession.load(p,new integer(5));**

* In the above line we are try to retrieve the product record whose product id is 5 and that record is not available in database table. when we run this program we get an exception that is **org.hibernate.object not found exception.**

**Note:** Once if load method is executed hibernate gets the object and add the pojo class object to 1st level cache.

**Program (by using get() method):**

|  |
| --- |
| package info.inetsolv;  import org.hibernate.Session;  import org.hibernate.SessionFactory;  import org.hibernate.Transaction;  import org.hibernate.cfg.Configuration;  public class RetrieveRecordsUsingGet {  public static void main(String[] args) {  Configuration cfg=new Configuration();  cfg.configure();  SessionFactory sf=cfg.buildSessionFactory();  Session hsession=sf.openSession();  //Transaction tx=hsession.beginTransaction();  //info.inetsolv.Product p=new info.inetsolv.Product();  //step6: Ask hibernate load the record from database  //Product p=(Product)hsession.get(Product.class,2);  //System.out.println(p.getPid());  //System.out.println(p.getPname());  //System.out.println(p.getPrice());  Object o=hsession.get(Product.class, 2);  if(o!=null){  Product p=(Product)o;  System.*out*.println(p.getPid());  System.*out*.println(p.getPname());  System.*out*.println(p.getPrice());  }else{  System.*out*.println("data is not available");  }  //step7: End the transaction.  tx.commit();  //step8: Close the session.  hsession.close();  }  } |

**Output:**

2

Ptwo

2000.0

**What is the difference between load() method and get() method?**

|  |  |
| --- | --- |
| **load()** | **get()** |
| 1. Only use the load() method if you are sure that the object exists. | 1. If you are not sure that the object exists, then use one of the get() methods. |
| 1. load() method will throw an exception if the unique id is not found in the database. | 1. get() method will return null if the unique id is not found in the database. |
| 1. load() just returns a proxy by default and database won’t be hit until the proxy is first invoked. | 1. get() will hit the database immediately. |

**Requirement:** Develop one hibernate application to delete a record from product table whose product id is 2.

**To delete a record from database by using hibernate we have two approaches:**

**Approach 1:** In approach 1 to delete the record from the table 1st we are going to retrieve the record and then we make the object as to be deleted. To load the object we use the load() method. To mark the object as to be deleted we use a method delete().

**Program:**

|  |
| --- |
| package info.inetsolv;  public class DeleteRecord {  public static void main(String[] args) {  Configuration cfg=new Configuration();  cfg.configure();  SessionFactory sf=cfg.buildSessionFactory();  Session hsession=sf.openSession();  Transaction tx=hsession.beginTransaction();  info.inetsolv.Product p=new info.inetsolv.Product(); **step1**  //step6: Ask hibernate load the record from database  hsession.load(p, 3); **step2**  //step7: mark the object as to be deleted  hsession.delete(p); **step3**  //step7: End the transaction.  tx.commit(); **step4**  System.*out*.println("record successfully deleted");  hsession.close();  }  } |

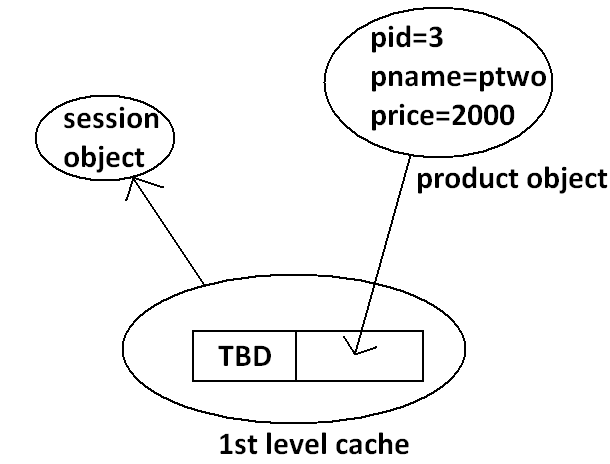
**Step 1:** When the step1 is executed it will create the pojo class object and all the instance variables will be initialized with the default values.

**Step 2:** When the step2 is executed hibernate will retrieve the records and store it in pojo class object. If the record is not available it throws the exception.

**Step 3:** When the step3 is executed the object will be marked as to be deleted in the 1st level cache.

**Step 4:** When the step4 is executed the hibernate checks are there any registered objects in 1st level cache if it is available based on registration code it will pick appropriate query and replace the question mark with values and send the query to the database.

**Diagram:**

****

**Note:** In this approach if we try to delete the record whose record is really not available it will through the exception the name of the exception is **org.hibernate.object not found exception**.

**Approach 2:** In this approach whichever record we want to delete to that table create the pojo class object and supply the primary key value.

Mark the pojo class object as to be deleted. We can do this with the help of a method delete.

**Program:**

|  |
| --- |
| public class DeleteRecordapproach2 {  public static void main(String[] args) {  //standard code from step1 to step4  info.inetsolv.Product p=new info.inetsolv.Product();  p.setPid(5); step1  hsession.delete(p); step2  tx.commit(); step3  System.*out*.println("record successfully deleted");  hsession.close();  }  } |

**Step 1:** When the step1 is executed the primary key value will be placed inside the product object.

**Step 2:** When the step2 is executed 1st hibernate sends a select query to the database to verify whether the record is available or not. If the record is available hibernate marks the object to be deleted otherwise hibernate will not mark the object to be deleted.

**Step 3:** When step3 is executed hibernate checks are there any objects are registered as to be deleted if it is registered hibernate sends the delete query to the database.

**Note:** **In this approach if we try to delete the record whose record is really not available it will not through the exception.**

**Updation:** To update the data inside database hibernate uses a dirty object concept.

**To update a record inside database by using hibernate we have two approaches:**

**Approach 1:** To update the data inside database hibernate uses a dirty object concept.

**What is a dirty object?**

* When the object is modified with a new value then the object is called as dirty object.

**Program:**

|  |
| --- |
| package info.inetsolv;  //import the packages  **public** **class** UpdateRecord {  **public** **static** **void** main(String[] args) {  //standard code  info.inetsolv.Product p=**new** info.inetsolv.Product();  //load the record from database  hsession.load(p, 2); step1  //update the record  p.setPname("lenovo"); step2  //step7: End the transaction.  tx.commit(); step3  //step8: Close the session.  System.*out*.println("record successfully updated");  hsession.close();  }  } |

**Step 1:** When the step1 is executed hibernate retrieves the record and store the data inside pojo class object. Hibernate ads the pojo class object to 1st level cache. Hibernate clones the object which is added to 1st level cache.

**Step 2:** In the step2 we are modifying the state of the actual object.

**Step 3:** When step3 is executed hibernate checks the actual object with cloned object if there is any difference in the actual object and cloned object. Hibernate issues the update query.

**Note:** If both objects are same the hibernate will not send the update query to the database.

**Approach 2:** In this approach whichever record we would like to update represent the updated data in the form of pojo class object. Call a method update to update the data in the database.

**Note:** **The update() method updates all the columns based on primary key**. The following is an example of updating a record in the hibernate.

**Program:**

|  |
| --- |
| **package** info.inetsolv;  //import the packages  **public** **class** UpdateRecordApproach2 {  **public** **static** **void** main(String[] args) {  //standard code from step1 to step4  //step5: to which table we want to interact to that table create the pojo class object.  info.inetsolv.Product p=**new** info.inetsolv.Product();  //represent the updated data in the form of pojo class object  p.setPid(1);  p.setPname("del");  p.setPrice(35000d);  //call update() method to update the record  hsession.update(p);  tx.commit();  System.*out*.println("record successfully updated");  hsession.close();  }  } |

**Note:** The disadvantage with 2nd approach is hibernate updates some of the columns with null values.

**What is the responsibility of evict() method?**

* evict() method is used to remove any specified object from 1st level cache.

**Example:** hsession.evict(p);

**Example program:**

|  |
| --- |
| **package** info.inetsolv;  //import the packages  **public** **class** ExampleOfEvictMethod {  **public** **static** **void** main(String[] args) {  //standard code from step1 to step4  //step5: to which table we want to interact to that table create the pojo class object.  info.inetsolv.Product p=**new** info.inetsolv.Product(); step1  hsession.load(p, 3); step2  p.setPname("HP"); step3  p.setPrice(25000d);  hsession.evict(p); step4  tx.commit();  hsession.close();  }  } |

**Step 1:** When the step1 is executed it will create pojo class object and will be initialized with default values.

**Step 2:** When the step2 is executed hibernate retrieves the data from database and place it in pojo class object and pojo class object will be added to 1st level cache.

**Step 3:** When the step3 is executed we are making (or) modifying the data of pojo class object the object becomes dirty object.

**Step 4:** When step4 is executed the product object will be removed from 1st level cache.

**What is the responsibility of merge() method?**

* merge() method is used to add any specific object to 1st level cache.

**Example:** hsession.merge(p);

**Example program:**

|  |
| --- |
| **package** info.inetsolv;  //import the packages  **public** **class** ExampleOfMergeMethod {  **public** **static** **void** main(String[] args) {  //standard code from step1 to step4  //step5: to which table we want to interact to that table create the pojo class object.  info.inetsolv.Product p=**new** info.inetsolv.Product(); step1  p.setPid(3);  p.setPname("hp"); step2  p.setPrice(25000d);  hsession.merge(p); step3  tx.commit(); step4  hsession.close();  }  } |

**Step 1:** When step1 is executed we create the pojo class object and it will be initialized with default values.

**Step 2:** When step2 is executed we are storing the data inside pojo class object.

**Step 3:** When step3 is executed the product object is added to 1st level cache.

**Step 4:** When step4 is executed hibernate checks are there any objects are available in 1st level cache. Hibernate checks whether the record is available in database or not. **If it is not available** hibernate will issue **insert query**. **If the record is available** in database hibernate **issues update** **query**.

**Clear():** The clear() method removes all the objects from first level cache.

**Example:** hsession.clear();

**Note:** evict() method is remove the specified object from 1st level cache but clear() method is remove all the objects from 1st level cache.

**flush():** This method send all the objects which are available in 1st level cache to the Database server.

**Note:** By default when we call the commit() method hibernate internally call this method.

**Procedure to develop hibernate applications manually:**

**Step 1:** Download hibernate software from [www.hibernate.org](http://www.hibernate.org) website. [We need to download core module. We get a zip file].

**Step 2:** Copy all the hibernate related jar files inside a folder lib.

**Step 3:** Create one command file which can set the class path to all the jar files.

**Example:**

|  |
| --- |
| set CLASSPATH=lib\antlr-2.7.6.jar;lib\cglib-2.2.jar;lib\commons-collections-3.1.jar;lib\dom4j-1.6.1.jar;lib\ehcache-1.2.3.jar;lib\ejb3-persistence.jar;lib\hibernate3.jar;lib\hibernate-annotations.jar;lib\hibernate-commons-annotations.jar;lib\hibernate-entitymanager.jar;lib\hibernate-validator.jar;lib\javassist-3.9.0.GA.jar;lib\jta-1.1.jar;lib\log4j-1.2.14.jar;lib\ojdbc14.jar;lib\slf4j-api-1.5.8.jar;lib\slf4j-log4j12-1.5.8.jar;.; |

**sethibclasspath.cmd**

**Step 4:** Create hibernate configuration file.

|  |
| --- |
| <?xml version="1.0"?>  <!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">  <hibernate-configuration>  <session-factory>  <property name="dialect">  org.hibernate.dialect.Oracle9Dialect  </property>  <property name="connection.url">  jdbc:oracle:thin:@localhost:1521:xe  </property>  <property name="connection.username">system</property>  <property name="connection.password">admin</property>  <property name="connection.driver\_class">  oracle.jdbc.driver.OracleDriver  </property>  <property name="show\_sql">true</property>  <property name="hbm2ddl.auto">update</property>  <mapping resource="Product.hbm.xml" />  </session-factory>  </hibernate-configuration> |

**Step 5:** Create Hbm file for the database table.

|  |
| --- |
| <?xml version="1.0"?>  <!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">  <hibernate-mapping>  <class name="info.inetsolv.Product" table="PRODUCT" schema="system">  <id name="pid" type="java.lang.Integer">  <column name="PID" precision="5" scale="0"/>  <generator class="assigned"/>  </id>  <property name="pname" type="java.lang.String">  <column name="PNAME" length="20"/>  </property>  <property name="price" type="java.lang.Double">  <column name="PRICE" precision="10"/>  </property>  </class>  </hibernate-mapping> |

**Step 6:** Create a Pojo class.

|  |
| --- |
| package info.inetsolv;  public class Product {  int pid;  String pname;  double price;  public int getPid() {  return pid;  }  public void setPid(int pid) {  this.pid = pid;  }  public String getPname() {  return pname;  }  public void setPname(String pname) {  this.pname = pname;  }  public double getPrice() {  return price;  }  public void setPrice(double price) {  this.price = price;  }  } |

**Step 7:** Develop a java application which interacts with database.

|  |
| --- |
| import org.hibernate.\*;  import org.hibernate.cfg.\*;  class InsertRecord  {  public static void main(String[] args)  {  Configuration cfg=new Configuration();  cfg.configure("hibernate.cfg.xml");  SessionFactory sf=cfg.buildSessionFactory();  Session hsession=sf.openSession();  Transaction tx=hsession.beginTransaction();  info.inetsolv.Product p=new info.inetsolv.Product();  p.setPid(1);  p.setPname("salt");  p.setPrice(10d);  hsession.save(p);  tx.commit();  hsession.close();  }  } |

**How do we run the above application?**

E:\Manual Hibernate Application>**javac -d . Product.java**

E:\Manual Hibernate Application>**sethibclasspath.cmd**

E:\Manual Hibernate Application>set CLASSPATH=lib\antlr-2.7.6.jar;lib\cglib-2.2.

jar;lib\commons-collections-3.1.jar;lib\dom4j-1.6.1.jar;lib\ehcache-1.2.3.jar;li

b\ejb3-persistence.jar;lib\hibernate3.jar;lib\hibernate-annotations.jar;lib\hibe

rnate-commons-annotations.jar;lib\hibernate-entitymanager.jar;lib\hibernate-vali

dator.jar;lib\javassist-3.9.0.GA.jar;lib\jta-1.1.jar;lib\log4j-1.2.14.jar;lib\oj

dbc14.jar;lib\slf4j-api-1.5.8.jar;lib\slf4j-log4j12-1.5.8.jar;.;

E:\Manual Hibernate Application>**javac InsertRecord.java**

E:\Manual Hibernate Application>**java InsertRecord**

log4j:WARN No appenders could be found for logger (org.hibernate.cfg.Environment).

log4j:WARN Please initialize the log4j system properly.

Hibernate: insert into system.PRODUCT (PNAME, PRICE, PID) values (?, ?, ?)

**Requirement:** Representing the relational data in the form of objects by using jdbc API?

**Step 1:** Develop the java bean to represent the relational data in the form of objects.

|  |
| --- |
| package info.inetsolv;  public class Product {  int pid;  String pname;  double price;  public int getPid() {  return pid;  }  public void setPid(int pid) {  this.pid = pid;  }  public String getPname() {  return pname;  }  public void setPname(String pname) {  this.pname = pname;  }  public double getPrice() {  return price;  }  public void setPrice(double price) {  this.price = price;  }  } |

**Step 2:** Develop the jdbc application to interact with database.

|  |
| --- |
| import java.sql.\*;  import java.util.\*;  class RetrieveRecords  {  public static void main(String[] args)throws SQLException  {  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","admin");  Statement stmt=con.createStatement();  ResultSet rs=stmt.executeQuery("select \*from product");  ArrayList a=new ArrayList();  System.out.println("before size is :"+a.size());  while(rs.next())  {  info.inetsolv.Product p=new info.inetsolv.Product();  p.setPid(rs.getInt("pid"));  p.setPname(rs.getString("pname"));  p.setPrice(rs.getDouble("price"));  a.add(p);  }  System.out.println("After the loop the size is :"+a.size());  Iterator itr=a.iterator();  while(itr.hasNext())  {  info.inetsolv.Product p=(info.inetsolv.Product)itr.next();  System.out.print(p.getPid()+"\t");  System.out.print(p.getPname()+"\t");  System.out.println(p.getPrice());  }  }  } |

**Run the above application:**

E:\JDBC &ORM1>**javac -d . Product.java**

E:\JDBC &ORM1>**javac RetrieveRecords.java**

Note: RetrieveRecords.java uses unchecked or unsafe operations.

Note: Recompile with -Xlint:unchecked for details.

E:\JDBC &ORM1>**java RetrieveRecords**

Before size is :0

After the loop the size is: 2

1 salt 10.0

2 rice 20.0

* Like JDBC, hibernate also can be integrated with web based applications. Because of this reason hibernate is called as reusable component.

**Procedure to use hibernate in web based applications:**

**Step1:** To work with hibernate in MyEclipse IDE we have to create new java project(web project).

**Step 2:** Create a package with the name info.inetsolv.

**Step 3:** Select the project and go to **MyEclipse menu** item and choose **project capabilities** and select **add** **Hibernate** **capabilities. (**This step same as standaloneapplication**).**

**Step 4:** Generate hbm files and POJO classes. **(**This step same as standaloneapplication**).**

**Step 5:** Develop the jsp with html code to display the form.

|  |
| --- |
| <body bgcolor="red">  <pre>  <center>  <form action="InsertProductValues.jsp">  <h1>Enter the product details</h1><h3>  product id :<input type="text" name="pid"><br><br>  product name :<input type="text" name="pname"><br><br>  product price:<input type="text" name="price"><br><br></h3>  <input type="submit" value="store">  </form>  </center>  </pre>  </body> |

**ProductForm.jsp**

**Step 6:** Develop the jsp.

|  |
| --- |
| <%@page import="org.hibernate.\*" %>  <%@page import="org.hibernate.cfg.\*" %>  <%  String pid=request.getParameter("pid");  String pname=request.getParameter("pname");  String price=request.getParameter("price");  %>  <%  Configuration cfg=new Configuration();  System.out.println("Configuration object created");  cfg.configure();  System.out.println("configure() called");  SessionFactory sf=cfg.buildSessionFactory();  System.out.println("SessionFactory object created");  Session hsession=sf.openSession();  Transaction tx=hsession.beginTransaction();  info.inetsolv.Product p=new info.inetsolv.Product();  p.setPid(pid);  p.setPname(pname);  p.setPrice(new Double(price));  hsession.save(p);  tx.commit();  hsession.close();  %>  data is stored in the database successfully.<br>  <a href="ProductForm.jsp">click here to add another record</a> |

**InsertProductValues.jsp**

**Step 7:** Deployee the project in the server.

* In the above web based application whenever we click on store button every time it calls the configure() and buildSessionFactory() methods. Calling these methods repeatedly in the project not recommended.
* We should call the configure() and buildSessionFactory() methods only once in the project. We should not call these methods repeatedly.
* In a project we should call the configure(), buildSessionFactory() methods only once in the project life time . To achieve this we use singleton design pattern.

**Design pattern:** Design pattern is nothing but a best solution for repeatedly occurred problems.

**What is a singleton design pattern?**

* Creating only one object to a class is called as singleton design pattern.
* The singleton design pattern solves the problems of calling the configure() method and buildSessionFactory() methods repeatedly in the project.
* Singleton design pattern make sure the configure() method and buildSessionFactory() methods called only once in the life time of the project.
* One of the project lead has developed **Cone class**. In the class we have a method **performBigTask()**. This method takes lot of time to execute. We need to make sure that this method is not repeatedly call in the project. The following is the Cone class.

|  |
| --- |
| class Cone  {  public void performBigTask()  {  System.out.println("calling the performBigTask() method");  System.out.println("performBigTask() method is executed for 10 minutes");  }  } |

**Cone.java**

* The following class make sure we can create Cone object only once and call the performBigTask() method only once.

|  |
| --- |
| class MyProjectSingleTon  {  private static Cone c;  static  {  c=new Cone();  c.performBigTask();  }  public static Cone getConeObject()  {  return c;  }  private MyProjectSingleTon()  {}  } |

**MyProjectSingleTon.java**

* The following java code is trying to get the Cone class object by using MyProjectSingleTon class. This class always returns only one object to Cone class.

|  |
| --- |
| public class MyApp  {  public static void main(String[] args)  {  Cone c1=MyProjectSingleTon.getConeObject();  Cone c2=MyProjectSingleTon.getConeObject();  System.out.println(c1);  System.out.println(c2);  }  } |

**MyApp.java**

* If we want to execute any statements or methods and to create any object only once it must be written in static block and variables must be in static because to access the variables from static block. For understand see above programs.
* The following singleton design pattern make sure the configure() method and buildSessionFactory() methods are called only once.

|  |
| --- |
| package info.inetsolv;  import org.hibernate.SessionFactory;  import org.hibernate.cfg.Configuration;  public class MyHibernateSingleTon  {  private static Configuration cfg;  private static SessionFactory sf;  static  {  cfg=new Configuration();  cfg.configure();  System.out.println("called the configure() method");  sf=cfg.buildSessionFactory();  System.out.println("called the buildSessionFactroy() method");  }  public static SessionFactory getSessionFactroyObject()  {  System.out.println("return the SessionFactroy object");  return sf;  }  private MyHibernateSingleTon()  {}  } |

**MyHibernateSingleTon.java**

* In the web based application i.e. in InsertProductValues.jsp to get the SessionFactory object we are using the following code.

|  |
| --- |
| SessionFactory sf=info.inetsolv.MyHibernateSingleTon.getSessionFactroyObject();  System.out.println(sf);  Session hsession=sf.openSession(); |

**InsertProductValues.jsp**

* From now onwards anybody who would like to get the SessionFactory object has to get with the help of MyHibernateSingleTon design pattern.
* In the applications we always use singleton design pattern. Whenever we use the IDE’s like MyEclipse IDE, RAD, NetBeans, and Eclipse automatically they develop the singleton design pattern program.
* When we develop hibernate application we always use SessionFactory object which is written by using singleton design pattern.
* When we use the programs like OraSF.java(oracle SessionFactory) directly we can get the Session object and perform the operations. By using OraSF.java itself we can close the Session object.
* The following the modified hibernate code which make sure that the configure() method and buildSessionFactory() method only once.

|  |
| --- |
| <%@page import="org.hibernate.\*" %>  <%@page import="org.hibernate.cfg.\*" %>  <%  String pid=request.getParameter("pid");  String pname=request.getParameter("pname");  String price=request.getParameter("price");  %>  <%  Session hsession=info.inetsolv.OraSF.getSession();  Transaction tx=hsession.beginTransaction();  info.inetsolv.Product p=new info.inetsolv.Product();  p.setPid(pid);  p.setPname(pname);  p.setPrice(new Double(price));  hsession.save(p);  tx.commit();  info.inetsolv.OraSF.closeSession();  %>  data is stored in the database successfully.<br>  <a href="ProductForm.jsp">click here to add another record</a> |

**InsertProductValues.jsp**

**Generators:**

**What is a generator?**

* Generator is one which helps us in generating the primary key values.
* We are using only one generator in the project.
* Generally in the project we should not allow the user to generate primary key values. Application should be able to generate the primary key values automatically.
* We can generate the primary key values by applying some algorithms. They are:

1. Find the max number of a particular table and add 1 to it and find the next primary key value.

**select max(pid)+1 from product;**

* As part of a database we have “**sequences**”. These will be able to generate next primary key value.
* Instead of we write the logic to generate primary key values hibernate has given predefined classes. These classes can generate the next primary key values. All these classes are available in a package “org.hibernate.id”.
* Hibernate can generate the primary key values. Hibernate guys has supplied predefined classes to generate the primary key values. All these classes are part of “**org.hibernate.id**” package.
* Hibernate identifies whether user is responsible to generate primary key values (or) hibernate is responsible to generate primary key values based on a tag generator.
* The following are some of the generator class names:

1. **GUIDGenerator.class**
2. **IncrementGenerator.class**
3. **HiLoGenerator.class**
4. **SequenceGenerator.class**

* All the generator classes are mapped with keywords.

1. **Assigned**
2. **Increment**
3. **Sequence**
4. **native**

**<generator class="assigned" />:**

* When the hibernate finds the above code it understands that user is responsible to supply the primary key values. If **class="assigned"** hibernate will not generate the primary key values.
* If **class=”assigned”** hibernate excepts the user to supply the primary key values. If he has not supply hibernate will through the exception **org.hibernate.id.IdentifierGenerationException**

**What is assigned?**

* Assigned is a keyword this keyword is mapped to a class. This class is executed and generates the primary key values.

**Increment:**

* When we use increment hibernate is responsible to generate primary key value. We have to add the following tag if hibernate need to generate the primary key value.

**<generator class="increment"/>**

* When the hibernate finds the above tag hibernate calls increment generator class to find the next primary key value and supply the primary key value to pojo class.
* Increment generator class internally uses the following logic to find the maximum no of primary key value and increment by 1.

**Select max(eno) from emp;**

**Hilo: <generator class="hilo" />:**

* When we use the above tag the hibernate generates a high value and then followed by a low value. Hibernate internally uses random function to generate a high value and then followed by a low value.

**Sequence: <generator class="sequence" />:**

* When the above line is executed hibernate internally creates one sequence and that from sequence it will get the primary key value.

**Native:** **<generator class="native" />:**

* When the above generator class is evaluated it will generate the sequence value based on the database. When we run this application in oracle it will generate the sequence because sequence is default generator for oracle. When run the same application in mysql it uses auto increment feature of mysql.

**Note:** Native will generate the primary key values based on the “**database vendor**”.

**Requirement:** Generate the primary key values by using hibernate software?

**Step 1:** Create the Product.hbm.xml file.

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">  <!--  Mapping file autogenerated by MyEclipse Persistence Tools  -->  <hibernate-mapping>  <class name="info.inetsolv.Product" table="PRODUCT" schema="VBR">  <id name="pid" type="java.lang.String">  <column name="PID" length="38" />  <generator class="**guid**" />  </id>  <property name="pname" type="java.lang.String">  <column name="PNAME" length="10" />  </property>  <property name="price" type="java.lang.Double">  <column name="PRICE" precision="10" />  </property>  </class>  </hibernate-mapping> |

**Product.hbm.xml**

**Step 2:** Develop the jsp with html code to display the form.

|  |
| --- |
| <body bgcolor="red">  <pre>  <center>  <form action="InsertProductValues.jsp">  <h1>Enter the product details</h1><h3>  product name :<input type="text" name="pname"><br><br>  product price:<input type="text" name="price"><br><br></h3>  <input type="submit" value="store">  </form>  </center>  </pre>  </body> |

**ProductForm.jsp**

**Step 3:** Develop the jsp to capture the data from the form and store the data into to database server.

|  |
| --- |
| <%@page import="org.hibernate.\*" %>  <%@page import="org.hibernate.cfg.\*" %>  <%  String pname=request.getParameter("pname");  String price=request.getParameter("price");  %>  <%  Session hsession=info.inetsolv.OraSF.getSession();  Transaction tx=hsession.beginTransaction();  info.inetsolv.Product p=new info.inetsolv.Product();  p.setPname(pname);  p.setPrice(new Double(price));  hsession.save(p);  tx.commit();  info.inetsolv.OraSF.closeSession();  %>  data is stored in the database successfully.<br>  <a href="ProductForm.jsp">click here to add another record</a> |

**InsertProductValues.jsp**

**What is hibernate reverse engineering?**

* Hibernate reverse engineering means generating pojo classes and hbm files from the tables is called as hibernate reverse engineering.

**Can we use primitive data types as part of pojo classes?**

* Primitive data types occupy less amount of memory when companied with wrapper classes.
* By using MyEclipse IDE we can generate the pojo classes with primitive data types (or) wrapper classes.
* If we want to generate the pojo classes with primitive data types we have to chosen an option **hibernate types**. If we want to generate pojo classes with wrapper classes we have to chosen an option **java types**.
* In java a primitive data type cannot hold a null value. For example:
* Int a =null (invalid)
* float f-=null (invalid)
* double d=null (invalid)
* When we try to assign like above the java compiler will not allow us to compile the program.

**Program:**

|  |
| --- |
| public class PrimitiveDataType{  public static void main(String args[]){  int a=null;  System.out.println(a);  }  } |

**Errors:**

D:\>javac PrimitiveDataType.java

PrimitiveDataType.java:3: incompatible types

Found : <nulltype>

Required: int

int a=null;

Note: in the above program we assign a null value to the primitive data type. Primitive data types cannot hold null values. Because the compiler throw the error.

* If we want to assign a null value we can use reference data types. The following is valid
* Integer i=null (valid)
* Double d=null (valid)
* Float f=null (valid)
* In hibernate as part of the pojo classes if we use primitive data types application will fail. This is because if any of the columns is having a null value we cannot store null value inside primitive data type. Because of this reason it’s always recommended to use wrapper classes rather than primitive data types.

**Can we use primitive data types as part of pojo classes?**

**Note:** while asking the interviewer tell below answer only.

* Yes, we can use primitive data types as part of pojo classes. But if we want to use primitive data types as part of pojo classes the database table should not contain null values.
* If the database table contains null values while retrieving the values from database table the application will fail. Because primitive data types cannot hold null values.
* Because of this it is not recommended to use primitive data types as part of pojo classes. It always recommended using wrapper classes only.

**What are the tags are available in hibernate configuration file?**

|  |
| --- |
| <hibernate-configuration>  <session-factory>  <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>  <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>  <property name="connection.username">vbr</property>  <property name="connection.password">admin</property>  <property name="connection.driver\_class">oracle.jdbc.driver.OracleDriver</property>  <property name="myeclipse.connection.profile">jdbc driver</property>  <mapping resource="org/student/Product.hbm.xml" />  <mapping resource="org/student/Emp.hbm.xml" />  </session-factory>  </hibernate-configuration> |

**What are the tags are available in hibernate mapping file?**

|  |
| --- |
| <hibernate-mapping>  <class name="org.student.Product" table="PRODUCT" schema="VBR">  <id name="pid" type="java.lang.Integer">  <column name="PID" precision="5" scale="0" />  <generator class="assigned" />  </id>  <property name="pname" type="java.lang.String">  <column name="PNAME" length="20" />  </property>  <property name="price" type="java.lang.Double">  <column name="PRICE" precision="10" />  </property>  </class>  </hibernate-mapping> |

**Can we provide any extra information to the hibernate softwate (or) hibernate configuration file?**

* Yes we can provide extra information to the hibernate software or hibernate configuration file. If we want to provide any extra information to hibernate software or hibernate configuration file we does that by adding a properties to hibernate configuration file.

**Example:**

**hbm2ddl.auto:** We add this property in hibernate configuration file by using a tag called as property. The following is the configuration of property in hibernate configuration file.

|  |
| --- |
| <hibernate-configuration>  <session-factory>  <property name="hbm2ddl.auto">update</property>  </session-factory>  </hibernate-configuration> |

* **Hbm2ddl.auto property creates the required tables in the data base.**

**show\_sql:** This property is used to display the query used by hibernate.

|  |
| --- |
| <hibernate-configuration>  <session-factory>  <property name="show\_sql">true</property>  </session-factory>  </hibernate-configuration> |

**How do you represent the primary key in hbm files?**

* By using <id> tag we can represent the primary key.
* Primary key represent with id tag.
* Every property will take name attribute and value attribute.
* Every property will having column name.

**How do you matain only one hbm file?**

* In a project if we want we can matain only one hbm file. But this is not recommended.

**Example:**

|  |
| --- |
| <hibernate-mapping>  <class name=*"org.student.Product"* table=*"PRODUCT"* schema=*"VBR"*>  <id name=*"productid"* type=*"java.lang.Integer"*>  <column name=*"PID"* precision=*"5"* scale=*"0"* />  <generator class=*"assigned"* />  </id>  <property name=*"productname"* type=*"java.lang.String"*>  <column name=*"PNAME"* length=*"20"* />  </property>  <property name=*"productprice"* type=*"java.lang.Double"*>  <column name=*"PRICE"* precision=*"10"* />  </property>  </class>  <class name=*"org.student.Emp"* table=*"EMP"* schema=*"VBR"*>  <id name=*"eno"* type=*"java.lang.Integer"*>  <column name=*"ENO"* precision=*"5"* scale=*"0"* />  <generator class=*"assigned"* />  </id>  <property name=*"name"* type=*"java.lang.String"*>  <column name=*"NAME"* length=*"20"* />  </property>  <property name=*"address"* type=*"java.lang.String"*>  <column name=*"ADDRESS"* length=*"30"* />  </property>  <property name=*"salary"* type=*"java.lang.Double"*>  <column name=*"SALARY"* precision=*"10"* />  </property>  </class>  </hibernate-mapping> |

**Can we change hbm file name?**

* Yes, we can change hbm file name if we change hbm file name we have to configure in hibernate configure file.

**Can we have the table column names and pojo class property names as different?**

* Yes, we can have the table column names and pojo class names as different. If we change the property names we have to update the hbm file also.

**Procedure to change the property names in pojo class:**

**Step 1:** In the pojo class change the setter methods and getter methods according to our convenient.

**Example:**

|  |
| --- |
| package org.student;  public class Product  {  private Integer productid;  private String productname;  private Double productprice;  public Integer getProductid() {  return productid;  }  public void setProductid(Integer productid) {  this.productid = productid;  }  public String getProductname() {  return productname;  }  public void setProductname(String productname) {  this.productname = productname;  }  public Double getProductprice() {  return productprice;  }  public void setProductprice(Double productprice) {  this.productprice = productprice;  }    } |

**Step 2:** Change the hbm files which uses the above properties.

|  |
| --- |
| <hibernate-mapping>  <class name="org.student.Product" table="PRODUCT" schema="VBR">  <id name="productid" type="java.lang.Integer">  <column name="PID" precision="5" scale="0" />  <generator class="assigned" />  </id>  <property name="productname" type="java.lang.String">  <column name="PNAME" length="20" />  </property>  <property name="productprice" type="java.lang.Double">  <column name="PRICE" precision="10" />  </property> |

HQL(hibernate query language)

* Hibernate software supporting different types of query API’s. They are:

1. HQL
2. Criteria API
3. Native Sql

* All the above three API’s in hibernate is used to send the query to the database using hibernate software.

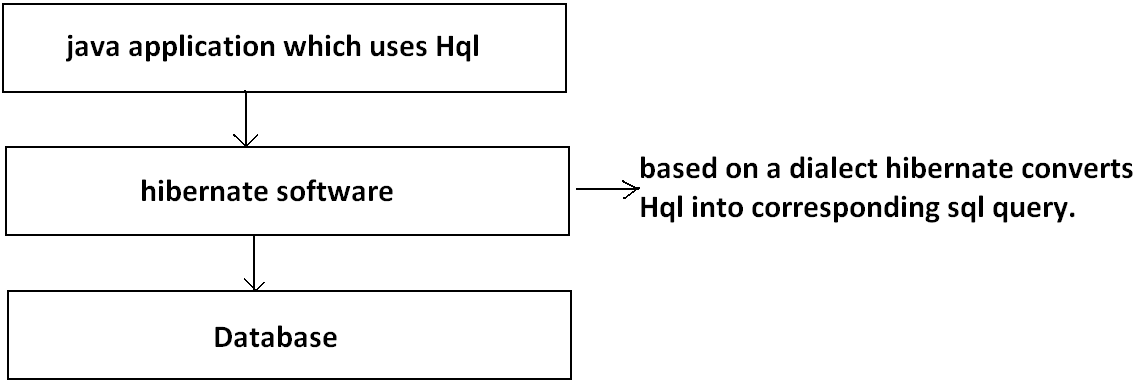
**Why we use these queries rather than Sql queries?**

* In hibernate we use Hql rather than sql queries. If we use the Hql queries we no need to change any query to communicate with any database.
* If we use jdbc by using normal queries, if there is a change in the database we have to test the queries in the new database.
* We can use the query API to communicate with any database by using hibernate.
* Bu using hibernate query API without changing the java code and queries we can connect to any database.

**Note:** By using Hql with hibernate we can connect to any database without code change.

* We cannot send Hql query directly to database. We give the Hql query to hibernate software. Hibernate software converts Hql into corresponding database sql.

**Diagram:**



* Hibernate software converts Hql query into sql query **based on a property called as** **dialect**.
* As part of hibernate software they have provided couple of predefined classes in a package **org.hibernate.dialect**.

**Note:** org.hibernate.dialect package is available in **hibernate3.jar** file.

* The classes which are available in this package are responsible to convert the Hql query in to the corresponding database query.
* As part of hibernate configuration file compulsory we have to provide the property with the name **dialect**.

**Example:** hibernate.cfg.xml

|  |
| --- |
| <session-factory>  <property name="dialect">  org.hibernate.dialect.Oracle9Dialect  </property>  </session-factory> |

**What is a dialect?**

* A dialect is a java class. As part of hibernate software they have provided couple of predefined dialect classes. All the predefined dialect classes are available in a package **org.hibernate.dialect**.

**What is the responsibility of the dialect?**

* The responsibility of dialect class is converting the Hql query into corresponding database query.
* To convert Hql query into corresponding database queries as part of hibernate configuration file compulsory we have to provide the property with the name dialect.

|  |
| --- |
| * The Hql queries will not start with the select keyword. * The Hql query doesn’t contain the table names. The Hql queries contain the name of the pojo class. |

* Write an Hql query which can retrieve all the records from product table.

|  |
| --- |
| **Sql:** select \*from **product**  Name of the Table  **Hql:** from **info.inetsolv.Product**    Name of the POJO class |

**Requirement:** Develop hibernate application which uses Hql query to retrieve the records from the product table.

**Program:**

|  |  |
| --- | --- |
| package info.inetsolv;  import java.util.ArrayList;  import java.util.Iterator;  import java.util.List;  import org.hibernate.Query;  import org.hibernate.Session;  import org.hibernate.SessionFactory;  import org.hibernate.Transaction;  import org.hibernate.cfg.Configuration;  public class RetrieveRecordsUsingHql {  public static void main(String[] args) {  //**step 1:** Create the configuration object and call the configure method.  Configuration cfg=new Configuration();  cfg.configure();  //**step 2:** Get the SessionFactory object with the help of configuration object.  SessionFactory sf=cfg.buildSessionFactory();  //**step 3:** Get the session object with the help of SessionFactory object.  Session hsession=sf.openSession();  //**step 4:** Start the transaction. To start the transaction creates the Transaction object with the help of session object.  Transaction tx=hsession.beginTransaction();  String hqlquery="from info.inetsolv.Product";  //**step 6:** Represent the query in the form of query object  Query query=hsession.createQuery(hqlquery);  //**step 7:** Send hqlquery to the database with the help of list() method.   |  | | --- | | List det=query.list();  //converting the List object into ArrayList object.  ArrayList list=(ArrayList)det;  Iterator i=list.iterator();  **(or)**  ArrayList list=(ArrayList)query.list();  Iterator i=list.iterator(); |   while(i.hasNext()){  Product p=(Product)i.next();  System.*out*.print(p.getPid()+"\t");  System.*out*.print(p.getPname()+"\t");  System.*out*.println(p.getPrice());  }  tx.commit();  hsession.close();  }  } |

**Output:**

1 pone 1000.0

2 ptwo 2000.0

* In jdbc if we want to send any query to the database we have to represent the query in the form of statement object.

**Example:**

|  |
| --- |
| Statement stmt=con.createStatement();  String query="select \*from product";  ResultSet rs=stmt.executeQuery(query); |

* In hibernate if we want to send Hql query to the hibernate 1st we have to represent the Hql query in the form of query object.

**Example:**

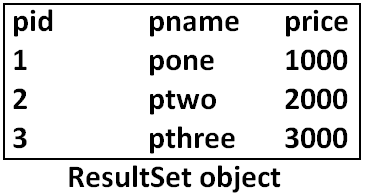
|  |
| --- |
| String hqlquery="from info.inetsolv.Product";  Query query=hsession.createQuery(hqlquery); |

* Once if represent the Hql query in the form of query object we can send the query object to hibernate by using list() method.
* The return type of the list() method is **java.util.List**.

**What will happen when we call the list() method on the query object?**

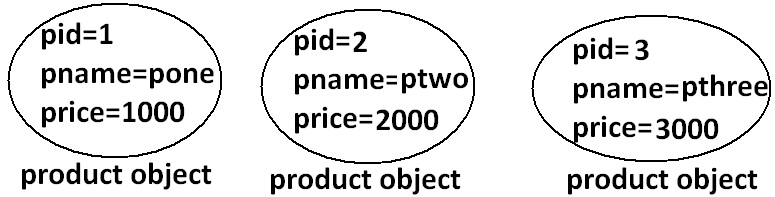
* The list() method get the Hql query from the query object and find the underlying database with the help of dialect class.
* Now the list() method converting the Hql query into corresponding database sql query and send it to the database. The database will execute the sql query and send the ResultSet object to the hibernate software.

**Diagram:**



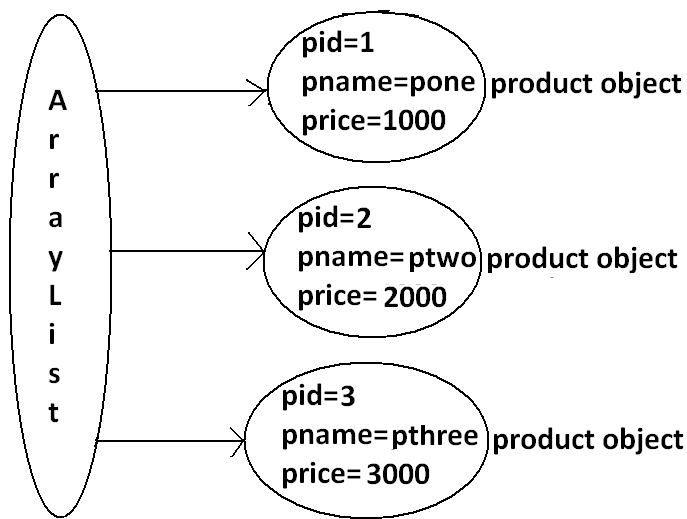
* Hibernate gets the records from the ResultSet object and place the data in the corresponding pojo class objects.

**Diagram:**



* Hibernate ads all these pojo class objects to ArrayList object.

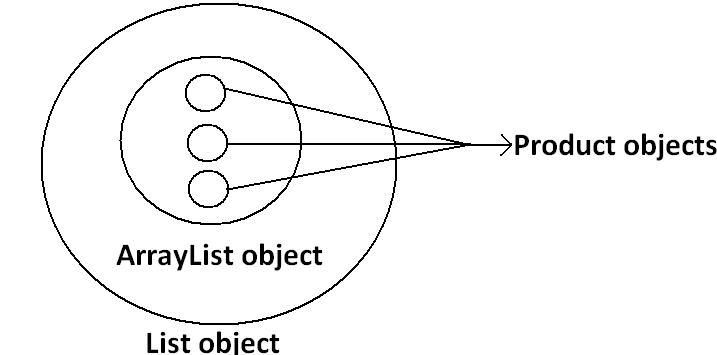
**Diagram:**



* Now the hibernate returns the ArrayList object to the list() method.

**Note:** list() method is caller to hibernate software because of this reason hibernate software will return the ArrayList object in the form of list object.

**Diagram:**



**What is a Sql query?**

* A Sql query is composed of set of clauses. The output of the sql query is not in the form of object.
* Hql query is also contains set of clauses. They are:

1. From
2. Select
3. Where
4. Group by
5. Order by………..etc.

* By using Hql through hibernate software we can communicate to any database.
* Hql queries are case sensitive.
* When we send Hql query to the database by using hibernate software the output of the Hql query is in the form of object(list object).

**What is the difference between sql and Hql?**

* In Sql we write Sql query by using table name and column name.
* In Hql we write Hql query by using pojo class name and properties.

**Using where clause in Hql:**

**Requirement:** Develop hibernate application which uses Hql query to find all the products whose price is greater than 1000 and less than 4000.

**Sql:** Select \*from product where price>1000 and price<5000

**Hql:** From info.inetsolv.Product where price>1000 and price<5000

**Program:**

|  |
| --- |
| package info.inetsolv;  import java.util.\*;  import org.hibernate.\*;  import org.hibernate.cfg.\*;  public class WhereClauseInHql {  public static void main(String[] args) {  //standard code from 1 to 4 steps  String hqlquery="from info.inetsolv.Product where price>1000 and price<5000";  //step6:represent the query in the form of query object  Query query=hsession.createQuery(hqlquery);  //step7:send hqlquery to the database with the help of list() method.  ArrayList list=(ArrayList)query.list();  Iterator i=list.iterator();  while(i.hasNext()){  Product p=(Product)i.next();  System.*out*.print(p.getPid()+"\t");  System.*out*.print(p.getPname()+"\t");  System.*out*.println(p.getPrice());  }  tx.commit();  hsession.close();  }  } |

**Output:**

2 ptwo 2000.0

3 pthree 3000.0

4 pfour 4000.0

**Requirement:** Develop hibernate application which uses Hql query to retrieve the record from product table whose PID=5.

**Sql:** select \*from product where PID=5

**Hql:** from info.inetsolv.Product where PID=5

**Requirement:** Develop hibernate application which uses Hql query to retrieve all the records whose price is greater than 2000.

**Sql:** Select \*from product where price>2000

**Hql:** From info.inetsolv.Product where price>2000

**Positional parameters in hibernate:**

* Positional parameter means **a query with question marks**.
* In jdbc we can use question marks as part of prepared statement.
* Like jdbc prepared statement we can use question marks as part of Hql queries.

**What is the difference between jdbc positional parameters and hibernate positional parameters?**

* In jdbc the positional parameter index starts with 1(one).
* In hibernate the positional parameter index starts with 0(zero).

**Can you have positional parameters as part of Hql?**

* Yes we can have positional parameters as part of Hql.

**Requirement:** Develop a hibernate application using Hql to retrieve the records whose productid is 1 and whose price is 100 [use positional parameter]

**Sql:** Select \*from product where PID=? And price=?

**Hql:** From info.inetsolv.Product where productid=? And productprice=?

**Program:**

|  |
| --- |
| package info.inetsolv;  //import the packages  public class RetrieveRecordsUsingAnd {  public static void main(String[] args) {  //standard code from 1 to 4 steps  //write the Hql query  String hqlquery="from info.inetsolv.Product where pid=? And price=?";  //step6:represent the Hql query in the form of query object  Query query=hsession.createQuery(hqlquery);  //passing the value to question marks.  query.setInteger(0,1);  query.setInteger(1,1000);  List productList=query.list();  Iterator i=productList.iterator();  while(i.hasNext()){  Product p=(Product)i.next();  System.out.println(p.getPid());  System.out.println(p.getPrice());  }  tx.commit();  hsession.close();  }  } |

**Output:**

1 1000.0

**Named positional parameters in hibernate**

* In hibernate instead of question marks we supply the variable names in the query. If we supply variable names in the query those queries are called as named parameters.

**What is the difference between positional parameters and named parameters?**

**Positional parameters:**

* Positional parameters will improve the performance.
* But positional parameters will not improve the readability of the code.

**Named parameters:**

* Named parameters will improve the readability of the code.
* But named parameters will not improve the performance.

**Requirement:** Develop a hibernate application using Hql to retrieve the records whose price is greater than 1000 and whose price less than 4000[use named parameters].

**Sql:** Select \*from product where price>1000 and price<4000

**Hql:** From info.inetsolv.Product where price>:minvalue and price<:maxvalue

**Note:** In Hql query variable name starts with colon. While passing the values to the variable the variable does not start with colon.

**Program:**

|  |
| --- |
| package info.inetsolv;  //import the packages  public class RetrieveRecordsUsingNamedParameters {  public static void main(String[] args) {  //standard code from 1 to 4 steps  //write the Hql query  String hqlquery="from info.inetsolv.Product where price>:minvalue and price<:maxvalue";  //step6:represent the query in the form of query object  Query query=hsession.createQuery(hqlquery);  //passing values to the variables.  query.setInteger("minvalue",1000);  query.setInteger("maxvalue",4000);  List productList=query.list();  Iterator i=productList.iterator();  while(i.hasNext()){  Product p=(Product)i.next();  System.out.print(p.getPid()+"\t");  System.out.println(p.getPrice());  }  tx.commit();  hsession.close();  }  } |

**Output:**

2 2000.0

3 3000.0

Note: we cannot insert a record with Hql query directly.

**Hql queries with select clause:**

**Requirement:** Develop hibernate application using Hql to retrieve the data from the specific columns.

* To implement this requirement in our Hql query we have to use the “select” keyword. The following are the sql and Hql queries.

**Sql:** Select \*from product

**Hql:** Select pid,pname,price from info.inetsolv.Product

Property names

**Program:**

|  |
| --- |
| package info.inetsolv;  //import the packages  public class RetrieveRecordsUsingHqlSelect {  public static void main(String[] args) {  //standard code from 1 to 4 steps.  //write the Hql query.  String hqlquery="select pid,pname,price from info.inetsolv.Product";  //represent the Hql query in the form of query object.  Query query=hsession.createQuery(hqlquery);  List productList=query.list();  Iterator i=productList.iterator();  while(i.hasNext()){  Object o = i.next();  Object item[]=(Object[])o;  System.out.print(item[0]+"\t");  System.*out*.print(item[1]+"\t");  System.*out*.println(item[2]);  }  tx.commit();  hsession.close();  }  } |

**Output:**

1 pone 1000.0

2 ptwo 2000.0

3 pthree 3000.0

4 pfour 4000.0

5 pfive 5000.0

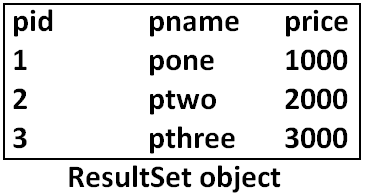
**When we send a Hql query which contains the select clause to the hibernate it performs the following steps:**

**Step 1:** Hibernate converts Hql query into corresponding sql query.

**Step 2:** Hibernate sends the sql query to database.

**Step 3:** Hibernate gets the ResultSet object with data.

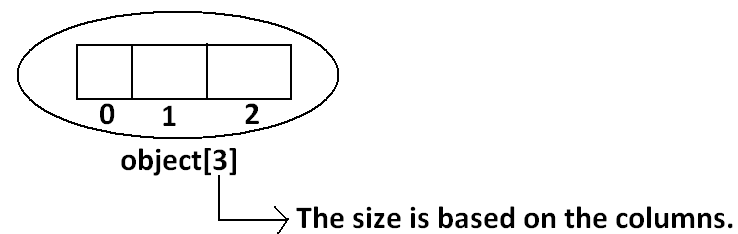
**Diagram:**



**Step 4:** Hibernate creates object array with the size which is matching to the no of columns selected by the user.

**Note:** Here no of columns is 3(three) because of this reason hibernate will create the object array with size 3(three).

**Diagram:**

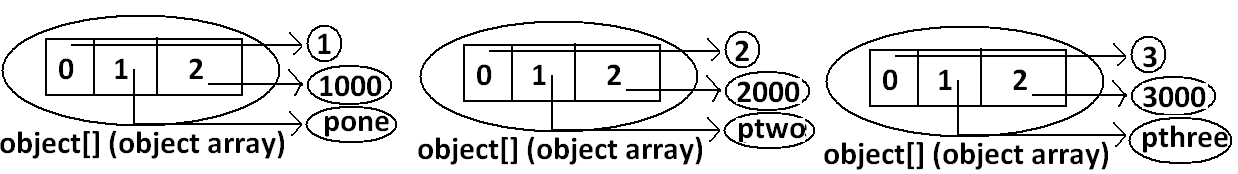


**What is there in the object array?**

* Data is stored in the form of indexes.

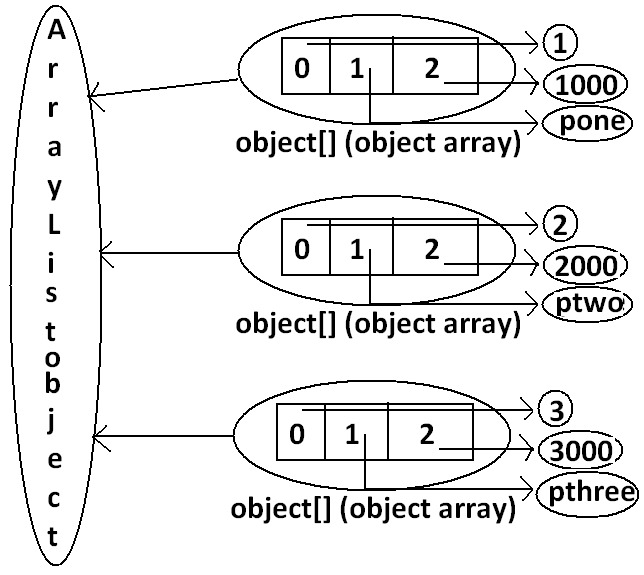
**Step 5:** Hibernate represents every record in the object array.

**Diagram:**



**Step 6:** Hibernate add object array to ArrayList.

**Diagram:**



**How do you retrieve the data from the table of a specific columns by using hibernate?**

* By using “select clause” we can retrieve the data from the table of specific columns.

**How do you retrieve the data from the table of a specific rows by using hibernate?**

* By using “where clause” we can retrieve the data from the table of a specific rows.

**Hql queries with update clause:**

* By using Hql if we want to “**update a record (or) delete a record**” we use a method “**executeUpdate**” on the query object.

**Requirement:** Develop hibernate application which update the record in the database by using Hql query.

**Sql:** Update product set price=20000 where pid=2

**Hql:** Update info.inetsolv.Product set price=20000 where pid=2

**Program:**

|  |
| --- |
| package info.inetsolv;  //import the packages  public class UpdateRecordUsingHql{  public static void main(String[] args) {  //standard code from 1 to 4.  //write the Hql query.  String hqlquery="update info.inetsolv.Product set price=20000 where pid=2";  //step6:represent the Hql query in the form of query object  Query query=hsession.createQuery(hqlquery);  //send Hql query to the database with the help of executeUpdate() method.  int noofupdatedRecords=query.executeUpdate();  System.*out*.println("updated records are/is:"+noofupdatedRecords);  tx.commit();  hsession.close();  }  } |

**Output:**

Updated records are/is: 1

**Hql queries with delete clause:**

**Requirement:** Develop a hibernate application which deletes the record from the database by using Hql query.

**Sql:** Delete from product where pid=1

**Hql:** Delete from info.inetsolv.Product where pid=1

**Program:**

|  |
| --- |
| package info.inetsolv;  //import the packages  public class DeleteRecordUsingHql{  public static void main(String[] args) {  //standard code from 1 to 4.  //write the Hql query.  String hqlquery="delete from info.inetsolv.Product where pid=2";  //step6:represent the Hql query in the form of query object.  Query query=hsession.createQuery(hqlquery);  int noofdeletedRecords=query.executeUpdate();  System.*out*.println("deleted records are/is:"+noofdeletedRecords);  tx.commit();  hsession.close();  }  } |

**Output:**

Deleted records are/is: 1

**Bulk update and delete records using Hql:**

**Hql queries with Bulk update records:**

**Requirement:** Develop a hibernate application which update all the records in the product table whose price>1000 using Hql.

**Sql:** Update product set price=price+500 where price >1000

**Hql:** Update info.inetsolv.Product set price=price+1000 where price>1000

**Program:**

|  |
| --- |
| package info.inetsolv;  //import the packages.  public class BulkUpdateRecordsUsingHql{  public static void main(String[] args) {  //standard code from 1 to 4.  //write the Hql query.  String hqlquery="update info.inetsolv.Product set price=price+1000 where price>1000";  //step6:represent the Hql query in the form of query object  Query query=hsession.createQuery(hqlquery);  //send Hql update query to the database using executeUpdate() method.  int noofupdatedRecords=query.executeUpdate();  System.*out*.println("updated records are/is:"+noofupdatedRecords);  tx.commit();  hsession.close();  }  } |

**Output:**

Updated records are/is: 4

**Hql queries with Bulk delete records:**

**Requirement:** Develop a hibernate application which delete all the records in the product table whose price>6000 using Hql.

**Sql:** Delete from product where price>1000;

**Hql:** Delete from info.inetsolv.Product where price>1000

**Program:**

|  |
| --- |
| package info.inetsolv;  //import the packages.  public class BulkDeleteRecordsUsingHql{  public static void main(String[] args) {  //standard code from 1 to 4.  //write the hql query.  String hqlquery="delete from info.inetsolv.Product where price>1000 ";  //**step 6:**represent the Hql query in the form of query object  Query query=hsession.createQuery(hqlquery);  //send Hql query to the database with the help of executeUpdate() method.  int noofdeletedRecords=query.executeUpdate();  System.*out*.println("updated records are/is:"+noofdeletedRecords);  tx.commit();  hsession.close();  }  } |

**Output:**

Updated records are/is: 4

**Hql queries with delete clause:**

**Requirement:** Develop a hibernate application which deletes the record from the database by using hql query.

**Sql:** Delete from product where pid=56

**Hql:** Delete from info.inetsolv.Product where ProductId=56

* Hql is not supporting the insert queries directly. We can use Hql to copy the record from one database table to another database table.
* By using Hql we can copy the records from one database table into another database table by using the following queries.

**Example:**

**Sql:** Insert into emp(eno,ename,salary) select pid,pname,price from product

**Hql:** Insert into info.inetsolv.Emp(eno,ename,salary) select pid,pname,price from info.inetsolv.Product

* **To find the no of records in the table the following query is suitable.**

**Example 1:**

**Sql:** Select count(\*) from Product;

**Hql:** Select count(\*) from info.inetsolv.Product;

**Example 2:**

**Sql:** Select count(\*)from Product group by pid;

**Hql:** Select count(\*) from info.inetsolv.Product group by pid;

**Legacy Database**

* We need to develop one java application which can store the EMP details in the database. The following are the details which we want to store in the database.

**Design1:** In this design we create one table. This table is going to hold all the EMP details.

**EMP Table:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Eno | Name | Salary | Street | City | State |
| 1 | Bhaskar | 30,000 | Srnagar | Hyd | AP |
| 1 | Bhaskar | 40,000 | Kukatpally | hyd | AP |

* In the above database table the data is being getting redundant. It is not recommended to hold the redundant in database. To remove the redundant we use normalization.

**Design2:** In the year of 2000 we have come up with the design2 in this design we have **Two** tables they are:

1. Emp
2. EmpAddr

**Emp Table:**

|  |  |  |
| --- | --- | --- |
| **Eno** | **Name** | **Salary** |
| 1 | Bhaskar | 10,000 |
| 2 | vijay | 20,000 |
| 3 | VijayBhaskar | 30,000 |

**EmpAddr Table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Eno(pk)** | **Addrno(pk)** | **Street** | **City** | **State** |
| 1 | 1 | Srnagar | Hyd | AP |
| 1 | 2 | YellaReddyGuda | Hyd | AP |
| 2 | 1 | KPHB | Hyd | AP |

**Design3:** As part of this design we contain two tables. They are

1. **Emp**
2. **Empaddr**

* In this design we have removed composite primary key. The address table contains a single column as a primary key.

**Emp Table:**

|  |  |  |
| --- | --- | --- |
| **Eno(pk)** | **Name** | **Salary** |
| 1 | Bhaskar | 10,000 |
| 2 | vijay | 20,000 |
| 3 | VijayBhaskar | 30,000 |

**EmpAddr:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **AddrSeq** | **Eno** | **Addrno** | **Street** | **City** | **State** |
| 1 | 1 | 1 | Srnagar | Hyd | 10,000 |
| 2 | 1 | 2 | YellaReddyGuda | Hyd | 20,000 |
| 3 | 2 | 1 | KPHB | Hyd | 30,000 |

* The customer has got a requirement to implement a new project on design two. Customer wants to use hibernate software.
* **Hibernate will not give best performance if it contains composite primary key**.
* Sometimes in the projects we need to apply hibernate software on legacy databases.

**What is a legacy database?**

* A database which contains “composite primary key” is called as legacy databases.

**Query1:**

SQL> create table emp(eno number(5) primary key,

2 name varchar2(10),

3 salary number(10,2));

**Query2:**

SQL> create table empaddr(eno number(5),

2 addrno number(5),

3 street varchar2(10),

4 city varchar2(20),

5 state varchar2(20),

6 primary key(eno,addrno));

* If we need to deal with the legacy databases that is which contains composite primary key we need to generate the following files. They are:

1. **Hibernate configuration file**
2. **Hbm files**
3. **Pojo classes**

* We will not find any change in hibernate configuration file.
* We required two hbm files. They are:

1. **Emp.hbm.xml**
2. **Empaddr.hbm.xml**

* When we observe the above two files we can find the difference in hbm files which contains composite primary key. That is we find the difference in **empaddr.hbm.xml**.
* **The following is the difference in hbm file.**

**Empaddr.hbm.xml:**

|  |
| --- |
| <hibernate-mapping>  <class name="info.inetsolv.Empaddr" table="EMPADDR" schema="VBR">  **<composite-id** name="id" class="info.inetsolv.EmpaddrId">  <key-property name="eno" type="java.lang.Integer">  <column name="ENO" precision="5" scale="0" />  </key-property>  <key-property name="addrno" type="java.lang.Integer">  <column name="ADDRNO" precision="5" scale="0" />  </key-property>  **</composite-id>**  <property name="street" type="java.lang.String">  <column name="STREET" length="10" />  </property>  <property name="city" type="java.lang.String">  <column name="CITY" length="20" />  </property>  <property name="state" type="java.lang.String">  <column name="STATE" length="20" />  </property>  </class>  </hibernate-mapping> |

* For the two tables **EMP** and **Empaddr** we need to generate **Three** pojo classes. They are:

1. **Emp.java**
2. **Empaddrid.java**
3. **Empaddr.java**

* There is no change in **EMP.java** pojo class. For EMP address we required two pojo classes they are **EmpAddrid.java**. this pojo class contains two properties[only the composite primary key columns]
* Empaddrid.java is called as composite primary key class.

**What is a composite primary key class?**

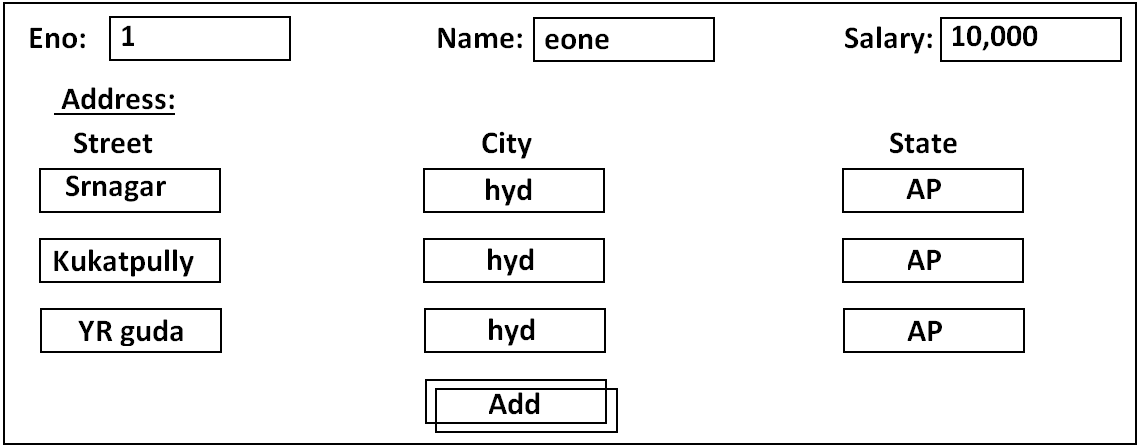
* A class which contains composite primary key is called as composite primary key class.

The composite primary key class contains the properties for composite primary keys and equals() method and hashcode() method.

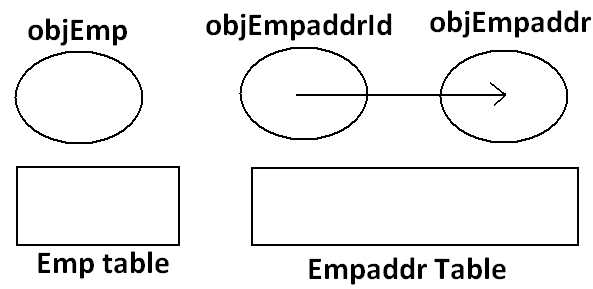
**Example:**

|  |
| --- |
| package info.inetsolv;  public class EmpaddrId {  private Integer eno;  private Integer addrno;  //provide setter and getter methods.  public boolean equals(Object other) {  }  public int hashCode() {  }  } |

**Assume this is jsp:**



**Diagram:**



**Program:**

|  |
| --- |
| **package** info.inetsolv;  import org.hibernate.Session;  import org.hibernate.Transaction;  public class LegacyExample {  public static void main(String[] args) {  Session hsession =OraSF.*getSession*();  Transaction tx=hsession.beginTransaction();  //create the Emp pojo class object  Emp objEmp =**new** Emp();  //Represent the data in the form of object.  objEmp.setEno(1);  objEmp.setName("eone");  objEmp.setSalary(10000d);  //create the EmpaddrId object  EmpaddrId objEmpaddrId1= **new** EmpaddrId();  //Represent the data in the form of object.  objEmpaddrId1.setEno(1);  objEmpaddrId1.setAddrno(1);  //create the Empaddr object  Empaddr objEmpaddr1=**new** Empaddr();  //Represent the data in the form of object.  objEmpaddr1.setStreet("Srnagar");  objEmpaddr1.setCity("hyd");  objEmpaddr1.setState("AP");  //Establish the relation between two tables  objEmpaddr1.setId(objEmpaddrId1);  //Ask hibernate to save the data in the database.  hsession.save(objEmp);  hsession.save(objEmpaddr1);  //End the transaction.  tx.commit();  System.*out*.println("Record successfully inserted");  //Close the session  OraSF.*closeSession*();  }  } |

**By using above program Insert multiple address at a time:**

|  |
| --- |
| package info.inetsolv;  import org.hibernate.Session;  import org.hibernate.Transaction;  public **class** LegacyExample {  public static void main(String[] args) {  Session hsession =OraSF.*getSession*();  Transaction tx=hsession.beginTransaction();  //create the Emp pojo class object  Emp objEmp =**new** Emp();  //Represent the data in the form of object.  objEmp.setEno(1);  objEmp.setName("eone");  objEmp.setSalary(10000d);  //create the EmpaddrId object  EmpaddrId objEmpaddrId1= **new** EmpaddrId();  //Represent the data in the form of object.  objEmpaddrId1.setEno(1);  objEmpaddrId1.setAddrno(1);  //create the Empaddr object  Empaddr objEmpaddr1=**new** Empaddr();  //Represent the data in the form of object.  objEmpaddr1.setStreet("Srnagar");  objEmpaddr1.setCity("hyd");  objEmpaddr1.setState("AP");  //Establish the relation between two tables  objEmpaddr1.setId(objEmpaddrId1);  EmpaddrId objEmpAddrId2 =**new** EmpaddrId();  objEmpAddrId2.setEno(1);  objEmpAddrId2.setAddrno(2);  Empaddr objEmpaddr2=**new** Empaddr();  //represent the data in the form of object  objEmpaddr2.setStreet("kukatpally");  objEmpaddr2.setCity("chennai");  objEmpaddr2.setState("AP");  objEmpaddr2.setId(objEmpAddrId2);  //Ask hibernate to save the data in the database.  hsession.save(objEmp);  hsession.save(objEmpaddr1);  hsession.save(objEmpaddr2);  //End the transaction.  tx.commit();  System.*out*.println("Record successfully inserted");  //Close the session  OraSF.*closeSession*();  }  } |

* Hibernate internally uses equals() method of composite primary key class. This is used to check whether the objects are same or not.

**Relationships:**

* In the database server the data will be stored inside the tables. Between these tables we will be having the relationships.
* The following are the relationships in the database server.

1. One to One relationship
2. One to Many relationship
3. Many to One relationship
4. Many to Many relationship

**One to One relationship:** In this relationship a record in one table is mapped with only one record in another table. These relationships are called as one to one relationship.

**One to Many relationship:** One record in one table is associated with more than one record in another table is called as one to many relationship.

**Many to One relationship:** Multiple records in one table is mapped with only one record in another table is called as many to one relationship.

* When we are dealing with hibernate we no need to bother about the relationship between the database tables. Hibernate and IDE will take care of the relationships between the tables in the database server.
* The following are two tables which are establish with one to many relationship.

**Table 1:**

**SQL> create table emp(eno number(5) primary key,**

**2 name varchar2(20),salary number(10,2));**

**Table 2:**

**SQL> create table empaddr(addrseq number(5)primary key,**

**2 eno number(5)references emp(eno),**

**3 addrno number(5),**

**4 street varchar2(20),**

**5 city varchar2(20),**

**6 state varchar2(20));**

* When we generate hbm files and pojo classes we have generated two pojo classes. They are:

1. **Emp.java**
2. **Empaddr.java**

* We have generated two hbm files. They are:

1. **Emp.hbm.xml**
2. **Empaddr.hbm.xml**

**How many properties are available in EMP.java?**

* In the Emp.java we have three regular properties and one extra property which represent the relationship between EMP table and other tables.

|  |
| --- |
| package org.student;  import java.util.HashSet;  import java.util.Set;  public class Emp implements java.io.Serializable {  private Integer eno;  private String name;  private Double salary;  private Set empaddrs = new HashSet(0);  public Integer getEno() {  return eno;  }  public void setEno(Integer eno) {  this.eno = eno;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public Double getSalary() {  return salary;  }  public void setSalary(Double salary) {  this.salary = salary;  }  public Set getEmpaddrs() {  return empaddrs;  }  public void setEmpaddrs(Set empaddrs) {  this.empaddrs = empaddrs;  }  } |

**Emp.java**

* For EMP table in the hbm file to represent one to many relationship it uses a tag **<set>**. In this to maintain one to many relationship we use a tag **<one-to-many>**.

|  |
| --- |
| <hibernate-mapping>  <class name="org.student.Emp" table="EMP" schema="VBR">  <id name="eno" type="java.lang.Integer">  <column name="ENO" precision="5" scale="0" />  <generator class="assigned" />  </id>  <property name="name" type="java.lang.String">  <column name="NAME" length="20" />  </property>  <property name="salary" type="java.lang.Double">  <column name="SALARY" precision="10" />  </property>  <**set** name="empaddrs" **cascade="all"**>  <key>  <column name="ENO" precision="5" scale="0" />  </key>  <one-to-many class="org.student.Empaddr" />  </**se**t>  </class>  </hibernate-mapping> |

**Emp.hbm.xml**

* In the Empaddr table we have the relationship with **eno** column of Emp table. From the Empaddr table to Emp table we have **many-to-one** relationship.
* “When we have **many-to-one** relationship to represent one relationship in the Empaddr pojo class we create the object to a class which is having one relationship”. In our example Empaddr table is having many-to-one relationship with Emp table.
* In the Empaddr pojo class we create the Object to Emp pojo class.
* The following is the pojo class for Empaddr.

|  |
| --- |
| package org.student;  public class Empaddr implements java.io.Serializable {  private Integer addrseq;  private Emp emp;  private Integer addrno;  private String street;  private String city;  private String state;  public Integer getAddrseq() {  return addrseq;  }  public void setAddrseq(Integer addrseq) {  this.addrseq = addrseq;  }  public Emp getEmp() {  return emp;  }  public void setEmp(Emp emp) {  this.emp = emp;  }  public Integer getAddrno() {  return addrno;  }  public void setAddrno(Integer addrno) {  this.addrno = addrno;  }  public String getStreet() {  return street;  }  public void setStreet(String street) {  this.street = street;  }  public String getCity() {  return city;  }  public void setCity(String city) {  this.city = city;  }  public String getState() {  return state;  }  public void setState(String state) {  this.state = state;  }  } |

**Empaddr.java**

* To represent many-to-one relationship in the hbm file we use the following tags.

|  |
| --- |
| <hibernate-mapping>  <class name=*"org.student.Empaddr"* table=*"EMPADDR"* schema=*"VBR"*>  <id name=*"addrseq"* type=*"java.lang.Integer"*>  <column name=*"ADDRSEQ"* precision=*"5"* scale=*"0"* />  <generator class=*"assigned"* />  </id>  **<many-to-one** name=*"emp"* class=*"org.student.Emp"* fetch=*"select"*>  <column name=*"ENO"* precision=*"5"* scale=*"0"* />  **</many-to-one>**  <property name=*"addrno"* type=*"java.lang.Integer"*>  <column name=*"ADDRNO"* precision=*"5"* scale=*"0"* />  </property>  <property name=*"street"* type=*"java.lang.String"*>  <column name=*"STREET"* length=*"20"* />  </property>  <property name=*"city"* type=*"java.lang.String"*>  <column name=*"CITY"* length=*"20"* />  </property>  <property name=*"state"* type=*"java.lang.String"*>  <column name=*"STATE"* length=*"20"* />  </property>  </class>  </hibernate-mapping> |

**Empaddr.hbm.xml**

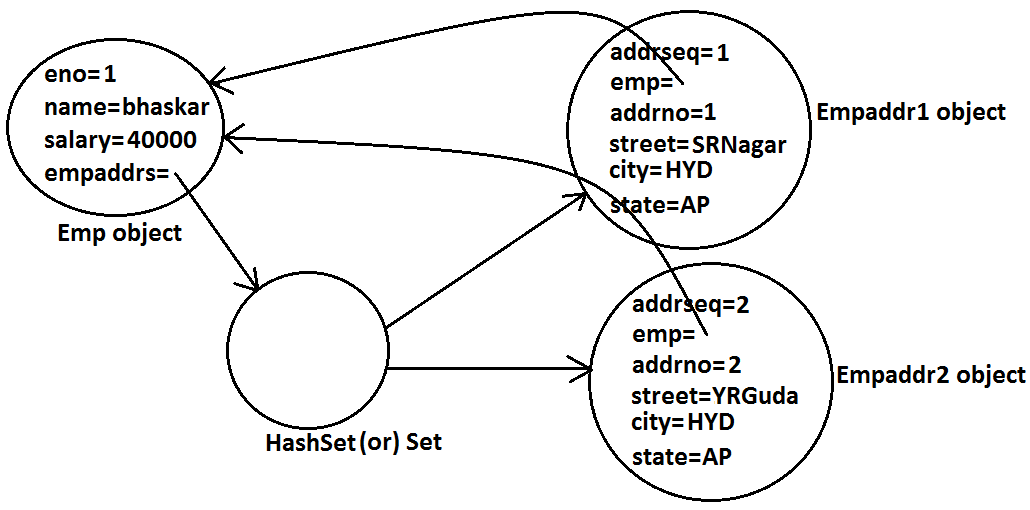
|  |
| --- |
| package org.student;  import org.hibernate.Session;  import org.hibernate.Transaction;  public class MyApp {  public static void main(String[] args) {  Session hsession=OraSF.*getSession*();  Transaction tx=hsession.beginTransaction();  org.student.Emp e=new org.student.Emp();  e.setEno(4);  e.setName("prasad");  e.setSalary(17000d);  org.student.Empaddr addr1=new org.student.Empaddr();  addr1.setAddrseq(1);  addr1.setAddrno(1);  addr1.setStreet("SRNagar");  addr1.setCity("HYD");  addr1.setState("AP");  addr1.setEmp(e);  org.student.Empaddr addr2=new org.student.Empaddr();  addr2.setAddrseq(2);  addr2.setAddrno(2);  addr2.setStreet("YRGuda");  addr2.setCity("HYD");  addr2.setState("AP");  addr2.setEmp(e);  java.util.Set s=e.getEmpaddrs();  s.add(addr1);  s.add(addr2);  hsession.save(e);  tx.commit();  OraSF.*closeSession*();  }  } |

* As part of the hbm file we have to specify one attribute **cascade="all".** This means if we perform any operation on the primary table it will carry out the operations on the dependent tables also.
* The **cascade** attribute can take multiple values. They are update, delete, insert (or) update-delete, insert-delete and etc. instead of specific value we can specify a value “**all”**.
* In the above program we create the object to Emp.java and create the object to Empaddr.java. In the EMP object we store one employee details and in the Empaddr table we store two employee address details in one EMP table of eno.
* Actually we save the data for every object. But we are using relations, in the Emp.java it generate one object is **Set** with instance variable. Like below.

|  |
| --- |
| package org.student;  import java.util.HashSet;  import java.util.Set;  public class Emp implements java.io.Serializable {  private Integer eno;  private String name;  private Double salary;  private Set empaddrs = new HashSet(0);  //provide setter and getter methods. |

* Actually Empaddr table of eno is assigned with EMP table of eno like this we create the table for EMP and Empaddr in database.
* Whenever we will doing hibernate reverse engineering then hibernate provide pojo classes and hbm files for EMP and Empaddr table.
* In the emp.java generate like above instance variable and Empaddr.java having instance variables and create the reference variable to emp.java. By this reference variable, we communicate with emp.java.
* Whenever we create the Empaddr object then we add the EMP object to Empaddr object. For understand see above program.
* In the above program we get the reference variable of Set object of emp.java and add Empaddr objects to Set object.
* Then we save the Set object(a Set) only. Then automatically store the data of all objects. But must be available **cascade=”all”** inside the <set> tag of Emp.hbm.xml file.
* If it is not available it will store only the data of which object is given as input of save() method. If **cascade=”all”** is available objects are executed circle wise and store the data in database. See the diagram given below.

**Diagram:**

****

* Like above will do one to many relations and Empaddr objects are added to Set object and Set object is get from the EMP object.
* In the relationships hibernate uses an attribute **cascade=”all”**. This attribute specify to hibernate saying hibernate has to find the relationships between all the tables.

**Example:**

|  |
| --- |
| <**set** name="empaddrs" **cascade="all"**>  <key>  <column name="ENO" precision="5" scale="0" />  </key>  <one-to-many class="org.student.Empaddr" />  </**se**t> |

**hbm.xml**

* From the above configuration we understood that hibernate automatically takes care of relationships. Whenever we call a save method on any object hibernate find the parent table and its child tables. so hibernate takes care of storing the data into parent table as well as child table.
* By using hibernate we would like to delete the records from parent table. The parent table is having some records in child table also. Now the hibernate takes care of deletions the records automatically from the child table as well as parent table.
* In this scenario we have to specify the cascading operation to delete.

**Example:**

|  |
| --- |
| <**set** name="empaddrs" **cascade="delete"**>  </**se**t> |

* If hibernate would like to take care of all the relationship between the tables we specify cascade=”all”.

**Example:**

|  |
| --- |
| <**set** name="empaddrs" **cascade="all"**>  </**se**t> |

**What is inverse and why do we required it?**

* When we use **inverse=”true”** hibernate insert record into parent table as well as child tables by using 3 queries. I.e. hibernate is finding the foreign key values and directly supply the foreign key values in the insert query. By default inverse=”true”.
* When we specify **inverse=”false”** first the hibernate insert the records into emp table and Empaddr table(parent and child) and the hibernate find the foreign key value and perform updation in the child table with the foreign key value.

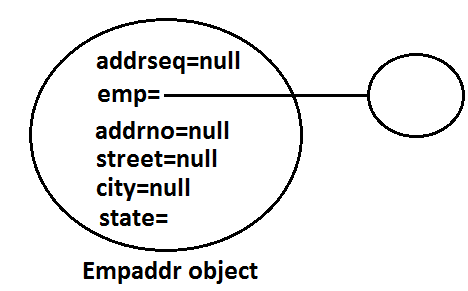
**How do you represent one-to-many?**

* By using **<set>** tag we can represent one-to-many relationship.

**How do you represent many-to-one?**

* By using a tag **<many-to-one>** we can represent many-to-one relationship.
* To represent many-to-one relationship in the pojo class we have to use the reference of the table which contains many to one relationship.

**Diagram:**

****

* In the hbm files to the <set> tag we have an attribute inverse.

|  |
| --- |
| <set name="empaddrs" inverse="true">  </set> |

* When we specify **inverse=”true”** hibernate takes care of managing the circular relationship.
* If hibernate need to store data in all the dependent tables we have to specify an attribute **cascade=*"all"***.

**Example:**

|  |
| --- |
| <set name=*"empaddrs"* inverse=*"true"* cascade=*"all"*>  </set> |

* When we specify **cascade=”all”** the hibernate updates (or) insert (or) delete all the records from dependent tables.

**Types of loading**

* To retrieve the data from database hibernate uses 2 types of loading techniques. They are:

1. **Aggressive loading**
2. **Lazy loading**

* By default hibernate uses lazy loading.

**What is lazy loading?**

* When we send a query to the database if hibernate gets the data from specific table then we call it as lazy loading.

**What is aggressive loading?**

* When we send a query to the database if it retrieves the data from all the dependent tables then we call hibernate as aggressive loading.

**How do you make hibernate as aggressive loading?**

* If we want to make hibernate as aggressive loading we need to specify the lazy attribute to **false**.
* The following tag shows how to make hibernate as aggressive.

**<set name=”empaddrs” inverse=”true” cascade=”all” lazy=”false”>**

**</set>**

* In the projects it is not recommended to use aggressive loading always we use lazy loading.

**Note:** loading mechanism/technique is applicable only for retrieve the data from database.

**Named queries:**

* Named queries will improve the performance of java application. If we write HQL queries in java every time we send the request the query has to be translated in to corresponding SQL query.

**Example:**

|  |
| --- |
| Query query=hsession.createQuery("from info.inetsolv.Product");  List list=query.list(); |

* When the above java code executed every time the HQL query will be converted into corresponding SQL query. Because of this we are getting the performance issues.
* Named queries will resolve the problem of converting HQL queries into corresponding SQL queries for multiple times.

**Procedure to use named queries:**

**Step 1:** Configure the named query in hbm file as shown below.

|  |
| --- |
| <hibernate-mapping>  <class………………..>  </class>  <query name="q1">  from info.inetsolv.Product  </query>  </hibernate-mapping> |

* Whenever the server reads the contents from hbm files the server translates the HQL query to corresponding SQL queries.
* Because of singleton design pattern the server reads hbm files only one time.
* In our application we have to write the java code to get the named queries from hbm files. To get the named queries we use a method getNamedQuery().
* The following example demonstrate use of named queries.

|  |
| --- |
| Query query=hsession.createQuery("from info.inetsolv.Product");  List list=query.list();  System.*out*.println(list); |

**Criteria API:**

**What is the main advantage of Criteria API?**

* Criteria API is used to represent the queries in the form of objects.
* Hibernate guys has provided an interface “org.hibernate.criteria interface”.
* Criteria api is mainly used in search operations.
* Criteria api gives better performance than Hql queries in case of search operation.

**Where do use criteria api?**

* In case of search operations only we use criteria API.

**What is a criteria?**

* Retrieve the data from database based on some conditions is called as criteria.
* To get the criteria object we use createCriteria() method.

**How do you convert Product class into class object?**

**Class c=Class.forName(“info.inetsolv.Product”);**

* **The following criteria example retrieves the data from Product table?**

|  |
| --- |
| package info.inetsolv;  import java.util.\*;  import org.hibernate.Criteria;  import org.hibernate.Session;  import org.hibernate.Transaction;  public class Myapp{  public static void main(String[] args) {  Session hsession=OraSF.getSession();  Transaction tx=hsession.beginTransaction();  //createCriteria() method takes the class object of the class.  Criteria c=hsession.createCriteria(Product.class);  List list=c.list();  Iterator i=list.iterator();  while(i.hasNext())  {  Product p=(Product)i.next();  System.out.print(p.getPid()+"\t");  System.out.print(p.getPname()+"\t");  System.out.println(p.getPrice());  }  }  } |

* Adding restrictions is nothing but calling Restriction class.
* If we want retrieve the data by using different criteria we have to add restrictions. Hibernate has provided predefined “Restriction” class in “org.hibernate.criteria package”.
* To add the restriction to the criteria we use a method add().

|  |
| --- |
| package info.inetsolv;  import java.util.\*;  import org.hibernate.Criteria;  import org.hibernate.Session;  import org.hibernate.Transaction;  import org.hibernate.criterion.Restrictions;  public class Myapp{  public static void main(String[] args) {  Session hsession=OraSF.*getSession*();  Transaction tx=hsession.beginTransaction();  Criteria c=hsession.createCriteria(Product.class);  //c.add(Restrictions.eq("pid","1"));  //c.add(Restrictions.eq("pname", "ptwo"));  //c.add(Restrictions.eq("price", 3000d));  List list=c.list();  Iterator i=list.iterator();  while(i.hasNext())  {  Product p=(Product)i.next();  System.*out*.print(p.getPid()+"\t");  System.*out*.print(p.getPname()+"\t");  System.*out*.println(p.getPrice());  }  }  } |

* Always criteria API is used in combination of restrictions.
* Like we can add the restriction to search for appropriate names in the database by using “like” restriction.

|  |
| --- |
| package info.inetsolv;  import java.util.\*;  import org.hibernate.Criteria;  import org.hibernate.Session;  import org.hibernate.Transaction;  import org.hibernate.criterion.Restrictions;  public class Myapp{  public static void main(String[] args) {  Session hsession=OraSF.getSession();  Transaction tx=hsession.beginTransaction();  Criteria c=hsession.createCriteria(Product.class);  //c.add(Restrictions.like("pname", "%p%"));  c.add(Restrictions.like("pname", "p%"));  List list=c.list();  Iterator i=list.iterator();  while(i.hasNext())  {  Product p=(Product)i.next();  System.out.print(p.getPid()+"\t");  System.out.print(p.getPname()+"\t");  System.out.println(p.getPrice());  }  }  } |

**Between:** This Restriction is used to find the values between 2 numbers.

|  |
| --- |
| package info.inetsolv;  import java.util.\*;  import org.hibernate.Criteria;  import org.hibernate.Session;  import org.hibernate.Transaction;  import org.hibernate.criterion.Restrictions;  public class Myapp{  public static void main(String[] args) {  Session hsession=OraSF.*getSession*();  Transaction tx=hsession.beginTransaction();  Criteria c=hsession.createCriteria(Product.class);  c.add(Restrictions.*between*("price",2000d,4000d));  List list=c.list();  Iterator i=list.iterator();  while(i.hasNext())  {  Product p=(Product)i.next();  System.*out*.print(p.getPid()+"\t");  System.*out*.print(p.getPname()+"\t");  System.*out*.println(p.getPrice());  }  }  } |

* Fetchsize means how many records the database retrieves from the table.
* setFirstResult(2) will return the first two records of the result.

|  |
| --- |
| package info.inetsolv;  import java.util.\*;  import org.hibernate.Criteria;  import org.hibernate.Session;  import org.hibernate.Transaction;  import org.hibernate.criterion.Restrictions;  public class Myapp{  public static void main(String[] args) {  Session hsession=OraSF.*getSession*();  Transaction tx=hsession.beginTransaction();  Criteria c=hsession.createCriteria(Product.class);  //c.setFetchSize(5);  c.setFirstResult(2);  System.*out*.println();  List list=c.list();  Iterator i=list.iterator();  while(i.hasNext())  {  Product p=(Product)i.next();  System.*out*.print(p.getPid()+"\t");  System.*out*.print(p.getPname()+"\t");  System.*out*.println(p.getPrice());  }  }  } |

**Native SQL:**

* In hibernate we can execute the normal SQL queries which can be directly executed in database.
* To work with native SQL we use a method createSQLQuery().
* When we call the createSQLQuery() method it will return SQLQuery object.

|  |
| --- |
| SQLQuery s=hsession.createSQLQuery("select \*from product");//native SQL (or) Normal SQL Query |

* We send the native SQL to the database by using a method list(). When the native SQL is executed the data will be placed in Object[] array and it will be added to list() object.
* To get the data from list object we have to get the Object[] array and from the Object[] array we get the data.

|  |
| --- |
| package info.inetsolv;  import java.util.\*;  import org.hibernate.SQLQuery;  import org.hibernate.Session;  import org.hibernate.Transaction;  public class Myapp{  public static void main(String[] args) {  Session hsession=OraSF.*getSession*();  Transaction tx=hsession.beginTransaction();  SQLQuery s=hsession.createSQLQuery("select \*from product");  List list=s.list();  Iterator i=list.iterator();  while(i.hasNext())  {  Object o[]=(Object[])i.next();  System.*out*.print(o[0]+"\t");  System.*out*.print(o[1]+"\t");  System.*out*.println(o[2]);  }  }  } |

* By using native SQL if we want to represent the data in the form of objects (or) entity we use a method addEntity().

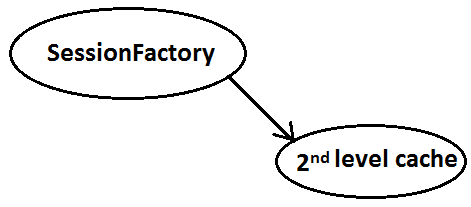
**Example:**

|  |
| --- |
| package info.inetsolv;  import java.util.\*;  import org.hibernate.SQLQuery;  import org.hibernate.Session;  import org.hibernate.Transaction;  public class Myapp{  public static void main(String[] args) {  Session hsession=OraSF.*getSession*();  Transaction tx=hsession.beginTransaction();  SQLQuery s=hsession.createSQLQuery("select \*from product");  s.addEntity(Product.class);  List list=s.list();  Iterator i=list.iterator();  while(i.hasNext())  {  Product p=(Product)i.next();  System.*out*.print(p.getPid()+"\t");  System.*out*.print(p.getPname()+"\t");  System.*out*.println(p.getPrice());  }  }  } |

**Second level cache:**

* Whenever we got the SessionFactory object it will create the second level cache object.

**Diagram:**



* The hibernate uses 2nd level cache when we retrieve the records from database.
* We can enable the 2nd level cache by using a tag <cache>. When we enable the 2nd level cache and retrieve the records the hibernate will add the object to 1st level cache as well as 2nd level cache also.
* When we want to retrieve for 2nd time if the object is available in 2nd level cache hibernate will not send query to the database.
* Instead of sending a query hibernate returns the object from 2nd level cache.
* Most of the projects we will not enable the 2nd level cache. If we enable the 2nd level cache if someone updates the data in database table the updated records will not be reflected in 2nd level cache until we restart the server.

**ORM levels:**

* ORM is categorized into 4 levels. They are:

1. Pure relational (stored procedure)
2. Light objects mapping (JDBC) (or) Light ORM
3. Medium object mapping (or) Medium ORM
4. Full object Mapping (composition, inheritance, polymorphism, persistence by reachability) (or)Full ORM

**Pure relational ORM:**

* Pure relational orm means we are not representing any data in the form of object.
* If we write the business logic in procedures we call them as pure relational orm.

**Light ORM:**

* Representing the data in the form of Resultset object is called as light orm.

**Medium ORM:**

* Representing the data by using the Resultset object and procedures is called as medium orm.

**Full ORM:**

* Representing the entire data by using objects is called as full orm.

**Hibernate states:**

* Hibernate follows three states. They are:

1. Transient state (or) ideal state
2. Persistent state
3. Detached state

**Transient state:**

* Transient state means when the object is created and store the data in the object, it will not add to first level cache memory. (or)
* The instance is not associated with any persistence context. It has no persistent identity or primary key value.

**Persistent state:**

* Persistent state means when the object is created to pojo class and store the data in the object, then this object is added to first level cache memory. (or)
* The instance is currently associated with a persistence context. It has a persistent identity (primary key value).

**Detached state:**

* Detached state means whenever the object is deleted from the first level cache memory. (or)
* The instance was once associated with a persistence context, but that context was closed, or the instance was serialized to another process.
* Whenever we write the !DoCTYPE in hbm files and configuration files, whatever mentioned in dtd files(hibernate-config.dtd, hibernate-mapping.dtd)in that files we copy the !DoCTYPE and past our hbm and configuration files.
* For dtd files we can extract the jar file is **hibernate3.jar**.

**SERVLETS**

**Web Technologies:** With the help of java software we can develop three types of applications.

They are:

1. Standalone applications
2. Applets
3. Web based applications (Servlet, jsp, Structs, ….etc)

**Standalone applications:** Stand alone application runs on the client computer. They consume the resources of the client computer. [Ram, hard disk, processor…etc].

* Every stand alone applications having **public static void** **main().**

**Disadvantages of standalone application:**

* Every stand alone application requires an installer to install software.
* Standalone applications occupy the client memory.

**Applets:** initially Sun Microsystems has released applets to work with web based applications. Applet runs on the client computer. Because of so many security reasons sun Microsystems has deprecated applet API.

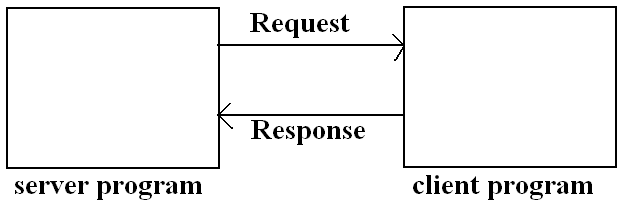
**Web based applications (or) server side applications:**

* The application which runs inside a server is called web based application.
* The web based applications are dependent on client server computing (architecture).
* As part of web based applications we developed Servlets, jsp programs. These programs run inside server. That is the reason these programs are called as server side programs.
* As part of the web based applications two parties are involved in executing the program. They are:

1. server program
2. Client program

* In the client server programs client sends request to the server program. Once if server receives the request server understand the request and process the request. Server gets the output and sends it to the client. The server sends the output to the client in the format of response.

**Diagram:**



* All the web based applications are dependent on a protocol calls as **HTTP protocol** (hyper text transfer protocol).
* To develop a web based application we required Web server program. Web server program can be developed in java language. The web server program can accept multiple requests at the same time and it can process multiple requests at the same time.
* If we want to develop our own web server it takes lot of time. Because of these reasons we use the web server programs developed by server vendors.
* The following are the most popular web servers are available in the market.

They are:

* Web logic

1. Tomcat
2. JBose
3. Sun one server
4. Praghathi J2EE sever(HYD)
5. IIS…etc

* There are so many web client programs are available in the market. They are:

1. Internet Explorer
2. Firefox
3. Mozilla
4. Nets scape navigator
5. Opera
6. Crome…..etc

* The web client program is responsible to establish connection with web server program after the connection is established. The web client program can send request to the server.

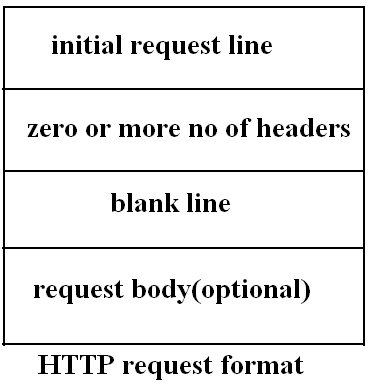
**Note:** All the web server programs and web client programs are developed based on HTTP protocol.

* HTTP protocol is an open specification (anyone in this world can be used).
* To develop any web based applications we use web server program. The web server program is developed in java language by following **http protocol**. The web client programs also developed using http protocol. The web client sends the request to the server using **HTTP request format**. The web client receives the response in **HTTP response format**.

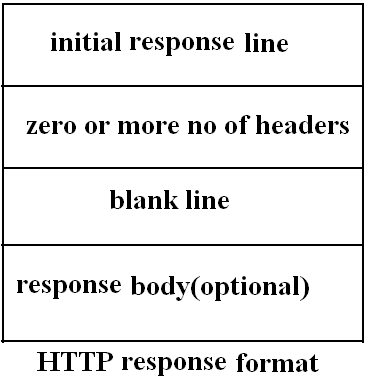
**Note:** As a programmer we should aware of http request format and http response format.

* HTTP request format if any client wants to send a request to the server they have to follow the following http request format.

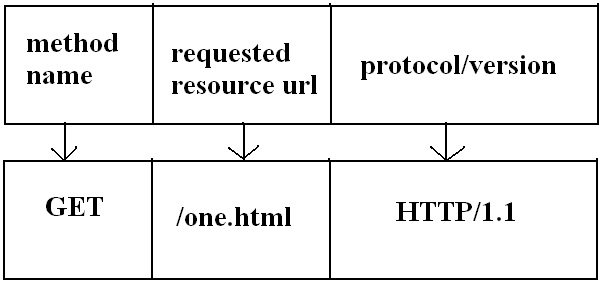
**HTTP request format diagram:**

****

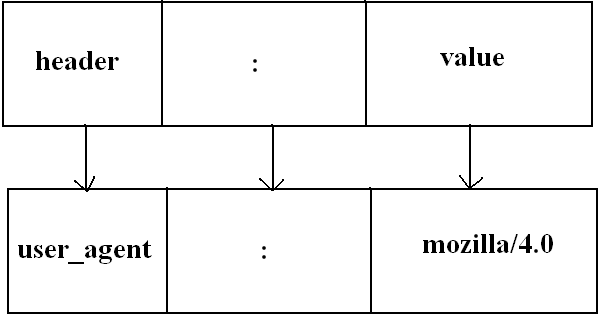
**HTTP response format diagram:**



**Initial Request line:**



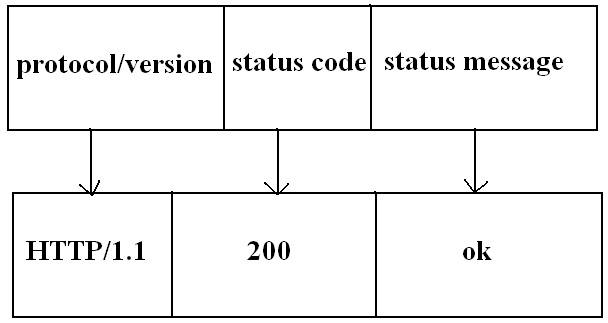
* As part of initial request line will be having method name, requested resource URI (uniform resource identifier), protocol/version number.
* There are different types of methods are available which are supported by http protocol. They are:
  + - 1. GET
      2. PUT
      3. POST
      4. DELETE
      5. TRACE
      6. HEAD
      7. LOCATE
* As part of http protocol we use **“\r\n”** to enter into next line.
* Every header will be having three pieces



* Here colon is a separator.
* Every header will be having name and value.
* The following are headers sent by the client to the server.

1. **Accept**
2. **Accept language**
3. **User\_agent**
4. **Connection**
5. **Host**

**Initial Response Line:**



**Status codes:**

* + 1xx : information
  + 2xx : success message
  + 3xx : redirection
  + 4xx : client request resource is not available in the server.
  + 5xx : problem is occurred while processing your request.

**The following are headers sent by the server to the client:**

* Server
* Date
* Contentment type
* Content type header is used by the client program to identify what the content is sent by server to client.
* The client renders the data based on Content-Type header.
* Content type header can take the following values.
* **Content-type: text/html**
* **Content-type: text/xml**
* **Content-type: application/pdf**
* **Content-type: application/ms-application…..etc.**

**1xx:** If we got a status code 1xx it indicates the server has given information to client.

**2xx:** We get this status code when the request resource of the client is available in the server.

**3xx:** We get this status code when the request resource was redirect to another server.

**4xx:** When the requested resource of client is not available in the server then we get 4xx.

**5xx:** When the requested resource is being getting executed in the server and if encounters a problem then the server will send 5xx message.

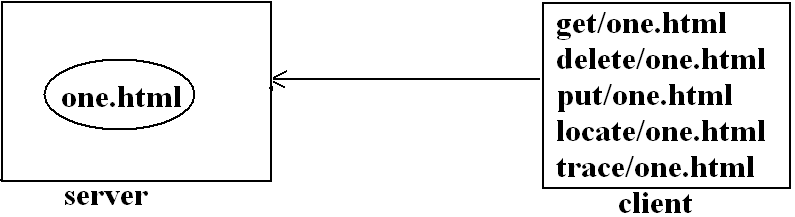
* For every status code is associated with status message. For example 200 is associated with ok and 404 is associated with not found and etc.

**Get() method:** This is the default method supported by every browser. This method is used if we want to execute a resource in the server and get the response of the request.

Note: every server supports this method.

**Put() method:** We can use put() method if we want to place a file inside the server. None of the servers supports this method this is because of security reason(virus).

**Diagram:**

****

**Delete():** Delete() method is used to delete the resource from the server. This method is not supported by any server because of security reasons.

**Trace() and locate():** These methods are used to search an item(file) inside the server. These methods are not supported by the servers because of security reasons.

**Note:** None of the server’s supports locate () and trace () methods because of the security reasons.

**Head():** This method is used to get the header information of the server to the client.

**Note:** Every server supports this header method.

**What is the difference between get() and post() methods?**

* Both the methods are used to send the request to the server. get () method sends the data to the server by appending the data to the URL. The following is an example for the same.

[**http://localhost:9000/userlogin.html/**](http://localhost:9000/userlogin.html/)**?**

Uname=myusername&pwd=mypassword.

* When we use the post method the data will not be append to the URL. Instead of that the browser places the data **inside http request format body** and sends it to the server.

**Procedure to develop any web based applications:**

**Step1:** Create a folder with the name as our project name. [**Myproject (or) LMS (or) DMTK**].

**Step2:** Create a folder with the name **WEB\_INF**. [This folder is called as private folder because the files which are available inside this folder can only accessible by the server. The client cannot access files which are available inside WEB\_INF directory].

**Step3:** Create the following two folders inside capital WEB\_INF directory. They are:

1. **Classes**
2. **Lib**

**Step4:** Create a file whose name is **web.xml** in this file we have to add the following root tag.



**Step5:** A project is a collection of **html resource files**. Develop the html files and place them inside our project folder [Myproject (or) LMS (or) DMTK] outside to **WEB\_INF** folder.

* After we develop the application **we are responsible to deploy** the project in the server.

**What is the meaning of deployment?**

* Placing our project inside **a specific folder of server**/**web** **server** is called deployment.
* Every project contains set of resource files.

**Which files are considered as resource files?**

* HTML, JavaScript, css, Servlets, jsp’s and etc are called as resources.
* After we are ready with web based application we need to **deploy** the project in the server.

**Approach1:** Generally every server vendor provides a folder the programmer is responsible to copy the project inside that folder. In case of **Tomcat** the name of the folder is **webapps**.

**Note:** We can find the folder in the following path **C:\Program Files\Apache Software Foundation\Tomcat 6.0.**

* After the deployment is done we need to perform unit testing to check whether the project is **deploy or not** we have to follow the following steps to carry out the unit testing.

**Step1:** Open the web client program.

**Step2:** Use the following URL to access the resources available in the project.

[**http://localhost:8000/myproject/one.html**](http://localhost:8000/myproject/one.html)

**Approach2:** In this approach we create **a war file** and deliver the war file to the customer. Customer can place **war file** inside the **webapps directory** and start the server.

**Procedure to create a war file:**

**Step1:** Open the command prompt goes the folder where our project is available.

**Step2:** Issue the following jar command to create the war file

**C:\work\myproject>jar –cvf myproject.war .**

**Approach3**: We can deploy the **war file** by using the **server admin console**. The following is the URL to access the tomcat admin console (**this is real time approach**).

[**http://localhost:8000/**](http://localhost:8000/)

* After we got an admin console choose **tomcat manager** and provide the **username** and **password**.

**Developing the project in weblogic server:**

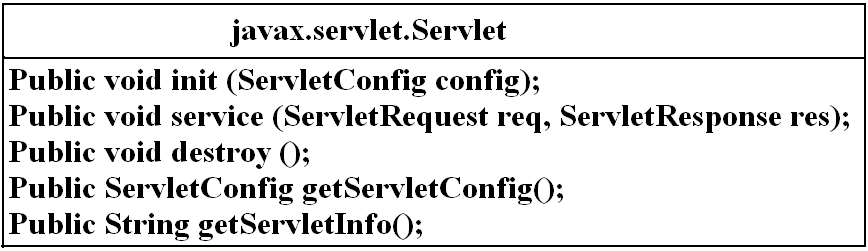
**Approach1:** In the weblogic server we can find a folder **autodeploy.** If you **copy, past your** project inside **autodeploy** folder and when we start the weblogic server, automatically the project will be deployed.

**Approach2:** As part of weblogic server, we can deploy our web based application, through **weblogic server admin console**. Once if we login to the admin console, we can find an option **deployment**. We have to follow the instructions given in the admin console, to perform the deployment.

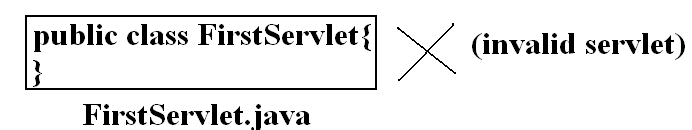
**What is a Servlet program?**

* Servlet is an open specification released by sun micro system. Servlet API is used to develop dynamic server side applications. (Or)
* **Servlet is a java program (or) an object** which provides the implementation of Servlet interface directly (or) indirectly.
* Sunmicrosystem has released all the classes and interfaces inside the following two packages. They are:

1. Javax.servlet
2. Javax.servlet.http



* The following are the most important interfaces of Servlets. They are:
* Javax.servlet.Servlet
* Javax.servlet.ServletConfig
* Javax.servlet.ServletRequest
* Javax.servlet.ServletResponse
* Javax.servlet.ServletContext
* The following program is not call as a Servlet.



* The following program is called as a Servlet program.



**Note:** A Servlet program should not contain public static void main().

* If we want to share data in two methods we have to declare the **instance variables**.

**Procedure in developing a Servlet program:**

**Step1:** Write a Servlet program

|  |
| --- |
| public class FirstServlet implements Servlet {  //to share the data more than one method we use the instance variables  //create the instance variables  ServletConfig config;  //create the default constructor  public FirstServlet(){  System.out.println ("FirstServlet object is created ..............");  }  //init () method acting as a setter() method  public void init(ServletConfig config){  this.config=config;  System.out.println ("first Servlet init() is executed ..........");  }  public void service(ServletRequest req, ServletResponse res){  System.out.println ("first Servlet service() is executed.......");  }  public void destroy(){  System.out.println ("first Servlet destroy() is executed......");  }  //getServletConfig () method acting as getter() method  public ServletConfig getServletConfig(){  return config;  }  public String getServletInfo(){  return "first Servlet example program......";  }} |

**Step2:** Compile the Servlet program. Before compile copy the **servlet-api.jar** file to the current working directory from the following location.

**C:\Program Files\Apache Software Foundation\Tomcat 6.0\lib.**

* Set the class path to the **servlet-api.jar** by using the following command.

**D:\work>set classpath=servlet-api.jar;.;**

**Step3:** Copy the **Servlet class file** inside the **classes’ directory** of our web based application.

**Step4:** Provide the information about Servlet inside **web.xml** (deployment descriptor).

**Web.xml file:**

|  |
| --- |
| <web-app>  <servlet>  <servlet-name>mfs</servlet-name>  <servlet-class>FirstServlet</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>mfs</servlet-name>  <url-pattern>/fs</url-pattern>  </servlet-mapping>  </web-app> |

**Step5:** Deploy the project in **server and perform unit testing**.

**Compile the Servlet program:**

D:\work>javac FirstServlet.java

FirstServlet.java:1: cannot find symbol

Symbol: class Servlet

public class FirstServlet implements Servlet{

FirstServlet.java:4: cannot find symbol

Symbol: class ServletConfig

Location: class FirstServlet

ServletConfig config;

FirstServlet.java:10: cannot find symbol

Symbol: class ServletConfig

Location: class FirstServlet

public void init(ServletConfig config){

FirstServlet.java:14: cannot find symbol

Symbol: class ServletRequest

Location: class FirstServlet

public void service(ServletRequest req, ServletResponse res){

FirstServlet.java:14: cannot find symbol

Symbol: class ServletResponse

Location: class FirstServlet

public void service(ServletRequest req, ServletResponse res){

FirstServlet.java:21: cannot find symbol

Symbol: class ServletConfig

Location: class FirstServlet

public ServletConfig getServletConfig(){

* To resolve the above errors we have to import the import statement the import statement is **import javax.servlet.\*;**

**After import the javax.servlet.\*;**

|  |
| --- |
| import javax.servlet.\*;  public class FirstServlet implements Servlet{  //to share the data more than one method we use the instance variables  //create the instance variables  ServletConfig config;  //create the default constructor  public FirstServlet(){  System.out.println ("FirstServlet object is created ..............");  }  //init() method acting as a setter() method  public void init(ServletConfig config){  this.config=config;  System.out.println ("first Servlet init () is executed ..........");  }  public void service(ServletRequest req, ServletResponse res){  System.out.println ("first Servlet service() is executed.......");  }  public void destroy(){  System.out.println ("first Servlet destroy () is executed......");  }  //getServletConfig () method acting as getter () method  public ServletConfig getServletConfig(){  return config;  }  public String getServletInfo(){  return "first Servlet example program......";  }} |

**ERRORS:**

D:\work>javac FirstServlet.java

FirstServlet.java:1: package javax.servlet does not exist

import javax.servlet.\*;

FirstServlet.java:2: cannot find symbol

Symbol: class Servlet

public class FirstServlet implements Servlet{

FirstServlet.java:5: cannot find symbol

Symbol: class ServletConfig

Location: class FirstServlet

ServletConfig config;

FirstServlet.java:11: cannot find symbol

Symbol: class ServletConfig

Location: class FirstServlet

public void init(ServletConfig config){

FirstServlet.java:15: cannot find symbol

Symbol: class ServletRequest

Location: class FirstServlet

public void service (ServletRequest req, ServletResponse res) {

FirstServlet.java:15: cannot find symbol

Symbol: class ServletResponse

Location: class FirstServlet

public void service(ServletRequest req, ServletResponse res){

FirstServlet.java:22: cannot find symbol

Symbol: class ServletConfig

Location: class FirstServlet

public ServletConfig getServletConfig(){

* To resolve the above errors we have to set the **class path** for jar file. The **Tomcat** jar file name is **servlet-api.jar;.;**

**AFTER SET THE CLASSPATH:**

|  |
| --- |
| import javax.servlet.\*;  public class FirstServlet implements Servlet{  //to share the data more than one method we use the instance variables  //create the instance variables  ServletConfig config;  //create the default constructor  public FirstServlet(){  System.out.println ("FirstServlet object is created ..............");  }  //init () method acting as a setter () method  public void init(ServletConfig config){  this.config=config;  System.out.println ("first Servlet init () is executed ..........");  }  public void service(ServletRequest req, ServletResponse res){  System.out.println ("first Servlet service () is executed.......");  }  public void destroy(){  System.out.println ("first Servlet destroy () is executed......");  }  //getServletConfig () method acting as getter () method  public ServletConfig getServletConfig(){  return config;  }  public String getServletInfo(){  return "first Servlet example program......";  }  } |

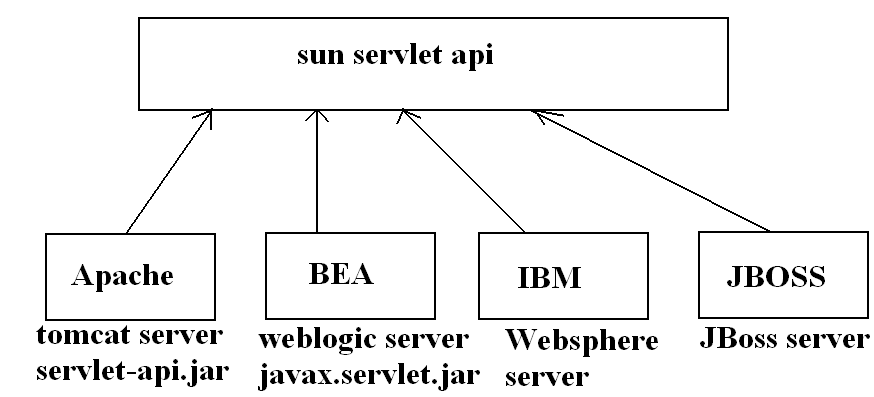
**OUTPUT OF THE COMMAND PROMPT:**

D:\work>javac FirstServlet.java

D:\work>

* After compile the above program copy the **.class** file into **classes directory.**

**Diagram:**

****

* From the above diagram we are understood that sun Microsystems has released Servlet API. The companies like Apache, Bea, IBM, Jboss and etc has provided the implementation to Servlet API.

**Example:**



* All the server vendors have provided the implementation of Servlet API. The implementation is called as software. Every server vendor creates jar file and deliver this with the server software.

**How do you compile your Servlet program?**

* To compile our Servlet program we have to set the class path to a jar file which provides the implementation of Servlet API.
* To find the name of the jar file which provides the implementation of Servlet API we have to read the documentation.
* In case of weblogic the name of the jar file which provides the implementation of Servlet API we can find this jar file in weblogic installation folder.

**C:\bea\modules**.

* In this folder we can find the Servlet implementation.
* In case of tomcat the names of the jar file which provides the implementation of Servlet-api.jar. We can find this jar file in tomcat installation folder is:

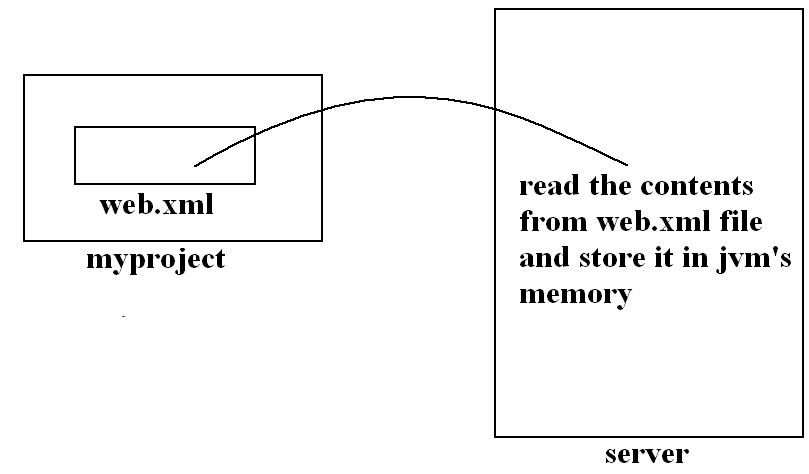
**C:\Program Files\Apache Software Foundation\Tomcat 6.0\lib.**

**What will happen when we deploy a web based application inside a server:**

* When we deploy a web based application as part of any server the server will perform the following steps.
* The server search for web.xml(deployment descriptor) file. If it is available server will validate whether web.xml file valid or not.
* If the web.xml file is valid the server reads the contents from web.xml file and stores the information in server’s jvm’s memory.
* To read the contents from web.xml file server internally uses the **Sax parser** (or) **Dom parser**. Most of the people use only Sax parser because of memory management.

**Note:** Sax parser and Dom parser is a programs.

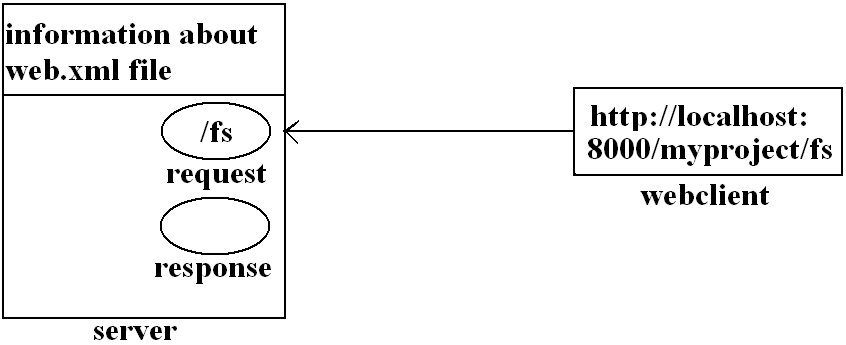
**Diagram:**

****

**What will happen when a client sends request to the server?**

* Whenever we send the request to the server from the client, the client sends the request in http request format.
* Whenever server receives the request the server will create two objects they are request and response.
* The server will read the contents from http request format and place it inside request object.
* The server will remove request object and response object when the response is committed (sends) to the client.

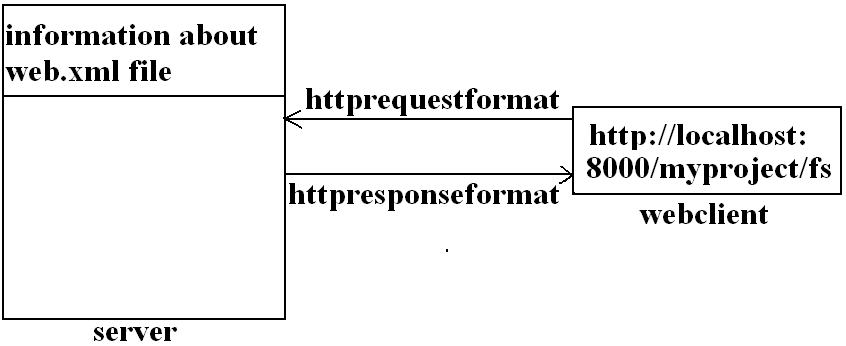
**Diagram:**



**When the server removes request and response object?**

* When the response is committed (sends) to the client the server will remove request object and response object.

**Diagram:**



**When the server creates request object and response object?**

* Whenever client sends request to the server the server will create request object and response object.
* When the client sends the following URL to the server it will perform some more steps.
* [**http://localhost:8000/myproject/fs**](http://localhost:8000/myproject/fs)
* When we send the above URL the server creates request object and response object. The server will place the requested resource name in request object.



* Now the server will open the request object and get the resource name /fs. The server checks is there any URL pattern is available in web.xml whose path matches to /fs. If it is available the server will find the corresponding Servlet name.
* Now the server starts search for is there any Servlet tag is available is Servlet name is **ABC** if the server finds the Servlet name then it finds Servlet class name.
* Now the server checks is there any Servlet object is available with that class name if it is not available the server will create the Servlet object.
* If it is available the server will not create the Servlet object.

**Summary:**

* Server **is responsible to create Servlet object. Server matains only one Servlet object.**
* **The Servlet object will be created by the server when we send the request for the 1st time.**
* Once if Servlet object is created the server will call the init() method.

**When the server removes the Servlet object?**

* When we undeploy the project the server removes the Servlet object.
* When we stop the server the server will remove the Servlet object.

**Note:** The server will call the destroy() method before it removes Servlet object.

**When the service() method will be called?**

* For every request we send from the client the server will execute the service() method.

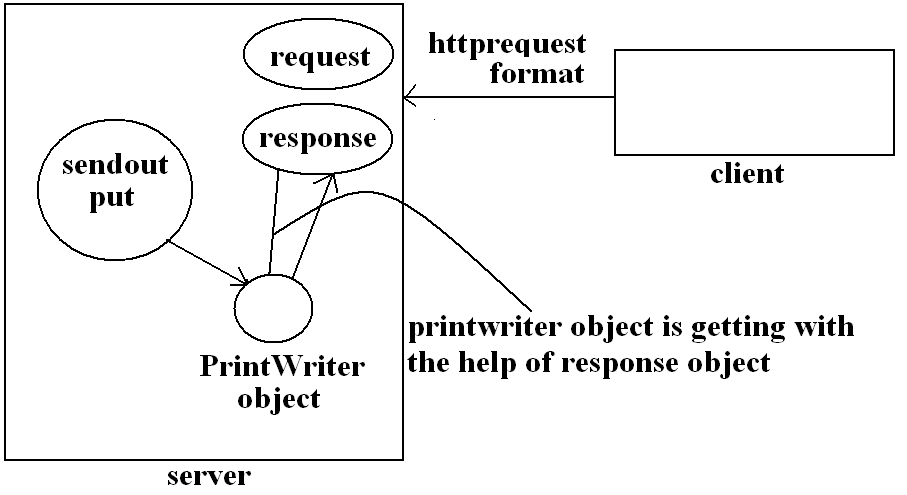
**When destroy() method is called by the server?**

* When we undeploy (or) when we stop the project from admin console the server will call destroy () method.

**Requirement:** Develop a Servlet program which sends **welcome message** to the client when ever client sends the request.

* A Servlet program cannot send an output directly to client. A Servlet places the data inside the response object. If we place the data in response object the data will be automatically send to client.
* If we want to place the data in response object we need to use PrintWriter object.
* PrintWriter object only can place the data inside response object.

**Diagram:**



**How do you get the PrintWriter object?**

* To get the PrintWriter object we take the help of response object. The following code shows how you get the PrintWriter object.

**PrintWriter out=response.getWriter();**

* As part of the PrintWriter object we have two methods. They are:

1. **Print();**
2. **Write();**

* These two methods can help us to write the contents into PrintWriter object. Once PrintWriter object get the data automatically PrintWriter object places the data inside response object. From response object the server sends the data to the client.

**Why PrintWriter object places the data inside response object?**

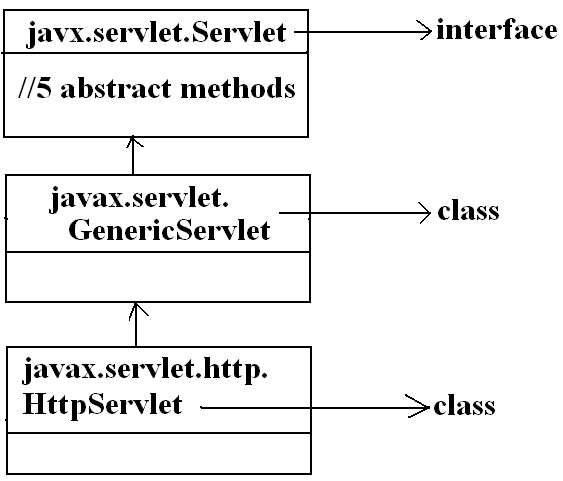
* PrintWriter object places the data inside response object because we got the PrintWriter object with the help of response object.

**Program:**

|  |
| --- |
| import javax.servlet.\*;  import java.io.\*;  public class FirstServlet implements Servlet {  //to share the data more than one method we use the instance variables  //create the instance variables  ServletConfig config;  //init () method acting as a setter() method  public void init(ServletConfig config){  this.config=config;  System.out.println ("first Servlet init() is executed ..........");  }  public void service(ServletRequest request, ServletResponse response)throws IOException{  PrintWriter out=response.getWriter();  out.println("welcome to servlet");  out.write("second line");  }  public void destroy(){  System.out.println ("first Servlet destroy() is executed......");  }  //getServletConfig () method acting as getter() method  public ServletConfig getServletConfig(){  return config;  }  public String getServletInfo(){  return "first Servlet example program......";  }} |

**Second way of developing Servlet program:**

**Requirement:** Develop a Servlet program using HttpServlet which sends the **“welcome** **message”** to the client whenever client sends the request.

****

**Program:**

|  |
| --- |
| import javax.servlet.http.\*;  import java.io.\*;  public class MyExample extends HttpServlet{  public void service(HttpServletRequest request,HttpServletResponse response)throws IOException{  PrintWriter out=response.getWriter();  out.write("o/p from httpservlet");  }} |

**Requirement**: Develop a Servlet program which sends **html page** as output to the client.

**Program:**

|  |
| --- |
| import javax.servlet.http.\*;  import java.io.\*;  public class FirstServlet3 extends HttpServlet{  public void service(HttpServletRequest request,HttpServletResponse response)throws IOException{  PrintWriter out=response.getWriter();  out.print("<html>");  out.print("<head>");  out.print("<title>");  out.print("<THIS IS MY WEBPAGE>");  out.print("</title>");  out.print("<body bgcolor=red>");  out.print("<h1><font size=4 face=arial fontcolor=green>welcome to bhaskarreddy<font></h1><br>");  out.print("</body>");  out.print("</head>");  out.print("</html>");  }} |

**Web.xml:**

|  |
| --- |
| <web-app>  <servlet>  <servlet-name>sp3</servlet-name>  <servlet-class>FirstServlet3</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>sp3</servlet-name>  <url-pattern>/fsp3</url-pattern>  </servlet-mapping>  </web-app> |

**URL PATTERN:**

[**http://localhost:7000/webapp/fsp3**](http://localhost:7000/webapp/fsp3)

**OUTPUT:**

Welcome to bhaskarreddy

* The above Servlets is sending a static content to the client every time when we send the request to the Servlet it will execute and send the same content.
* The above Servlet is called as a static Servlet because it will not change its output.
* By using Servlets we can develop 2 types of Servlets they are:

1. **Servlets which generates static contents**
2. **Servlets which generates dynamic contents.**

* In the projects we should not develop the Servlets which sends the static content because it is not recommended to use static content Servlet.
* In the projects we have to use the Servlet to generate dynamic contents only.

**Requirement:** Develop a Servlet program which generates the **date and time** and send output to the client.

**Program:**

|  |
| --- |
| import javax.servlet.http.\*;  import java.io.\*;  import java.util.\*;  public class DTServlet extends HttpServlet{  public void service(HttpServletRequest request,HttpServletResponse response)throws IOException{  Date d=new Date();  PrintWriter out=response.getWriter();  out.print(d);  }} |

**WEB.XML:**

|  |
| --- |
| <web-app>  <servlet>  <servlet-name>dts</servlet-name>  <servlet-class>DTServlet</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>dts</servlet-name>  <url-pattern>/dts</url-pattern>  </servlet-mapping>  </web-app> |

**OUTPUT:**

Fri Dec 10 11:10:41 IST 2010

**Requirement:** Develop a Servlet program which gets data from **product table** and display to the user.

**Step1:**

**Program:**

|  |
| --- |
| //import the statements  import javax.servlet.http.\*;  import java.sql.\*;  import java.io.\*;  public class RetrieveServlet extends HttpServlet{  //define the service method to get the data from database  public void service(HttpServletRequest request,HttpServletResponse response){  try{  //create the PrintWriter object to display the data to the client.  //PrintWriter out=response.getWriter();  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","bhaskar");  Statement stmt=con.createStatement();  ResultSet rs=stmt.executeQuery("select \*from product");  //create the PrintWriter object to display the data to the client.  //PrintWriter object is also created at the top of the driver manager class.  PrintWriter out=response.getWriter();  while(rs.next())  {  //display the data to the client with the help of printwriter object  out.print(rs.getString(1)+"\t");  out.print(rs.getString(2)+"\t");  out.println(rs.getString(3));  }}  catch(Exception e){  }}} |

**Web.xml file:**

|  |
| --- |
| <web-app>  <servlet>  <servlet-name>RetrieveServlet</servlet-name>  <servlet-class>RetrieveServlet</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>RetrieveServlet</servlet-name>  <url-pattern>/rs</url-pattern>  </servlet-mapping>  </web-app> |

**Step2:** To compile the above Servlet program we need to set the class path to **servlet-api.jar and** **ojdbc14.jar.**

**Step3:** To run the above Servlet program we have to copy **ojdbc14.jar** inside the lib directory of our project.

**Note:** We can copy ojdbc14.jar inside the server lib directory also.

**Note:** If multiple projects want to use a jar file we place the jar file inside server lib folder. (Or)

If only one project want to use a jar file we place it in project lib folder.

**Step4:** Deploy the project inside the server and perform unit testing

**Output of the RetrieveServlet:**

1 ptwo 2000

3 pthree 3000

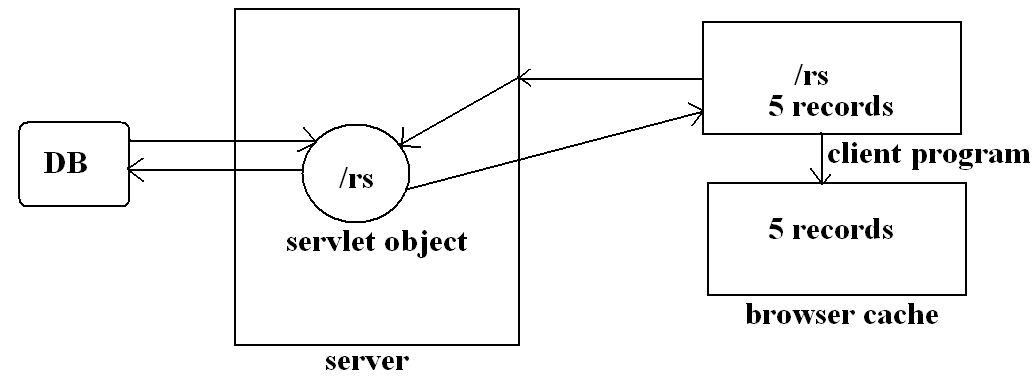
4 pfour 4000

6 psix 6000

90 pninety 9000

**Problem:** With the default setting of **Internet Explorer** we have facing problem with the above database RetrieveServlet program. That is when **we insert a new record into a database (or)** **delete a record from the database (or) update the record** and send request to the server we are not getting the updated records but if press **refresh** (or) **f5** it is working fine.

**Diagram:**



* When people develop the browser program to improve the performance they use a memory called as **cache** **memory** (this memory is part of browsers memory).
* Whenever we send request to the server the server send output to client program (or) browser. The browser program stores the output in its cache memory. The output is stored in the form of files to that file it will assign the URL.
* When the client send request for second time the browser checks whether the data is available in the browser’s cache or not. If it is not available the browser send request to server. If the data is available in cache the browser will not send the request. The browser sends the data which is stored in cache to the client.
* When we press the **refresh button (or) FS** the browser will not check for data in the cache memory directly it will send request to the server.

**Problem statement:**

* We have developed a servlet program which retrieves the data from database and send it to the client. After we implements this requirements if we modify the data in database (or) delete the data from database the updated data is not being getting reflected in the client program.

**Reason:**

* This is because of browsers cache memory.
* The browser stores the files in the cache memory if the file is got expired then only the client(browser) send request to server otherwise the client(browser) will not send the request.
* By default the expiry date is none.
* To resolve this problem we need to set the expiry date to the content which is storing inside browsers cache.
* We can resolve the above problem by setting the expires date header.

**Requirement:** Develop a servlet program which can read the contents from product table in database and send output to client.

**Program:**

|  |
| --- |
| import javax.servlet.http.\*;  import java.io.\*;  import java.sql.\*;  import java.util.\*;  public class RetrieveServlet1 extends HttpServlet{  public void service(HttpServletRequest request,HttpServletResponse response){  try{  java.util.Date d=new java.util.Date();  long time=d.getTime();  time=time-(60\*60\*1000);  response.setDateHeader("Expires",time);  PrintWriter out=response.getWriter();  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","lms","abc");  Statement stmt=con.createStatement();  ResultSet rs=stmt.executeQuery("select \*from product");  while(rs.next())  {  out.print(rs.getString(1));  out.print(rs.getString(2));  out.println(rs.getString(3));  }}  catch(Exception e){  }}} |

**Output:**

1 camera 5000

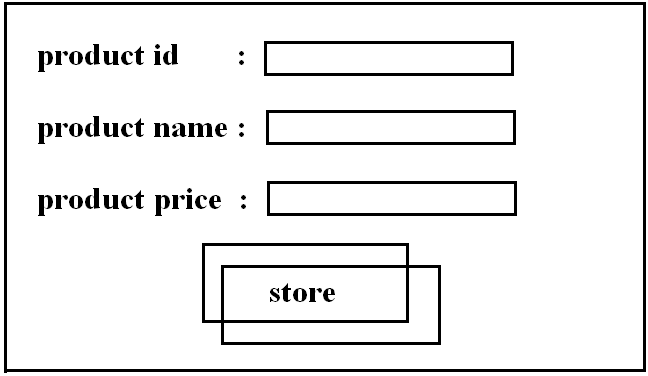
2 television 500000

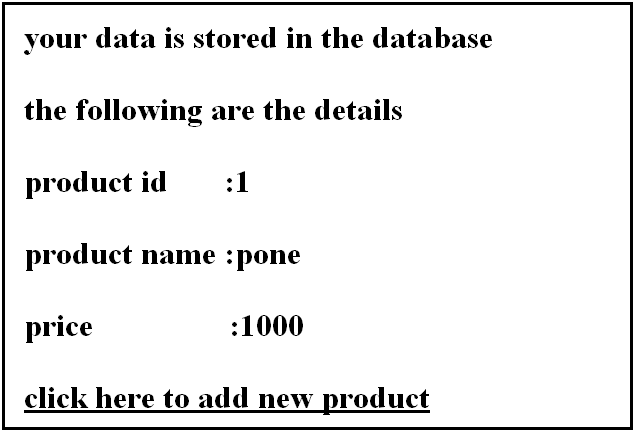
3 mobile 50000

**Requirement:** We need to develop an application which can capture the data from the user and store it in database. From the user we need to capture productid, product name and price.

* The following is the prototype of our program to store the data inside database.

**Prototype:**

****

****

* We can implements the above requirement in two ways.

**Approach1:** We need to develop two different Servlets they are:

1. **DisplayFormServlet.java**

* This servlet is responsible to generate html form.

1. **ProductStoreServlet.java**

* This servlet is responsible to store the data inside database.

**Note:** If we use Servlets, Servlets uses memory unnecessary in the server to create the objects that’s why static servlet is not recommended.

**Disadvantage:** The disadvantage of this approach is we are developing servlet which generates static content.

**Approach2:** In thisapproachwe are generating 2 programs. They are:

1. **Html file**

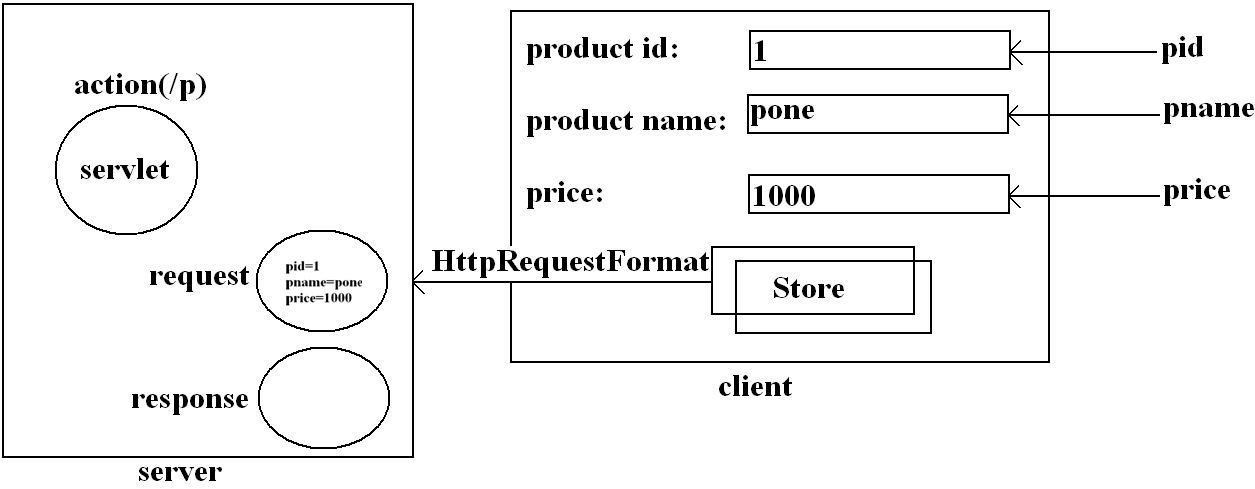
* This program is responsible to display form.

**Note:** For every text field we need to assign one variable name.

1. **ProductStoreServlet.java**

* This servlet is responsible to store the data inside database.
* To read the contents from html form we use a method **request.getParameter(fieldname)** this method returns a string value.

**Diagram:**

****

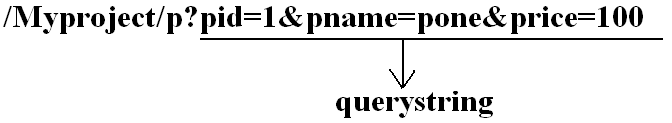
**What will happen when the user fills the form and click on store button?**

**Step1:** When the user clicks on store button the browser is responsible to capture the data from all the fields of html file.

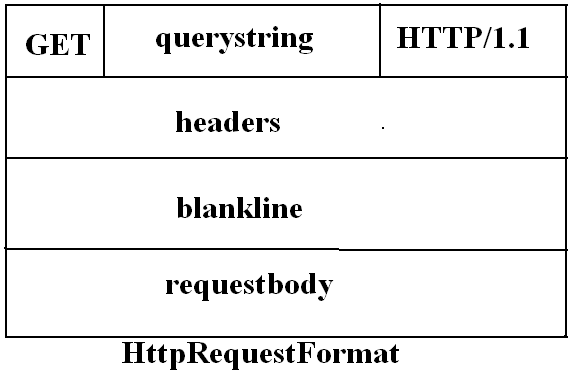
**Step2:** The browser places the data inside the field names of html file.

**Step3:** The browser checks the value of action attribute and gets its value.

**Step4:** The browser is responsible to append the data to URL in the form of query string. The following is the query string.

****

**Step5:** The browser is responsible to create http request format and send it to server.

****

**Step6:** The browser sends http request format to server.

**Step7:** The server will create request object and response object and the server reads the contents from httprequestformat and places the data in request object.

**Step8:** As a java programmer we write the code in servlet to get the data from request object. In Servlets to get the data from request object we use a method request.getParameter().

**Step8:** The **request.getParameter** takes the field name as input. If the data is available in request object with the given field it returns the data in the form of string. If the data is not available in request object it returns **null value** to our string variable

**Html program:**

|  |
| --- |
| <html>  <head>  <title>productdetails</title>  </head>  <body bgcolor="green" text="blue"><center>  <form action="/webapp/pss">  <h1>Enter the product details</h1>  <h3>product id :<input type="text" name="pid"><br>  product name :<input type="text" name="pname"><br>  product price :<input type="text" name="price"><br></h3>  <input type="submit" value="store">  </form>  </center>  </body>  </html> |

**ProductStoreServlet.java program:**

|  |
| --- |
| import javax.servlet.http.\*;  import java.io.\*;  import java.sql.\*;  public class ProductStoreServlet extends HttpServlet{  public void service(HttpServletRequest request,HttpServletResponse response)throws IOException{  String productid=request.getParameter("pid");  String productname=request.getParameter("pname");  String price=request.getParameter("price");  try{  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","lms","abc");  PreparedStatement pstmt=con.prepareStatement("insert into product values(?,?,?)");  pstmt.setString(1,productid);  pstmt.setString(2,productname);  pstmt.setString(3,price);  pstmt.executeUpdate();  }  catch(SQLException e){  System.out.println("error occured in try block");  }  PrintWriter out=response.getWriter();  out.print("<body bgcolor='yellow' text='red'>");  out.print("<center>");  out.print("<h1>Entered data successfully stored in database<h1>");  out.print("product id is:"+productid);  out.print("<br>product name is:"+productname);  out.print("<br>product price is:"+price);  out.print("<br><a href='/webapp/product.html'>click here to add new product</a></center></body>");  }} |

**Web.Xml:**

|  |
| --- |
| <web-app>  <servlet>  <servlet-name>ProductStoreServlet</servlet-name>  <servlet-class>ProductStoreServlet</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>ProductStoreServlet</servlet-name>  <url-pattern>/pss</url-pattern>  </servlet-mapping>  </web-app> |

**Output:**

# Entered data successfully stored in database

# Product id is: 1 product name is: pone product price is: 1000 [click here to add new product](http://localhost:8888/webapp/product.html)

# How do you send error messages to the client?

# We can send the error message to the client by using a method sendError() this method is part of HttpServletResponse.

# We have to provide this method in the catch block only that is we have to throw (or) send an error message to the client when the exception is occurred.

# This method takes two arguments. They Are:

# Errorcode (statuscode)

# Errormessage (statusmessage)

# Example: response.sendError(599,”got the problem in the servlet”);

# Generally we will not write this code in the project on behalf of this we use errorpages which are available in jsp and struts.

# Example: The following is the servlet example of sendError() method example.

**Program:**

|  |
| --- |
| import javax.servlet.http.\*;  import java.sql.\*;  import java.io.\*;  public class SendErrorUsingServlet extends HttpServlet{  public void service(HttpServletRequest request,HttpServletResponse response)throws IOException{  try{  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","lms","123");  System.out.println("connection object is:"+con);  }  catch(Exception e)  {  response.sendError(555,"error is:"+e.getMessage());  }}} |

**Output:**

HTTP Status 555 - error is: ORA-01017: invalid username/password; logon denied

ServletContext:

ServletConfig:

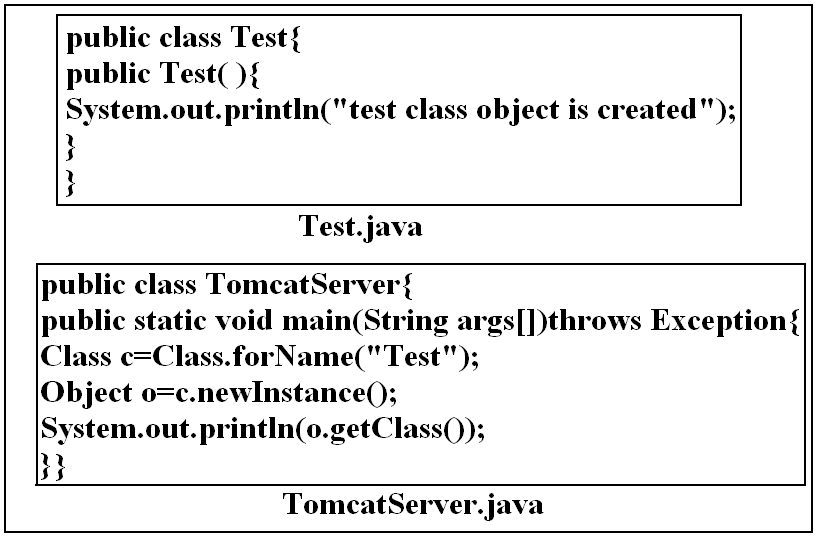
**Can we provide the constructors as part of Servlets?**

* We can provide the constructors as part of Servlet. But the Servlets will accept **only default constructors**. If we provide parameterized constructors the server will fail in creating the Servlet object.
* When we try to run the below program server will fail in creating the Servlet object.

|  |
| --- |
| public class DBservlet extends HttpServlet{  public DBservlet(int a){  System.out.println("DBservlet object is created");  }} |

* If I want to create an object to a class without using new operator we can use class.forName();
* We want to create an object to a class using class.forName() method compulsory the class must contain a default constructor.

**Example:**

****

**Output:**

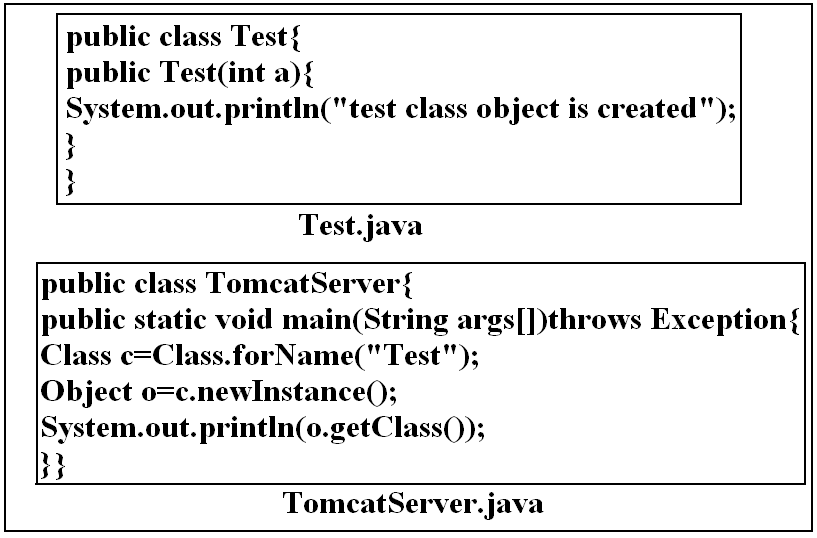
E:\servlet>java TomcatServer

Test class object is created

Class Test

* We cannot create an object to any class using class.forName() method if it doesn’t contains default constructor (or) if it contains only parameterized constructors.

**Example:**

****

**Output:**

**Note:** While running the TomcatServer.java it throws the following Exception. Because we cannot create an object to a parameterized constructor class.

E:\servlet>java Tomcat Server

Exception in thread "main" java.lang.InstantiationException: Test

at java.lang.Class.newInstance0(Unknown Source)

at java.lang.Class.newInstance(Unknown Source)

at TomcatServer.main(TomcatServer.java:4)

* As part of the Tomcat Server we can find the following code to create the Servlet object dynamically.

**Program:**

|  |
| --- |
| public static void main(String args[]){  Class c=Class.forName("FirstServlet");  Object o=c.newInstance();  Servlet s=(Servlet)o;  s.init(.......);  s.service(.......);  } |

**Content- Type:** Giving information about content is called content type.

* As part of the Servlet we use setContentType() method to specify what type of content is been sent by the server to the client.
* We write (or) provide setContentType() method in the beginning of the service() method of the Servlet.

**Example:**

|  |
| --- |
| import java.io.\*;  import javax.servlet.http.\*;  public class ContentTypeXml extends HttpServlet{  public void service(HttpServletRequest request,HttpServletResponse response)throws IOException{  response.setContentType("text/xml");  PrintWriter out=response.getWriter();  out.print("<student>");  out.print("<sno>1</sno>");  out.print("<name>studentname</name>");  out.print("</student>");  }  } |

**Output:**

<student>

<sno>1</sno>

<name>studentname</name>

</student>

**Example:**

|  |
| --- |
| import java.io.\*;  import javax.servlet.http.\*;  public class ContentType extends HttpServlet{  public void service(HttpServletRequest request,HttpServletResponse response)throws IOException{  response.setContentType("text/html");  PrintWriter out=response.getWriter();  out.print("<html>");  out.print("<head>");  out.print("<title>sample document</title></head>");  out.print("<h1>this is line one</h1>");  out.print("<h1>this is line two</h1>");  out.print("<h1>this is line three</h1>");  out.print("<h1>this is line four</h1>");  out.print("</body></html>");  }} |

**Output:**

This is line one

This is line two

This is line three

This is line four

* By default the server sends text/html as output to client. We can specify different values to the content type they are:
* Text/html
* Text/xml
* Application/pdf
* Application/msword
* Application/msexcel and etc.

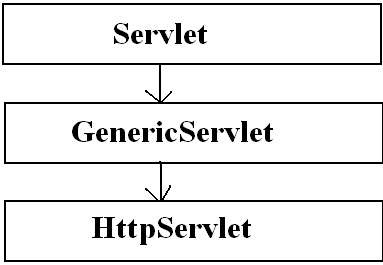
**How many ways to create Servlet object? (Or)**

**How many ways to develop the Servlets?**

* Most of the people say we have 3 options to develop a Servlet program. They are:

1. By using Servlet interface
2. By using GenericServlet
3. By using HttpServlet

**Diagram:**



* From the above diagram GenericServlet class inheriting the properties of Servlet interface and HttpServlet class inheriting the properties GenericServlet.

**Option1: By using Servlet interface**

**Program:**

|  |
| --- |
| import javax.servlet.\*;  public class FirstServlet1 implements Servlet {  ServletConfig config;  public void init(ServletConfig config){  this.config=config;  System.out.println ("first Servlet init() is executed ..........");  }  public void service(ServletRequest req, ServletResponse res){  System.out.println ("first Servlet service() is executed.......");  }  public void destroy(){  System.out.println ("first Servlet destroy() is executed......");  }  public ServletConfig getServletConfig(){  return config;  }  public String getServletInfo(){  return "first Servlet example program......";  }} |

**Output:**

First Servlet init() is executed ..........

First Servlet service() is executed.......

**Option2: by using GenericServlet**

* Sunmicrosystem has released a class GenericServlet this class provides the implementation of Servlet interface. GenericServlet is an abstract class this is because service() method is an abstract as part of GenericServlet. We can develop the Servlet by using generic Servlet.

**Program:**

|  |
| --- |
| import javax.servlet.\*;  public class SecondServlet extends GenericServlet{  public void service(ServletRequest request,ServletResponse response){  System.out.println("FirstServlet service() method executed");  }} |

**Output:**

FirstServlet service() method executed

**Option3: by using HttpServlet**

* We can develop a Servlet by using HttpServlet class. This class inherits the properties of GenericServlet. HttpServlet class is an abstract class. Sunmicrosystem has declared as abstract class because no one should be able creates the object directly.

**Program:**

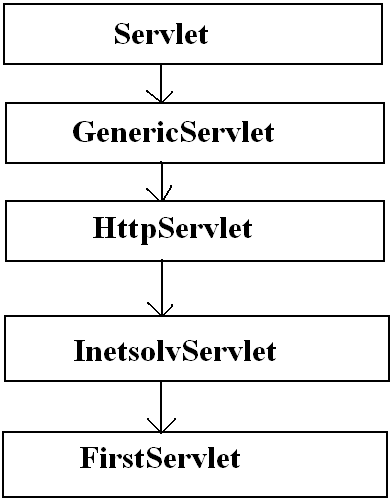
|  |
| --- |
| import javax.servlet.http.\*;  public class SecondServlet extends HttpServlet{  public void service(HttpServletRequest request,HttpServletResponse response){  System.out.println("second servlet service() method executed");  }} |

**Output:**

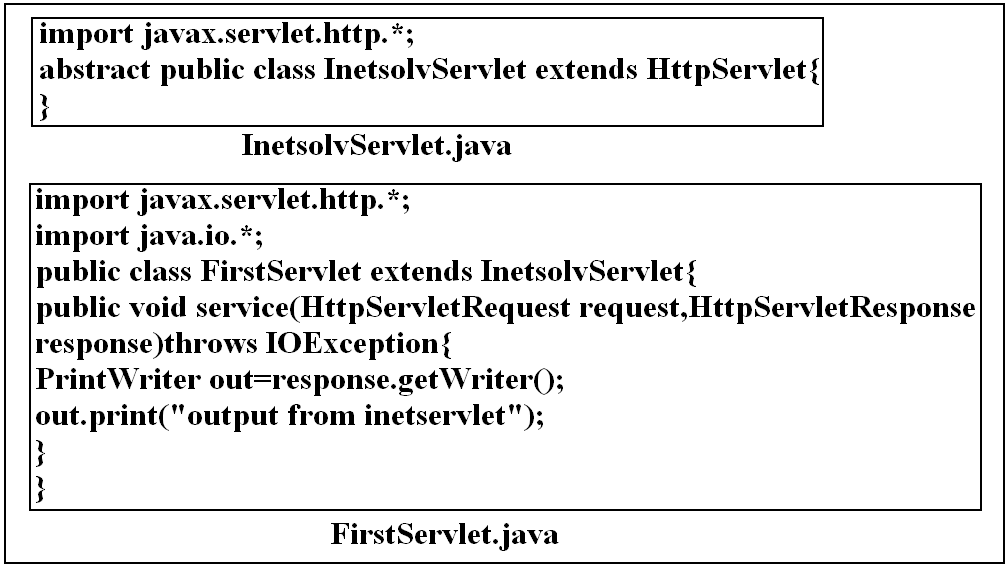
Second Servlet service() method executed

**Option4:**

**Diagram:**



**Program:**



* We can develop the Servlet program by using Servlet interface (or) GenericServlet (or) HttpServlet (or) InetsolvServlet (or) FirstServlet.
* Like this we can create the Servlet classes in so many ways.
* Most of the people say there is only one way available to create Servlet object. That is using a Servlet interface directly (or) indirectly.

**When server creates object to a Servlet?**

**The following are the scenarios when the server is creating the Servlet object:**

**Scenarios1:** When we deploy the project and when we send the request to the server to execute the Servlet for the 1st time the server is creating Servlet object.

**Scenarios2:** To the Servlet we can add a tag**<load-on-startup>**.

|  |
| --- |
| <servlet>  <servlet-name>FirstServlet<servlet-name>  <servlet-class>FirstServlet</servlet-class>  <load-on-startup>1</load-on-startup>  </servlet> |

* When we deploy a project which contains **<load-on-startup>** tag to that Servlet the server will create the Servlet object at the time of project deployment itself.
* The server will create the object to a Servlet which contains load-on-startup tag at the time of deployment of the project.

**Can we give –ve values to <load-on-startup> tag?**

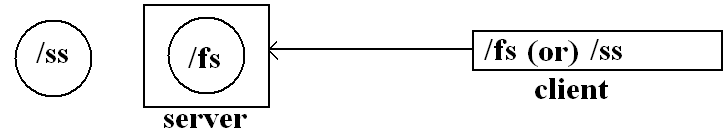
* No we cannot give the –ve values to <load-on-startup> tag. Because the <load-on-startup> tag takes only a +ve number. If we give the –ve number the server will not create the object.
* The number indicates the **order/priority** to the server when it has created the Servlet object.

**Scenarios3:** whenever we modified the Servlet class file and send the request to the server the server will check the loaded class modification time and the hard disk class file modification time if both are different the server will remove the old Servlet object and create the object to new Servlet.

**Note:** scenario3 doesn’t work in tomcat server. It works all the application servers(weblogic, jboss, websphere and etc).

**Scenarios4:** The server will create the object based on its memory management. If there is no free space is available in memory server finds which object is not in use from long time. The server removes that object from jvm’s memory and create object to new Servlet.

**Diagram:**



**Explain Servlet life cycle?**

* Every Servlet will be having a life cycle. It is creating the object and executing the init() method from that point onwards every time it executes service() method.
* Before the server removes the object it will call the destroy() method.
* Init(), service(), destroy() methods are called as Servlet life cycle methods.



* Server is responsible to remove the Servlet object. The server calls destroy() method before it removes the Servlet object.
* As part of the service() method we can call the destroy() method manually. Even if we call the destroy() method manually the server will not remove the Servlet object.

**Example:**

|  |
| --- |
| import javax.servlet.\*;  public class FirstServlet implements Servlet{  ServletConfig config;  //create the default constructor  public FirstServlet(){  System.out.println ("FirstServlet object is created ..............");  }  public void init(ServletConfig config){  this.config=config;  System.out.println ("first Servlet init () is executed ..........");  }  public void service(ServletRequest req, ServletResponse res){  System.out.println ("first Servlet service () is executed.......");  //here we can call the destroy() method manually  destroy();  }  public void destroy(){  System.out.println ("first Servlet destroy () is executed......");  }  public ServletConfig getServletConfig(){  return config;  }  public String getServletInfo(){  return "first Servlet example program......";  }} |

**Output:**

FirstServlet object is created..............

First Servlet init() is executed.............

First Servlet service() is executed.......

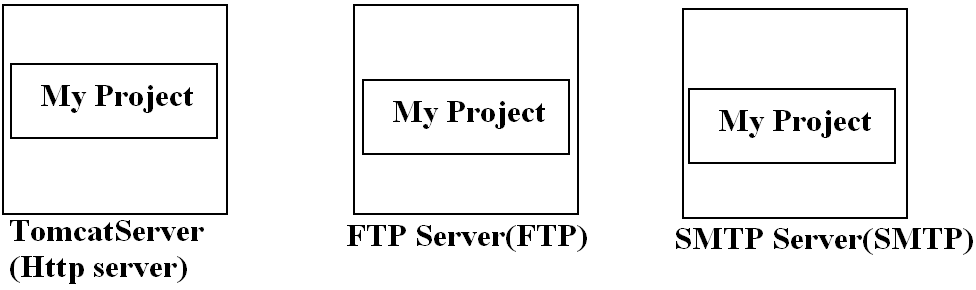
First Servlet destroy() is executed......

First Servlet service() is executed.......

First Servlet destroy() is executed......

* According to Sunmicrosystem specification once if we develop the Servlet program we can run the program on any server which uses any protocol.
* According to Sunmicrosystem we can run the Servlet application on HTTP server as well as FTP server as well as SMTP server.

**Diagram:**

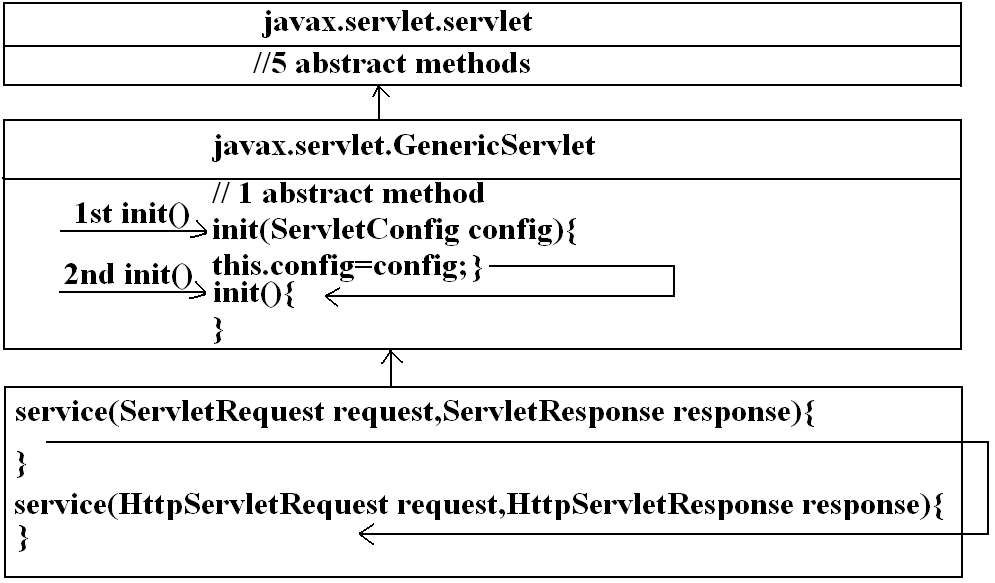


* Practically if we want to run the Servlet program the server must provide the implementation of Servlet API.
* Till today only http server people is providing the implementation of Servlet API. Till now either SMTP server (or) FTP server has not provide the implementation of Servlet API.

**What is the difference between javax.servlet and Javax.servlet.http?**

* All the classes and interfaces which are available in javax.servlet package can be used in all the servers(like HTTPservers, FTPservers, SMTPservers).
* All the classes and interfaces which are available in javax.servlet.http package can be used only in http servers.

**Diagram:**



* **Sunmicrosystem has given GenericServlet class. This class provides the implementation of Servlet interface.**
* As part of GenericServlet class Sunmicrosystem has provided the implementation to 4(four) methods only. They are init(), destroy(), getServletConfig(), getServletInfo().
* GenericServlet class has not provided the implementation of service() method.
* Sunmicrosystem has provided 2(two) init() methods as part of GenericServlet. They are:
  + - 1. **First init()method(init(ServletConfig config))**
      2. **Second init(init()) method**

**Note:** The init() method which takes ServletConfig as parameter is called first init() method.

**Note:** The init() method which does not takes any parameter is called second init() method.

**The following is the code of GenericServlet:**

**GenericServlet program:**

|  |
| --- |
| import javax.servlet.\*;  public abstract class GenericServlet implements Servlet{  // declare instance variable  ServletConfig config;  public void init(ServletConfig config){  //code to initilize config  this.config=config;  //calling the 2nd init() method from 1st init() method  init();  }  public void init(){  }  public void destroy(){  }  public ServletConfig getServletConfig(){  return config;  }  public String getServletInfo(){  return "GenericServlet";  }  abstract public void service(ServletRequest request,ServletResponse response);  } |

Sunmicrosystem has given HttpServlet class. As part of this class they have provided two service methods. They are:

1. **First service() method(service(ServletRequest request, ServletResponse response))**
2. **Second service() method(service(HttpServletRequest request, HttpServletResponse response))**

**Note:** The service() method which takes ServletRequest and ServletResponse as parameters is called 1st service() method.

**Note:** The service() method which takes HttpServletRequest and HttpServletResponse as parameters is called 2nd service() method.

**The following is the code of HttpServlet class:**

|  |
| --- |
| import javax.servlet.http.\*;  import javax.servlet.\*;  public abstract class HttpServlet extends GenericServlet{  public void service(ServletRequest request, ServletResponse response){  //code to typecast ServletRequest, ServletResponse objects into HttpServletRequest and HttpServletResponse HttpServletRequest req=(HttpServletRequest)request;  HttpServletResponse res=(HttpServletResponse)response;  //calling the 2nd service() method from 1st service() method  service(req,res);  }  public void service(HttpServletRequest request, HttpServletResponse response){  //some code is there we will see the code later  }  } |

**The following steps will happen when we send request to the server to execute the servlet object**:

**Step1:** Server is responsible to create servlet object.

**Step2:** Server calls 1st init() method on the above created servlet object.

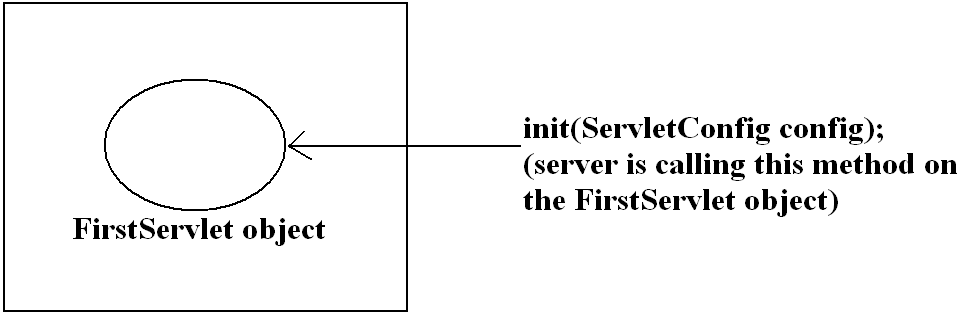
**Step3:** Server calls 1st service() method on the above created servlet object.

* **As part of our Servlets its always recommended to use second init() method.**

|  |
| --- |
| public class FirstServlet extends HttpServlet{  public void init(){  System.out.println("FirstServlet 2nd init() method");  }  public void service(HttpServletRequest request,HttpServletResponse response)throws IOException{  PrintWriter out=response.getWriter();  out.print("o/p from FirstServlet 2nd service() method");  }} |

* The above FirstServlet program contains the 2nd init() method. When the user sends the request to the above FirstServlet program first the server will create the FirstServlet object.
* When the Servlet object is created the server will call the 1st init() method on the FirstServlet object.

**Diagram:**



**server**

* The JVM checks for 1st init() method on 1st Servlet object. If it is available the JVM executes the 1st init() method. If it is not available the JVM checks for 1st init() method in the 1st servlet super class. The FirstServlet super class is HttpServlet.
* Now the JVM check for 1st init() method on HttpServlet. As the init() methods are not available in HttpServlet the JVM checks its super class of HttpServlet that is generic servlet.
* Now the JVM search for 1st init() method in generic servlet class. Now the JVM will execute the 1st init() method of GenericServlet.
* The GenericServlet 1st init() method calls 2nd init() method.
* Now the JVM checks for 2nd init() method on the current object that is FirstServlet.
* If the 2nd init() method is available in 1st Servlet the JVM will execute it otherwise the JVM checks for 2nd init() method on its super classes.
* Generally as part of init() method we provide the code which need to be executed only once and the code which need to be executed at the time of Servlet object is created.
* For example opening the connection from database (or) reading the contents from a file and storing its information server’s jvm’s memory.
* **It’s not recommended to use 1st init() method as part of our Servlet.**
* If we use the 1st init() method the following program is fail in executing.

**Example:**

|  |
| --- |
| import javax.servlet.http.\*;  import javax.servlet.\*;  import java.io.\*;  public class FirstServlet1 extends HttpServlet{  //The init() method which takes ServletConfig as parameter is called as 1st init() method.  //The following method is a 1st init() method.  public void init(ServletConfig config)throws ServletException{  System.out.println("First Servlet 1st init() method");  }  public void service(HttpServletRequest request,HttpServletResponse response)throws IOException{  ServletConfig config=getServletConfig();  System.out.println("config object value is :"+config);  String uname=config.getInitParameter("uname");  PrintWriter out=response.getWriter();  out.println("o/p from FirstServlet 2nd service() method");  }}  **Output:**  When we execute the above servlet it will give java.lang.NullPointerException |

**Reason:**

**Step1:** When we send the request to aboveservlet the server will create 1st servlet object and tries to call 1st init() method.

**Step2:** As the 1st init() method is available in our servlet it will execute the 1st init() method of 1st servlet.(GenericServlet 1st init() method will not executed).

**Step3:** When the server execute the service() method we get the config object using a method getServletConfig this method is part of GenericServlet.

**Step4:** The getServletConfig() method of GenericServlet returns the config object of GenericServlet. The config object is holding “null” value. So getServletConfig method returns null value.

**Example:**

|  |
| --- |
| import javax.servlet.http.\*;  import javax.servlet.\*;  import java.io.\*;  public class FirstServlet1 extends HttpServlet{  public void init(ServletConfig config)throws ServletException{  System.out.println("First Servlet 1st init() method");  }  public void service(HttpServletRequest request,HttpServletResponse response)throws IOException{  ServletConfig config=getServletConfig();  System.out.println("config object value is :"+config);  }}  **Output:**  First Servlet 1st init() method  config object value is :null |

* When we call any method on null object it will through the exception.
* If we want to resolve this problem compulsory we have to execute the GenericServlet 1st init() method.
* To execute GenericServlet 1st init() method in our 1st init() method we have to use the following code.

Super.init(config);

**Program:**

|  |
| --- |
| import javax.servlet.http.\*;  import javax.servlet.\*;  import java.io.\*;  public class FirstServlet1 extends HttpServlet{  public void init(ServletConfig config)throws ServletException{  super.init(config);  System.out.println("1st init() method");  }  public void service(HttpServletRequest request,HttpServletResponse response)throws IOException{  ServletConfig config=getServletConfig();  System.out.println("config object value is:"+config);  String var1=config.getInitParameter("var1");  System.out.println("var1 value is :"+var1);  PrintWriter out=response.getWriter();  out.println("o/p from FirstServlet 2nd service() method");  }}  **Output:**  O/p from FirstServlet 2nd service() method. |

* When we send the request to above servlet the server creates the object for 1st servlet class. After the object is created the server will call the 1st init() method.
* For every request sent to server it executes (or) call the 1st service() method on the 1st servlet object.
* Now the jvm checks for 1st service() method on 1st servlet object. As the 1st service() method is not available on the 1st servlet object it checks for 1st service() method on its super class that is HttpServlet class.
* The jvm executes the 1st service method of HttpServlet class. The 1st service() method of HttpServlet class calls the second service() method. Now the JVM checks for second service() method as part of 1st servlet class.
* As the 2nd service() method is available on the 1st servlet it will execute the second service() method of first servlet class.

**Example servlet of doGet() method and doPost() method:**

|  |
| --- |
| public class FirstServlet extends HttpServlet{  public void doGet(HttpServletRequest request,HttpServletResponse response)throws IOException{  PrintWriter out=response.getWriter();  out.print("welcome to servlets");  }  public void doPost(HttpServletRequest request,HttpServletResponse response)throws IOException{  PrintWriter out=response.getWriter();  out.print("welcome to servlets");  }}  **Output:**  welcome to Servlet |

* Generally most of the people try to create their own Servlet as part of projects that’s servlet will have their own methods.

**The following is an example of InetsolvServlet:**

|  |
| --- |
| package info.inetsolv;  import javax.servlet.http.\*;  public class InetsolvServlet extends HttpServlet{  public void init(){  myInit();  }  public void service(HttpServletRequest request,HttpServletResponse response){  myService(request,response);  }  public void myInit(){  }  public void myService(HttpServletRequest request,HttpServletResponse response){  }  } |

* The following is a program developed by a team member based on InetsolvServlet.

|  |
| --- |
| package info.inetsolv;  import javax.servlet.http.\*;  public class FirstServlet extends InetsolvServlet{  public void myInit(){  System.out.println("my init() is executed");  }  public void myService(HttpServletRequest request,HttpServletResponse response){  System.out.println("myservice() is executed");  }} |

**Step1:** First we compile the InetsolvServlet. To compile the InetsolvServlet we use the following command.

**D:\bhaskar>javac InetsolvServlet.java**

* After compile the above program it will generate info folder. The info folder containing the InetsolvServlet.class file.

**Step2:** in 2nd step we compile the FirstServlet.java. To compile the FirstServlet.java we use the following command.

**D:\bhaskar>javac -d . FirstServlet.java**

* After compile the above program it will generate info folder. The info folder containing the InetsolvServlet.class file and FirstServlet.class file.

**Note:** Because the two classes are having info.inetsolv package. That’s why the two “.classes” are placed inside info folder. The info folder is placed inside classes folder of our project.

* If the server contains cache then the cache is called as server’s cache.
* If the client contains cache then the cache is called as client cache.

**Why WEB-INF folder is a private folder?**

* The resources which are available inside WEB-INF folder can be access by only server. We cannot access the resources which are available inside the WEB-INF. We can access the resources which are available outside the WEB-INF folder only.

**What is the get() method and post() method?**

**Get() method:** When we use get() method the browser captures the data from html form and create query string. The following is an example of query string.

<http://localhost:8000/webapp/store.jsp?username=sone&pwd=mypassword&fname=sone+father&mname=sone+mother>.

* When we use the get() method as the data is being getting appended to the url if we are having any sensitive data it will be display to the user.
* The query string can accept maximum of 1024 characters.

**Post() method:** To send the sensitive data we use this method. By using a post method the data will be sending to the server by appending into httprequest format.

**Filters**

**Filter:** Filter is an object which provides the implementation of Filter interface directly (or) indirectly.

* We call the filters as “FrontControllDesign” patterns.
* As part of the web based applications we can develop multiple filters to perform the filtering task. They are:

1. Authentication Filters
2. Logging and Auditing Filters
3. Image conversion Filters
4. Data compression Filters
5. Encryption Filters
6. Tokenizing Filters
7. Filters that trigger resource access events
8. XSL/T filters
9. Mime-type chain Filter

* Filter interface contains three methods. They are:

1. Public void init([FilterConfig](file:///G:\softwares\java%20EE\javax\servlet\FilterConfig.html) filterConfig)
2. public void doFilter(ServletRequest request,ServletResponse response,FilterChain chain)
3. void destroy()

* The following is an example of filter class and its configuration.

**Program:**

|  |
| --- |
| import javax.servlet.\*;  public class FilterOne implements Filter{  FilterConfig fconfig;  public FilterOne(){  System.out.println("FilterOne object is created");  }  public void init(FilterConfig config){  this.fconfig=config;  System.out.println("FilterOne init() method is executed");  }  public void doFilter(ServletRequest response request,ServletResponse,FilterChain chain){  System.out.println("FilterOne doFilter() method is executed");  }  public void destroy(){  System.out.println("FilterOne destroy() method is executed");  }} |

* The following is the configuration of the filter.

**Web.xml:**

|  |
| --- |
| <web-app>  <filter>  <filter-name>FilterOne</filter-name>  <filter-class>FilterOne</filter-class>  </filter>  <filter-mapping>  <filter-name>FilterOne</filter-name>  <url-pattern>/fo</url-pattern>  </filter-mapping>  </web-app> |

**Output:**

FilterOne object is created

FilterOne init() method is executed

FilterOne doFilter() method is executed

FilterOne destroy() method is executed

**When the server creates filter object?**

* Whenever we deploy the project inside the server the server will create filter object. After filter project is created immediately the server will call init() method of the filter.
* Sunmicrosystem recommends to use /\*(slash star) as URL pattern to the filters. This is because every request has to go through the filters.
* Note: The following is the filter life cycle.

**Step1:** When we deploy a webbased application which contains the filters the server will create the filter object immediately.

**Step2:** After filter object is created server is responsible to call the init() method.

**Step3:** For every request the server will execute the doFilter() method.

**Step4:** The server will remove the filter object when we stop the webbased application.

**What is a filter chaining?**

* If multiple filters are involved in executing a single request is called as filter chaining.
* To achieve filter chaining we have to call a method chain.doFilter(). This method is part of FilterChain interface.
* Whenever we use Filter in every doFilter() method of Filter class we must call chain.doFilter() method.

**What will happen when chain.doFilter is executed?**

* When the server execute chain.doFilter() method it checks are there any filters are configure or not. If the filters are configure it will execute the remaining filters. If it has not found any filter it will forward the request to the appropriate resource.

**Which filter executed first if multiple filters are configuring in the project?**

* It’s all based on the configuration file.
* The following is an example of Logging Filter.

|  |
| --- |
| import javax.servlet.\*;  import javax.servlet.http.\*;  import java.io.\*;  public class LoggingFilter implements Filter{  FilterConfig config;  public void init(FilterConfig config){  this.config=config;  }  public void doFilter(ServletRequest request,ServletResponse response,FilterChain chain)throws IOException,ServletException{  HttpServletRequest req=(HttpServletRequest)request;  ServletContext application=config.getServletContext();  application.log(req.getRequestURI());  chain.doFilter(request,response);  }  public void destroy(){  System.out.println("LoggingFilter destroy() method is executed");  }} |

* The request will be stored in log folder of the server. The request is available Inside local host text document of log folder.
* In the filter every request go throw the filter object because of this reason filters are called as “**Front Controller Design Pattern**”.

**JSP**

**Jsp:** java server pages.

* As an alternative technology to a Servlet, sunMicrosystem has released jsp technology.
* Jsp’s are used to develop service side (or) web based applications(like Servlets).
* Servlets are used to develop the web based applications. If we want to develop a Servlet program to display welcome message, we have to carry out lot of steps and it has taken more amount of time. To implement the same requirement using jsp’s it has taken very less amount of time.
* Jsp’s takes less amount of development time when compared with Servlets.

**Advantages of jsp:**

1. Jsp’s have better productivity than Servlets. I.e. we can develop the jsp’s very quickly than Servlets. (Or)
2. If we use jsp’s in the project it will improve the productivity of developer.

**Note:** Productivity means efficiency of finishing the task. (Or)

1. If we develop the applications based on jsp’s the developer productivity will be improved(delivers the project quickly).
2. If we use jsp’s we no need to write (or)provide the java code in jsp’s.[if we design the project carefully].(or)
3. If we design the project carefully we can remove the java code from jsp’s (By using jsp action tags and custom tags).
4. By using jsp’s we can separate out business logic and presentation logic.

* Sunmicrosystem has released jsp technology. This technology inherits so many properties from ASP(Active server pages).
* Microsoft has released ASP technology to simplify the development of web based applications. To develop the web based applications using ASP we need to learn scripting language rather than programming language.
* Most of the average developers attracted towards ASP this is because learning a scripting language is easier than learning a programming language.
* Sunmicrosystem also released jsp’s by calming that we can develop jsp’s without learning the java code.

**Note:** If we carefully design the project we can eliminate java code completely from jsp’s.

**Note:** We use the jsp action tags (or) custom tags (or) JSTL to remove java code from jsp’s.

* We have developed a manual servlet to display a welcome message to the user. We are able to finish the code in 8 lines. The same program we have implemented using jsp’s.
* In jsp we have provided only one line this is converted into a servlet which contains 65 lines. When we execute the both programs we can say manually written servlet program takes less amount of time, when compared with automatically generated servlet.
* When we compare jsp and servlet with respect to performance Servlets will give a very good performance with respect to automatically generated servlet (or) jsp.
* Our project lead will design perform the analysis of using the appropriate technology and its development cost.
* The project lead gets the approval from customer and start working on the project based on the customer recommendations.
* Because of the project cost most of the people prefers using jsp’s rather than Servlets.
* Even though Servlets gives a better performance.
* As part of the testing we carried out the following types of testing.

1. **Unit testing**
2. **Integration testing**
3. **System testing**
4. **Smoke testing**
5. **UAT(user acceptance testing)**

**Unit testing:** A developer will carry out unit testing. Whatever code developed by that developer he will perform the unit testing on his code.

**Integration testing:** As part of integration testing we combined all the modules and then carried out the testing. Developers are the responsible to perform integration testing.

**System testing:** The tester will perform the system testing(project testing).

**Smoke testing:** We perform the smoke testing when we Deployee the project on the customer’s server. If we develop the jsp’s, all the jsp’s will be converted into Servlets when we carry out the smoke testing i.e. we are accessing the jsp for the 1st time.

**Procedure to develop a simple jsp program:**

**Step 1:** Every jsp program ends with an extension “.jsp”.

**Step 2:** The jsp programs will be placed inside the project folder and outside to WEB-INF folder.

* As part of any J2EE server we will be having servlet container and jsp container.
* Servlet container is responsible to run the servlet applications.
* Jsp container is responsible to run the jsp programs.
* As part of every jsp container we will be having jsp compiler(JSPC tool).
* JSPC compiler is responsible to run our jsp program. (JSP compiler is responsible to convert our jsp program into .java file.)
* When we send the request for the 1st time to the server, server will generate servlet.
* As part of jsp’s it is the responsibility of server to invoke (or) call the jsp compiler program. Jsp compiler program accepts “.jsp file” as input.

**Procedure to deployee the project in the weblogic server:**

**Step 1:** After we deployee the project in auto deployee directory the server will take care of deploying the project.

**Step 2:** When we send the request to weblogic server to access the jsp the weblogic server is invoke the jsp compiler.

**Step 3:** The weblogic server jsp compiler converts jsp program into servlet program.

**Step 4:** The java compiler converts the servlet program into “.class file”.

**Step 5:** The weblogic server deletes the generated servlet program(.java file).

**Note:** We can find the generated class file in the following location of weblogic server.

**C:\bea\user\_projects\domains\bhaskar\servers\AdminServer\tmp\\_WL\_user**

**Step 6:** Generally when we deployee the project and send request to the server the server will invoke the jsp compiler.

* Every server vendor will provide a feature of invoking (or) calling the jsp compiler manually also.

**Procedure to invoke the jsp compiler manually in weblogic server:**

**Step 1:** Create the folder and develop a jsp inside the folder.

**Step 2:** Enter into the bin directory of our domain(student/bin).

**Step 3:** Execute **setDomainEnv.cmd**

|  |
| --- |
| C:\>cd bea  C:\bea>cd user\_projects  C:\bea\user\_projects>cd domains  C:\bea\user\_projects\domains>cd bhaskar  C:\bea\user\_projects\domains\bhaskar>cd bin  C:\bea\user\_projects\domains\bhaskar\bin>setDomainEnv.cmd  C:\bea\user\_projects\domains\bhaskar> |

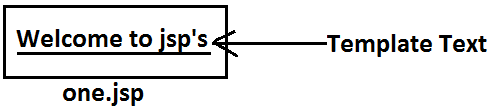
**Step 4:** The above command sets the path and classpath environment variables.

**Step 5:** Move to the folder where our jsp file is available and execute the following command.

**E:\vbr>java weblogic.jspc -keepgenerated one.jsp**

* The weblogic server follows different naming conventions when they develop servlet program.

**What will happen if we create one simple jsp and send request to the server for the 1st time?**

****

**Step 1:** First time when a client send request to one.jsp to the server. The server will invoke jsp compiler program.

**Note:** Every server vendor is responsible to develop the jsp compiler on their own. We can find the jsp compiler as part of Tomcat as well as Weblogic as well as JBose.

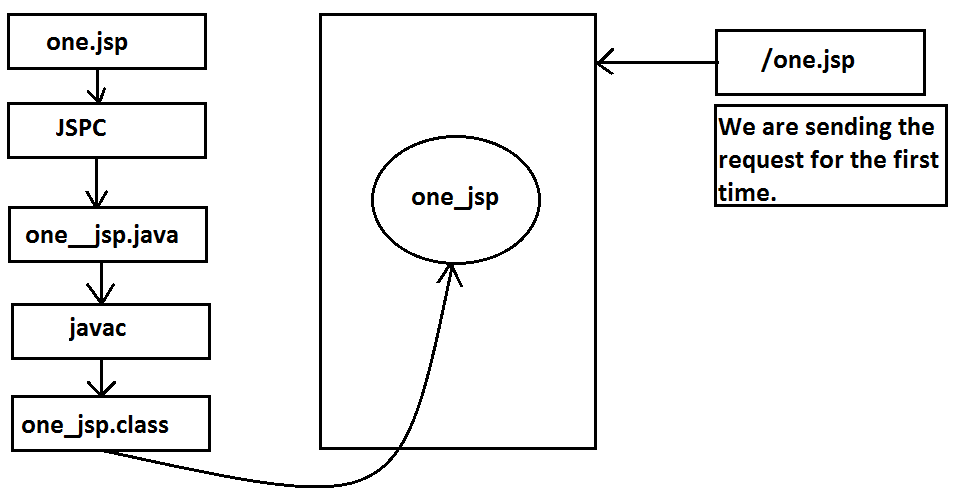
**Note:** Jsp compiler is a java program it is developed in java language.

**Step 2:** The jsp compiler experts “.jsp file” as input. Jsp compiler is responsible to translate .jsp program into Servlet program.

**Step 3:** After the Servlet is generated the java compiler is responsible to compile the servlet program.

**Step 4:** The server will create the servlet object for the generated class file.

**Diagram:**

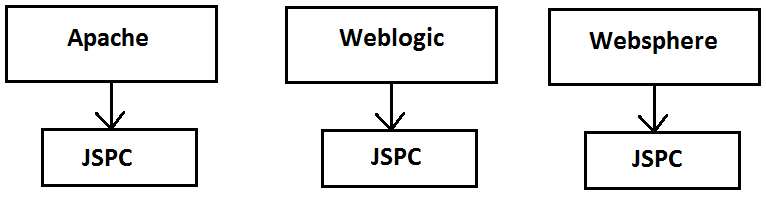


**Step 5:** All the above process will happen only once that is when we send request to the server for the 1st time.

**What will happen when we send request to the server for the second time?**

* The server will execute the existing servlet object.
* According to sunmicrosystem specifications every server vendor is responsible to generate (or) develop jsp compiler program as well as every server vendor is responsible to develop one servlet program. Apache guys has used (or) developed HttpJspBase servlet. Weblogic guys developed JspBase servlet.

**Diagram:**



* Every jsp compiler generates the servlet code by following the rules.
* The following is an example of the servlet generated by Apache group (or) Tomcat.

|  |
| --- |
| public class one\_jsp extends HttpJspBase{  //some standard code  public void \_jspService(HttpServletRequest request, HttpServletResponse response){  //standard code  }  } |

* We have created the following one.jsp file. The server has translated the jsp program into corresponding servlet program.

**Program:**

|  |
| --- |
| Welcome to jsp’s |

|  |
| --- |
| JSPC |

|  |
| --- |
| public class one\_jsp extends HttpJspBase{  public void \_jspService(HttpServletRequest request, HttpServletResponse response){  //standard code  }  } |

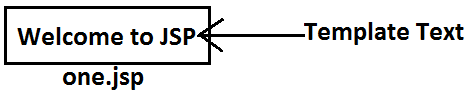
* We can provide the following elements as part of every jsp program.

1. **Template Text**
2. **Scriptlet**
3. **JSP Declarations**
4. **JSP Directives**
5. **JSP Expression**
6. **JSP Action Tags**
7. **JSP Custom tags**
8. **EL expression**

**Template Text:**

* **I**f we want to send any output to the client and display to the client we use template text element.

**Example:**

****

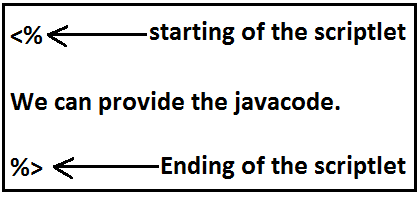
* Whenever the JSP compiler encounters template text it will convert the template text into corresponding java code. The template text will be converted into java code as shown below.

|  |
| --- |
| public class one\_jsp extends HttpJspBase{  public void \_jspService(HttpServletRequest request, HttpServletResponse response){  //standard code  out.write("welcome to jsp.");  } } |

**Scriptlet:**

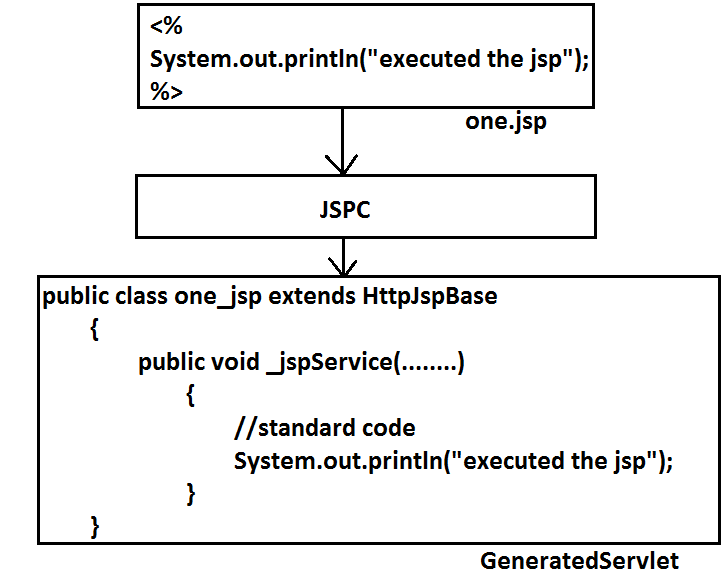
* Scriptlets are used to write the java code in jsp’s.

**Syntax:**

****

* We can use any no. Of Scriptlets as part of jsp.

**Example:**



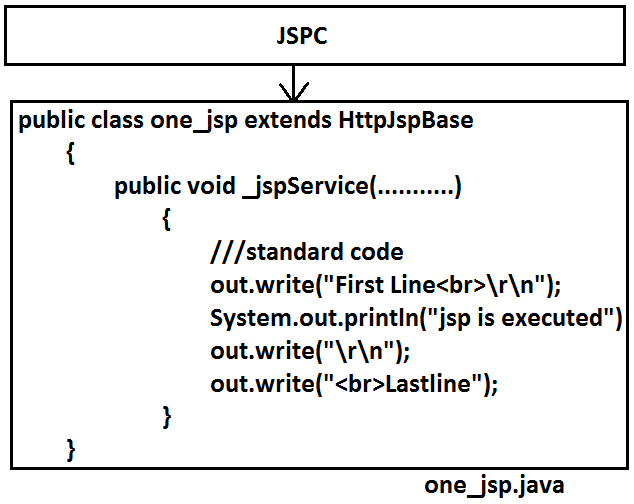
* When we try to execute the following jsp it will fail in running the jsp program. This is because of syntax problem available in generated Servlet.

**Example:**

|  |
| --- |
| FirstLine<br>  <%  System.out.println("jsp is executed")  %>  <br>LastLine |

**One.jsp**

* When the above JSP is converted into Servlet it has converted as shown below.



* The java compiler cannot convert “.java file” into “.class file” if it is having syntax errors.

|  |
| --- |
| <%  int a;  a=10;  %>  <%  int b;  b=20;  %>  <%  int c=a+b;  System.out.println(c);  %> |

* When the JSP compiler converts the above JSP into “.java file” it places all the Scriptlet code inside \_jspService() method. The following is the generated Servlet.

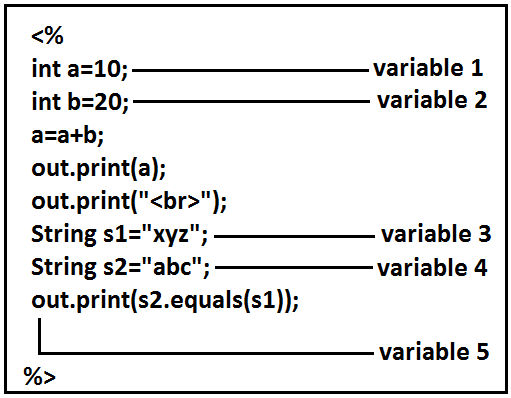
|  |
| --- |
| public void \_jspService(...........){  int a;  a=10;  int b;  b=20;  int c=a+b;  System.out.println(c);  } |

* If we declared any java variables inside the Scriptlets those variables become local variables to \_jspService() method.
* In JSP if Scriptlet is responsible to send the output to the client we use “out variable” directly as part of the Scriptlet as shown below.

|  |
| --- |
| <%  int a=10;  int b=20;  int c=a+b;  out.print(c);  %> |

**One.jsp**

**Example:**



* In the above JSP there are 5 variables are available. We have declared the 4 variables they are a, b, s1 and s2. We have not declared a available **out**.
* Without declaring the out variable we are able to use the out variable.
* Any variables if we are able to use without declarations those variables are called as implicit variables.

**Implicit variables:**

* The variables which can be used without declaring by JSP developer is called **as implicit variables.**
* The following are 9 implicit variables which can be used directly in the JSP without declaring.

1. **request**
2. **response**
3. **pageContext**
4. **session**
5. **application**
6. **config**
7. **out**
8. **page**
9. **exception**

**Note:** The implicit variables local variables to \_jspService() method. We can not use outside the \_jspService() method.

* The following jsp returns which client has sent the request to the server.

|  |
| --- |
| <%  String name=request.getHeader("User\_agent");  out.print(name);  %> |

* JSP’s takes little bit of more amount of time when we try to access the JSP for the 1st time.
* This is not at all a problem because of the following reasons.
* When we deliver a project to the customer we perform smoke test on the project which is deliver to the customer. When we perform the smoke test all the JSP’s will be converted into Servlets.
* We can precompile the jsp’s and then deploy the project. We can perform the pre compilation by using JSPC.
* We can configure the jsp’s in web.xml file similar to Servlets. The following are the tags to configure jsp into deployment descriptor.

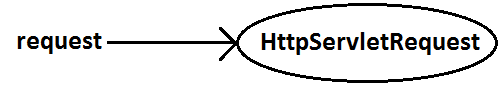
|  |
| --- |
| <web-app>  <servlet>  <servlet-name>a</servlet-name>  <jsp-file>/one.jsp</jsp-file>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>a</servlet-name>  <url-pattern>/a</url-pattern>  </servlet-mapping>  </web-app> |

* Generally in most of the projects because of security reasons people place the files inside WEB-INF folder. WEB-INF is a private folder. The client cannot access the resources which are available inside WEB-INF.
* If we want to access the resources which are available in WEB-INF we need to configure those resources in web.xml file.

**Request implicit variable:**

* We can use this request implicit variable directly in the Scriptlet. We no need to declare the request variable.
* Request implicit variable is holding HttpServletRequest object.

**Diagram:**

****

**Example 1:**

|  |
| --- |
| <%  String name=request.getHeader("user-agent");  out.print(name);//Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.2.24) Gecko/20111103 Firefox/3.6.24  %> |

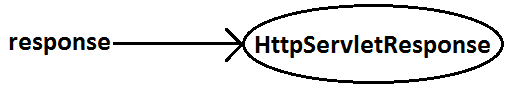
**Example 2:**

|  |
| --- |
| <%  String name=request.getHeader("accept-language");  out.print(name);//en-us,en;q=0.5  %> |

**Response implicit variable:**

* The response object also can be used directly in the jsp Scriptlets.
* Response implicit variable is holding HttpServletResponse object.

**Diagram:**



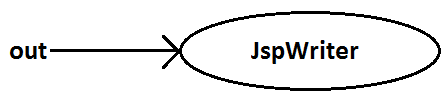
**Example:**

|  |
| --- |
| <%  response.sendError(555,"my own error");  %> |

**Out implicit variable:**

* Out is an implicit variable of a jsp which can be used directly in the Scriptlets.
* Out implicit variable is holding JspWriter object.

**Diagram:**

****

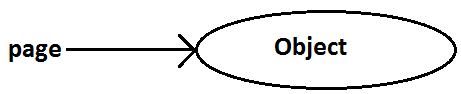
**Example:**

|  |
| --- |
| **<%**  **out.print("welcome to jsp's");**  **%>** |

**Page implicit variable:**

* page implicit variable holds currently executing Servlet object.

**Diagram:**

****

**Example:**

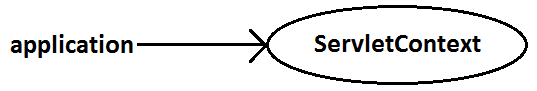
|  |
| --- |
| <%  out.print(page);//org.apache.jsp.one\_jsp@1bf7b23  %> |

**Note:** The implicit variables local variables to \_jspService() method.

**Application implicit variable:**

* This implicit variable holds an object of type ServletContext.

**Diagram:**

****

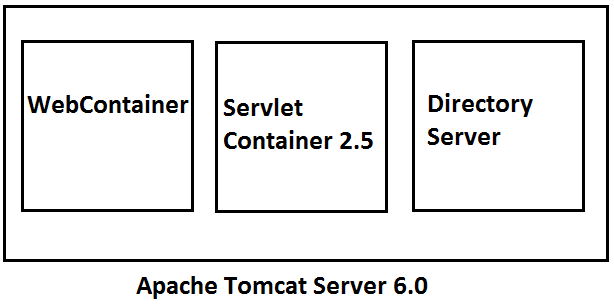
* Servlet program is going to use Servlet container to run our applications.
* “Servlet uses application object to communicate with Servlet container”. The methods which are part of application object will help us in finding the Servlet container information.

**Example:**

|  |
| --- |
| <%  out.print(application.getServerInfo());//Apache Tomcat/6.0.20  %> |

* getServerInfo() method tells us the name of the server and its version who is running the Servlet application.
* Once if sunmicrosystem releases Servlet specification tomcat guys (or) any other server vendors are responsible to provide the implementation of Servlet specification.

**Diagram:**

****

* As part of tomcat 6.0 the Servlet container version no is 2.5.

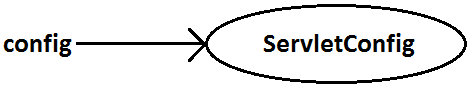
**Example:**

|  |
| --- |
| <%  out.print(application.getServerInfo()+"<br>");//Apache Tomcat/6.0.20  out.print(application.getMajorVersion()+"<br>");//2  out.print(application.getMinorVersion());//5  %> |

**Config implicit variable:**

* We can use this object directly in JSP. This object is used by Servlet container to pass the information to Servlet during Servlet initialization.

**Diagram:**

****

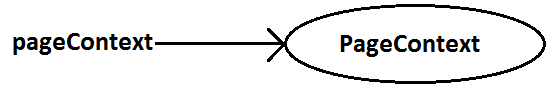
**Example:**

|  |
| --- |
| <%  out.print(config.getServletName());//jsp  %> |

**pageContext:**

* pageContext is an implicit variable. We get this variable whenever we have send request to the service() method.

**Diagram:**



* **pageContext is a local variable to \_jspService() method**. The scope of the pageContext variable is until the service() method is executed (or) until the request is passes.
* As part of pageContext object we have couple of methods to get all the other implicit variables. The methods are:

1. getRequest()
2. getResponse()
3. getServletConfig()
4. getServletContext()
5. getPage()
6. getSession()
7. getException()
8. getOut()

* PageContext class available in javax.servlet.jsp package.

**Exception:**

* Exception implicit variable can be used in the case of error pages only.

**Note:**

* If the JSP is not error page we can not use exception implicit variable.

**Example:**

|  |
| --- |
| <%@ page isErrorPage="true"%>  <%  out.print(exception);//null  %> |

**Jsp Declarations:**

public class <class\_name>{

//instance variables

//static variables

//instance methods

//static methods

}

* As a developer we can provide the instance variables, instance methods, static variables and static methods as part of a class.
* **Jsp is a Servlet. Servlet is a java program (or) class.**
* **To provide instance variables, instance methods, static variables and static methods as part of the generated servlet we use jsp declarations.**
* **We cannot declare a method inside a method, but we can declare a method inside a class.**

**Declare a method inside a class:**

|  |
| --- |
| public class Test{  public void methodone(){  System.out.println("method one");  }} |

**Note:** we can declare a method inside a class.

**Declare a method inside a method:**

|  |
| --- |
| public class Test{  public void methodone(){  public void methodtwo();  System.out.println("method one");  }} |

**Note:** We cannot declare a method inside a method this is because of java rules. If we declare a method inside a method the java compiler throw the following errors.

**Errors:**

E:\>javac Test.java

Test.java:3: **illegal start of expression**

public void methodtwo();

Test.java:3: **illegal start of expression**

public void methodtwo();

Test.java:3: **';' expected**

public void methodtwo();

* To resolve the above errors we cannot declare a method inside a method.

**Define a method inside a method:**

|  |
| --- |
| public class Test{  public void methodone(){  public void methodtwo(){}  System.out.println("method one");  } } |

**Note:** We cannot define a method inside a method this is because of java rules. If we define a method inside a method the java compiler throw the following errors.

**Errors:**

E:\>javac Test.java

Test.java:3: illegal start of expression

public void methodtwo(){}

Test.java:3: illegal start of expression

public void methodtwo(){}

Test.java:3: ';' expected

public void methodtwo(){}

* To resolve the above errors we cannot define a method inside a method.
* **By using a Scriptlet we cannot declare instance variables, instance methods this is because of java rules.**
* **The following program is throw an exception when we run the jsp.**

|  |
| --- |
| <%  public void methodOne()  {  System.out.println("we are in methodOne()");  }  %> |

**One.jsp**

* When we try to run above jsp program the jsp compiler translates **.jsp** **file** into **.java file**. While it is converting the jsp into java program it has placed all the Scriptlet code inside **\_jspService() method**. As part of a Scriptlet we have provided the code of method. The following is the generated **.java** **file** for the above **one.jsp**.

|  |
| --- |
| public class one\_jsp extends HttpJspBase{  public void \_jspService(......){  //Standard code  public void methodOne() {  System.out.println("we are in methodOne()");  } }} |

**one\_jsp.java**

* When the java compiler tries to compile the above **.java file** into **.class** it will fail because in java we cannot provide a method inside a method.

**Jsp Declarations:**

* To provide instance variables, instance methods, static variables and static methods as part of the generated servlet we use jsp declarations.

**Syntax of jsp declarations:**

|  |
| --- |
| <%!  //instance variables  //static variables  //Instance methods  //static methods  %> |

**Note:** In java inside the class we can provide only variables and methods. We should not provide the java code inside the class.

**Example:**

public class Test{

//provide the java code inside the class.

System.out.println("welcome to java");

public void methodone(){

System.out.println("welcome to methodone");

}

}

* If we provide the java code inside the class the java compiler throws the following errors.

**Errors:**

E:\>javac Test.java

Test.java:2: <identifier> expected

System.out.println("welcome to java");

Test.java:2: illegal start of type

System.out.println("welcome to java");

* To resolve the above errors we cannot provide the java code inside the class.

**Example:**

|  |
| --- |
| <%!  int a;  public void methodOne()  {  System.out.println("we are in jsp declarations");  }  %>  <%  methodOne();  %> |

|  |
| --- |
| **JSPC** |

|  |
| --- |
| public class one\_jsp extends HttpJspBase  {  int a;  public void methodOne()  {  System.out.println("we are in jsp declarations");  }  public void \_jspService(........)  {  //standard code  methodOne();  } |

* In the above jsp file we have use jsp declarations and we have declared one instance variable and one method.
* In jsp’s if we want to call the methods from \_jspService() method. We need to call the method directly. To call the method directly from \_jspService() method we have to use a Scriptlet.

**Example:**

|  |
| --- |
| <%!  public void methodOne(){  int a=10;  int b=20;  int c=a+b;  out.print(c);  }  %>  <%  methodOne();  %> |

* The above program will fail when we try to run the program this is because we cannot access the implicit variables in the jsp declarations.

**Note:** The local variables of one method cannot be accessible in another method.

**Example:**

|  |
| --- |
| public class Test{  public void methodone(){  int a;  a=10;  methodtwo();  }  public void methodtwo(){  System.out.println(a);  }  } |

**Note:** We cannot access the local variables from one method to another method. If we try to access the local variables from one method to another method the java compiler throw the following errors.

**Error:**

E:\>javac Test.java

Test.java:8: cannot find symbol

Symbol: variable a

Location: class Test

System.out.println(a);

* To resolve the above error we can’t try to access the local variables from one method to another method. **(Or)**
* If we want to access the local variables from one method to another method while calling the method we have to pass the values as method arguments. The following program is an example to access the local variables from one method to another method.

**Example:**

|  |
| --- |
| public class Test{  public void methodone(){  int a;  a=10;  methodtwo(a);  }  public void methodtwo(int a){  System.out.println(a);  }  } |

* If we want to access the implicit variables in the jsp declarations we have to pass the values as method arguments. The following program is an example program.

**Example:**

|  |
| --- |
| <%!  public void methodOne(JspWriter out)  {  int a=10;  int b=20;  int c=a+b;  try  {  out.print(c);  }catch(java.io.IOException e)  {}  }  %>  <%  methodOne(out);  %> |

**Explain jsp life cycle?**

* As part of jsp the following 3 methods are called as jsp life cycle methods. They are:

1. **jspInt()**
2. **\_jspService()**
3. **jspDestroy()**

**Example:**

|  |
| --- |
| <%!  public void jspInit()  {  System.out.println("jspInit() method called");  }  public void jspDestroy()  {  System.out.println("jspDestroy() method called");  }  %>  <%  out.println("Output from service() method"+"<br>");  out.println("updated the jsp");  %> |

**Output on the server:**

jspInit() method called

jspDestroy() method called

**Output on the client:**

Output from service() method  
updated the jsp

* Whenever the server has created the servlet object for the jsp file the server calls jspInit() method. For every request sent by the client to the server it will call \_jspService() method.
* Whenever there is a modification in jsp file server is responsible to regenerate servlet program and create the servlet object. When the new object is created the server will unload the old servlet object[when server is unloading the servlet object it will call the jspDestroy() method].

**Jsp Expressions:**

* **We can use expressions as part of jsp’s.**
* When we use the expressions the expressions will be evaluated and send output to the client.

**Example:**

|  |
| --- |
| <%  int a=10;  int b=20;  out.print(a);//Here a is a java expression  %> |

**One.jsp**

* When the above java expression is evaluated it will send the value 10 as output to the client.

|  |
| --- |
| <%  int a=10;  int b=20;  out.print(a+"<br>");//java expression  out.print(a+b+"<br>");//java expression  String s1="abc";  out.print(s1.equals("<br>"+"abc"));//java expression  java.util.ArrayList a1=new java.util.ArrayList();//java expression  a1.add("one");  out.print("<br>"+a1.toString());//java expression  %> |

**Note:** From the above one.jsp program display the value is a java expression, creating the object is also a java expression, compare the strings is also a java expressions and etc.

**Note:** **jsp expressions are alternative to java expressions.**

**Jsp expressions:**

* To simplify the java expressions we can use jsp expressions in jsp.

**Syntax of jsp expression:**

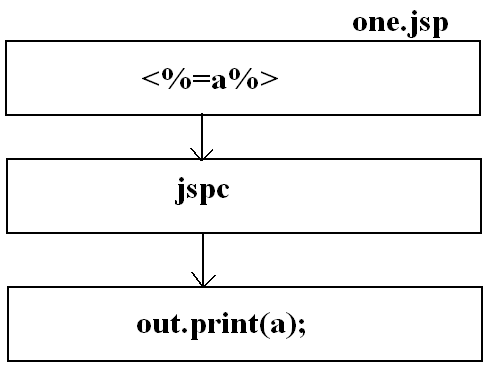
|  |
| --- |
| **<%= javaexpression %>** |

**Example of jsp expression:**

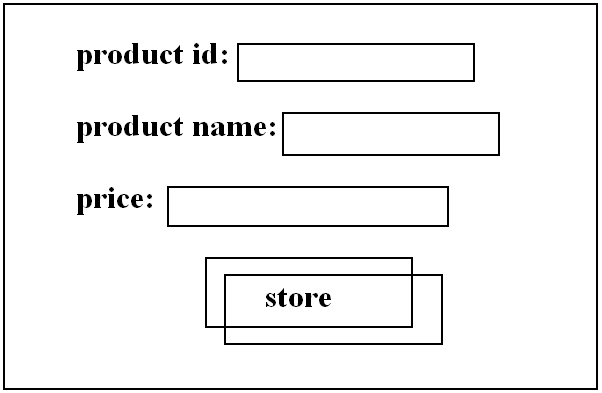
|  |
| --- |
| <%  int a=10;  int b=20;  %>  <%=a%><br>//10  <%=b%><br>//20  <%=request%>//org.apache.catalina.connector.RequestFacade@118278a |

**One.jsp**

* For the following one.jsp program server will generate the corresponding Servlet.



**Requirement:** Develop a jsp application which will be able to capture the data from html form and send output to the client. The following is the prototype of the application.



* To implement the above requirement we need to develop 2 programs. They are:

1. Html file(This file is responsible to display the form to the user)
2. Jsp file(This file is responsible to store the data in database and display the information to the user)

**Html file:**

|  |
| --- |
| <form action="store.jsp">  product id:<input type="text" name="pid"><br><br>  product name:<input type="text" name="pname"><br><br>  product price:<input type="text" name="price"><br><br>  <input type="submit" value="store">  </form> |

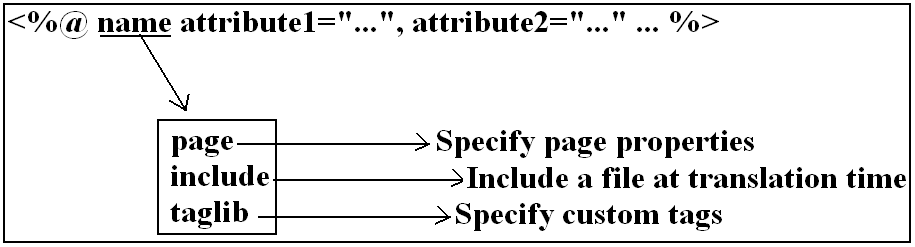
**Store.jsp:**

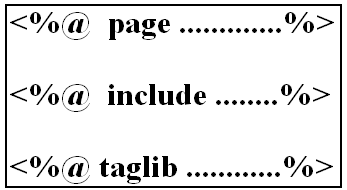
|  |
| --- |
| <%@ page import="java.sql.\*"%>  <%  String pid=request.getParameter("pid");  String pname=request.getParameter("pname");  String price=request.getParameter("price");  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  Connection objcon=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","vbr","admin");  PreparedStatement objpstmt=objcon.prepareStatement("insert into product values(?,?,?)");  objpstmt.setString(1,pid);  objpstmt.setString(2,pname);  objpstmt.setString(3,price);  objpstmt.executeUpdate();  objcon.close();  %>  product id:<%=pid%><br>  product name:<%=pname%><br>  product price:<%=price%><br>  <a href="product.html">click here to add new product</a> |

**Jsp Directives:**

* Jsp directive is an instruction given to the jsp compiler. (Or)
* Jsp directive is an instruction that is given to the jsp compiler.
* There are **3 jsp directives** are available in jsp’s. They are:

1. **Page directive**
2. **Include directive**
3. **Taglib directive**

**Syntax of jsp directive: Example:**



* When Microsoft has released **ASP** programming language they said a developer who knows any scripting language can develop the applications using **ASP**. A java script developer can develop the ASP applications by using java script.
* According to Sunmicrosystem jsp specification a **Jsp** developer can develop the jsp application using any scripting language that means as part of the **Jsp** Scriptlet we can provide **JavaScript** (**or)** **VBscript (or) Jscript (or) perlscript and etc**.
* We can develop the jsp’s by using java also. The jsp compiler is responsible to convert the scripting language into java code.
* **Infact(actually) there are no server vendor has developed the jsp compiler which can support a scripting language.**

**Note:** Earlier a server called as **resin** supports JavaScript as part of jsp Scriptlets.

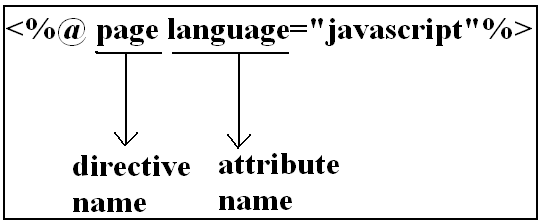
* Every J2EE server which is available in the market are **supporting only one language that is java**.

**Language page directive:**

**What is the purpose of the language page directive?**

* Language page directive is used to specify what is the content inside the Scriptlet.

**Syntax of the language page directive:**

****

**Example for the language page directive:**

|  |
| --- |
| <%@ page language="javascript"%>  <%  var a;  a=10;  %> |

**One.jsp**

* When we try to run one.jsp program jsp compiler gets one instruction that is the language which is used as part of the Scriptlet is javascript.
* Now the jsp compiler is responsible to translate javascript into corresponding java code.
* When we run the above one.jsp program in Tomcat, Weblogic (or) Websphere we get an error message this is because the jsp compiler is not able to convert javascript code into java code.
* This jsp program invoking a server which can translate java script into java.

**Note:** As of today no server is supporting java scripting as part of scriptlets.

**Why the server vendors are not interested to provide the implementation of the scripting language?**

* To provide the tool which converts java scripting language into java code will required lot of logic to be written in the jsp compiler. To provide the logic in the jsp compiler we need more no of developers which cost us more.
* **If we didn’t provide any language page attribute by default it treats the code available in the Scriptlet as “java”.**

**Example:**

|  |
| --- |
| <%  int a=10;  out.print(a);//10  %> |

**One.jsp**

* When we try to run the above one.jsp program the jsp compiler has not found any language page directive so by default the jsp compiler treats the code available in the Scriptlet is java.

**Note:** If there is no language page directive in the jsp, jsp compiler considered the code in the Scriptlet as java code.

**Import page directive:**

**What is the purpose of the import page directive?**

* By using import page directive we can import the different type of packages in the generated Servlet. (or)
* Import page directive is used to import the appropriate package. (or)
* To import the packages we can use import page directive.

**Syntax of the import page directive:**

|  |
| --- |
| **<%@page import=”packagename”%>** |

**Example for the import page directive:**

|  |
| --- |
| <%@page import="java.util.\*"%>  <%  ArrayList a=new ArrayList();  a.add("one");  out.print(a);//[one]  %> |

* If we want to import multiple import statements we can import them by using import page directive by using coma operator.

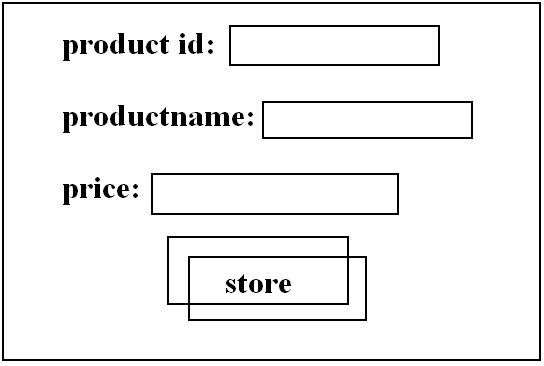
**Example:**

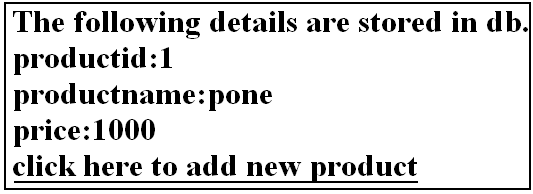
|  |
| --- |
| **<%@page import=”java.util.\*,java.sql.\*,java.net.\*”%>** |

* It is not recommended to use import page directive to import multiple packages. It’s always recommended to use multiple import page directives to import the packages.

**Requirement:** Implement an application to store the data into product table. The following is the prototype of application.

**Prototype:**





**Step1:** To implement the above requirement we need to develop **2 programs**. They are:

Html file (This file is responsible to display form to the user).

**Product.html:**

|  |
| --- |
| <html><head><title>productdetails</title>  </head><body>  <form action="/webappjsp/store.jsp">  ProductId:<input type="text" name="pid"><br>  product name:<input type="text" name="pname"><br>  price:<input type="text" name="price"><br>  <input type="submit" value="store">  </form></body></html> |

**Step2:** Develop the jsp which can capture the data from html form and store the data in database.

**Store.jsp:**

|  |
| --- |
| <%@ page import="java.sql.\*"%>  <%  String pid=request.getParameter("pid");  String pname=request.getParameter("pname");  String price=request.getParameter("price");  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","lms","abc");  PreparedStatement pstmt=con.prepareStatement("insert into product values(?,?,?)");  pstmt.setString(1,pid);  pstmt.setString(2,pname);  pstmt.setString(3,price);  pstmt.executeUpdate();  %>  ProductId:<%=pid%><br>  product name:<%=pname%><br>  price:<%=price%><br>  <a href="/webappjsp/product.html">click here to add new product</a>. |

**Output:**

ProductId: 3  
product name: pthree  
price: 3000  
[**click here to add new product**](http://localhost:8888/webappjsp/product.html)

* As part of the Servlet interface they have provided a method getServletInfo() method. This method returns a string. This method returns the purpose of the development of Servlet.

**Example:**

|  |
| --- |
| public class FirstServlet implements Servlet{  public String getServletInfo(){  return “storing the data in database”;  }} |

**Info page directive**

**What is the purpose of the info page directive?**

* Info page directive tells us the purpose of jsp file. (or)
* When we use this page directive it tells the purpose of jsp file.

**Syntax of the info page directive:**

|  |
| --- |
| **<%@page info=”Here we can provide about jsp”%>** |

**Example of the info page directive:**

|  |
| --- |
| <%@page info=”This is used to store the data in the DB”%> |

**One.jsp**

**What will happen when we use info page directive?**

* Whenever the jsp compiler encounters the info page directive it will convert the page directive into corresponding getServletInfo() method in the generated servlet.

**Diagram:**

|  |
| --- |
| <%@page info="This jsp is used to store the data in database"%> |

|  |
| --- |
| JSPC |

|  |
| --- |
| public class one\_jsp extends HttpJspBase{  public String getServletInfo() {  return "This jsp is used to store the data in database";  }  ...........;  } |

* When we run the above program with the values 1, pone, 1000 and click on store button the application has capture the data and store the data in database. We have run the same application **by giving pid values as abc, ptwo, 2000** and click on store button. Now the application has displayed an exception to the end user.
* We should not display the exceptions to the end user instead of that we **should display a** **proper error message** **this can be down with jsp error pages**.

**IsErrorPage directive**

**What is the purpose of isErrorPage directive?**

* isErrorPage directive is used to indicate that jsp is an error page.

**Syntax of the isErrorPage directive:**

|  |
| --- |
| **<%@page isErrorPage="true||false"%>** |

**One.jsp**

**Example of the isErrorPage directive:**

|  |
| --- |
| <%@page isErrorPage="true"%>  A problem has occured in the project. please call the project admin to resolve the problem. |

**errorPage directive**

**What is the purpose of ErrorPage directive?**

* errorPage directive is used to display the errorpage we use errorPage directive.

**Syntax of the ErrorPage directive:**

|  |
| --- |
| **<%@page errorPage=”/isErrorPagename.jsp”%>** |

**Example of the ErrorPage directive:**

|  |
| --- |
| <%@page errorPage=”/projecterror.jsp”%> |

**Procedure to develop an error page:**

* The page directive **can take multiple attributes. They are:**

1. **language**
2. **extends**
3. **import**
4. **session**
5. **buffer**
6. **autoFlush**
7. **isThreadSafe**
8. **info**
9. **errorPage**
10. **isErrorPage**
11. **contentType**

**Procedure to implement errorpages in jsp:**

**Step 1:** To display the errorpage to the user we need to develop one error page which displays error message to the client. Compulsory every errorpage must use isErrorPage directive. The following one.jsp is an error page of our project.

|  |
| --- |
| <%@page isErrorPage=”true”%>  Problem is occured while processing your request. Please contact system admin. |

**One.jsp**

**Step 2:** In the jsp if the error is encountered we need to display the errorpage. To display the error page we use error page directive.

|  |
| --- |
| <%@ page import="java.sql.\*"%>  <%@ page errorPage="/one.jsp"%>  ………………………………………………………  ……………………………………………………… |

**Store.jsp**

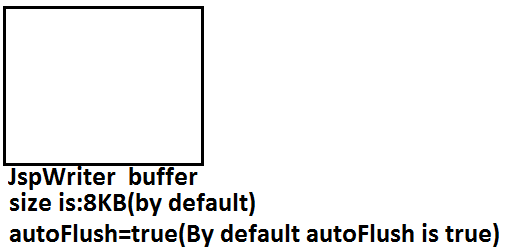
**contentType:** contentType page directive is used to set what is the content send by the jsp to the client program.

**Example:**

|  |
| --- |
| <%@page contentType="text/xml"%>(or)  <%@page contentType="text/html"%>  <html>  <p>welcome to jsp</p>  <p>welcome to jsp</p>  </html> |

* Whenever we are executing a jsp internally the jsp creates jsp buffer object. This buffer is used to add the data given by the application.
* By default the jsp writer buffer size 8kb. To the buffer we have one more property autoFlush. By default the autoFlush value is “**true**”.
* The application uses this buffer. Application starts adding the contents to buffer when it is filled and if autoFlush value is true it sends output to the client and clears the buffer space.

**Diagram:**

****

* In jsp by using page directive we can increase the buffer size.

**Example:**

|  |
| --- |
| <%@page autoFlush="true"%>  <%@page buffer="10kb"%>  <%  for(int i=0;i<2000000;i++)  {  out.println(i+"<br>");  }  %> |

* When we use the above page directive autoFlush and if the value is false and the buffer size is filled the server will not clear the buffer size.
* If the buffer size is filled and autoFlush is false and if we are trying to add the data to buffer it will through an exception jsp buffer overflow.

**What is the difference between JspWriter and PrintWriter objects?**

* Whenever we got the JspWriter object it will allocate a buffer size and starts adding contents to buffer. Whereas the PrintWriter object doesn’t contain the buffer whenever we add the data to PrintWriter object immediately it will be sent to the client.
* When we try to access any website by default we will not except the end user to remember the URL. To access any website the default behavior is after the project name we use “**/**” and click on enter.
* For example to access our web based application “webapp” we except the end user to type the URL as shown below.

**http://localhost:8888/webapp/**

* When we use the above URL by default the server uses (or) search for index.html. If it is available the server sends output to the client. If it is not available the server displays 404 error. In our project the default page name is product.html. If the server needs to pick product.html as default we need to provide the information of default file list in web.xml file.

|  |
| --- |
| <web-app>  <welcome-file-list>  <welcome-file>one.jsp</welcome-file>  <welcome-file>product.html</welcome-file>  </welcome-file-list>  </web-app> |

**Web.xml**

* Similar to Servlet based on our project requirement we can configure jsp file name inside web.xml file. The following is the configuration to configure a jsp in web.xml file.

|  |
| --- |
| <web-app>  <servlet>  <servlet-name>one</servlet-name>  <jsp-file>/one.jsp</jsp-file>  </servlet>  <servlet-mapping>  <servlet-name>one</servlet-name>  <url-pattern>/a</url-pattern>  </servlet-mapping>  </web-app> |

**Web.xml**

* Similar to Servlets we can use <load-on-startup> tag for the jsp programs if we add <load-on-startup> tag for the jsp programs at the time of deployment the jsp compiler converts jsp into Servlets.
* Now the server will create the corresponding Servlet object.
* There are two different types of protocols are available. They are:

1. **Statefull protocol**
2. **Stateless protocol**

**Note:** **http protocol is a stateless protocol.**

**Statefull protocol:**

* If the server is able to remember the conversation which is happening with the client then the server is called as Statefull server.

**Stateless protocol:**

* If the server is not able to remember the conversation which is happening with the client then it is called as stateless protocol.
* Http protocol is a stateless protocol. All the servers which are built on the http protocol are also stateless. These servers are called as stateless servers.

**Requirement:** We want to design an application for registering the user to register the user we need to capture the data from user and store the data into the database. We need to capture username, pwd, fathername, mothername and store the data inside database.

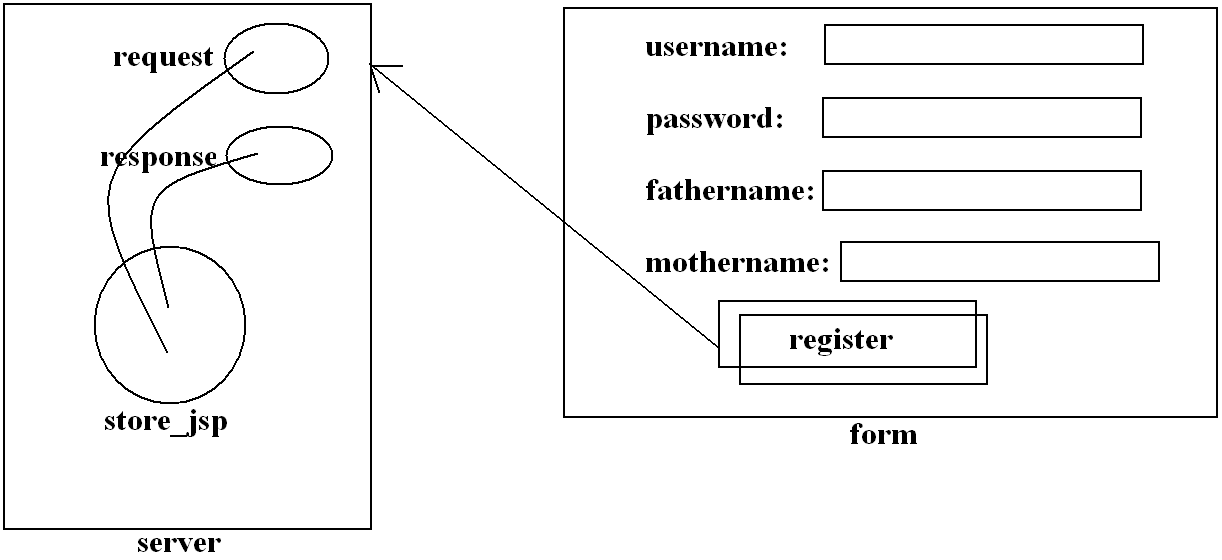
* To implement this application we can design the forms in two ways.

**Approach 1:** In this approach a form contain all the fields which we need to capture from the user (This form contains all the fields’ example 50).

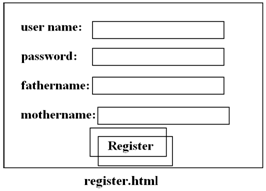
* If we have follow approach1 we need to develop 2 programs. They are:

1. Html form (which displays the all fields).
2. A jsp which can capture the data from all the fields of the application and store the data into the database.

**Diagram:**

****

**Prototype:**





**Register.html:**

|  |
| --- |
| <html><head><title>productdetails</title></head><body>  <form action="/webappjsp/store.jsp">  <center><pre>  <h1>Enter the user details</h1>  <h3>username :<input type="text" name="uname"><br>  password :<input type="password" name="pwd"><br>  fathername:<input type="text" name="fname"><br>  mothername:<input type="text" name="mname"><br>  <input type="submit" value="register"><br></h3>  </form></pre></center></body></html> |

**Store.jsp:**

|  |
| --- |
| <%@ page import="java.sql.\*"%>  <%  String username=request.getParameter("uname");  String password=request.getParameter("pwd");  String fathername=request.getParameter("fname");  String mothername=request.getParameter("mname");  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","lms","abc");  PreparedStatement pstmt=con.prepareStatement("insert into register values(?,?,?,?)");  pstmt.setString(1,username);  pstmt.setString(2,password);  pstmt.setString(3,fathername);  pstmt.setString(4,mothername);  pstmt.executeUpdate();  %>  username:<%=username%><br>  password:<%=password%><br>  fathername:<%=fathername%><br>  mothername:<%=mothername%><br>  <a href="/webappjsp/register.html">click here to add new details</a> |

**Output:**

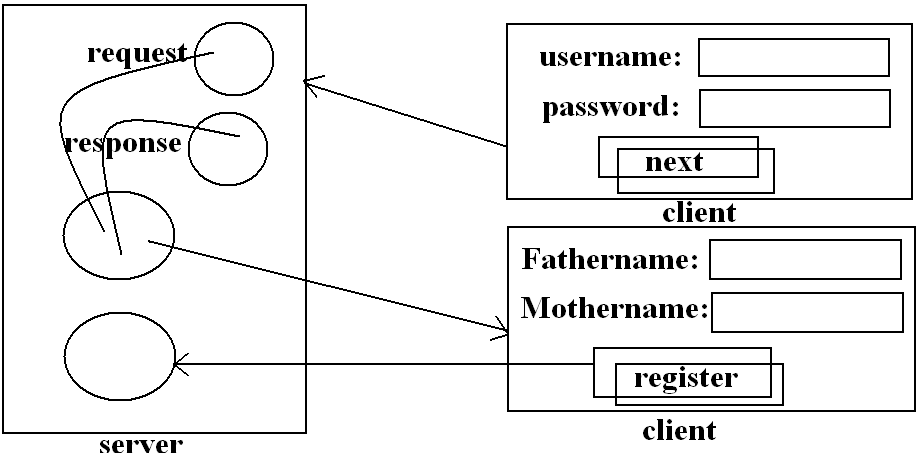
Username: uname  
password: password  
father name: father name  
mother name: mother name  
[click here to add new details](http://localhost:8888/webappjsp/register.html)

**Approach 2:** In this approach we capture the data from the user in multiple stages. Some of the fields we capture in form 1 and the remaining fields we capture in form 2 and finally click on store button to store the data in the database.

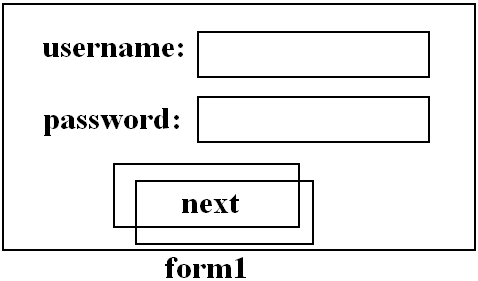
In this approach to implement the above requirement we need to develop 3 programs. They are:

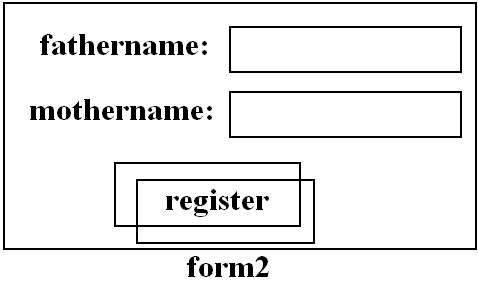
1. **Html file which displays the 1st form.**
2. **A jsp which can capture the data entered by the user in the 1st form and generate 2nd form.**
3. **A jsp which can capture the data from all the fields of the application and display to the user.**

**Diagram:**

****

**Prototype:**



****

**Form1.html:**

|  |
| --- |
| <html><head><title>productdetails</title></head>  <body bgcolor="green">  <form action="/webappjsp/two.jsp">  <center><pre>  <h1>Enter the user details</h1><h3>  username :<input type="text" name="uname"><br>  password :<input type="password" name="pwd"><br>  <input type="submit" value="next"><br></h3>  </form></pre></center></body></html> |

**Two.jsp:**

|  |
| --- |
| <%  String username=request.getParameter("uname");  String password=request.getParameter("pwd");  %>  <body bgcolor="red"><center><pre>  <form action="/webappjsp/store.jsp">  <h1>Enter Fathername and Mothername of the user </h1><h3>  fathername :<input type="text" name="fname"><br>  mothername :<input type="text" name="mname"><br>  <input type="submit" value="store"><br></h3>  </form></center></pre></body> |

**Store.jsp:**

|  |
| --- |
| <%@ page import="java.sql.\*"%>  <%  String username=request.getParameter("uname");  String password=request.getParameter("pwd");  String fathername=request.getParameter("fname");  String mothername=request.getParameter("mname");  %>  <body bgcolor="violet"><center><pre><h1>  username :<%=username%><br>  password :<%=password%><br>  fathername:<%=fathername%><br>  mothername:<%=mothername%><br>  <a href="/webappjsp/form1.html">click here to add new details</a></h1></center></pre></body> |

**Note:** When we run the above application it is not able to capture the data from old forms. This is because of http protocol is a stateless protocol.

**Output:**

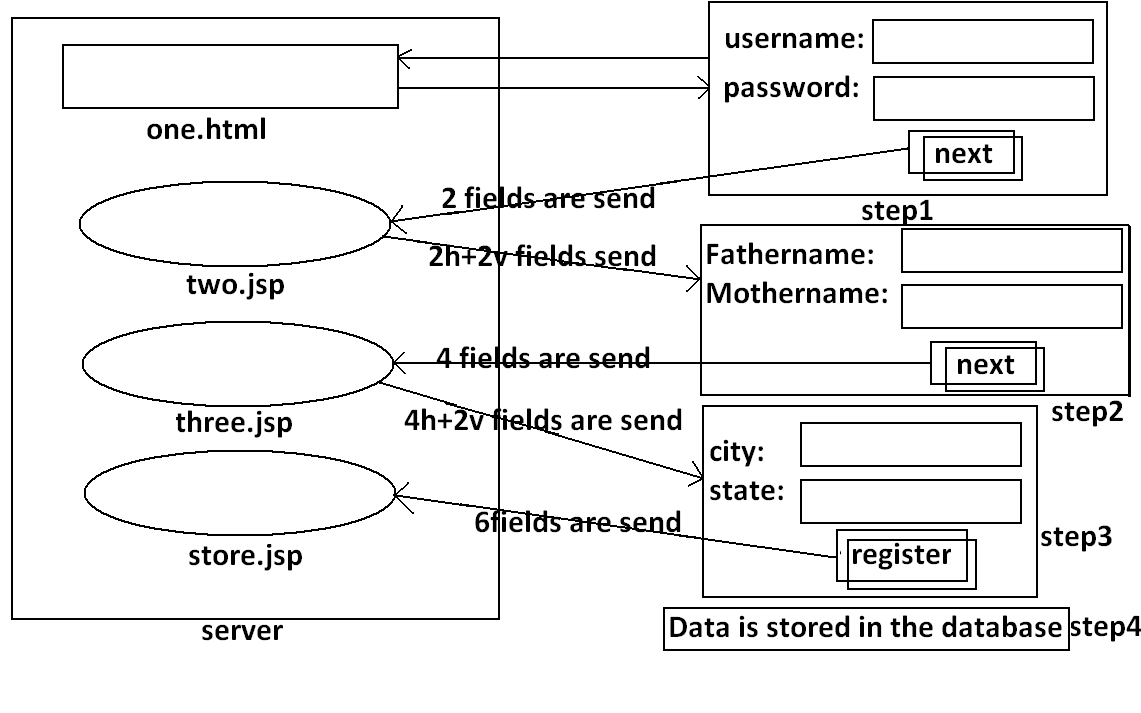
# Username: null Password: null fathername: fathername mothername: mothername [click here to add new details](http://localhost:8888/webappjsp/form1.html)

# We can use any of the following techniques to make (or) to act server as a stateful protocol.

1. **Hidden variables**
2. **Cookies**
3. **Sessions**
4. **Sessions with URL rewriting**

**Hidden variables:**

**Diagram:**



**One.html:**

|  |
| --- |
| <html><head><title>productdetails</title></head>  <body bgcolor="green">  <form action="/webapp/two.jsp">  <center><pre>  <h1>Enter the user details</h1><h3>  username :<input type="text" name="uname"><br>  password :<input type="password" name="pwd"><br>  <input type="submit" value="next"><br></h3>  </form></pre></center></body></html> |

**Two.jsp:**

|  |
| --- |
| <%  String username=request.getParameter("uname");  String password=request.getParameter("pwd");  %>  <body bgcolor="red"><center><pre>  <form action="/webapp/three.jsp">  <h1>Enter Fathername and Mothername of the user </h1><h3>  <input type="hidden" name="uname" value="<%=username%>"><br>  <input type="hidden" name="pwd" value="<%=password%>"><br>  Fathername :<input type="text" name="fname"><br>  Mothername :<input type="text" name="mname"><br>  <input type="submit" value="store"><br></h3>  </form></center></pre></body> |

**Three.jsp:**

|  |
| --- |
| <%  String username=request.getParameter("uname");  String password=request.getParameter("pwd");  String fathername=request.getParameter("fname");  String mothername=request.getParameter("mname");  %>  <body bgcolor="red"><center><pre>  <form action="/webapp/store.jsp">  <h1>Enter Fathername and Mothername of the user </h1><h3>  <input type="hidden" name="uname" value="<%=username%>"><br>  <input type="hidden" name="pwd" value="<%=password%>"><br>  <input type="hidden" name="fname" value="<%=fathername%>"><br>  <input type="hidden" name="mname" value="<%=mothername%>"><br>  city :<input type="text" name="city"><br>  state :<input type="text" name="state"><br>  <input type="submit" value="store"><br></h3>  </form></center></pre></body> |

**Store.jsp:**

|  |
| --- |
| <%@ page import="java.sql.\*"%>  <%  String username=request.getParameter("uname");  String password=request.getParameter("pwd");  String fathername=request.getParameter("fname");  String mothername=request.getParameter("mname");  String city=request.getParameter("city");  String state=request.getParameter("state");  %>  <body bgcolor="violet"><center><pre><h1>  username :<%=username%><br>  password :<%=password%><br>  fathername:<%=fathername%><br>  mothername:<%=mothername%><br>  city:<%=city%><br>  state:<%=state%><br>  <ahref="/webapp/form1.html">click here to add new details</a>  </h1></center></pre></body> |

**Output:**

Username: Bhaskar

Password: Bhaskar

Father name: Anjireddy

Mother name: Anjamma

City: Ongole

# State: Andhrepredesh

# [Click here to add new details](http://localhost:8888/webappjsp/form1.html)

# We would like to design an application which contains 50 fields in the form. Instead of displaying 50 fields in a single form we would like to display 10 fields in a form.

# To implement this requirement we are going to develop 5 jsp programs which display the form. We develop store.jsp to capture the data from 5 forms and store it in database.

# To make this application as Statefull we use hidden variables. The following diagram shows the data transfer between client and server.

# Diagram:

# 

# In the above application when the user interacting with the server more amount of data is being getting transfer between client and server.

# It is not recommended to use hidden variables technique in case of an application which deals with multiple fields in the forms.

# Where we use hidden variables in the project/application?

# We use hidden variables in the project/application in case of search operations.

# Advantage of hidden variables:

# The major advantage of hidden variables is they occupy very less amount of memory in the client computer.

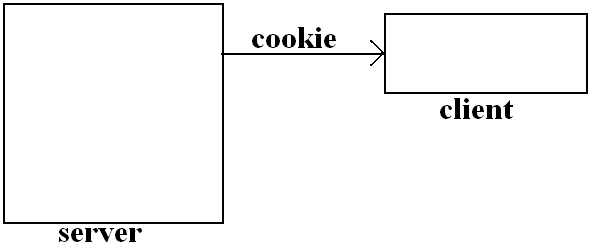
**Disadvantage of hidden variables:**

* If we use the hidden variables the data will be transferred between server program and client program for multiple times. **Because of hidden variables the network traffic will be** **increased**.

**Cookies:**

* Cookie is a small piece of information sent by the server to the client (or)
* Cookie is a small piece of information set by the server on the client memory.

**Diagram:**

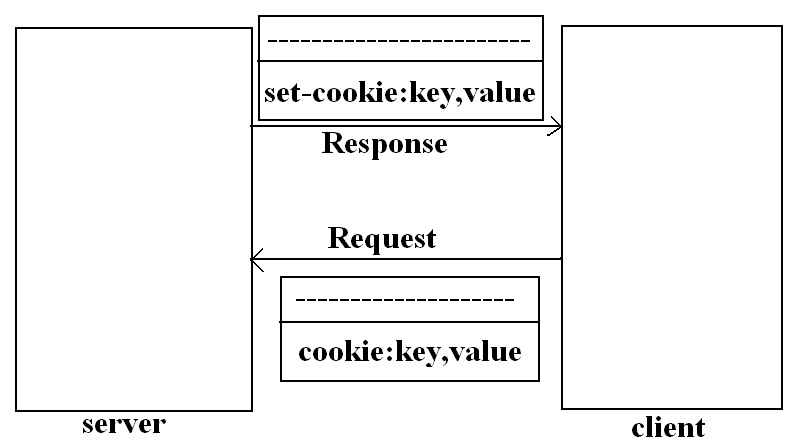


* Every cookie contains the name and value.
* When the server sent cookie to the client the cookies are stored in the client memory under **server’s domain name**.

**Note:** As part of any client the cookies will be stored with the domain name.

* Most of the browser vendors have restricted the maximum no of cookies per domain to 15 only.
* Whenever the client send request to the server all the cookies which are associated to that server will be send back to server.
* When the cookies are sending back to the server the cookies are appended to httprequest format in the form of header and send it to the server.

**Diagram:**



* In java cookie is an object to represent the cookie we use a class **javax.servlet.http.cookie**.

**Procedure to send cookie to the client:**

**Step 1:** To represent a cookie use an object cookie. Cookie contains name and value.

**Example:** **Cookie c1=new cookie("uname","ramesh");**

**Step 2:** To send the cookie to the client we have to add the cookie to response object so that the server sends cookie to client.

**Example:** **response.addCookie(c1);**

* The following jsp sends 2 cookies to the client when we execute the jsp.

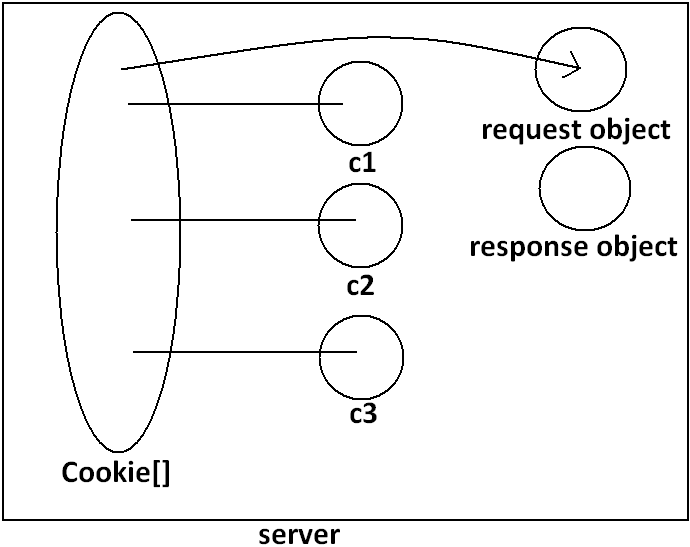
**Program:** sendcookies.jsp

|  |
| --- |
| <%  Cookie c1=new Cookie("uname","bhaskar");  Cookie c2=new Cookie("pwd","bhaskarpassword");  c1.setMaxAge(60);  response.addCookie(c1);  out.println(c1);  out.print("<br>");  c2.setMaxAge(120);  response.addCookie(c2);  out.println(c2);  out.print("<br>");  %>  **Output:**  javax.servlet.http.Cookie@1415056  javax.servlet.http.Cookie@1014e21 |

**Note:** By default every jsp program creates one cookie whose name is “JSESSIONID”.

* Whenever we send a request to the server the browser checks if any cookies are associated to that server, the browser sends all the cookies to the server. The server create objects to every cookie and all the cookie objects add the cookie array object the cookie array object is bound to the request object.

**Diagram:**



* The following jsp program can receive all the cookies from client and display the cookies.

**Program:** retrievecookies.jsp

|  |
| --- |
| <%  Cookie c[]=request.getCookies();  if(c!=null){  for(int i=0;i<c.length;i++){  out.print("name of the cookie :"+c[i].getName()+"<br>");  out.print("value of the cookie :"+c[i].getValue()+"<br>");  }  }else{  out.print("there are no cookies");  }  %>  **Output:**  There are no cookies.  Note: when we run the first time this program we got this output.  Note: when we run the second time this program we got the following output.  **Output:**  name of the cookie :JSESSIONID value of the cookie :774409564A658ABB30A2387D33D1F8DD |

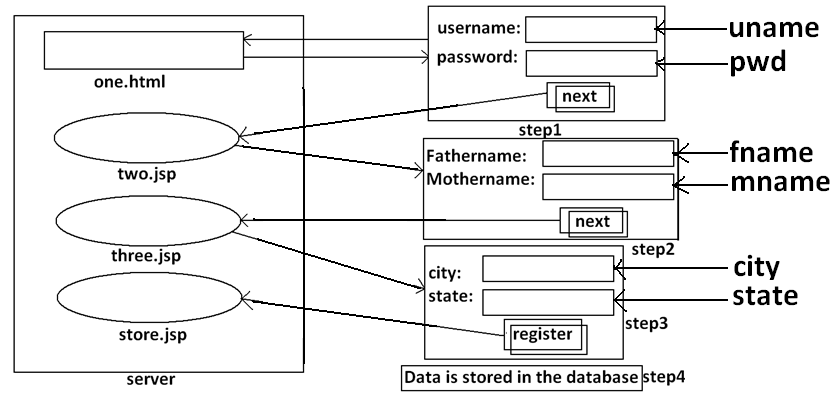
**How do we read a specific cookie value?**

**Program:** specificcookie.jsp

|  |
| --- |
| <%  String uname=null;  String pwd=null;  Cookie c[]=request.getCookies();  if(c!=null){  for(int i=0;i<c.length;i++){  if(c[i].getName().equals("uname")){  out.print(c[i].getName()+"<br>");  out.print(c[i].getValue()+"<br>");  }  if(c[i].getName().equals("pwd")){  out.print(c[i].getName()+"<br>");  out.print(c[i].getValue()+"<br>");  }  if(c[i].getName().equals("JSESSIONID")){  out.print(c[i].getName()+"<br>");  out.print(c[i].getValue()+"<br>");  }}  }else{  out.print("no cookies are available");  }  %>  **Output:**  no cookies are available  Note: when we run the first time this program we got this output.  Note: when we run the second time this program we got the following output.  **Output:**  name of the cookie :JSESSIONID value of the cookie :2AD03BE812F393C1E4094F5317841AC8 |

**Requirement:** The following program captures the data from multiple forms and stores the data inside database. To implement this requirement we use a technique cookies.

**Diagram:**



**Step 1:** Develop html form which displays the following form.

|  |
| --- |
| **File name:One.html**  <html><head><title>user details</title></head>  <body bgcolor="wheat">  <center><pre>  <form action="two.jsp">  <h1>Enter user details</h1><h3>  user name:<input type="text" name="uname"><br>  password:<input type="password" name="pwd"><br>  <input type="submit" value="next"></h3>  </form></center></pre></body></html> |

**Step 2:** When the user files the above form and click on next button the server starts executing two.jsp file. The two.jsp is responsible to capture the data entered by the user in the form1 and create cookie objects and generate second form and send it to the client.

|  |
| --- |
| **File name: two.jsp**  <%  //capture the data from the form  String captureusername=request.getParameter("uname");  String capturepassword=request.getParameter("pwd");  //to store the data first we have to create cookie object. the data will be stored in the form of key and value.  Cookie cookieobj1=new Cookie("username",captureusername);  Cookie cookieobj2=new Cookie("password",capturepassword);  //to send the cookie to the client we have to add the cookie to response object  //so that the server sends cookie to the client.  response.addCookie(cookieobj1);  response.addCookie(cookieobj2);  %>  <body bgcolor="green"><center><pre><form action="three.jsp">  <h1>Enter user fathername and mothername</h1>  <h3>  father name:<input type="text" name="fname"><br>  mother name:<input type="text" name="mname"><br>  <input type="submit" value="next">  </h3></form></center></pre></body> |

**Step 3:** When the user files the second form and click on next button the server starts executing three.jsp. Three.jsp is responsible to capture the data entered by the user in the form2 and create cookie objects and generate third form and send it to the client.

|  |
| --- |
| **File name: three.jsp**  <%  //capture the data from the form  String capturefathername=request.getParameter("fname");  String capturemothername=request.getParameter("mname");  //represent the data in the form of cookie  Cookie cookieobj3=new Cookie("fname",capturefathername);  Cookie cookieobj4=new Cookie("mname",capturemothername);  //send cookie to the client  response.addCookie(cookieobj3);  response.addCookie(cookieobj4);  %>  <body bgcolor="green"><center><pre><form action="/cookieproject/four.jsp">  <h1>Enter user fathername and mothername</h1>  <h3>  city name:<input type="text" name="city"><br>  state name:<input type="text" name="state"><br>  <input type="submit" value="register">  </h3></form></center></pre></body> |

**Step 4:** When the user files the 3rd form and click on register button we need to execute four.jsp. Four.jsp is responsible to capture the data from the form as well as capture the values of user name, password, father name and mother name cookies and store the data in database.

|  |
| --- |
| **Filename: four.jsp**  <%  String username=null;  String pwd=null;  String fname=null;  String mname=null;  //create cookie array object with the help of getCookies() method.  Cookie c[]=request.getCookies();  if(c!=null){  for(int i=0;i<c.length;i++){  Cookie c1=c[i];  if(c1.getName().equals("username")){  username=c1.getValue();  }  if(c1.getName().equals("password")){  pwd=c1.getValue();  }  if(c1.getName().equals("fname")){  fname=c1.getValue();  }  if(c1.getName().equals("mname")){  mname=c1.getValue();  }}}  else{  out.print("no cookies are set");  }  String city=request.getParameter("city");  String state=request.getParameter("state");  %>  <body bgcolor="yellow"><pre>  <center><h1>  The user details are</h1><h3>  Username :<%=username%><br>  Password :<%=pwd%><br>  Father name:<%=fname%><br>  Mother name:<%=mname%><br>  City :<%=city%><br>  State :<%=state%><br>  </h3></pre></center></body> |

**Note:** The above application will fail if the client doesn’t accept cookies.

* There are two types of cookies are available. They are:

1. **Persistence cookies**
2. **Non Persistence cookies**

**What is a persistence cookie?**

* A cookie which is stores inside the computer hard disk is called as persistence cookie. This cookie contains life time. Once if life time expires automatically the cookie will be removed from hard disk.

**What is non persistence cookie?**

* These cookies store inside the browsers memory is called as non persistence cookie. Whenever we close the browser these cookies will be removed. By default we get non persistence cookies only.
* If we want to make a cookie as a persistent cookie we have to set the “maxage” to the cookie. This method takes a +ve number as input and for those many no of seconds the cookie will be stored in the browser.

**Example:**

Cookie c1=new Cookie(“username”,”bhaskarreddy”);

C1.setMaxAge(6000);

**Disadvantages of cookies:**

* Cookies are meant for to store small amount of data only. We cannot use cookies to manage huge amount of data.
* If the client doesn’t accept cookies the application will not work.
* By using cookies we should not store the sensitive data (secured data is called sensitive).

**Advantages of cookies:**

* We can store small amount of data in cookies.

**Sessions:**

* The sessions will resolve the problem which available in hidden variables and cookies.

**What is a session?**

* A session is an object which provides the implementation of **HttpSession** interface directly (or) indirectly. (or)
* An HttpSession object can hold conversational state across multiple requests from the same client.
* In other words, it persists for an entire session with a specific client.

**How do the client and container exchange session id information?**

* The client and container exchange the session id information through cookies.

**When the server create session object?**

* Whenever we send a request to the server for the 1st time on behalf the client the server will create one session object.
* Whenever the session object is created on the server the server is responsible to generate one uniqueid and associate the uniqueid to the session object.
* The server is responsible to generate JSESSIONID cookie and associate the session object id to the JSESSIONID and send the cookie to client. We use this cookie for future conversation.
* In Servlets to represent the session objects we use an interface javax.servlet.http.HttpSession. The server is responsible to create an object to session.
* According to the J2EE it is an API. Once if an API is released anyone in this world can provide the implementation.
* In case of Servlets apache guys has provided the implementation of servlet API. Apache guys are provided a class which provides the implementation of HttpSession interface that class is called session object.
* In case of weblogic BEA guys has provided the implementation of HttpSession interface that class is called as session class.

**The following are the scenarios when the server creates the new session object?**

* If there are no cookies sent by the client to the server then the server creates the new session object.
* If the client send cookies and if doesn’t contain the JSESSIONID cookie then the server will create a new session object.
* If the client and JSESSIONID cookie and if the valid session object is not available in the server, then the server will create the new session object.

**What is the meaning of creating the session object?**

* Creating the session object is nothing but creating the object to a class which provides the implementation of **HttpSession** interface.

**Note:** The name of the class which provides the implementation of HttpSession interface is varying between servers to server.

**Example:**

|  |
| --- |
| import javax.servlet.http.\*;  import javax.servlet.\*;  import java.io.\*;  public class FirstServlet extends HttpServlet{  public void service(HttpServletRequest request,HttpServletResponse response)throws IOException{  //creating the session object with the help of getSession method.  HttpSession session=request.getSession(true); step1  PrintWriter out=response.getWriter();  out.println("session object is : "+session);  out.println("session id is : "+session.getId());  out.println("is new? : "+session.isNew());  }}  **Output:**  Session object is: org.apache.catalina.session.StandardSessionFacade@c2b2f6  Session id is: 1B1E996A0CE38CDFC64EC5A46F4C76E9  Is new? : True |

* In the above program when the step1 is executed the server checks whether the request send by the client for the first time or not.
* If it is first time the server will create a new session object and generates JSESSIONID cookie and send it to the client.
* If the request is for the second time the server will not create a new session object the server will return the old session object.
* The server is responsible to create the session object. The server creates the session object based on client. **The server will create one session object per client for the first time.**

**What will happen when we access request.getSession(true)?**

* When this code is executed the server will check on behalf of the client whether the session object is available or not. If it is not available the server will create new session object on behalf the client. If the session object is already available the server will not create new session object.

**What will happen when we access request.getSession(false)?**

* When this code is executed the server will check on behalf the client whether session object is available or not. If it is available it will not create a new session object. If the session object is not available on behalf of the client it will not create the new session object.
* If the client doesn’t accept cookies the server keeps on creating new session objects. **To** **resolve this problem we use sessions with URL rewriting**.

**What will happen when we send the request for the second time to the server?**

* When we send the request for the 2nd time the browser automatically send all the cookies which are associated to that domain.
* Now the server checks whether the JSESSIONID cookie is available as part of the client request or not.
* If it is available it gets the value of JSESSIONID cookie and check is there any session object is available with that id or not.
* If it is available the server will not create a new session object, the server starts using the old session object only.

**To remove the session object from the server we can use the following 3 techniques?**

**Technique 1:** On the session object we can call a invalidate() method. When we call the invalidate() method it is the responsibility of the server to remove the session object from the servers memory.

* Generally we call invalidate() method when the user clicks on logout button or anchor tag.

**Example:**

|  |
| --- |
| import javax.servlet.http.\*;  import javax.servlet.\*;  import java.io.\*;  public class FirstServlet extends HttpServlet{  public void service(HttpServletRequest request,HttpServletResponse response)throws IOException{  //creating the session object with the help of getSession method.  HttpSession session=request.getSession(true);  PrintWriter out=response.getWriter();  session.setAttribute("vijay","vijay");  session.setAttribute("bhaskar","bhaskar");  String name=(String)session.getAttribute("vijay");  out.println("name:"+name);  session.invalidate();  out.println("session object deleted");  String name1=(String)session.getAttribute("bhaskar");  out.print("name:"+name1);  }}  **Output:**  Session already invalidated |

**Technique 2:** If the user doesn’t click on logout button and if you close the button the server will not remove the session object from its memory.

* By default all the servers will removes the session object after every 30 minutes.

**Example**:

|  |
| --- |
| //import the packages  public class SecondServlet extends HttpServlet{  public void service(HttpServletRequest request,HttpServletResponse response)throws IOException{  //creating the session object with the help of getSession method.  HttpSession session=request.getSession(true);  PrintWriter out=response.getWriter();  out.println("session id is : "+session.getId());  if(session.isNew()){  out.println("This is a new session object");  }  else{  out.println("This is not a new session object");  }}}  **Output:**  session id is : 0FD97AF74B0E45EC050BB9FE62618DCC  This is not a new session object |

**Web.xml**

|  |
| --- |
| <web-app>  <session-config>  <session-timeout>1</session-timeout>  </session-config>  </web-app> |

* We are configured session timeout as part of web.xml file.
* When we configure the session time out in web.xml file it will override the default session time out of server.
* The server will remove every session object it will not use for past 1 minute.

**Technique 3:**

* We can specify the session time out in a servlet program (or) jsp by using a method **setMaxInactiveInterval()** method. In this method we set the time out in seconds.

**Example:**

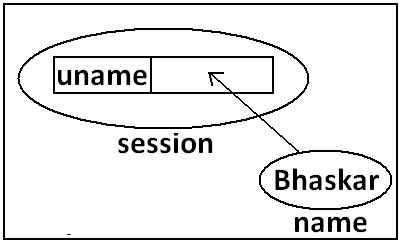
|  |
| --- |
| //import the packages  public class FirstServlet extends HttpServlet{  public void service(HttpServletRequest request,HttpServletResponse response)throws IOException{  //creating the session object with the help of getSession method.  HttpSession session=request.getSession(true);  PrintWriter out=response.getWriter();  out.println("session id is : "+session.getId());  session.setMaxInactiveInterval(30);  if(session.isNew()){  out.println("This is a new session object");  }  else{  out.println("This is not a new session object");  }}}  **Output:**  session id is : 15A8B4C641A443E60D0AC88F9B07AD24  This is a new session object |

* By default the server will give priority to a session object if is specified in java program then it goes to web.xml otherwise servers default timeout.
* In every jsp the implicit variable session will be initialized with request.getSession(true).
* We can store the data as part of session object. To store the data inside session object we use a method setAttribute().

**Example:** session.setAttribute(“uname”,”bhaskar”);

* When the above code is executed it will add Bhaskar object to session scope with a given key uname.

**Diagram:**



* When we add an object to a session scope the object will be converted or type casted to the super class object java.lang.Object. I.e. the string value Bhaskar will be type casted to object as shown below.

Object o=Bhaskar

* To get the data which is stored in session object we use a method getAttribute().

**Example:** session.getAttribute(“uname”);

**Note:** The getAttribute() method returns the value in the form of Object type.

**Note:** When we use a getAttribute() method for non available key the method returns null value.

* To remove an object from session object we use a method removeAttribute().

**Example:** session.removeAttribute(“uname”);

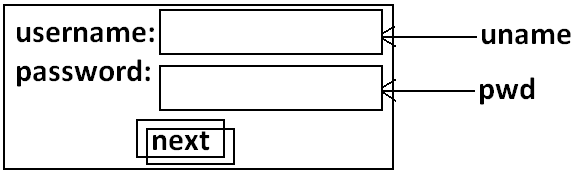
* To store the data first we got a session object. In jsp we no need to create session object. By default the session object is true(request.getSession(true)).
* But in Servlets we have to create the session object. We create the session object in Servlets with the help of request.getSession(true).
* The following application is implemented by using sessions.

**This application is implemented with jsp and Servlets:**

**Implemented using jsp:**

**Step 1:** Develop one.html file which displays username and password fields.

**Diagram:**



**Program:**

|  |
| --- |
| **One.html**  <html><head><title>user details</title></head><body bgcolor="wheat"><center><pre>  <form action="two.jsp"><h1>Enter user details</h1><h3>  user name:<input type="text" name="uname"><br>  password:<input type="password" name="pwd"><br>  <input type="submit" value="next"></h3>  </form></center></pre></body></html> |

**Step 2:** Develop two.jsp which capture the data from the form and store the data in session object and generate three.jsp.

**Program:** two.jsp

|  |
| --- |
| <%@ page import="javax.servlet.http.\*"%>  <%  //capture the data from the form  String captureusername=request.getParameter("uname");  String capturepassword=request.getParameter("pwd");  //store username and password in session object  session.setAttribute("username",captureusername);  session.setAttribute("password",capturepassword);  %>  <body bgcolor="green">  <center><pre>  <form action="three.jsp">  <h1>Enter user fathername and mothername</h1>  <h3>  father name: <input type="text" name="fname"><br>  mother name:<input type="text" name="mname"><br>  <input type="submit" value="next">  </h3></form></center></pre></body> |

**Step 3:** Develop three.jsp which capture the data from two.jsp and store the data in session object and generate four.jsp.

**Program:** three.jsp

|  |
| --- |
| <%  //capture the data from the form  String capturefathername=request.getParameter("fname");  String capturemothername=request.getParameter("mname");  //store fathername and mothername in session object  session.setAttribute("fathername",capturefathername);  session.setAttribute("mothername",capturemothername);  %>  <body bgcolor="green">  <center><pre>  <form action="/webappjsp/four.jsp">  <h1>Enter user fathername and mothername</h1>  <h3>  city name:<input type="text" name="city"><br>  state name:<input type="text" name="state"><br>  <input type="submit" value="register">  </h3></form></center></pre></body> |

**Step 4:** Develop store.jsp program which gets the data from session object as well as get the data from three.jsp and store the data in database.

**Program:** store.jsp

|  |
| --- |
| <%  //the following code get the data from the session object  String username=(String)session.getAttribute("username");  String password=(String)session.getAttribute("password");  String fathername=(String)session.getAttribute("fathername");  String mothername=(String)session.getAttribute("mothername");  //capture the data from the form  String city=request.getParameter("city");  String state=request.getParameter("state");  %>  <body bgcolor="yellow"><pre>  <center><h1>  The user details are</h1><h3>  Username :<%=username%><br>  Password :<%=password%><br>  Fathername:<%=fathername%><br>  Mothername:<%=mothername%><br>  City :<%=city%><br>  State :<%=state%><br>  </h3></pre></center></body> |

**Implemented using Servlets:**

**Step 1:** Develop one.html file which displays username and password fields.

|  |
| --- |
| **One.html**  <html><head><title>user details</title></head>  <body bgcolor="wheat">  <center><pre>  <form action="sup">  <h1>Enter user details</h1><h3>  user name:<input type="text" name="uname"><br>  password:<input type="password" name="pwd"><br>  <input type="submit" value="next"></h3>  </form></center></pre></body></html> |

**Step 2:** Develop a servlet program (SessionGetUnameAndPwd.java) which capture the data from the form and store the data in session object and generate SessionGetFnameAndMothername.java.

**Program:** SessionGetUnameAndPwd

|  |
| --- |
| import javax.servlet.http.\*;  import java.io.\*;  public class SessionGetUnameAndPwd extends HttpServlet{  public void service(HttpServletRequest request,HttpServletResponse response)throws IOException{  PrintWriter out=response.getWriter();  //get session object with the help of getSession() method  HttpSession session=request.getSession(true);  //capture the data from the form  String captureusername=request.getParameter("uname");  String capturepassword=request.getParameter("pwd");  //store username and password in session object  session.setAttribute("username",captureusername);  session.setAttribute("password",capturepassword);  out.print("<body bgcolor='green'>");  out.print("<center>");  out.print("<pre>");  out.print("<form action='sfm'>");  out.print("<h1>Enter user fathername and mothername</h1>");  out.print("<h3>");  out.print("Father name:");  out.print("<input type='text' name='fname'><br>");  out.print("Mother name:");  out.print("<input type='text' name='mname'><br>");  out.print("<input type='submit' value='next'>");  out.print("</h3>");  out.print("</form>");  out.print("</center>");  out.print("</pre>");  out.print("</body>");  }} |

**Step 3:** Develop a servlet program (SessionGetFnameAndMothername.java) which capture the data from the servlet and store the data in session object and generate SessionGetCityAndState.java.

**Program:** SessionGetFnameAndMothername

|  |
| --- |
| import javax.servlet.http.\*;  import java.io.\*;  public class SessionGetFnameAndMothername extends HttpServlet{  public void service(HttpServletRequest request,HttpServletResponse response)throws IOException{  PrintWriter out=response.getWriter();  //get session object with the help of getSession() method  HttpSession session=request.getSession(true);  //capture the data from the form  String capturefathername=request.getParameter("fname");  String capturemothername=request.getParameter("mname");  //store fathername and mothername in session object  session.setAttribute("fathername",capturefathername);  session.setAttribute("mothername",capturemothername);  out.print("<body bgcolor='green'>");  out.print("<center>");  out.print("<pre>");  out.print("<form action='scs'>");  out.print("<h1>Enter user fathername and mothername</h1>");  out.print("<h3>");  out.print("city name:");  out.print("<input type='text' name='city'><br>");  out.print("state name:");  out.print("<input type='text' name='state'><br>");  out.print("<input type='submit' value='register'>");  out.print("</h3>");  out.print("</form>");  out.print("</center>");  out.print("</pre>");  out.print("</body>");  }} |

**Step 4:** Develop a servlet (SessionGetCityAndState.java) program which gets the data from session object as well as get the data from SessionGetFnameAndMothername.java and store the data in database.

**Program:** SessionGetCityAndState

|  |
| --- |
| import javax.servlet.http.\*;  import java.io.\*;  public class SessionGetCityAndState extends HttpServlet{  public void service(HttpServletRequest request,HttpServletResponse response)throws IOException{  PrintWriter out=response.getWriter();  HttpSession session=request.getSession(true);  //the following code get the data from the session object  String username=(String)session.getAttribute("username");  String password=(String)session.getAttribute("password");  String fathername=(String)session.getAttribute("fathername");  String mothername=(String)session.getAttribute("mothername");  //capture the data from the form  String city=request.getParameter("city");  String state=request.getParameter("state");  out.print("<body bgcolor='yellow'><pre>");  out.print("<center><h1>");  out.print("The user details are</h1><h3>");  out.println("<br>Username :"+username);  out.println("<br>Password :"+password);  out.println("<br>Fathername:"+fathername);  out.println("<br>Mothername:"+mothername);  out.println("<br>City :"+city);  out.println("<br>State :"+state);  out.print("</h3>");  out.print("</pre>");  out.print("</center>");  out.print("</body>");  }} |

**Advantages of the sessions:**

* In sessions we can store huge amount of data.

**Disadvantages of the cookies:**

* If the client doesn’t accept cookies the application will not work.

**Sessions with URL rewriting**

* URL rewriting ads the SESSIONID to the end of all the URL’s in the HTML that you write to the response.
* If the client doesn’t accept cookies the session tracking is failed. To resolve this problem we use response.encodeURL() method. This method changes the URL with JSESSIONID and sends it to the client.
* We need to place all the URL’s which are in the project has to be placed inside response.encodeURL() method.
* We need to modify above two.jsp program. The following two.jsp program shown below.

**Example:**

**Step 1:** Program: one.jsp

|  |
| --- |
| <html><head><title>user details</title></head>  <body bgcolor="wheat"><center><pre>  <form action='<%=response.encodeURL("/webappjsp/two.jsp")%>'>  <h1>Enter user details</h1><h3>  user name:<input type="text" name="uname"><br>  password:<input type="password" name="pwd"><br>  <input type="submit" value="next"></h3>  </form></center></pre></body></html> |

**Step 2:** program: two.jsp

|  |
| --- |
| <%@ page import="javax.servlet.http.\*"%>  <%  //capture the data from the form  String captureusername=request.getParameter("uname");  String capturepassword=request.getParameter("pwd");  //store username and password in session object  session.setAttribute("username",captureusername);  session.setAttribute("password",capturepassword);  %>  <body bgcolor="green"><center><pre>  <form action='<%=response.encodeURL("/webappjsp/three.jsp")%>'>  <h1>Enter user fathername and mothername</h1>  <h3>  father name:<input type="text" name="fname"><br>  mother name:<input type="text" name="mname"><br>  <input type="submit" value="next">  </h3></form></center></pre></body> |

**Step 3:** program: three.jsp

|  |
| --- |
| <%  //capture the data from the form  String capturefathername=request.getParameter("fname");  String capturemothername=request.getParameter("mname");  //store fathername and mothername in session object  session.setAttribute("fathername",capturefathername);  session.setAttribute("mothername",capturemothername);  %>  <body bgcolor="green"><center><pre>  <form action='<%=response.encodeURL("/webappjsp/four.jsp")%>'>  <h1>Enter user fathername and mothername</h1>  <h3>  city name:<input type="text" name="city"><br>  state name:<input type="text" name="state"><br>  <input type="submit" value="register">  </h3></form></center></pre></body> |

**Step 4:** program: four.jsp

|  |
| --- |
| <%  //the following code get the data from the session object  String username=(String)session.getAttribute("username");  String password=(String)session.getAttribute("password");  String fathername=(String)session.getAttribute("fathername");  String mothername=(String)session.getAttribute("mothername");  //capture the data from the form  String city=request.getParameter("city");  String state=request.getParameter("state");  %>  <body bgcolor="yellow"><pre><center><h1>  The user details are</h1><h3>  Username :<%=username%><br>  Password :<%=password%><br>  Fathername:<%=fathername%><br>  Mothername:<%=mothername%><br>  City :<%=city%><br>  State :<%=state%><br>  </h3></pre></center></body> |

Scoped Variables:

* In Jsp’s we have four scoped variables are available. They are:

1. Pagescope
2. Requestscope
3. Sessionscope
4. Applicationscope

* In Servlets we have three scoped variables are available. They are:

1. Requestscope
2. Sessionscope
3. Applicationscope(contextscope)

* **Scoped variables are used to store the data.**
* By using scoped variables we can transfer data from one application/layer to another application/layer.
* We can use the following three methods to deal with scoped variables. They are:

1. setAttribute(key,object);
2. getAttribute(key);
3. removeAttribute(key);

* setAttribute() method is used to store the data in scoped variables.
* getAttribute() method is used to get the data from scoped variables.
* removeAttribute() method is used to remove the data from scoped variables.

**PageScope:** If we add/store an object to a PageContext object then we call it as a pagescope. (Or)

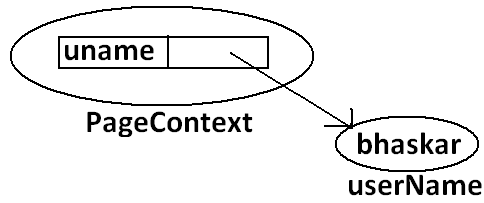
* If we store an object inside pagecontext object then it is called page scope.

**Example:** one.jsp

|  |
| --- |
| <%  String userName="Bhaskar";  pageContext.setAttribute("uname",userName);  Object o=pageContext.getAttribute("uname");  String name=(String)o;  out.print("value is : "+name);  %>  **(Or)**  <%  pageContext.setAttribute("uname","bhaskar");  out.print("value of uname is : "+pageContext.getAttribute("uname"));  %> |

* In the above jsp program we have added userName object to pagescope with the key.

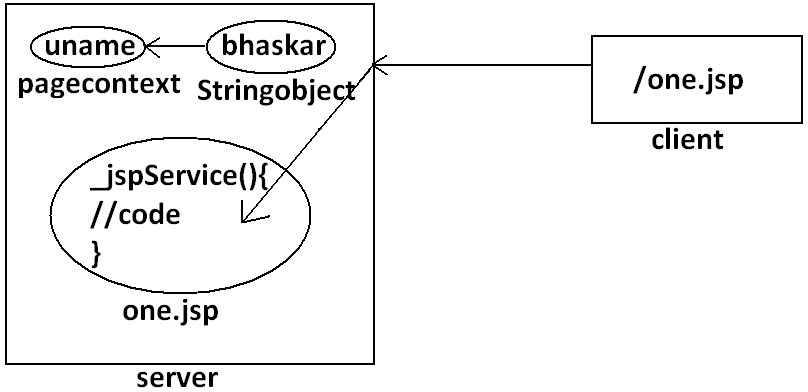
**Diagram:**



* As part of the every jsp program we have pageContext implicit variable. This variable will be created whenever \_jspService() method is executing. This variable will be removed by the server after it has executed \_jspService() method.
* Generally we use the pagecontext object to store the data which can be accessible until the jsp is executed.

**Note:** If we store any data in pagecontext it will be there in the object until the jsp is executed.

**Diagram:**



**Requestscope:** If we store the data in request object then we call it as a request scope.

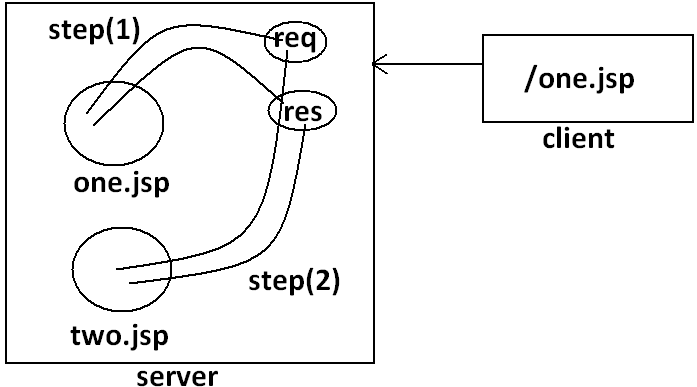
**Example:** one.jsp

|  |
| --- |
| <%  request.setAttribute("uname","vijayabhaskar");  out.print("value of requestscope is: "+request.getAttribute("uname"));  %> |

**When do we use requestscope?**

* In a single request if multiple resources are executing and if they want to share the data we use request scope. (or)
* We use the requestscope if multiple Servlets (or) jsp’s want to share the data.

**Diagram:**



**Note:** If we store the data in requestscope the data will be there in request object until the request is processed.

**Sessionscope:** If we store the data in session object then we call it as a session scope.

**Example:**

|  |
| --- |
| <%  session.setAttribute("uname","vijayabhaskar");  out.print("value of sessionscope is: "+session.getAttribute("uname"));  **%>** |

**When do we use session scope?**

* If we want to store the client specific data then we use session object. If we store the data in a session scope until the session is ended the user can access it.

**Applicationscope:** If we store the data in application (ServletContext) object then we call it as an application scope (ServletContext).

**Example:**

|  |
| --- |
| <%  application.setAttribute("uname","vijayabhaskar");  out.print("value of applicationscope is: "+application.getAttribute("uname"));  %> |

**When do we use application scope?**

* If the data is common to all the clients then we store the data in application scope(ServletContext).
* Generally as part of application scope we store the information like name of the project, background color, foreground color this data is not specific to any client. This data can be accessible by every client for every request.

Request Dispatcher:

* Request dispatcher is an interface of javax.servlet package. This interface contains 2 methods. They are:

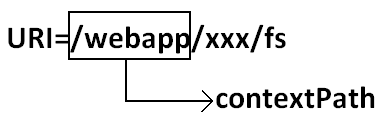
1. Void forward(ServletRequest request,ServletResponse response)
2. Void include(ServletRequest request,ServletResponse response)

**RequestDispatcher:**

* RequestDispatcher is used to dispatch a request from one resource to another resource.
* We use the RequestDispatcher to execute multiple resources in a single request.
* The following is the URL which is used to access 1st servlet.



* Generally from the URL we got URI.



* From the URI if we remove context path we get path of the resource.

**Example:**



**Absolute path:** Any path which starts with “/” (slash) is called as absolute path.

**Example:** /xxx/fs

**Relative path:** Any Path which doesn’t start with “/” (slash) is called as relative path.

**Example:** xxx/fs

* If we want to dispatch a request from one resource to another resource we need to get the RequestDispatcher object.
* We can get the RequestDispatcher object in 2 ways. They are:

1. By using ServletContext object
2. By using ServletRequest object

* To get the RequestDispatcher object we use a method getRequestDispatcher(string path). This method taken path of a resource as argument.

RequestDispatcher rd=application.getRequestDispatcher(String path);

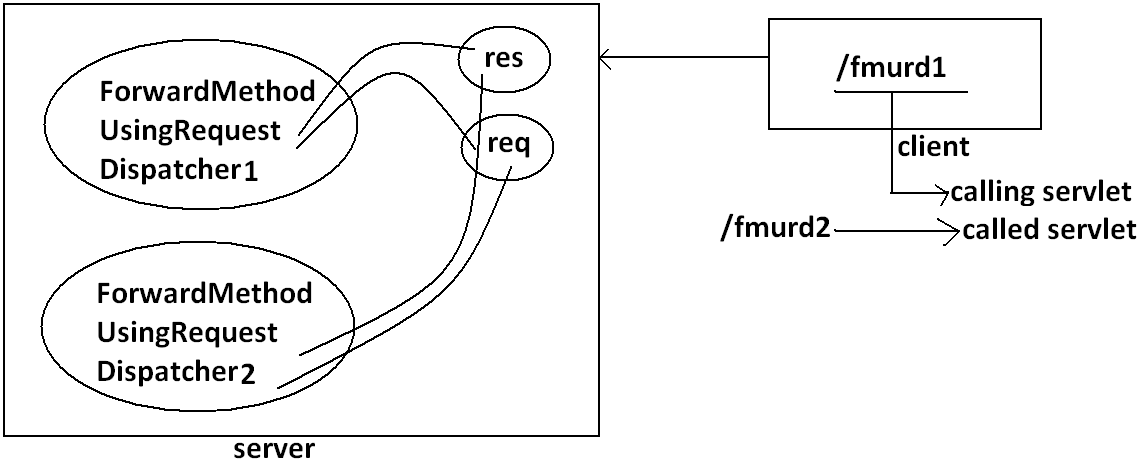
**Example program:** ForwardMethodUsingRequestDispatcher1.java

|  |  |  |
| --- | --- | --- |
| |  | | --- | | import javax.servlet.\*;  import javax.servlet.http.\*;  import java.io.\*;  public class ForwardMethodUsingRequestDispatcher1 extends HttpServlet{  public void service(HttpServletRequest request,HttpServletResponse response)throws IOException,ServletException{  System.out.println("ForwardMethodUsingRequestDispatcher1 class is executed");  PrintWriter out=response.getWriter();  out.println("output from ForwardMethodUsingRequestDispatcher1 class");  //get ServletConfig object with the help of getServletConfig() method  ServletConfig config=getServletConfig();  //get ServletContext object with the help of getservletcontext() method  ServletContext application=config.getServletContext();  //get RequestDispatcher object with the help of getRequestDispatcher() method  RequestDispatcher rd=application.getRequestDispatcher("/fmurd2");  //calling the forward() method (or) include() method on RequestDispatcher object.  rd.forward(request,response); (or)  rd.include(request,response);  }} |   **Program:** ForwardMethodUsingRequestDispatcher2.java   |  | | --- | | import javax.servlet.http.\*;  import java.io.\*;  public class ForwardMethodUsingRequestDispatcher2 extends HttpServlet{  public void service(HttpServletRequest request,HttpServletResponse response)throws IOException{  System.out.println("ForwardMethodUsingRequestDispatcher2 class is executed");  PrintWriter out=response.getWriter();  out.println("output from ForwardMethodUsingRequestDispatcher2 class");  }} | |

**Calling servlet:** If the client send a request to the servlet that servlet is called as a calling servlet.

**Called servlet:** If a servlet is calling another servlet then it is called as called servlet.

**Diagram:**



**What will happen when we call forward() method on the RequestDispatcher object?**

* When we call the forward() method it will remove the output generated by “calling servlet” and it will send only the output generated by the called servlet.

**What will happen when we call include() method on the RequestDispatcher object?**

* When we call the include method the output generated by “calling servlet” and “called servlet” will be included and send the output to the client.

**Servlet chaining**

**Servlet chaining:** When we send a request to a server if more than one servlet is executed in processing a single request then it is called as servlet chaining.

* **We can call multiple include() methods from a single resource.**

**Example:**

|  |  |  |
| --- | --- | --- |
| **Program:** IncludeMethodUsingRequestDispatcher1.java   |  | | --- | | import javax.servlet.\*;  import javax.servlet.http.\*;  import java.io.\*;  public class IncludeMethodUsingRequestDispatcher1 extends HttpServlet{  public void service(HttpServletRequest request,HttpServletResponse response)throws IOException,ServletException{  PrintWriter out=response.getWriter();  out.println("output from IncludeMethodUsingRequestDispatcher1 class");  ServletConfig config=getServletConfig();  ServletContext application=config.getServletContext();  RequestDispatcher rd=application.getRequestDispatcher("/imurd2");  //calling multiple resources from a single resource  rd.include(request,response);  rd.include(request,response);  rd.include(request,response);  } } |   **Program:** IncludeMethodUsingRequestDispatcher2.java   |  | | --- | | import javax.servlet.http.\*;  import java.io.\*;  public class IncludeMethodUsingRequestDispatcher2 extends HttpServlet{  public void service(HttpServletRequest request,HttpServletResponse response)throws IOException{  PrintWriter out=response.getWriter();  out.println("output from IncludeMethodUsingRequestDispatcher2 class");  } } |   **Output:**  output from IncludeMethodUsingRequestDispatcher1 class  output from IncludeMethodUsingRequestDispatcher2 class  output from IncludeMethodUsingRequestDispatcher2 class  output from IncludeMethodUsingRequestDispatcher2 class |

* **In a single resource we cannot call multiple forward() methods, when we call them we get an exception “IllegalStateException”.**

**Example:**

|  |  |  |
| --- | --- | --- |
| **Program:** ForwardMethodUsingRequestDispatcher1.java   |  | | --- | | import javax.servlet.\*;  import javax.servlet.http.\*;  import java.io.\*;  public class ForwardMethodUsingRequestDispatcher1 extends HttpServlet{  public void service(HttpServletRequest request,HttpServletResponse response)throws IOException,ServletException{  PrintWriter out=response.getWriter();  out.println("output from ForwardMethodUsingRequestDispatcher1 class");  ServletConfig config=getServletConfig();  ServletContext application=config.getServletContext();  RequestDispatcher rd=application.getRequestDispatcher("/imurd2");  rd.forward(request,response);  rd.forward(request,response);  rd.forward(request,response);  }} |   **Program:** ForwardMethodUsingRequestDispatcher2.java   |  | | --- | | import javax.servlet.http.\*;  import java.io.\*;  public class ForwardMethodUsingRequestDispatcher2 extends HttpServlet{  public void service(HttpServletRequest request,HttpServletResponse response)throws IOException{  PrintWriter out=response.getWriter();  out.println("output from ForwardMethodUsingRequestDispatcher2 class");  } } |   **Output:**  IllegalStateException: Cannot forward after response has been committed |

* The following code shows getting RequestDispatcher object using request object.

**Example:**

|  |  |  |
| --- | --- | --- |
| Program: IncludeMethodUsingRequestDispatcher1.java   |  | | --- | | import javax.servlet.\*;  import javax.servlet.http.\*;  import java.io.\*;  public class IncludeMethodUsingRequestDispatcher1 extends HttpServlet{  public void service(HttpServletRequest request,HttpServletResponse response)throws IOException,ServletException{  PrintWriter out=response.getWriter();  out.println("output from IncludeMethodUsingRequestDispatcher1 class");  //getting the RequestDispatcher object with the help of request object  RequestDispatcher rd=request.getRequestDispatcher("imurd2");  rd.include(request,response);  }  } |   **Program:** IncludeMethodUsingRequestDispatcher2.java   |  | | --- | | import javax.servlet.http.\*;  import java.io.\*;  public class IncludeMethodUsingRequestDispatcher2 extends HttpServlet{  public void service(HttpServletRequest request,HttpServletResponse response)throws IOException{  PrintWriter out=response.getWriter();  out.println("output from IncludeMethodUsingRequestDispatcher2 class");  }  } |   **Output:**  output from IncludeMethodUsingRequestDispatcher1 class  output from IncludeMethodUsingRequestDispatcher2 class |

**What is the difference between application.getRequestDiapatcher() and request.getRequestDispatcher()?**

* application.getRequestDiapatcher() method excepts the absolute path to a resource. If we give the relative path it will not work.

**Example:**

RequestDispatcher rd=application.getRequestDispatcher("/fmurd2");

Valid path

RequestDispatcher rd=application.getRequestDispatcher("fmurd2");

Invalid path

* request.getRequestDispatcher() method excepts a relative path to a resource.

**Example:**

RequestDispatcher rd=request.getRequestDispatcher("imurd2");

Relative path

**Jsp Action Tags:**

* Sun micro system has released jsp action tags to remove the java code from jsp’s. The following are the jsp action tags.

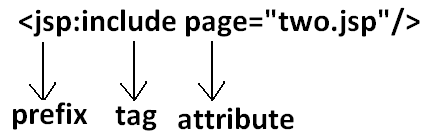
1. **<jsp:include/>**
2. **<jsp:forward/>**
3. **<jsp:useBean/>**
4. **<jsp:setProperty/>**
5. **<jsp:getproperty/>**

(Or)

1. **Include**
2. **Forward**
3. **useBean**
4. **setProperty**
5. **getproperty**

* Every jsp action tag need to call with prefix called as jsp.

**Example:**



**Include tag:**

* We use this include tag to include the contents of calling jsp as well as called jsp.

|  |
| --- |
| * **This tag is an alternative to RequestDispatcher.include().** |

**Example:**

|  |  |  |
| --- | --- | --- |
| **Program:** one.jsp   |  | | --- | | Output from one.jsp.  <%  out.print("<br>");  %>  <jsp:include page="two.jsp"/> |   **Program:** two.jsp   |  | | --- | | Output from two.jsp. |   **Output:**  Output from one.jsp.  Output from two.jsp. |

* When we send the request to above one.jsp we get the output from both one.jsp and two.jsp.
* When we send the request to one.jsp the server converts one.jsp into corresponding Servlets.
* Whenever the server encounters the action tag the server will evaluate the action tags.
* Evaluate means converting tag into corresponding java code.

**Forward Tag:**

* The forward tag will be able to forward the request to any specific resource.

**Example:**

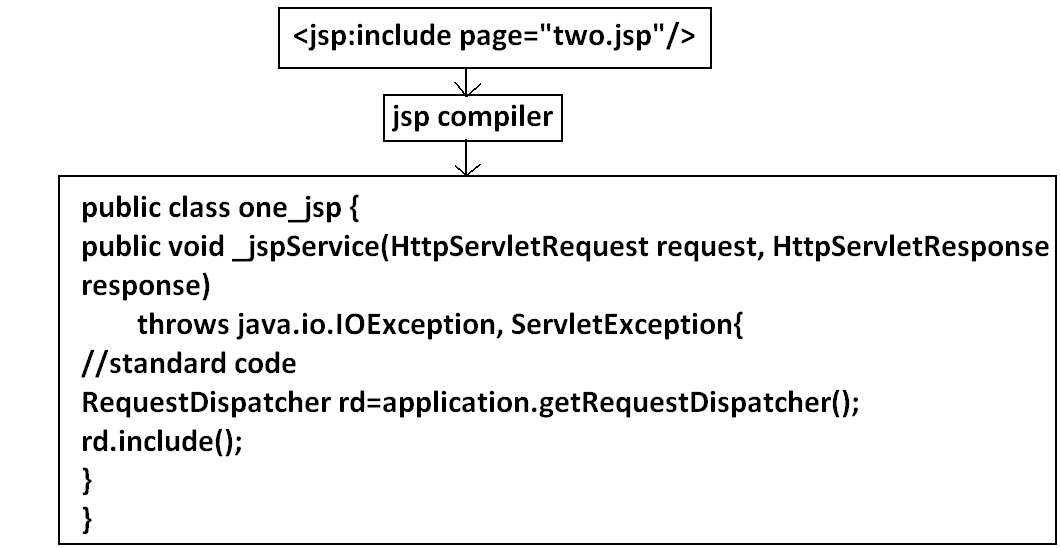
|  |  |  |
| --- | --- | --- |
| **Program:** one.jsp   |  | | --- | | Output from one.jsp.  <%  out.print("<br>");  %>  <jsp:forward page="two.jsp"/> |   **Program:** two.jsp   |  | | --- | | Output from two.jsp. |   **Output:**  Output from two.jsp. |

* When the jsp compiler encounters the tag it will be evaluated.

**What is the meaning of evaluated jsp tag?**

* For every tag the jsp compiler generates the corresponding java code and it will be added to the generated servlet. For example when the jsp compiler encounters <jsp:include………/> the tag is evaluated as shown below.

**Diagram:**



**Include Directive:** include directive is used to include the contents of two.jsp into one.jsp.

**Example:**

|  |  |  |
| --- | --- | --- |
| **Program:** one.jsp   |  | | --- | | Output from one.jsp include directive<br>.  <%@ include file="two.jsp"%> |   **Program:** Two.jsp   |  | | --- | | Output from two.jsp include directive. |   **Output:**  Output from one.jsp include directive output from two.jsp include directive. |

**How the include directive will work?**

* Whenever jsp compiler encounters “include directive” it starts creating a temporary jsp. To the temporary jsp the jsp compiler copies the contents of “calling jsp code” as well as “called jsp code”.
* Now the jsp compiler converts temporary jsp into corresponding servlet, now the server will run generated servlet.
* In “include directive” only one servlet is being executed in the server.
* In “include tag” two Servlets are executed in server.

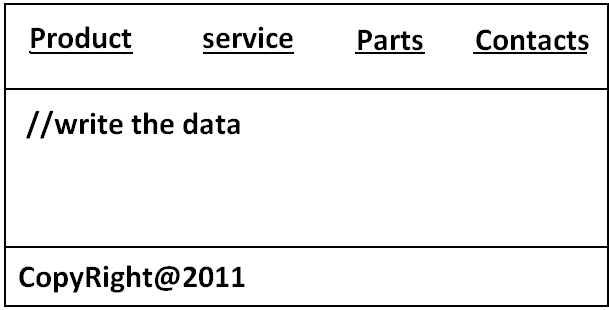
**What is the difference between include directive and include tag?**

* In case of “include directive” only one servlet is responsible to send the output to client(static include).
* In case of “include tag” two Servlets are responsible to send output to client(dynamic include).

**Note:**

* <jsp:include/> tag is called as dynamic include.
* Include directive is called as static include.
* The include directive can include only static resources whereas include tag can include dynamic resources.

**Requirement:** We need to develop the following application for my project?



* We need to develop the above web based application. When we access this website in all the pages we should be able to see the same menu options and bottom copy right information to implement this project we have two approaches.

**Approach 1:**

* Develop four jsp’s in all this jsp’s we need to provide the menu options in the top and copy right information in the bottom.

**Program 1: venture.jsp**

|  |
| --- |
| <body bgcolor="red"><center><h1>  VBR CONSTRUCTIONS PV.LTD<br>  <a href="/vbr1/venture.jsp">venture</a>  <a href="/vbr1/plates.jsp">plates</a>  <a href="/vbr1/individualhouses.jsp">individualhouses</a>  <a href="/vbr1/contacts.jsp">contacts</a><br>  This page will display the venture information of our company.<br>  </center><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br>  <br><br>  CopyRight to VBR CONSTRUCTIONS for year 2011  </h1>  </body> |

**Program 2: plates.jsp**

|  |
| --- |
| <body bgcolor="green"><center><h1>  VBR CONSTRUCTIONS PV.LTD<br>  <a href="/vbr1/venture.jsp">venture</a>  <a href="/vbr1/plates.jsp">plates</a>  <a href="/vbr1/individualhouses.jsp">individualhouses</a>  <a href="/vbr1/contacts.jsp">contacts</a><br>  This page will display the plates information of our company.<br>  </center><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br>  <br><br>  CopyRight to VBR CONSTRUCTIONS for year 2011  </h1>  </body> |

**Program 3: individualhouses.jsp**

|  |
| --- |
| <body bgcolor="yellow"><center><h1>  VBR CONSTRUCTIONS PV.LTD<br>  <a href="/vbr1/venture.jsp">venture</a>  <a href="/vbr1/plates.jsp">plates</a>  <a href="/vbr1/individualhouses.jsp">individualhouses</a>  <a href="/vbr1/contacts.jsp">contacts</a><br>  This page will display the individualhouses information of our company.<br>  </center><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br<br><br>  <br><br>  CopyRight to VBR CONSTRUCTIONS for year 2011  </h1>  </body> |

**Program 4: contacts.jsp**

|  |
| --- |
| <body bgcolor="cyan"><center><h1>  VBR CONSTRUCTIONS PV.LTD<br>  <a href="/vbr1/venture.jsp">venture</a>  <a href="/vbr1/plates.jsp">plates</a>  <a href="/vbr1/individualhouses.jsp">individualhouses</a>  <a href="/vbr1/contacts.jsp">contacts</a><br>  This page will display the contacts information of our company.<br>  </center><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br>  CopyRight to VBR CONSTRUCTIONS for year 2011  </h1>  </body> |

**Disadvantage:**

* In this approach the maintains of the project is difficult. Maintains of the project means changing the menu options (or) changing the copyright information. If we need to change one we have to change in all jsp’s.

**Approach 2:**

* In this approach we create “menu.jsp”. This jsp file contains the code to display all the menu items.
* We develop “footer.jsp”. This jsp contains the copyright information.

**Program 1:** **menu.jsp**

|  |
| --- |
| <body bgcolor="red">  <center>  <h1>  VBR CONSTRUCTIONS PV.LTD<br>  <a href="/vbr/venture.jsp">venture</a>  <a href="/vbr/plates.jsp">plates</a>  <a href="/vbr/individualhouses.jsp">individualhouses</a>  <a href="/vbr/contacts.jsp">contacts</a>  </center>  </h1>  </body> |

**Program 2:** **footer.jsp**

|  |
| --- |
| CopyRight to VBR CONSTRUCTIONS for year 2011 |

* In all other jsp’s we provide the code to include menu.jsp and footer.jsp.

**Program 3:** **venture.jsp**

|  |
| --- |
| <body bgcolor="green">  <center>  <jsp:include page="menu.jsp"/><br>  <h1>  This page will display the venture information of our company.<br>  </center>  <br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br>  <jsp:include page="footer.jsp"/>  </h1>  </body> |

**Program 4:** **plates.jsp**

|  |
| --- |
| <body bgcolor="yellow">  <center>  <jsp:include page="menu.jsp"/><br>  <h1>  This page will display the plates information of our company.<br>  </center>  <br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br>  <jsp:include page="footer.jsp"/>  </h1>  </body> |

**Program 5:** **individualhouses.jsp**

|  |
| --- |
| <body bgcolor="orange">  <center>  <jsp:include page="menu.jsp"/><br>  <h1>  This page will display the individualhouses information of our company.<br>  </center>  <br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br>  <jsp:include page="footer.jsp"/>  </h1>  </body> |

**Program 6: contacts.jsp**

|  |
| --- |
| <body bgcolor="cyan">  <center>  <jsp:include page="menu.jsp"/><br>  <h1>  This page will display the contacts information of our company.<br>  </center>  <br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br>  <jsp:include page="footer.jsp"/>  </h1>  </body> |

**response.sendRedirect():**

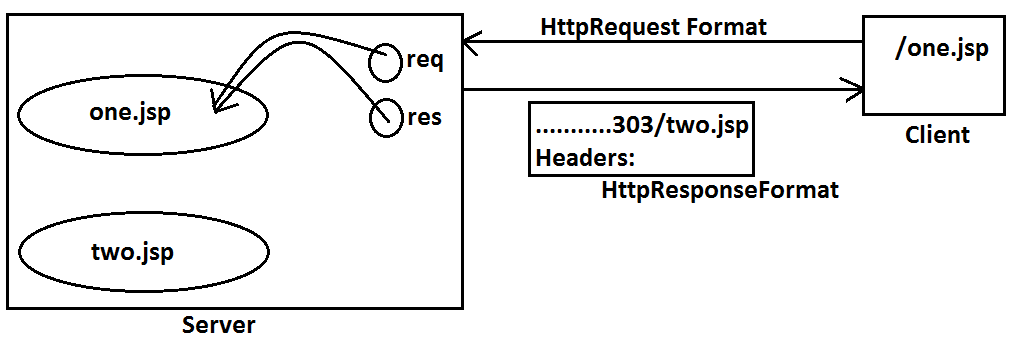
* response.sendRedirect() method is used to forward the request to the resource available in same server as well as the resources are available in another server also.

|  |
| --- |
| output from one.jsp  <%  //response.sendRedirect("/vbr1/two.jsp");  //response.sendRedirect("two.jsp");  response.sendRedirect("http://www.yupptv.com/ntv\_live.html");  %> |

**One.jsp**

* When we send request to above one.jsp the server will create request and response object to one.jsp request.
* The server will hand over the request and response objects to one.jsp. Whenever one.jsp has encounter sendRedirect() method the server will send the response back to client.
* As part of the response it contains the information of 3xx and to which resource the client has to redirect the request. This will be send to browser.

**Diagram:**

****

* Now the browser will automatically send a request to two.jsp.
* By using sendRedirect() method we can redirect request to a resource which is available in different server.
* By using RequestDispatcher we can send request to the resource which is available inside the same server.
* In the software 2 types of designs are there. They are:

1. **Monolithic Design**
2. **Modular design**

**What is a monolithic design?**

* According to monolithic design all the project code will be placed in a single file.
* All the project code means business logic and presentation logic.

**Example:**

|  |
| --- |
| **Product.jsp**  <body bgcolor="red"><pre><center>  <form action="store.jsp"><h1>Enter the product details</h1><h3>  product id :<input type="text" name="pid"><br><br>  product name :<input type="text" name="pname"><br><br>  product price:<input type="text" name="price"><br><br></h3>  <input type="submit" value="store"></form></center></pre></body> |
| **Store.jsp**  //code to import the package  <%@ page import="java.sql.\*"%>  <%   |  | | --- | | //code to capture the data from the form  String productid=request.getParameter("pid");  String productname=request.getParameter("pname");  String productprice=request.getParameter("price"); **Business Logic**  //code to store the data into the database  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  Connection con=DriverManager.getConnection ("jdbc:oracle:thin:@localhost:1521:xe","vbr","123");  PreparedStatement pstmt=con.prepareStatement("insert into product values(?,?,?)");  pstmt.setString(1,productid);  pstmt.setString(2,productname);  pstmt.setString(3,productprice);  pstmt.executeUpdate(); |   %>   |  | | --- | | <center><h1>Data is stored in database</h1>  product id:<%=productid%><br>  productname:<%=productname%><br> **Presentation Logic**  productprice:<%=productprice%><br>  <a href="product.jsp">click here to add new product</a></center> | |

**Note:** In the above example we club the business logic and presentation logic in **store.jsp**. According to sun Microsystems in java it’s not recommended to club business logic and presentation logic.

**What is a modular design?**

* According to modular design the project code will be divided into multiple pieces and we integrate these pieces.
* According to software monolithic design is not a good design this is because the matains of the project will be more.
* If we use modular design the matains of the project becomes easy.

**As part of the projects we will be having 3 different types of logics. They are:**

1. **Business logic**
2. **Presentation logic**
3. **Data access logic**

**Business logic:**

* If we develop a code based on the business rules of a company then it is called as business logic (logic is nothing but a code).
* Business logic will change application to application.
* Generally every project will have their own business logic.
* For example in case of AP online when we register the user they capture the information from the user and storing the data in database and generate password dynamically and send the password to mail id specified by the user.
* In case of railway ticket cancelation system the application calculates the ticket cancelation charges based on travel date and ticket cancelation date.
* If we take applications like **“way2sms”** they take the mobile number and message body and send it **SMS server**.
* In case of sending the emails we have a server called as **mail server** which helps us in sending an email.
* In case of mail application when the user clicks on sending button we need to write the code to send an email to the mail server.

**Presentation logic:**

* The code which is responsible to display the information to the user is called as presentation logic.

**Note:** According to sun Microsystems in java it’s not recommended to club business logic and presentation logic.

**What are the problems we face when we club business logic and presentation logic?**

**The problems are:**

1. The matain of the project is difficult.

**What is the matain of the project?**

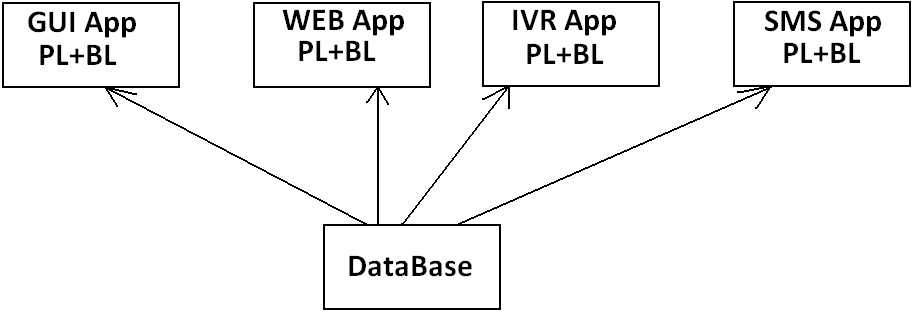
* If there is a change in one application we need to change in all the applications.

1. We cannot reuse the code.

**What is reusing the code?**

* Reusing the code means the same code can be used in other applications then it is called reusing the code.
* **If we develop the application by clubbing business logic and presentation logic** **whenever there is a change in one application we need to change all the applications**.

**Diagram:**



**GUI:** Graphical User Interface

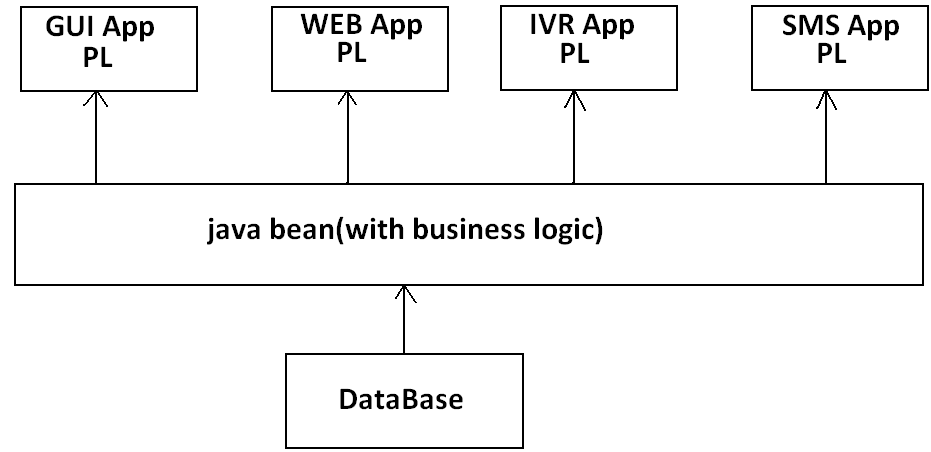
**WEB**:

**IVR**: Interactive Voice Response Applications

**SMS**: Simple Mail Server

* In the above application the business logic and presentation logic is clubbed in all the four applications. If there is a change in the business logic we have to change the business logic in all the four applications because of this the matains of the project becomes difficult. **To resolve this problem we write the business logic code as part of** **the java beans and all the applications can reuse the java beans.**

**Diagram:**



**What is a java Bean?**

* Java Bean is a simple java program which follows the rules defined as part of **java bean** **specification.**

**What is the use of Java Bean?**

* By using java bean we can separate out the business logic and presentation logic.
* Sunmicrosystem has released a specification (or) document with a set of rules. **If any** **java program which follow these set of rules then the java program is called as java bean.**
* Sunmicrosystem **has released 2 java bean specifications. They are:**

1. **Java bean specification**
2. **Enterprise java bean specification.**

**Rules of the java bean:**

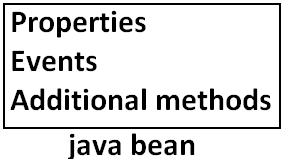
**Rule 1:** Every java bean program must contain the default constructor.

**Rule 2:** Every java bean program must be placed inside a package.

**Note:** This rule is not part of java bean specification.

**Rule 3:** A Java Bean Program Contain set of properties, events, and additional methods.

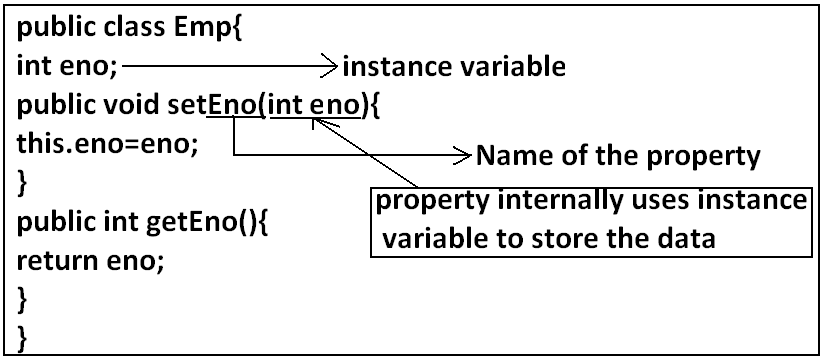
**Diagram:**



**What is a property?**

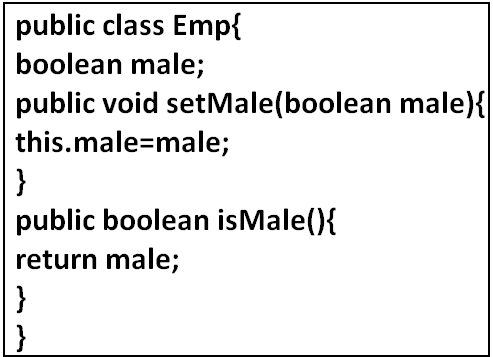
* A property which contains setter methods and getter methods is called property.
* **Generally a property contains setter method and getter method.**

**Example:**



* According to java bean specification even property can have either setter method or getter method.
* **According to java bean specification every setter method has to take an argument and every getter method must return a value.**
* According to java bean specification if the property returns a Boolean value the property must use a method is xxxxx().

**Example:**



**Note:** If we have property we can get the instance variables from the property.

**Example:**

|  |
| --- |
| public void setENO( int eno){  }  public int getEno(){ **setEno() method and getEno() method is called property**  **}** |

* From the property we can write the instance variable. While writing the instance variable the initial letter starts with small letter.
* From the above property we get the instance variable name is: **pid.**

**Note:** if we have instance variables we can write the properties by using instance variables.

**Example:** The instance variable name is: ename.

* from the instance variable we can write the property. While writing the property the instance variable first letter starts with capital letter.
* From the instance variable we can write the property like this.

**public void setEname(String ename){**

**}**

**public String getEname(){**

}

**Events:**

* According to java bean specification an event can contain addxxxxListener and removexxxxListener.

**Example:**

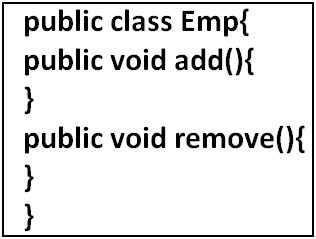
Public void addxxxxListener(xxxEvent)

Public void removexxxxListener(xxxEvent)

**Additional methods:**

* If a java program contains any other methods apart from properties and events we called them additional methods.

**Example:**

****

**What is the meaning of component model?**

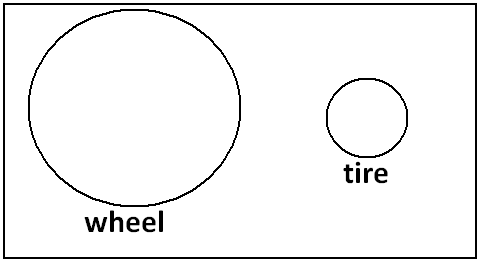
* **Java bean is called as component model technique. We can plug-in this component with any other software tools (or) applications.**

**Note:** The java bean we can integrate **Web Based** (or) **SMS** (or) **IVR** (or) **Standalone** applications.

* If we want to follow a component model the component has to be developed based on the specifications. If the components are not made with specifications it is very difficult to integrate with any other software tools (or) applications.

**Note:** For example here we cannot integrate tire and wheel because wheel is big size and tire is small size(They do not follow the specification).

**Diagram:**



**How many properties are having in our java bean?**

* Based on the no of fields in the form we are having properties in the java bean.

**Requirement:** Implement an application which stores the data in database for the product table. The application must support the business logic and presentation logic.

**Step 1:** Develop a form which displays PID, pname and price.

**Product.jsp:**

|  |
| --- |
| <body bgcolor="red"><pre><center>  <form action="store.jsp">  <h1>Enter the product details</h1><h3>  product id :<input type="text" name="pid"><br><br>  product name :<input type="text" name="pname"><br><br>  product price:<input type="text" name="price"><br><br></h3>  <input type="submit" value="store">  </form></center></pre></body> |

**Step 2:** develop a java bean which can store the data into database.

**ProductJB.java:**

|  |
| --- |
| package info.inetsolv;  import java.sql.\*;  public class ProductJB{  String pid;  String pname;  String price;  public void setPid(String pid){  this.pid=pid;  }  public String getPid(){  return pid;  }  public void setPname(String pname){  this.pname=pname;  }  public String getPname(){  return pname;  }  public void setPrice(String price){  this.price=price;  }  public String getPrice(){  return price;  }  public void storeData(){  try{  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  Connection con=DriverManager.getConnection ("jdbc:oracle:thin:@localhost:1521:xe","vbr","123");  PreparedStatement pstmt=con.prepareStatement("insert into product values(?,?,?)");  pstmt.setString(1,pid);  pstmt.setString(2,pname);  pstmt.setString(3,price);  pstmt.executeUpdate();  }  catch(SQLException e){  e.printStackTrace();  }}} |

**Step 3:** develop a jsp which captures the data from html form and create the java bean object and call the store method.

**Store.jsp:**

|  |
| --- |
| <%  //code to capture the data from the form  String pid=request.getParameter("pid");  String pname=request.getParameter("pname");  String price=request.getParameter("price");  // code to create the object to pojo class  info.inetsolv.ProductJB p=new info.inetsolv.ProductJB();  //code to represent the data in the form of object  p.setPid(pid);  p.setPname(pname);  p.setPrice(price);  p.storeData();  %>  <center><pre>  <h1>Data is stored in database</h1>  product id :<%=p.getPid()%><br> **(or)** product id :<%=pid%><br>  productname :<%=p.getPname()%><br> **(or)** productname :<%=pname%><br>  productprice:<%=p.getPrice()%><br> **(or)** productprice:<%=price%><br>  <a href="product.jsp">click here to add new product</a>  </center></pre> |

**Output:**

Data is stored in database

Product id: 5

Product name: pfive

Product price: 5000

Click here to add new product

* According to sunmicrosystem we should completely remove the java code from jsp’s. **In** **the projects we must use jsp’s as view component (view logic).**

**Can I write business logic as part of jsp?**

* Yes we can write the business logic as part of jsp’s. But it is not recommended.

**useBean tag:**

**Explain useBean tag:**

* useBean tag is part of jsp action tags.

**What is the use of useBean tag?**

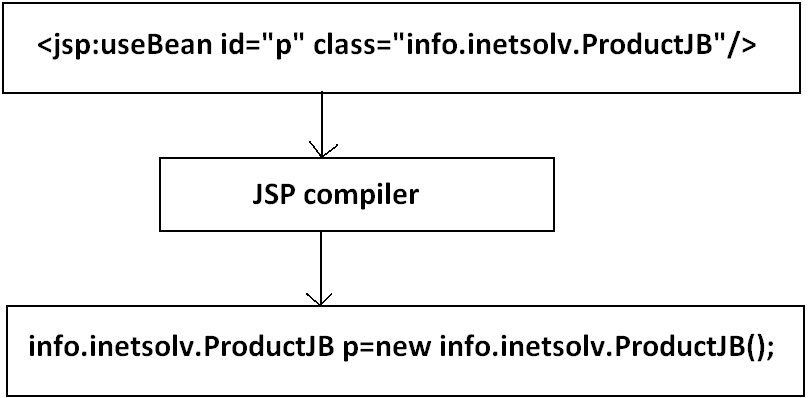
* useBean tag is used to create an object to any specified class.

**Example:**

**<jsp:useBean id="p" class="info.inetsolv.ProductJB"/>**

* When the above useBean tag is evaluated the jsp compiler converts the above tag into the following java code.

**Diagram:**



**Evaluated:** Evaluated means converting the tag into corresponding the java code.

* Whenever the useBean tag is evaluated by the jsp compiler, internally the jsp compiler creates the object to the specified class. By default the object is added to the page scoped variable with the key value. The following code shows how it works.

|  |
| --- |
| <jsp:useBean id="p" class="info.inetsolv.ProductJB"/>  info.inetsolv.Product p=new info.inetsolv.product();    **Note:** To store the object the jsp compiler uses the key name as object name. |

* The useBean tag takes an attribute **“scope”**. The scope attribute can take any of the following 4 scoped variables. They are:

1. Page(by default useBean tag takes the page scope)
2. Request
3. Session
4. application

**Example:**

<jsp:useBean id=”p” class=”info.inetsolv.ProductJB”/>

* When the above tag is evaluated the jsp compiler adds the following code.

Info.inetsolv.ProductJB p=new info.inetsolv.ProductJB();

PageContext.setAttribute(“p”,p);

**The behavior of useBean tag is:**

|  |
| --- |
| info.inetsolv.ProductJB p=null;  Object o=PageContext.getAttribute("p");  if(0==null){  p=new info.inetsolv.ProductJB();  PageContext.setAttribute("p",p);  }else{  p=(info.inetsolv.ProductJB)o;  } |

* When the above code is evaluated it will check whether an object is stored in pagecontext with a key p. if it is not available we are creating the object to ProductJB and adding it PageContext with a key p.
* If the object available in PageContext we are getting the object and storing in the p variable.

**When the useBean tag will create the object?**

* Whenever the useBean tag is evaluated 1st it checks whether the specified id is available in the specific key in the appropriate scoped variable or not. If it is not available the useBean tag creates the object and adds the object to specific scope with the given key. If it is already available then it will not create the object.

**What is the purpose of using set property?**

**SetProperty:** setProperty tag is used to call the appropriate setter() method of a property.

**Example:**

**How do you use it?**

<jsp:useBean id=”p” class=”info.inetsolv.ProductJB”/>

<jsp:setproperty name=”p” property=”pid” value=”777”/>

**What is the purpose of using get property?**

**getproperty:** getproperty tag is used to call the appropriate getxxx() method on the specified java bean.

**Example:**

**How do you use it?**

<jsp:getproperty name=”p” property=”pid”/>

* When the above tag is evaluated it will call a method getpid() on the object p.

**Note:** If any java code is available in the jsp we assume the jsp contains the business logic.

**The following store.jsp we are able to remove 50% of the java code.**

**Program:**

**Store.jsp:**

|  |
| --- |
| <%  //code to capture data from the form  String pid=request.getParameter("pid");  String pname=request.getParameter("pname");  String price=request.getParameter("price");  %>  <jsp:useBean id="p" class="info.inetsolv.ProductJB" scope="page"/>  <jsp:setProperty name="p" property="pid" value='<%=pid%>'/>  <jsp:setProperty name="p" property="pname" value='<%=pname%>'/>  <jsp:setProperty name="p" property="price" value='<%=price%>'/>  <%  //code to call store method  p.storeData();  %>  <center><pre>  <h1>Data is stored in database</h1>  Product id :<jsp:getProperty name="p" property="pid"/><br><br>  productname :<jsp:getProperty name="p" property="pname"/><br>  productprice:<jsp:getProperty name="p" property="price"/><br>  <a href="product.jsp">click here to add new product</a>  </center></pre> |

* In the above jsp we are calling the storeData() method as part of the scriptlet. We can use the following approach to remove the java code from jsp.
* The approach is change the storeData() method to setStoreData() which supports store Data property.

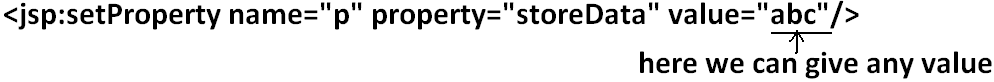
**Program:**

|  |
| --- |
| package info.inetsolv;  import java.sql.\*;  public class ProductJB{  String pid;  String pname;  String price;  public void setPid(String pid){  this.pid=pid;  }  public String getPid(){  return pid;  }  public void setPname(String pname){  this.pname=pname;  }  public String getPname(){  return pname;  }  public void setPrice(String price){  this.price=price;  }  public String getPrice(){  return price;  }  public void setStoreData(String dummy){  try{  //jdbc code  }  catch(SQLException e){  e.printStackTrace();  }  }  } |

**Note:** From the above program we are observing storeData() method is converted into setStoreData() method. This is because There is no way to call the storeData() method directly. Because of this reason we are converting the storeData() method to setStoreData() method. The setter method must take parameter because of that we are giving dummy value. We cannot use the dummy value in the method.

* Now we can call the above setStoreData() method by using setProperty tag.

**Example:**

 Because it is dummy value.

**Store.jsp:**

|  |
| --- |
| <%  String pid=request.getParameter("pid");  String pname=request.getParameter("pname");  String price=request.getParameter("price");  %>  <jsp:useBean id="p" class="info.inetsolv.ProductJB" scope="page"/>  <jsp:setProperty name="p" property="pid"/>  <jsp:setProperty name="p" property="pname"/>  <jsp:setProperty name="p" property="price"/>  <jsp:setProperty name="p" property="storeData" value="abc"/>  <center><pre>  <h1>Data is stored in database</h1>  Product id :<%=p.getPid()%><br>  Product name :<%=p.getPname()%><br>  Productprice :<%=p.getPrice()%><br>  <a href="product.jsp">click here to add new product</a>  </center></pre> |

**Note:** In the above program we writing the java code in the scriptlet.

* The following jsp will be able to capture the data from the form automatically.

**When the jsp will capture the data from the form automatically?**

* If the field name of the html form and property name of the java bean are same then only the jsp will capture the data from the form automatically.

**Note:** If field name and property name is same not require the following statements in the jsp.

|  |
| --- |
| <%  String pid=request.getParameter("pid");  String pname=request.getParameter("pname");  String price=request.getParameter("price");  %>  Because the jsp will capture the data from the form automatically. See the following program store.jsp. |

**Store.jsp:**

|  |
| --- |
| <jsp:useBean id="p" class="info.inetsolv.ProductJB" scope="page"/>  <jsp:setProperty name="p" property="pid"/>  <jsp:setProperty name="p" property="pname"/>  <jsp:setProperty name="p" property="price"/>  <jsp:setProperty name="p" property="storeData" value="abc"/>  <center><pre>  <h1>Data is stored in database</h1>  product id :<%=p.getPid()%><br>  productname :<%=p.getPname()%><br>  productprice:<%=p.getPrice()%><br>  <a href="product.jsp">click here to add new product</a>  </center></pre> |

* The problem with above jsp is if a form contains 100 fields we need to provide 100 setProperty tags. We can resolve the above problem by using following technique.

**<jsp:setProperty name=”p” property=”\*”/>**

**Program: store.jsp**

|  |
| --- |
| <jsp:useBean id="p" class="info.inetsolv.ProductJB" scope="page"/>  <jsp:setProperty name="p" property="\*"/>  <center><pre>  <h1>Data is stored in database</h1>  product id :<%=p.getPid()%><br>  productname :<%=p.getPname()%><br>  productprice:<%=p.getPrice()%><br>  <a href="product.jsp">click here to add new product</a>  </center></pre> |

**CUSTOM TAGS**

* Jsp action tags are not sufficient to remove the java code completely from jsp’s.
* Sunmicrosystem added functionality in jsp1.1 to develop our own tags.

**What are custom tags?**

* Developing our own tags according to the project requirement is called as custom tags.

**In interview we tell like this:**

**What are custom tags?**

* Initially sunmicrosystem has release jsp action tags. These action tags are not sufficient to remove the java code. Because of this reason sunmicrosystem added functionality in jsp1.1 to develop our own tags.
* Developing our own tags according to the project requirement is called as custom tags. Custom tags are used to remove the java code from view(jsp) component.

**What is the use of custom tags?**

* Custom tags are used to remove the java code from view component.

**Procedure to develop the custom tag library:**

**What is a tag library?**

* A tag library contains set of tags.

**Step 1:** A tag library contains set of tags. We need to identify the tags which are going to available in our tag library.

**Note:** we would like to provide add, sub, div, mul tags as part of our tag library.

**Step 2:** identify a name to the tag library.

**Step 3:** we need to clearly identify the functionality of every tag.

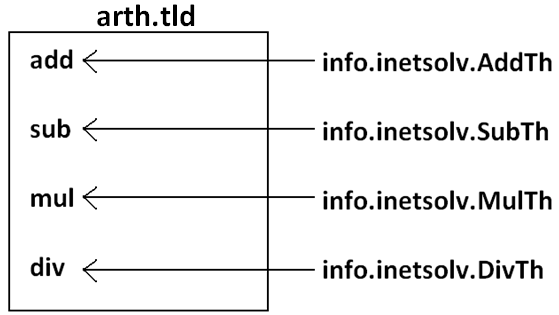
**Step 4:** develop “**tld”** (tag lib descriptor) file for a tag library this is used by jsp compiler to convert the tags into java code.

**Note:** descriptor gives information about something is called descriptor.

**Note:** **uri** means unique identification.

**Step 5:** For every tag we need to develop a class and the class is called as **Tag Class** (or) **Tag Handler Class**.

**Diagram:**

****

**Arth.tld file:**

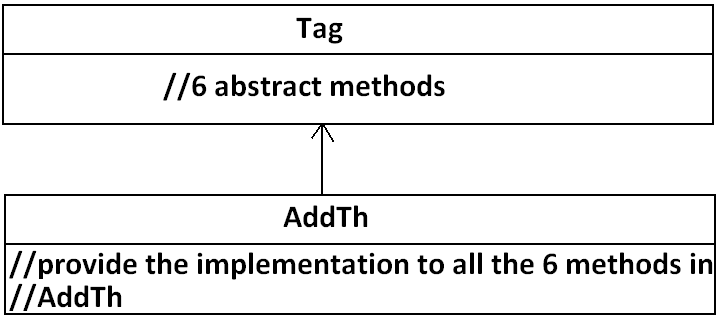
|  |
| --- |
| <taglib> //taglib is a root tag of tld file  <tlib-version>1.0</tlib-version>  <jsp-version>1.2</jsp-version>  <short-name>bhaskartags</short-name>  <uri>customtags</uri>  <description>sampli tag library</description>  <tag> //tag is a child tag of taglib  <name>add</name>  <tag-class>info.inetsolv.AddTh</tag-class>  <attribute>  <name>pone</name>  <required>true</required>  </attribute>  <attribute>  <name>ptwo</name>  <required>true</required>  </attribute>  </tag>  <tag>  <name>sub</name>  <tag-class>info.inetsolv.SubTh</tag-class>  <attribute>  <name>pone</name>  <required>true</required>  </attribute>  <attribute>  <name>ptwo</name>  <required>true</required>  </attribute>  </tag>  <tag>  <name>mul</name>  <tag-class>info.inetsolv.MulTh</tag-class>  <attribute>  <name>pone</name>  <required>true</required>  </attribute>  <attribute>  <name>ptwo</name>  <required>true</required>  </attribute>  </tag>  <tag>  <name>div</name>  <tag-class>info.inetsolv.DivTh</tag-class>  <attribute>  <name>pone</name>  <required>true</required>  </attribute>  <attribute>  <name>ptwo</name>  <required>true</required>  </attribute>  </tag>  </taglib> |

* Sunmicrosystem has released all the required interfaces and classes to develop the custom tags. They are placed inside a package **“javax.servlet.jsp.tagext”**.

**What is a Tag class (or) Tag handler class?**

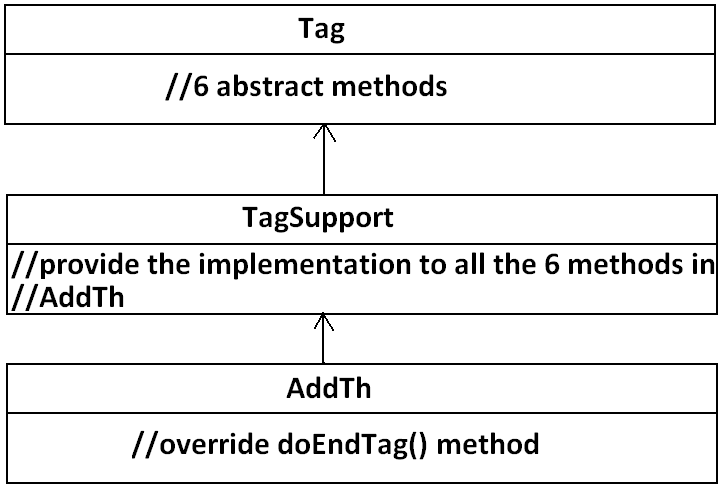
* A class which provides the implementation of tag interface directly (or) indirectly is called Tag Class.

**Diagram:**



* From the above diagram whenever we develop a TagHandler class we have to provide implementation to all the methods of Tag interface. But most of the code is same in all the Tag handler classes.

**Diagram:**

****

* From the above diagram we have understood that we are developing TagHandler class based on TagSupport class. If we develop the TagHandler class based on TagSupport we can reduce the common code.

**Develop the TagHandler classes:**

|  |
| --- |
| **AddTh.java**  package info.inetsolv;  import javax.servlet.jsp.tagext.\*;  import javax.servlet.jsp.\*;  import java.io.\*;  public class AddTh extends TagSupport{  int pone;  int ptwo;  public AddTh(){  System.out.println("AddTh object is created");  }  public void setPone(int pone){  this.pone=pone;  System.out.println("setpone is called:"+pone);  }  public void setPtwo(int ptwo){  this.ptwo=ptwo;  System.out.println("setpone is called:"+ptwo);  }  public int doEndTag(){  System.out.println("doEndTag of AddTh is called");  try{  int result=pone+ptwo;  JspWriter out=pageContext.getOut();  out.print("sum of pone and ptwo:"+result);  }catch(Exception e){  }  return EVAL\_PAGE;  }  } |

|  |
| --- |
| **SubTh.java**  package info.inetsolv;  import javax.servlet.jsp.tagext.\*;  import javax.servlet.jsp.\*;  import java.io.\*;  public class SubTh extends TagSupport{  int pone;  int ptwo;  public SubTh(){  System.out.println("SubTh object is created");  }  public void setPone(int pone){  this.pone=pone;  System.out.println("setpone is called:"+pone);  }  public void setPtwo(int ptwo){  this.ptwo=ptwo;  System.out.println("setpone is called:"+ptwo);  }  public int doEndTag(){  System.out.println("doEndTag of SubTh is called");  try{  int result=pone-ptwo;  JspWriter out=pageContext.getOut();  out.print("subtraction of pone and ptwo:"+result);  }catch(Exception e){  }  return EVAL\_PAGE;  }  } |

|  |
| --- |
| **MulTh.java**  package info.inetsolv;  import javax.servlet.jsp.tagext.\*;  import javax.servlet.jsp.\*;  import java.io.\*;  public class MulTh extends TagSupport{  int pone;  int ptwo;  public MulTh(){  System.out.println("MulTh object is created");  }  public void setPone(int pone){  this.pone=pone;  System.out.println("setpone is called:"+pone);  }  public void setPtwo(int ptwo){  this.ptwo=ptwo;  System.out.println("setpone is called:"+ptwo);  }  public int doEndTag(){  System.out.println("doEndTag of MulTh is called");  try{  int result=pone\*ptwo;  JspWriter out=pageContext.getOut();  out.print("multipication of pone and ptwo is:"+result);  }catch(Exception e){  }  return EVAL\_PAGE;  }  } |

|  |
| --- |
| **DivTh .java**  package info.inetsolv;  import javax.servlet.jsp.tagext.\*;  import javax.servlet.jsp.\*;  import java.io.\*;  public class DivTh extends TagSupport{  int pone;  int ptwo;  public DivTh(){  System.out.println("DivTh object is created");  }  public void setPone(int pone){  this.pone=pone;  System.out.println("setpone is called:"+pone);  }  public void setPtwo(int ptwo){  this.ptwo=ptwo;  System.out.println("setpone is called:"+ptwo);  }  public int doEndTag(){  System.out.println("doEndTag of DivTh is called");  try{  int result=pone/ptwo;  JspWriter out=pageContext.getOut();  out.print("division of pone and ptwo is:"+result);  }catch(Exception e){  }  return EVAL\_PAGE;  }  } |

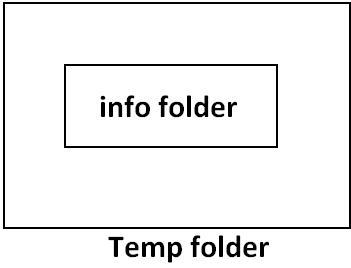
* To compile the above java program we required 2 jar files. They are:

1. **Servlet-api.jsr**
2. **Jsp-api.jar**

**Note:** After compile it generates one **info folder**. The info folder is placed inside one **Temp** **directory** and generate jar file.

* Create a jar file by considering all the TagHandler classes.
* Copy all the TagHandler classes to a temp directory. Use the following command to create a **jar file**.

**Diagram:**



**Command: F:\temp>jar -cvf arth.jar .**

**Note:** The generated jar file is placed inside a project lib directory.

* When someone gives the tag library they have to provide 2 files. They are:

1. **tld file**
2. **Jar file**

**Procedure to use tag library in our project:**

**Step 1:** copy the jar files into the lib folder of our project.

**Step 2:** place the tld file inside web-inf folder.

**Step 3:** configure the tag library in web.xml file.

|  |
| --- |
| **Web.xml:**  <web-app>  <taglib>  <taglib-uri>customtags</taglib-uri>  <taglib-location>/WEB-INF/arth.tld</taglib-location>  </taglib>  </web-app> |

**Step 4:** use the tags in jsp.

**What is the use of taglib directive?**

* Taglib directive is used to import the tag library.

|  |
| --- |
| **Custom.jsp**  <%@ taglib uri="customtags" prefix="inet"%>  <inet:add pone="100" ptwo="200"/><br>  <inet:sub pone="200" ptwo="100"/><br>  <inet:mul pone="100" ptwo="200"/><br>  <inet:div pone="200" ptwo="100"/><br> |

**<inet:add pone="100" ptwo="200"/>**

* Whenever the jsp compiler encounters the above tag it finds the prefix of the tag. Now the jsp compiler checks is there any taglib directive is available whose prefix matches to **“inet”**. If it is available jsp compiler find the **“uri”** of the tag library directive.
* If the jsp compiler has not found any prefix then it treats as **“Template text”**.

**Note:** Here tag is evaluated and display.

* Once if jsp compiler finds the uri the jsp compiler checks in web.xml file is there any tag library which is configure with that “uri”. If it is available it loads the **tld** file in jvm’s memory.

**Note:** The tld file will be loaded only once.

**<inet:addd pone="100" ptwo="200"/>**

* When the jsp compiler encounters a tag it checks whether the tag is available in the tld file or not. If it is not available it throws the error message saying tag is not available in tld file (No tag "addd" defined in tag library imported with prefix "inet").

**<inet:add pone="100"/>**

* Now the jsp compiler checks whether all the required attributes are available or not. If it is available it will evaluate the tag otherwise the jsp compiler throws the exception (According to the TLD or the tag file, attribute ptwo is mandatory for tag add).

**Note:** when we are configuring tld file in web.xml, there is no rule to say compulsory the uri, which is available in the tld must be same as the uri specified in web.xml.

**Example:**

|  |
| --- |
| **Arth.tld:**  <taglib>  <tlib-version>1.0</tlib-version>  <jsp-version>1.2</jsp-version>  <short-name>bhaskartags(any name we can give)</short-name>  <uri>**customtags**</uri>  <description>sampli tag library</description>  </taglib> |
| **Custom.jsp:**  <%@ taglib uri="**customtags**" prefix="inet"%>  <inet:add pone="200" ptwo="100"/><br> |
| **Web.xml:**  <web-app>  <tag-lib>  <taglib-uri>**customtags1**</taglib-uri>  <taglib-location>/WEB-INF/arth.tld</taglib-location>  </tag-lib>  </web-app |

**Approach2:**

**Step 1:** Create the **jar** file with all the TagHandler classes.

**Step 2:** Extract the above jar file. The jar file will generate 2 folders. They are:

1. **Info folder**
2. **META-INF folder**

**Step 3:** Place the **Tld** file with in META-INF folder and delete the above created jar file.

**Step 4:** generate the jar file with info folder and META\_INF folder. The meta-inf folder must contain the Tld folder.

**Step 5:** Place the jar file inside project lib folder.

**Step 6:** We no need configure in web.xml file.

**Step 7:** perform the unit testing.

JSTL:

* Sun Microsystems has released JSTL specification with all the common tags which can be used across multiple projects.
* Sunmicrosystem has released 4 tag libraries as part of JSTL specification they are:

1. **Core**
2. **SQL**
3. **Formating**
4. **Xml**

**Core tag library:** This **Tag** library contains set of tags which deals with the basic functionality of every project. This tag library contains the following tags. They are:

1. **out**
2. **If**
3. **Import**
4. **forEach**
5. **forTokens**
6. **Set**
7. **URL**
8. **When**
9. **Remove**
10. **Catch**
11. **Otherwise and etc.**

**Sql tag library:** This **Tag** library contains the tags which deal with database. The following are the tags of Sql tag library.

1. Query
2. Update
3. SetDatasource
4. Transaction

**Format tag library:** This **Tag** library is used to deal with “I18N applications”. The following are the tags of formatting tag library.

1. SetLocale
2. Bundle
3. Message
4. Formatedate

**Note:** We will be discussing the format tag library in struts.

**Xml tag library:** This **Tag** library is used to deal with xml files. The following are the tags

1. out
2. parse
3. transform

* Apache guys has provided the implementation of JSTL specification we can download the JSTL implementation from “**www.apache.org”**.
* Sunmicrosystem has release jstl1.0 as the initial version of JSTL. The current version of JSTL is 1.1.
* When we downloaded JSTL1.1 we can get all the versions of JSTL Tlds. That is 1.0 as well as 1.1.

**Procedure to configure JSTL in our webbased applications:**

**Step 1:** create a webbased application with a name **“MyJstlProject”**.

**Step 2:** copy the JSTL jar file into the lib folder of our project.

**Step 3:** copy the Tld files inside WEB-INF directory.

**Step 4:** configure the Tag libraries in web.xml file.

**Core tag library:**

**Out:** out tag is used to send the output to the client.

**Example:**

|  |
| --- |
| **One.jsp**  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>  <c:out value="welcome to jstl"/> |

* Value is an attribute of out tag.

**Note:** If we place the tld files inside jar file we no need to configure in web.xml.

* When the above one.jsp is executed the jsp compiler executes out tag. The jsp compiler sends the value attribute value as output to client.
* We should not use out tag to send static content (or) template text to the client. If we want to send static content to client it’s better to use template text rather than out tag.
* Out tag is mainly used in EL Expressions.
* El expressions work with “scoped variables”. EL Expressions cannot deal with java variables.

**What is the difference between JSP Expressions and EL Expressions?**

* Jsp Expressions deal with “java variables”.
* EL Expressions deal with scoped variables only.

**When we use EL Expressions?**

* If data is stored any of the scoped variables to get the data from scoped variables we use EL Expressions.

**Syntax of the EL Expression:**

**Syntax:** ${value}

**Example:**

|  |
| --- |
| **One.jsp**  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>  <%@ page isELIgnored="false"%>  <%  pageContext.setAttribute("var1","100");  %>  value of var1 is:<c:out value="${var1}"/>  **output:**  value of var1 is:100 |

* When the EL Expression is evaluated it checks for the variable in the scoped variable. If the variable is available in any scoped variable it gets the value and display the value. If it is not available in scoped variable it will display a “blank value”.
* When we use the EL Expression by default the jsp compiler checks in the scoped variables in the following order. They are:

1. Page
2. Request
3. Session
4. Application

|  |
| --- |
| **One.jsp**  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>  <%@ page isELIgnored="false"%>  <%  pageContext.setAttribute("var1","100");  pageContext.setAttribute("var2","300");  request.setAttribute("var2","200");  %>  value of var2 is:<c:out value="${var2}"/><br>  value of var2 is:<c:out value="${var2}"/><br>  **output:**  value of var2 is:300 value of var2 is:300 |

* In the above program we get the output 300 because by default el expression search for variable in pagescope.

|  |
| --- |
| **One.jsp**  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>  <%@ page isELIgnored="false"%>  <%  int a=10;  int b=20;  pageContext.setAttribute("var1","100");  pageContext.setAttribute("var2","300");  request.setAttribute("var2","200");  %>  value of var2 is:<c:out value="${var2}"/><br>  value of a is:<c:out value="${a}"/><br>  value of b is:<c:out value="${b}"/><br>  **output:**  value of var2 is:300 value of a is: value of b is: |

* We are dealing with java variables. EL Expressions cannot be used for java variables.

|  |
| --- |
| <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>  <%@ page isELIgnored="false"%>  <%  pageContext.setAttribute("var1","100");  pageContext.setAttribute("var2","200");  %>  sum of var1 and var2 is:<c:out value="${var1+var2}"/>  **output:**  sum of var1 and var2 is:300 |

* On the EL Expressions we can perform arithmetic operations, relational operations and logical operations.
* As part of JSTL we have some implicit variables. They are:

1. pagescope
2. requestScope
3. sessionScope
4. applicationScope

**Example:**

|  |
| --- |
| **Scopedvariables.jsp**  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>  <%@ page isELIgnored="false"%>  <%  pageContext.setAttribute("var1","100");  request.setAttribute("var1","200");  session.setAttribute("var1","300");  application.setAttribute("var1","400");  %>  value of pagescope is:${pageScope.var1}<br>  value of requestscope is:${requestScope.var1}<br>  value of sessionscope is:${sessionScope.var1}<br>  value of applicationscope is:${applicationScope.var1}  **output:**  value of pagescope is:100 value of requestscope is:200 value of sessionscope is:300 value of applicationscope is:400 |

**Another example:**

**Java bean program:**

|  |
| --- |
| **Emp.java**  package info.inetsolv;  public class Emp{  int eno;  String name;  public void setEno(int eno){  this.eno=eno;  }  public int getEno(){  return eno;  }  public void setName(String name){  this.name=name;  }  public String getName(){  return name;  }  } |

|  |
| --- |
| **One.jsp**  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>  <%@ page isELIgnored="false"%>  <%@ page import="java.util.\*"%>  <%  info.inetsolv.Emp e=new info.inetsolv.Emp();  e.setEno(1);  e.setName("eone");  request.setAttribute("empdetails",e);  %>  <%  ArrayList a=new ArrayList();  a.add("one");  a.add("two");  a.add("three");  a.add("four");  request.setAttribute("elements",a);  %>  Arraylist elements are:${elements}<br>  Employee number is:${empdetails.eno}<br>  Employee name is:${empdetails.name}  **Output:**  Arraylist elements are:[one, two, three, four] Employee number is:1 Employee name is: one |

**Param Tag:** In JSTL (or) EL Expressions we can use an implicit variable “param” this is used to capture the data from html form.

**Example:**

|  |
| --- |
| **One.jsp**  <center>  <form action="param.jsp">  value1:<input type="text" name="pone"><br><br>  value2:<input type="text" name="ptwo"><br><br>  <input type="submit" value="store">  </form>  </center> |

|  |
| --- |
| **Param.jsp**  <%@ page isELIgnored="false"%>  value of one is:${param.pone}<br>  value of two is:${param.ptwo}<br>  **output:**  value of one is:100 value of two is:200 |

**Set Tag:** set tag is used to store any data in a scoped variable.

**Example:**

|  |
| --- |
| **Set.jsp**  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>  <%@ page isELIgnored="false"%>  <c:set var="var1" value="100" scope="page"/>  <c:set var="var2" value="200" scope="request"/>  <c:set var="var3" value="300" scope="session"/>  <c:set var="var4" value="400" scope="application"/>  value of page scope is:${var1}<br>  value of request scope is:${var2}<br>  value of session scope is:${var3}<br>  value of application scope is:${var4}  **output:**  value of page scope is:100 value of request scope is:200 value of session scope is:300 value of application scope is:400 |

* When the above tag is evaluated the data will be stored in the pagescope with a key var1.
* If we didn’t specify the scope attribute by default it uses PageScope.

**Remove Tag:** This tag is used to remove the specific key from any scoped variables.

**Example:**

|  |
| --- |
| **Remove.jsp**  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>  <%@ page isELIgnored="false"%>  <c:set var="var1" value="100" scope="page"/>  <c:set var="var2" value="200" scope="request"/>  <c:set var="var3" value="300" scope="session"/>  <c:set var="var4" value="400" scope="application"/>  value of page scope is:${var1}<br>  value of request scope is:${var2}<br>  value of session scope is:${var3}<br>  value of application scope is:${var4}<br>  <c:remove var="var1" scope="page"/>  <c:remove var="var2" scope="request"/>  <c:remove var="var3" scope="session"/>  <c:remove var="var4" scope="application"/>  After removing the values from scoped variables are<br>  value of page scope is:${var1}<br>  value of request scope is:${var2}<br>  value of session scope is:${var3}<br>  value of application scope is:${var4}  **output:**  value of page scope is:100 value of request scope is:200 value of session scope is:300 value of application scope is:400 After removing the values from scoped variables are value of page scope is: value of request scope is: value of session scope is: value of application scope is: |

**If Tag:** This is used to evaluate (or) test the condition based on EL Expressions. If the test condition returns the “true” value the body of the tag will be evaluated otherwise it will not execute the tag body.

**Example:**

|  |
| --- |
| **If.jsp**  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>  <%@ page isELIgnored="false"%>  <c:if test="${1000<200}"> starting of the tag  Test condtion is true that's why body of the for loop is executed<br> Tag body  </c:if> End of the tag  Test condition is false that's why body of the for loop not executed  **Output:**  Test condition is false that's why body of the for loop not executed |

**Composite Tag:** if multiple tags are there then it is called as composite tag.

**Example:** choose Tag

* Choose, when, otherwise these tags are used to implement if else if conditions.

**Choose Tag:**

**How do you implement if else if in JSTL?**

* To implement this we need to combine three tags they are choose, when, otherwise.

**Example:**

|  |
| --- |
| **Choose.jsp**  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>  <%@ page isELIgnored="false"%>  <c:set var="var1" value="1000"/>  <c:choose>  <c:when test="${var1<200}">  we are in if block  </c:when>  <c:otherwise>  we are in else block  </c:otherwise>  </c:choose>  **Output:**  we are in else block |

**forEach:** This tag is used if we want to execute the same work for rapidly

**Note:** This tag is similar to for loop in our java.

**Example:**

|  |
| --- |
| **Foreach.jsp**  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>  <%@ page isELIgnored="false"%>  <c:forEach var="i" begin="0" end="10" step="1">  <c:out value="${pageScope.i}"/>  </c:forEach>  **Output:**  0 1 2 3 4 5 6 7 8 9 10 |

* ForEach tag is used to retrieve the data from Arraylist object (or) Vector object (or) etc.

**Example:**

|  |
| --- |
| **forEach1.jsp**  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>  <%@ page isELIgnored="false"%>  <%  java.util.ArrayList a=new java.util.ArrayList();  a.add("one");  a.add("two");  a.add("three");  a.add("four");  a.add("five");  a.add("six");  request.setAttribute("details",a);  %>  <c:forEach var="var1" items="${details}">  <c:out value="${var1}"/><br>  </c:forEach>  **Output:**  one two three four five six |

**forTokens:** This tag is used to spilt the string which contains deliminators (or) separator.

**Example:**

|  |
| --- |
| **ForTokens:**  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>  <%@ page isELIgnored="false"%>  <%  String name="vaka:vijaya:bhaskara:reddy";  pageContext.setAttribute("name",name);  %>  <c:forTokens var="i" items="${name}" delims=":">  <c:out value="${i}"/><br>  </c:forTokens>  **Output:**  vaka vijaya bhaskara reddy |

**Import Tag:** This tag is similar to jsp include. This will be including the output of multiple jsp into single jsp.

|  |
| --- |
| **Import.jsp**  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>  <%@ page isELIgnored="false"%>  output from import.jsp<br>  <c:import url="foreach.jsp"/><br>  <c:import url="fortokens.jsp"/>  **Output**  output from import.jsp 0 1 2 3 4 5 6 7 8 9 10  vaka vijaya bhaskara reddy |

**Redirect.jsp:** This tag is used to redirect the request to the resource which is available in a different server.

**Example:**

|  |
| --- |
| <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>  <%@ page isELIgnored="false"%>  <c:redirect url="http://www.yupptv.com/sakshitv.html"/> |

**Sql tag library**

* By using this tag library we can deal with database.