



# ADVANCED JAVA

---

## COURSE MATERIAL

---

 <sup>TM</sup> **NARESH** 

Opp.Satyam Theatre, Ameerpet, Hyderabad - 16  
Ph: 040-23746666, 23734842 Cell: 9000994007 / 08

*technologies*

B.O. Kaba Plaza, 3rd Floor, LB Road, Adyar, Chennai - 20. Phone : 044-6555 5557, 6555 5558

 [info@nareshit.com](mailto:info@nareshit.com)



[www.nareshit.com](http://www.nareshit.com)



[nareshit](#)



[nareshitech](#)

**Advanced java**

**By**

**Mr. Nataraj**

# JDBC

Java Database Connectivity or in short JDBC is a technology that enables the java program to manipulate data stored into the database.

**DBC** is Java application programming interface that allows the Java programmers to access database management system from Java code. It was developed by **JavaSoft**, a subsidiary of **Sun Microsystems**.

## Definition

**Java Database Connectivity** in short called as JDBC. It is a java API which enables the java programs to execute SQL statements. It is an application programming interface that defines how a java programmer can access the database in tabular format from Java code using a set of standard interfaces and classes written in the Java programming language.

JDBC has been developed under the Java Community Process that allows multiple implementations to exist and be used by the same application. JDBC provides methods for querying and updating the data in Relational Database Management system such as SQL, Oracle etc.

The Java application programming interface provides a mechanism for dynamically loading the correct Java packages and drivers and registering them with the JDBC **Driver Manager** that is used as a connection factory for creating JDBC connections which supports creating and executing statements such as SQL INSERT, UPDATE and DELETE. Driver Manager is the backbone of the JDBC architecture.

Generally all Relational Database Management System supports SQL and we all know that Java is platform independent, so JDBC makes it possible to write a single database application that can run on different platforms and interact with different Database Management Systems.

Java Database Connectivity is similar to Open Database Connectivity (ODBC) which is used for accessing and managing database, but the difference is that JDBC is designed specifically for Java programs, whereas ODBC is not depended upon any language.

In short JDBC helps the programmers to write java applications that manage these three programming activities:

1. It helps us to connect to a data source, like a database.
2. It helps us in sending queries and updating statements to the database and
3. Retrieving and processing the results received from the database in terms of answering to your query.

## JDBC Versions

1. The JDBC 1.0 API.
2. The JDBC 1.2 API.
3. The JDBC 2.0 Optional Package API.
4. The JDBC 2.1 core API.
5. The JDBC 3.0 API.
6. The JDBC 4.0 API.

## **Features of JDBC 1.0 API**

The JDBC 1.0 API was the first officially JDBC API launched consists of the following java classes and interfaces that you can open connections to particular databases.

This version includes a completely redesigned administration console with an enhanced graphical interface to manage and monitor distributed virtual databases.

**Features of JDBC 1.2 API**

1. It supports Updatable ResultSets.
2. The DatabaseMetaData code has been refactored to provide more transparency with regard to the underlying database engine.
3. New pass through schedulers for increased performance.

**Features of The JDBC 2.0 Optional Package API**

1. The use of DataSource interface for making a connection.
2. Use of JNDI to specify and obtain database connections.
3. It allows us to use Pooled connections, that is we can reuse the connections.
4. In this version the distributed transactions is possible.
5. It provides a way of handling and passing data using Rowset technology.

**Features of the JDBC 2.1 core API.**

1. Scroll forward and backward in a result set or has the ability to move to a specific row.
2. Instead of using SQL commands, we can make updates to a database tables using methods in the Java programming language
3. We can use multiple SQL statements in a database as a unit, or batch.
4. It uses the SQL3 datatypes as column values. SQL3 types are Blob, Clob, Array, Structured type, Ref.
5. Increased support for storing persistent objects in the java programming language.
6. Supports for time zones in Date, Time, and Timestamp values.
7. Full precision for java.math.BigDecimal values.

**Features of JDBC 3.0 API**

1. Reusability of prepared statements by connection pools.
2. In this version there is number of properties defined for the ConnectionPoolDataSource. These properties can be used to describe how the PooledConnection objects created by DataSource objects should be pooled.
3. A new concept has been added to this API is of savepoints.
4. Retrieval of parameter metadata.
5. It has added a means of retrieving values from columns containing automatically generated values.
6. Added a new data type i.e. java.sql.BOOLEAN.
7. Passing parameters to CallableStatement.
8. The data in the Blob and Clob can be altered.
9. DatabaseMetaData API has been added.

**Features of JDBC 4.0 :**

1. Auto-loading of JDBC driver class.
2. Connection management enhancements.
3. Support for RowId SAL type.
4. SQL exception handling enhancements.
5. DataSet implementation of SQL using Annotations.
6. SQL XML support

**JDBC 3.0 & JDBC 4.0**

<b>JDBC 3.0</b>	<b>JDBC 4.0</b>
<p><b>Features:</b></p> <ul style="list-style-type: none"> <li>- Reusability of prepared statements by connection pools.</li> <li>- In this version there is number of properties defined for the ConnectionPoolDataSource. These properties can be used to describe how the PooledConnection objects created by DataSource objects should be pooled.</li> <li>- A new concept has been added to this API is of savepoints: One of the useful new features is transactional savepoints. With JDBC 3.0, the transactional model is now more flexible.</li> <li>- Retrieval of parameter metadata.</li> <li>- It has added a means of retrieving values from columns containing automatically generated values.</li> <li>- Added a new data type i.e. java.sql.BOOLEAN.</li> <li>- Passing parameters to CallableStatement.</li> <li>- The data in the Blob and Clob can be altered: JDBC 3.0 introduces a standard mechanism for updating BLOB and CLOB data.</li> <li>- DatabaseMetaData API has been added.</li> <li>- It allows stored procedure parameters to be called by name.</li> </ul>	<p><b>Features:</b></p> <ul style="list-style-type: none"> <li>- Auto-loading of JDBC driver class: In JDBC 4 invoking the getConnection() on DriverManager will automatically load a driver.</li> <li>- Connection management enhancements: In jdbc it may happen that a Connection is lying idle or not closed in a pool, then it becomes stale over time. This will lead to the connection pool run out of resource due to stale connection.</li> <li>- Support for RowId data type: JDBC introduces support for ROWID, a data type that had been in use in database products even before it became part of the SQL.</li> <li>- SQL exception handling enhancements: JDBC 4 addresses the error handling beautifully. As databases are often remotely accessible resources, problems such as network failures is common and it can cause exceptions when executing a database operation. SQL statements can also cause exceptions. Prior to JDBC 4, most JDBC operations generated a simple SQLException.</li> <li>- SQL XML support.</li> <li>- DataSet implementation of SQL using Annotations: The JDBC 4.0 specification leverages annotations to allow developers to associate a SQL query with a Java class without a need to write a lot of code to achieve this association.</li> </ul>

**Relational Database Concepts**

An important part of every business is to keep records. We need to keep records of our customers, the employees of our company, the emails etc. To keep all the data individually is quite difficult and hectic job, because whenever if we need the record of a particular customer or an employee we need to search manually. It takes lot of time and still not reliable. Here comes the concept of databases.

**What is database?**

A database is an organized collection of information. A simple example of a database are like your telephone directory, recipe book etc.

A Relational model is the basis for any relational database management system (RDBMS). A relational model has mainly three components:

1. A collection of objects or relations.,
2. Operators that act on the objects or relations.
3. Data integrity methods.

To design a database we need three things:

1. Table
2. Rows
3. Columns

A table is one of the most important ingredient to design the database. It is also known as a *relation*, is a

two dimensional structure used to hold related information. A database consists of one or more tables.

A table contains **rows** : Rows is a collection of instance of one thing, such as the information of one employee.

A table contains the **columns**: Columns contains all the information of a single type. Each column in a table is a category of information referred to as a field.

One item of data, such as single phone number of a person is called as a **Data Value**.

### **ACID Properties:**

ACID properties are one of the important concept for databases. ACID stands for Atomicity, Consistency, Isolation, and Durability. These properties of a DBMS allow safe sharing of data. Without these properties the inaccuracy in the data will be huge. With the help of the ACID properties the accuracy can be maintained.

### **Normalization:**

Normalization is a design technique which helps the to design the relational databases. Normalization is essentially a two step process that puts data into tabular form by removing redundant data from the relational tables. A basic goal of normalization is to create a set of relational tables that are free of redundant data and the data should be consistent. Normalization has been divided into following forms.

1. **First Normal Form:** A relational table, by definition are in first normal form. All values of the columns are atomic. It means that it contains no repeating values.
2. A relational table is in second normal form if it is in 1NF and every non- key column is fully dependent upon the primary key.
3. A relational table is in third normal form (3NF) if it is already in 2NF and every non- key column is non transitively dependent upon its primary key. The advantage of having table in 3NF is that it eliminates redundant data which in turn saves space and reduces manipulation anomalies.

### **Understanding Common SQL statements**

The commonly used SQL statements are:

- 1): Select
- 2): Insert
- 3): Update
- 4): Delete

#### **SQL Select statement:**

The SELECT statement is used to select data from a table.

Syntax: **Select column\_names FROM table\_name;**

The result from a SQL query is stored in a result test. The SELECT statement has mainly three clauses.

- 1) Select
- 2) From
- 3) Where

The *Select* specifies the table columns that are retrieved. The *From* clause tells from where the tables has been accessed. The *Where* clause specifies which tables are used. The *Where* clause is optional, if not used then all the table rows will be selected.

We can see that we have used semicolon at the end of the select statement. It is used to separate each SQL statement in database systems which helps us to execute more than one SQL statement in the same

call to the server.

### **SQL INSERT Statement:**

This statement allows you to insert a single or multiple records into the database. We can specify the name of the column in which we want to insert the data.

Syntax: **Insert into table\_name values (value1, value2..);**

The Insert statement has mainly three clauses.

- 1) *Insert*: It specifies which table column has to be inserted in the table.
- 2) *Into*: It tells in which the data will be stored.
- 3) *Values*: In this we insert the values we have to insert.

We can also specify the columns for which we want to insert data.

### **The UPDATE Statement:**

The Update statement is used to modify the data in the table. Whenever we want to update or delete a row then we use the Update statement.

The syntax is :

**UPDATE table\_name Set column\_name = new\_value WHERE column\_name = some\_name;**

The Update statement has mainly three clauses.

- 1) *UPDATE*: It specifies which table column has to be updated.
- 2) *Set*: It sets the column in which the data has to be updated.
- 3) *Where*: It tells which tables are used.

### **SQL DELETE Statement:**

This delete statement is used to delete rows in a table.

Syntax:

**DELETE FROM table\_name WHERE column\_name = some\_name;**

The Delete statement has following clauses.

- 1) *Delete*: It specifies which table column has to be deleted.
- 2) *From*: It tells from where the Table has been accessed.
- 3) *Where*: It tells which tables are used.

### **Introduction to java.sql package**

This package provides the APIs for accessing and processing data which is stored in the database especially relational database by using the java programming language. It includes a framework where we different drivers can be installed dynamically to access different databases especially relational databases.

This java.sql package contains API for the following :

#### **1 Making a connection with a database with the help of DriverManager class**

DriverManager class: It helps to make a connection with the driver.

SQLPermission class: It provides a permission when the code is running within a Security

Manager, such as an Applet. It attempts to set up a logging stream through the DriverManager class.

Driver interface : This interface is mainly used by the DriverManager class for registering and connecting drivers based on JDBC technology.

DriverPropertyInfo class : This class is generally not used by the general user.

## **2. Sending SQL Parameters to a database:**

Statement interface: It is used to send basic SQL statements.

PreparedStatement interface: It is used to send prepared statements or derived SQL statements from the Statement object.

CallableStatement interface : This interface is used to call database stored procedures.

Connection interface : It provides methods for creating statements and managing their connections and properties.

Savepoint: It helps to make the savepoints in a transaction.

## **3. Updating and retrieving the results of a query:**

ResultSet interface: This object maintains a cursor pointing to its current row of data. The cursor is initially positioned before the first row. The next method of the resultset interface moves the cursor to the next row and it will return false if there are no more rows in the ResultSet object. By default ResultSet object is not updatable and has a cursor that moves forward only.

## **4. Providing Standard mappings for SQL types to classes and interfaces in Java Programming language.**

Array interface: It provides the mapping for SQL Array.

Blob interface : It provides the mapping for SQL Blob.

Clob interface: It provides the mapping for SQL Clob.

Date class: It provides the mapping for SQL Date.

Ref interface: It provides the mapping for SQL Ref.

Struct interface: It provides the mapping for SQL Struct.

Time class: It provides the mapping for SQL Time.

Timestamp: It provides the mapping for SQL Timestamp.

Types: It provides the mapping for SQL types.

## **5. Metadata**

DatabaseMetaData interface: It keeps the data about the data. It provides information about the database.

ResultSetMetaData: It gives the information about the columns of a ResultSet object.

ParameterMetaData: It gives the information about the parameters to the PreparedStatement commands.

## **6. Exceptions**

SQLException: It is thrown by the methods whenever there is a problem while accessing the data or any other things.

SQLWarning: This exception is thrown to indicate the warning.

BatchUpdateException: This exception is thrown to indicate that all commands in a batch update are not executed successfully.

DataTruncation: It is thrown to indicate that the data may have been truncated.

## **7. Custom mapping an SQL user- defined type (UDT) to a class in the java programming language.**

SQLData interface: It gives the mapping of a UDT to an instance of this class.

SQLInput interface: It gives the methods for reading UDT attributes from a stream.

SQLOutput: It gives the methods for writing UDT attributes back to a stream.

**DriverManager Class****The JDBC DriverManager**

The JDBC Driver Manager is a very important class that defines objects which connect Java applications to a JDBC driver. Usually Driver Manager is the backbone of the JDBC architecture. It's very simple and small that is used to provide a means of managing the different types of JDBC database driver running on an application. The main responsibility of JDBC database driver is to load all the drivers found in the system

properly as well as to select the most appropriate driver from opening a connection to a database. The Driver Manager also helps to select the most appropriate driver from the previously loaded drivers when a new open database is connected.

The DriverManager class works between the user and the drivers. The task of the DriverManager class is to keep track of the drivers that are available and handles establishing a connection between a database and the appropriate driver. It even keeps track of the driver login time limits and printing of log and tracing messages. This class is mainly useful for the simple application, the most frequently used method of this class is DriverManager.getConnection(). We can know by the name of the method that this method establishes a connection to a database.

The DriverManager class maintains the list of the Driver classes. Each driver has to be registered in the DriverManager class by calling the method DriverManager.registerDriver().

By calling the Class.forName() method the driver class get automatically loaded. The driver is loaded by calling the Class.forName() method. JDBC drivers are designed to tell the DriverManager about themselves automatically when their driver implementation class get loads.

This class has many methods. Some of the commonly used methods are given below:

1. **deregisterDriver(Driver driver)** : It drops the driver from the list of drivers registered in the DriverManager class.
2. **registerDriver(Driver driver)** : It registers the driver with the **DriverManager** class.
3. **getConnection(String url)** : It tries to establish the connection to a given database URL.
4. **getConnection(String url, String user, String password)** : It tries to establish the connection to a given database URL.
5. **getConnection(String url, Properties info)** : It tries to establish the connection to a given database URL.
6. **getDriver(String url)** : It attempts to locate the driver by the given string.
7. **getDrivers()** : It retrieves the enumeration of the drivers which has been registered with the DriverManager class.

**Understanding Data Source**

The JDBC API provides the **DataSource** interface as an alternative to the DriverManager for establishing the connection. A DataSource object is the representation of database or the data source in the Java programming language. DataSouce object is mostly preferred over the DriverManager for establishing a connection to the database.

**DataSource** object can be thought as a factory for making connections to the particular database that the **DataSource** instance represents.

DataSource has a set of properties that identify and describe the real world data source that it represents. The properties include information about the location of the database server, the network protocol use to communicate with the server the name of the database and so on.

DataSource object works with JNDI (Java Naming and Directory interface) naming service so application can use the JNDI API to access the DataSource object.

In short we can say that the **DataSource** interface is implemented to provide three kinds of connections:

**1). Basic DataSource class**

This class is provided by the driver vendor. It is used for portability and easy maintenance.

**2). To provide connection pooling.**

It is provided by the application server vendor or driver vendor. It works with ConnectionPoolDataSource class provided by a driver vendor. Its advantage is portability, easy maintenance and increased performance.

**3). To provide distributed transactions**

This class works with an **XADatasource** class, which is provided by the driver vendor. Its advantages are easy maintenance, portability and ability to participate in distributed transactions.

**Understanding Connection Object**

A **Connection** object represents a connection with a database. When we connect to a database by using connection method, we create a Connection Object, which represents the connection to the database. An application may have one or more than one connections with a single database or many connections with the different databases also.

We can use the Connection object for the following things:

1. It creates the **Statement**, **PreparedStatement** and **CallableStatement** objects for executing the SQL statements.
2. It helps us to **Commit** or **roll back** a JDBC transaction.
3. If you want to know about the database or data source to which you are connected then the **Connection** object gathers information about the database or data source by the use of **DatabaseMetaData**.
4. It helps us to close the data source. The **Connection.isClosed()** method returns true only if the **Connection.close()** has been called. This method is used to close all the connection.

Firstly we need to establish the connection with the database. This is done by using the method **DriverManager.getConnection()**. This method takes a string containing a URL. The **DriverManager** class, attempts to locate a driver that can connect to the database represented by the string URL. Whenever the **getConnection()** method is called the **DriverManager** class checks the list of all registered **Driver** classes that can connect to the database specified in the URL.

**Syntax:**

```
String url = "jdbc: odbc: makeConnection";  
  
Connection con = DriverManager.getConnection( url,"userID","password");
```

**Choosing a DB Driver:**

The type of driver depends on quite a few parameters : whether the application is internet based, whether it needs to support heterogeneous databases, the number of concurrent users, and so on. JDBC drivers are divided into four types or levels. Each type defines a JDBC driver implementation with increasingly higher levels of platform independence, performance, and deployment administration.:

**JDBC driver types**

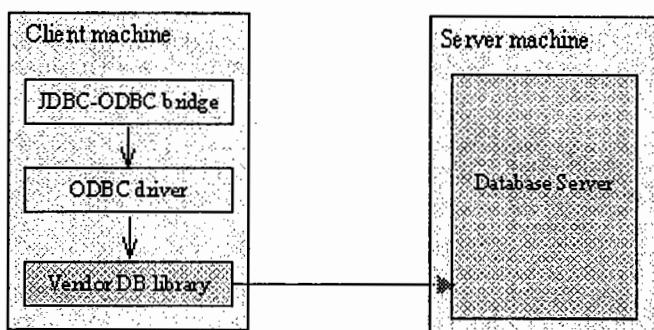
JDBC drivers are divided into four types or levels. Each type defines a JDBC driver implementation with increasingly higher levels of platform independence, performance, and deployment administration. The four types are:

- Type 1: JDBC-ODBC Bridge
- Type 2: Native-API/partly Java driver

- Type 3: Net-protocol/all-Java driver  
 Type 4: Native-protocol/all-Java driver

### Type 1: JDBC-ODBC Bridge

The type 1 driver, JDBC-ODBC Bridge, translates all JDBC calls into ODBC (Open DataBase Connectivity) calls and sends them to the ODBC driver. As such, the ODBC driver, as well as, in many cases, the client database code, must be present on the client machine. Figure 1 shows a typical JDBC-ODBC Bridge environment.



**Figure 1. Type 1: JDBC-ODBC Bridge**

#### Pros

The JDBC-ODBC Bridge allows access to almost any database, since the database's ODBC drivers are already available. Type 1 drivers may be useful for those companies that have an ODBC driver already installed on client machines.

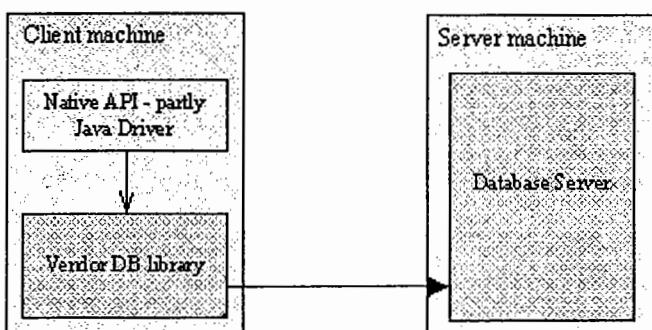
#### Cons

The performance is degraded since the JDBC call goes through the bridge to the ODBC driver, then to the native database connectivity interface. The result comes back through the reverse process. Considering the performance issue, type 1 drivers may not be suitable for large-scale applications.

The ODBC driver and native connectivity interface must already be installed on the client machine. Thus any advantage of using Java applets in an intranet environment is lost, since the deployment problems of traditional applications remain.

### Type 2: Native-API/partly Java driver

JDBC driver type 2 -- the native-API/partly Java driver -- converts JDBC calls into database-specific calls for databases such as SQL Server, Informix, Oracle, or Sybase. The type 2 driver communicates directly with the database server; therefore it requires that some binary code be present on the client machine.



**Figure 2. Type 2: Native-API/partly Java driver**

**Pros**

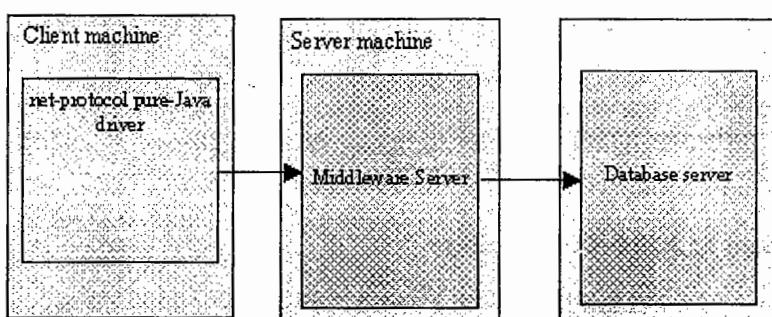
Type 2 drivers typically offer significantly better performance than the JDBC-ODBC Bridge.

**Cons**

The vendor database library needs to be loaded on each client machine. Consequently, type 2 drivers cannot be used for the Internet. Type 2 drivers show lower performance than type 3 and type 4 drivers.

**Type 3: Net-protocol/all-Java driver**

JDBC driver type 3 -- the net-protocol/all-Java driver -- follows a three-tiered approach whereby the JDBC database requests are passed through the network to the middle-tier server. The middle-tier server then translates the request (directly or indirectly) to the database-specific native-connectivity interface to further the request to the database server. If the middle-tier server is written in Java, it can use a type 1 or type 2 JDBC driver to do this.



**Figure 3. Type 3: Net-protocol/all-Java driver**

**Pros**

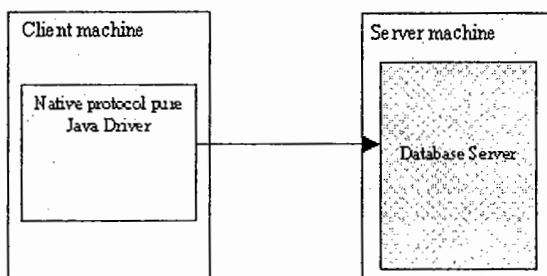
The net-protocol/all-Java driver is server-based, so there is no need for any vendor database library to be present on client machines. Further, there are many opportunities to optimize portability, performance, and scalability. Moreover, the net protocol can be designed to make the client JDBC driver very small and fast to load. Additionally, a type 3 driver typically provides support for features such as caching (connections, query results, and so on), load balancing, and advanced system administration such as logging and auditing.

**Cons**

Type 3 drivers require database-specific coding to be done in the middle tier. Additionally, traversing the recordset may take longer, since the data comes through the backend server.

**Type 4: Native-protocol/all-Java driver**

The native-protocol/all-Java driver (JDBC driver type 4) converts JDBC calls into the vendor-specific database management system (DBMS) protocol so that client applications can communicate directly with the database server. Level 4 drivers are completely implemented in Java to achieve platform independence and eliminate deployment administration issues.



**Figure 4. Type 4: Native-protocol/all-Java driver**

**Pros**

Since type 4 JDBC drivers don't have to translate database requests to ODBC or a native connectivity interface or to pass the request on to another server, performance is typically quite good. Moreover, the native-protocol/all-Java driver boasts better performance than types 1 and 2. Also, there's no need to install special software on the client or server. Further, these drivers can be downloaded dynamically.

**Cons**

With type 4 drivers, the user needs a different driver for each database.

**Choosing right Driver**

Here we will walk through initially about the types of drivers, availability of drivers, use of drivers in different situations, and then we will discuss about which driver suits your application best.

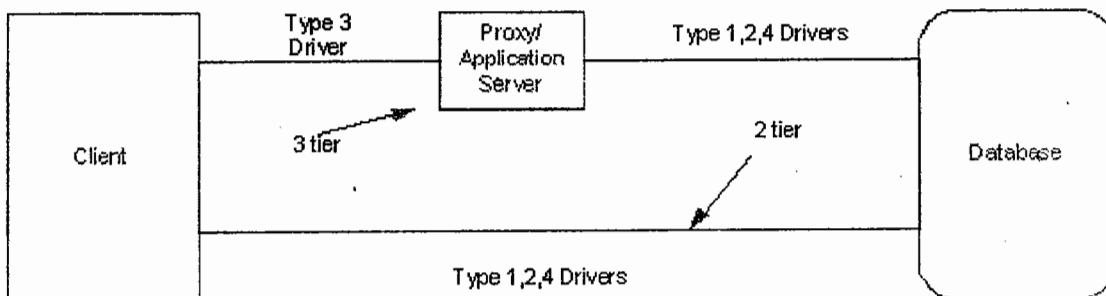
Driver is the key player in a JDBC application, it acts as a mediator between Java application and database. It implements JDBC API interfaces for a database, for example Oracle driver for oracle database, Sybase driver for Sybase database. It maps Java language to database specific language including SQL.

JDBC defines four types of drivers to work with. Depending on your requirement you can choose one among them.

Here is a brief description of each type of driver :

Type of driver	Tier	Driver mechanism	Description
1	Two	JDBC-ODBC	This driver converts JDBC calls to ODBC calls through JDBC-ODBC Bridge driver which in turn converts to database calls. Client requires ODBC libraries.
2	Two	Native API - Partly - Java driver	This driver converts JDBC calls to database specific native calls. Client requires database specific libraries.
3	Three	JDBC - Net -All Java driver	This driver passes calls to proxy server through network protocol which in turn converts to database calls and passes through database specific protocol. Client doesn't require any driver.
4	Two	Native protocol - All - Java driver	This driver directly calls database. Client doesn't require any driver.

Obviously the choice of choosing a driver depends on availability of driver and requirement. Generally all the databases support their own drivers or from third party vendors. If you don't have driver for your database, JDBC-ODBC driver is the only choice because all most all the vendors support ODBC. If you have tiered requirement ( two tier or three tier) for your application, then you can filter down your choices, for example if your application is three tiered, then you can go for Type three driver between client and proxy server shown below. If you want to connect to database from java applet, then you have to use Type four driver because it is only the driver which supports that feature. This figure shows the overall picture of drivers from tiered perspective.



This figure illustrates the drivers that can be used for two tiered and three tiered applications. For both two and three tiered applications, you can filter down easily to Type three driver but you can use Type one, two and four drivers for both tiered applications. To be more precise, for java applications( non-applet) you can use Type one, two or four driver. Here is exactly where you may make a mistake by

choosing a driver without taking performance into consideration. Let us look at that perspective in the following section.

Type 3 & 4 drivers are faster than other drivers because Type 3 gives facility for optimization techniques provided by application server such as connection pooling, caching, load balancing etc and Type 4 driver need not translate database calls to ODBC or native connectivity interface. Type 1 drivers are slow because they have to convert JDBC calls to ODBC through JDBC-ODBC Bridge driver initially and then ODBC Driver converts them into database specific calls. Type 2 drivers give average performance when compared to Type 3 & 4 drivers because the database calls have to be converted into database specific calls. Type 2 drivers give better performance than Type 1 drivers.

Finally, to improve performance

1. Use Type 4 driver for applet to database communication.
2. Use Type 2 driver for two tiered applications for communication between java client and the database that gives better performance when compared to Type1 driver
3. Use Type 1 driver if your database doesn't support a driver. This is rare situation because almost all major databases support drivers or you will get them from third party vendors.
4. Use Type 3 driver to communicate between client and proxy server ( weblogic, websphere etc) for three tiered applications that gives better performance when compared to Type 1 & 2 drivers.

#### **Class Summary**

<b>Date</b>	A thin wrapper around a millisecond value that allows JDBC to identify this as a SQL DATE.
<b>DriverManager</b>	The basic service for managing a set of JDBC drivers. <b>NOTE:</b> The DataSource interface, new in the JDBC 2.0 API, provides another way to connect to a data source.
<b>DriverPropertyInfo</b>	Driver properties for making a connection.
<b>SQLPermission</b>	The permission for which the SecurityManager will check when code that is running in an applet calls one of the setLogWriter methods.
<b>Time</b>	A thin wrapper around java.util.Date that allows JDBC to identify this as a SQL TIME value.
<b>Timestamp</b>	A thin wrapper around java.util.Date that allows the JDBC API to identify this as an SQL TIMESTAMP value.
<b>Types</b>	The class that defines the constants that are used to identify generic SQL types, called JDBC types.

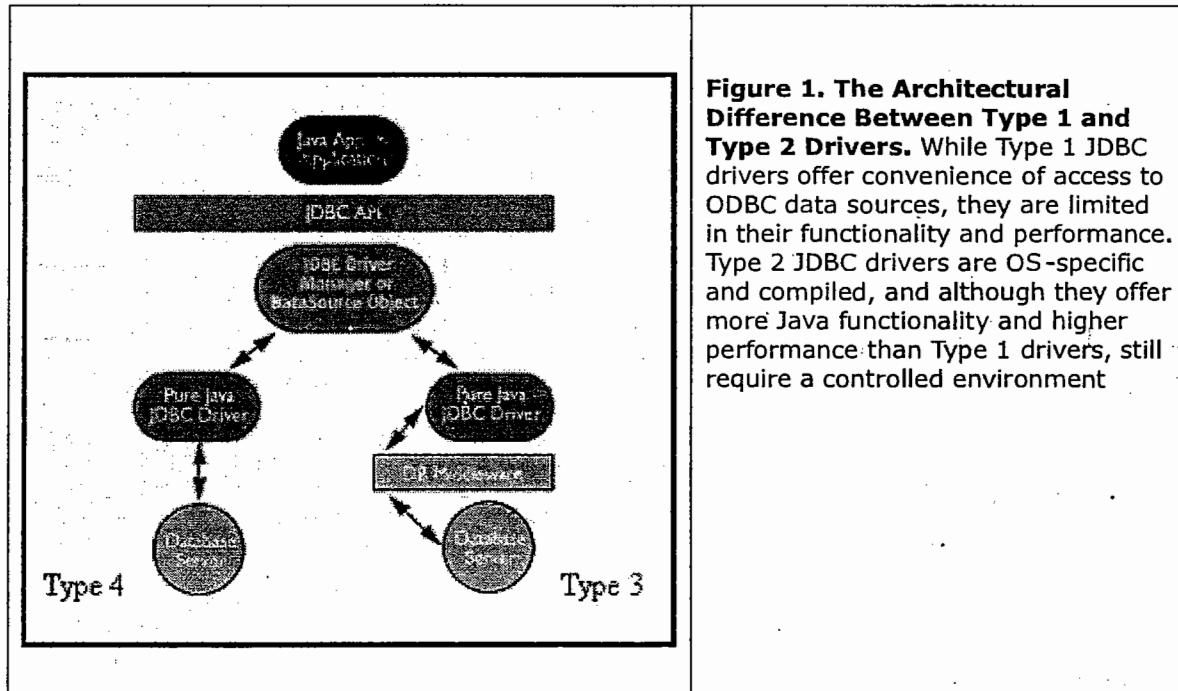
#### **Interface Summary**

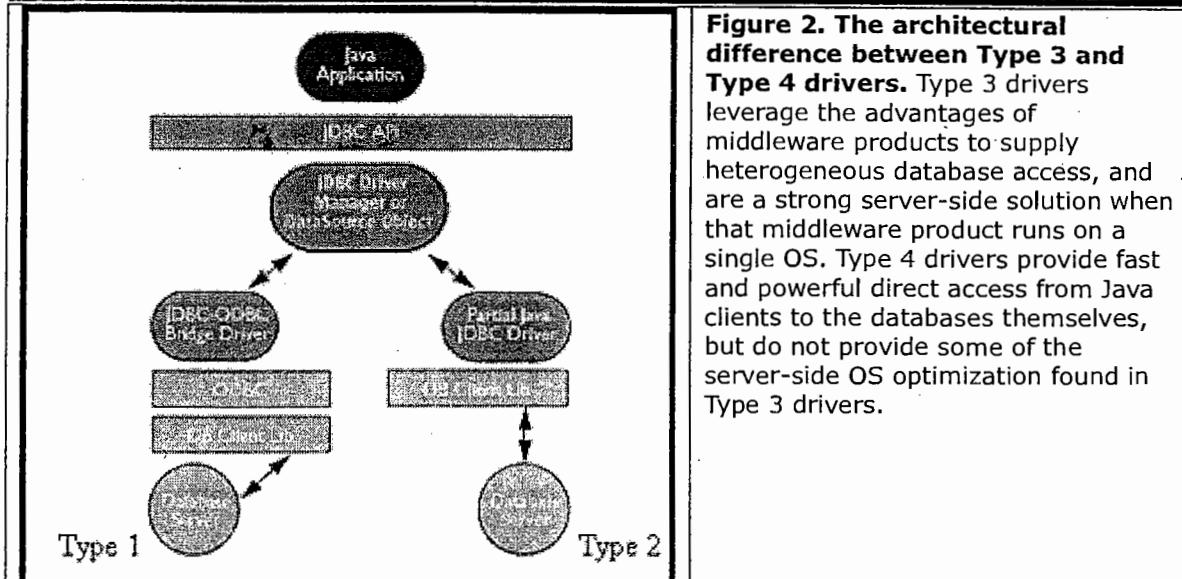
<b>Array</b>	The mapping in the Java programming language for the SQL type ARRAY.
<b>Blob</b>	The representation (mapping) in the Java™ programming language of an SQL BLOB value.
<b>CallableStatement</b>	The interface used to execute SQL stored procedures.
<b>Clob</b>	The mapping in the Java™ programming language for the SQL CLOB type.
<b>Connection</b>	A connection (session) with a specific database.
<b>DatabaseMetaData</b>	Comprehensive information about the database as a whole.
<b>Driver</b>	The interface that every driver class must implement.
<b>PreparedStatement</b>	An object that represents a precompiled SQL statement.
<b>Ref</b>	The mapping in the Java programming language of an SQL REF value, which is a reference to an SQL structured type value in the database.
<b>ResultSet</b>	A table of data representing a database result set, which is usually generated by executing a statement that queries the database.

<b><u>ResultSetMetaData</u></b>	An object that can be used to get information about the types and properties of the columns in a ResultSet object.
<b><u>SQLData</u></b>	The interface used for the custom mapping of SQL user-defined types.
<b><u>SQLInput</u></b>	An input stream that contains a stream of values representing an instance of an SQL structured or distinct type.
<b><u>SQLOutput</u></b>	The output stream for writing the attributes of a user-defined type back to the database.
<b><u>Statement</u></b>	The object used for executing a static SQL statement and obtaining the results produced by it.
<b><u>Struct</u></b>	The standard mapping in the Java programming language for an SQL structured type.

**getConnection() Methods:**

```
public static Connection getConnection(String url) throws SQLException
public static Connection getConnection(String url, String user, String pwd) throws SQLException
public static Connection getConnection(String url, Properties info) throws SQLException
```

**The Architectural Difference Between the All Drivers :**



**Figure 2. The architectural difference between Type 3 and Type 4 drivers.** Type 3 drivers leverage the advantages of middleware products to supply heterogeneous database access, and are a strong server-side solution when that middleware product runs on a single OS. Type 4 drivers provide fast and powerful direct access from Java clients to the databases themselves, but do not provide some of the server-side OS optimization found in Type 3 drivers.

### TYPE 5 JDBC Driver

In recent years, the Java ecosystem has evolved quite dramatically on a number of fronts. As a result, IT organizations are taking advantage of an array of new technologies that promise to streamline Java development cycles and reduce operating costs.

In complex enterprise computing environments, however, everything is connected and not all architectural components evolve in concert. Therefore, the introduction of advanced technologies at specific points in the architectural stack often exposes many downstream technological limitations.

As a result, businesses struggle to realize the return on investment promised by these new technologies and in some cases, even experience problems that take them a step backward rather than forward. Today, enterprise architects and developers experience this phenomenon when they task overmatched Type 4 JDBC database drivers with the ever-increasing demands of the modern Java environment.

### **The JDBC Driver: An Architectural Afterthought**

The JDBC API specification and the drivers it enables have certainly evolved over time, from the original JDBC-ODBC bridge to the native-protocol Type 4 drivers that are so prevalent today. At this point, however, that evolution is stagnant. The majority of Type 4 JDBC drivers essentially offer the same basic features and capabilities, so for many developers, JDBC access is essentially an architectural afterthought that requires little attention.

This perception grew with the increasing adoption of Object-Relational Mapping (ORM) technologies (JPA, Hibernate, and Spring, among others), or application servers such as JBoss that sit on top of JDBC. With these modern development platforms, many developers no longer program directly to JDBC.

With no access to underlying JDBC calls, developers almost never think about JDBC or what JDBC driver to use as part of determining a data access strategy. Therefore, product evaluation means simply selecting a JDBC driver based on "checklisting." If there is a free Type 4 JDBC driver available, the driver is automatically assumed to be adequate for any use case.

Beyond ORM adoption, there have been other key technological advancements at work in the Java universe. Virtualization now delivers massive scalability on an affordable growth curve, placing a much higher value than ever before on optimum performance throughout the entire application stack. The increasingly complex features of relational databases frequently involve complicated and proprietary implementations that make them all but inaccessible to most applications. These strategic technologies are quickly finding acceptance in the enterprise marketplace, directing renewed attention on JDBC components, and placing a revealing spotlight on the shortcomings on Type 4 drivers as well as the negative impact these shortcomings have on IT performance and costs.

#### Type 4 JDBC Driver: Glaring Limitations

Despite superiority over other JDBC architecture types, Type 4 drivers have failed to keep up with the evolutionary advancement of complimentary Java technologies. As a result, most Type 4 drivers come with glaring limitations in today's Java-based enterprise application environments.

**Slow or Inconsistent Performance.** The response time and data throughput performance of many Type 4 drivers is poor or inconsistent, particularly when deployed into certain runtime environments (such as different JVMs) or with modern data access models (ORMs and app servers).

**Unavailable or Inaccessible Functionality.** Enabling or tuning critical functionality with many Type 4 JDBC drivers requires access to JDBC code, which is not available to applications deployed with ORM frameworks or in app servers. New database or driver functionality may also not be available across all supported JVMs or environments.

**Poor Resource Efficiency.** Most Type 4 JDBC drivers use excessive amounts of CPU and memory resources during data access. Tuning options, if available, are inaccessible or limited. This leads to excessive usage of CPU resources and a disproportionately large memory footprint.

**Application Deployment Restrictions.** Most Type 4 JDBC drivers require multiple JAR files to support different JVMs or database versions. They also typically require the deployment of platform-dependent DLLs or shared libraries to support certain driver or database functionality, limiting what environments they can be deployed to.

**Proprietary Implementation.** Many Type 4 JDBC drivers require proprietary code for applications to leverage features such as BLOBS and CLOBS, high availability, and XA. As more and more data sources must be accessed from a single application, the amount of application code, and potential for bugs, increases significantly.

Many developers and architects find out the hard way, after a project has gone into production, that even the most common Type 4 JDBC drivers exhibit most, if not all, of these limitations. As businesses look to new technologies such as virtualization to improve performance and reduce costs, the cost of dealing with the deficiencies of Type 4 JDBC drivers will only increase.

Type 4 drivers have failed to keep up with advancements

#### Unlocking the Limitations: Taking the Next Step

There is a movement afoot to address these limitations of Type 4 JDBC drivers by proposing the next step down the evolutionary path of JDBC drivers. Such a new step has been dubbed "Type 5", where the problems caused by today's Type 4 JDBC drivers are solved but the benefits of Type 4 architecture are maintained. One proposed definition states that Type 5 JDBC drivers should provide the following:

**Unrestricted Performance.** Should be able to easily handle the performance demands of modern data-driven enterprise Java applications.

**Codeless Enhancement.** Should offer the ability to add, configure, and tune features without requiring access to or changes made to JDBC code.

**Resource Efficiency.** Should use resources such as CPU and memory as efficiently as possible in multiple environments.

**All-in-One Deployment.** Should be able to support multiple environments without requiring the deployment of multiple JAR files, or any native code libraries like DLLs.

**Streamlined Standardization.** Should not require the use of extensions to the JDBC specification no matter what functionality is required by the application.

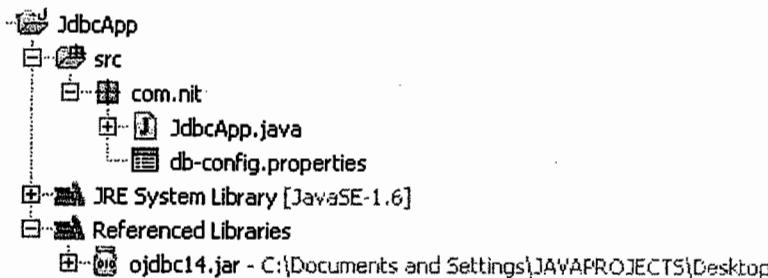
#### Type 5 JDBC Drivers: Evolution or Revolution?

In an "evolutionary" step, Type 5 JDBC drivers will build on all the positive features found with Type 4 products. Given the lack of advancement in the JDBC driver specification over the last decade, it's time that connectivity standards evolved with the myriad other advancements in the Java programming language. By delivering on the promise of unrestricted performance, codeless enhancement capabilities, optimized resource efficiency, an integrated deployment architecture, and streamlined standardization, developers and architects that use Type 5 JDBC drivers can mitigate the technical limitations imposed by outgunned Type 4 drivers that can no longer keep pace within the greater Java ecosystem.

(This is not used in Realtime )

**Solution two:-**

Store driver class name, database url, database username and database password in properties file and read this information from the properties file(Used in Real time).

**ExampleProgram****JdbcApp.java**

```

package com.nit;
import java.io.FileInputStream;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Properties;
public class JdbcApp {
    public static void main(String[] args) throws IOException, ClassNotFoundException, SQLException {
        FileInputStream fin=new FileInputStream("E:\\ADV_JAVA\\JdbcApp\\src\\com\\nit\\db-
config.properties");
        Properties p=new Properties();
        p.load(fin);
        String driverclass=p.getProperty("driverclass");
        String url=p.getProperty("url");
        String user=p.getProperty("dbuser");
        String pass=p.getProperty("dbpassword");
        Class.forName(driverclass);
        Connection con=DriverManager.getConnection(url,user,pass);
        System.out.println("connection"+ con);
    }
}
  
```

**db-config.properties**

```

driverclass=oracle.jdbc.driver.OracleDriver
url=jdbc:oracle:thin:@nit-11:1521:xe
dbuser=system
  
```

```

dbpassword=manager
  
```

**Solution three:-**

Store driver class name, database url, database username and database password in a XML file and read this information from the XML file(Used in Real time).It is a recommended

```
1 =====
2 App1(Simple Select operation)
3 =====
4 -----SelectTest.java-----
5 //SelectTest.java
6 import java.sql.*;
7 import java.util.*;
8 import java.io.*;
9 public class SelectTest
10 {
11     public static void main(String args[])throws Exception
12     {
13         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
14         Connection con=DriverManager.getConnection("jdbc:odbc:oradsn","scott","tiger");
15         // create Staement obj
16         Statement st=con.createStatement();
17         // execute the query
18         ResultSet rs=st.executeQuery("select * from student");
19         //print db table records
20         while(rs.next())
21         {
22             System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
23         }
24         //close all jdbc stream objects
25         rs.close();
26         st.close();
27         con.close();
28     }//main
29 }//class
30 =====
31 App2(Application to call SQL Aggragate Function)
32 =====
33 -----AggragateTest.java-----
34 //AggragateTest.java
35 import java.sql.*;
36 import java.util.*;
37 import java.io.*;
38 public class AggragateTest
39 {
40     public static void main(String args[])throws Exception
41     {
42         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
43         Connection con=DriverManager.getConnection("jdbc:odbc:oradsn","scott","tiger");
44         // create Staement obj
45         Statement st=con.createStatement();
46         // execute the query
47         ResultSet rs=st.executeQuery("select count(*) from emp");
48         if(rs.next())
49         {
50             System.out.println("The Number of Records are :"+rs.getInt(1));
51         }
52         //close all jdbc stream objects
53         rs.close();
54         st.close();
55         con.close();
56     }//main
57 }//class
58 =====
59 App3(Simple Delete operation taking sno value as criteria)
60 =====
```

```
61 -----DeleteTest.java-----
62 //DeleteTest.java
63 import java.sql.*;
64 import java.util.*;
65 public class DeleteTest
66 {
67     public static void main(String args[])throws Exception
68     {
69         // read input values from key board
70         Scanner sc=new Scanner(System.in);
71         System.out.println("Enter student no to delete record");
72         int no=sc.nextInt();
73
74         // create jdbc connection
75         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
76         Connection con=DriverManager.getConnection("jdbc:odbc:oradsn","scott","tiger");
77
78         //disable autocommit mode on DB s/w
79         con.setAutoCommit(false);
80
81         //create jdbc Statement obj
82         Statement st=con.createStatement();
83
84         //prepare query
85         String qry="delete from student where sno="+no;
86         System.out.println(qry);
87
88         // execute the query.....
89         int result=st.executeUpdate(qry);
90
91         //commit or rollback
92         con.commit(); // (or) con.rollback();
93
94         // process the result
95         if(result==0)
96             System.out.println("record not found to delete");
97         else
98             System.out.println(result+" no.of Record(s) found and deleted");
99
100        //close jdbc objects
101        st.close();
102        con.close();
103    }//main
104 } //class
105 =====
106 App4(Simple Insert operation)
107 =====
108 -----InsertTest.java-----
109 package com.nt;
110 //InsertTest.java (having Coding standards)
111 /* Application to insert record into DB table
112 * version :1.0
113 * author: Team-s */
114 import java.util.Scanner;
115 import java.sql.DriverManager;
116 import java.sql.Connection;
117 import java.sql.Statement;
118 import java.sql.SQLException;
119
120 public class InsertTest
```

```
121 {
122     public static void main(String args[])
123     {
124         Connection con=null;
125         Statement st=null;
126         Scanner sc=null;
127         try
128         {
129             // read record related input values from keyboard
130             int no=0;
131             String name=null,addrs=null;
132             sc=new Scanner(System.in);
133             if(sc!=null)
134             {
135                 System.out.println("Enter student no");
136                 no=sc.nextInt();
137                 System.out.println("Enter student name");
138                 name=sc.next();
139                 System.out.println("Enter student address");
140                 addrs=sc.next();
141             }//if
142             //prepare variable values as required for the SQL(insert) query
143             name=""+name+"";
144             addrs=""+addrs+"";
145
146             // create jdbc con with DB s/w
147             Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
148             con=DriverManager.getConnection("jdbc:odbc:oradsn","scott","tiger");
149
150             // create jdbc statement obj
151             if(con!=null)
152                 st=con.createStatement();
153
154             //prepare example query with direct values
155             //String qry="insert into student values(101,'raja','hyd')";
156
157             //frame the insert query with variables
158             String qry="insert into student values("+no+","+name+","+addrs+ ")";
159             System.out.println(qry);
160
161             // send and execute the query in DB s/w
162             int res=0;
163             if(st!=null)
164                 res=st.executeUpdate(qry);
165
166             //process the result
167             if(res==0)
168                 System.out.println("Record insertion Failed");
169             else
170                 System.out.println("Record inserted");
171         }//try
172         catch(ClassNotFoundException cnf) // to handle known exception
173         {
174             cnf.printStackTrace();
175         }
176         catch(SQLException se) // to handle known exception
177         {
178             se.printStackTrace();
179         }
180         catch(Exception e) // to unknown exceptions
```

```
181         {
182             e.printStackTrace();
183         }
184     finally
185     {
186         //close jdbc objects.....
187         try
188         {
189             if(st!=null)
190                 st.close();
191         }
192         catch(SQLException se)
193         {
194             se.printStackTrace();
195         }
196
197         try
198         {
199             if(con!=null)
200                 con.close();
201         }
202         catch(SQLException se1)
203         {
204             se1.printStackTrace();
205         }
206     }//finally
207 } //main
208 } //class
209
210 //>javac -d . InsertTest.java
211 //>java com.nt.InsertTest
212
213 =====
214 App5(Program to update the record based on the student number)
215 =====
216 -----UpdateTest.java-----
217 package com.nt;
218 //UpdateTest.java (with coding standards)
219 /* App to update student details
220 * version : 1.0
221 * author : Team-s*/
222
223 import java.util.Scanner;
224 import java.sql.DriverManager;
225 import java.sql.Connection;
226 import java.sql.Statement;
227 import java.sql.SQLException;
228
229 public class UpdateTest
230 {
231     public static void main(String args[])
232     {
233         Connection con=null;
234         Statement st=null;
235         Scanner sc=null;
236         try
237         {
238             // read input values from keyboard
239             sc=new Scanner(System.in);
240             int no=0;
```

```
241     String name=null;
242     String addrs=null;
243     if (sc!=null)
244     {
245         System.out.println("Enter existing student number");
246         no=sc.nextInt();
247         System.out.println("Enter new name:");
248         name=sc.next();
249         System.out.println("Enter new address:");
250         addrs=sc.next();
251     }//if
252
253     // register jdbc driver and establish the connection
254     Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
255     con=DriverManager.getConnection("jdbc:odbc:oradsn","scott","tiger");
256     //create Statement obj
257     if(con!=null)
258         st=con.createStatement();
259
260     // frame example qry with direct values
261     //String qry="update student set sname='maharaja',sadd='vizag' where sno=101";
262     String qry="update student set sname='"+name+"',sadd='"+addrs+"' where sno='"+no';
263     System.out.println(qry);
264
265     //send and execute the qry in db s/w
266     int res=0;
267     if(st!=null)
268         res=st.executeUpdate(qry);
269
270     // process results
271     if (res==0)
272     {
273         System.out.println("record not found");
274     }
275     else
276     {
277         System.out.println(res+"no.of records are updated");
278     }
279 }//try
280 catch(ClassNotFoundException cnf)//handles known exception
281 {
282     cnf.printStackTrace();
283 }
284 catch(SQLException se) //handles known exception
285 {
286     se.printStackTrace();
287 }
288 catch(Exception e) //handles unknown exceptions
289 {
290     e.printStackTrace();
291 }
292 finally
293 {
294     //close jdbc objs
295     try
296     {
297         if(st!=null)
298             st.close();
299     }
300     catch(SQLException se)
```

```
301         {
302             se.printStackTrace();
303         }
304     try
305     {
306         if(con!=null)
307             con.close();
308     }
309     catch(SQLException se)
310     {
311         se.printStackTrace();
312     }
313 } //finally
314 } //main
315 } //class
316 //>javac -d . UpdateTest.java
317 //>java com.nt.UpdateTest
318 =====
319 App6 (App to execute both select and non-select SQL Queries)
320 =====
321 -----SelectNonSelectTest.java-----
322 //SelectNonSelectTest.java (Example on execute(-) method)
323 import java.sql.*;
324 import java.util.*;
325 public class SelectNonSelectTest
326 {
327     public static void main(String args[])throws Exception
328     {
329         // read sql query (any) from keyboard
330         Scanner sc=new Scanner(System.in);
331         System.out.println("Enter the qry");
332         String qry=sc.nextLine();
333
334         // create jdbc con
335         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
336         Connection con=DriverManager.getConnection("jdbc:odbc:oradsn","scott","tiger");
337         // create jdbc statement obj
338         Statement st=con.createStatement();
339
340         //send and execute qry in db s/w
341         boolean flag=st.execute(qry);
342         // gather and process the result
343         if(flag==true) // when select sql qry is executed
344         {
345             ResultSet rs=st.getResultSet();
346             while(rs.next())
347             {
348                 System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
349             }
350             rs.close();
351         }//if
352         else // when non-select qry is executed
353         {
354             int rowcount=st.executeUpdate();
355             System.out.println("the no.of records that effected"+rowcount);
356         }
357
358         //close jdbc objs
359         st.close();
360     }
```

```
361         con.close();
362     }//main
363 }//class
364 =====
365 App7(Program to insert multiple records using PreparedStatement)
366 =====
367 -----PstInsertTest.java-----
368 //PstInsertTest.java
369 import java.sql.DriverManager;
370 import java.sql.Connection;
371 import java.sql.PreparedStatement;
372 import java.sql.SQLException;
373 import java.util.Scanner;
374
375 public class PstInsertTest
376 {
377     public static void main(String args[])
378     {
379         Connection con=null;
380         Scanner sc=null;
381         PreparedStatement ps=null;
382         try
383         {
384             //read input value from key board
385             sc=new Scanner(System.in);
386             int n=0;
387             if(sc!=null)
388             {
389                 System.out.println("Enter no.of students");
390                 n=sc.nextInt();
391             }
392             //create jdbc con object
393             Class.forName("oracle.jdbc.driver.OracleDriver");
394             con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
395
396             // create PreparedStatement obj
397             if(con!=null)
398                 ps=con.prepareStatement("insert into student values(?, ?, ?)");
399
400             // read each student details from keyboard, set them to query
401             // and execute the query
402             if(ps!=null && sc!=null)
403             {
404                 for(int i=1;i<=n;++i)
405                 {
406                     System.out.println("Enter "+i+" student details");
407                     System.out.println("Enter student no");
408                     int no=sc.nextInt();
409
410                     System.out.println("Enter student name");
411                     String name=sc.next();
412
413                     System.out.println("Enter student address");
414                     String addrs=sc.next();
415                     //set these values query place holders
416                     ps.setInt(1,no);
417                     ps.setString(2,name);
418                     ps.setString(3,addrs);
419                     //execute the query
420                     int res=ps.executeUpdate();
```

```
421             if(res==0)
422                 System.out.println(i+"student details are not inserted");
423             else
424                 System.out.println(i+"student details are inserted");
425         }//for
426     }//if
427 };//try
428 catch(ClassNotFoundException cnf) // to handle known exceptions
429 {
430     cnf.printStackTrace();
431 }
432 catch(SQLException se) // to handle known exceptions
433 {
434     se.printStackTrace();
435 }
436 catch(Exception e) // to hanlde unknown exceptions
437 {
438     e.printStackTrace();
439 }
440 finally
441 {
442     //close jdbc stream objects
443     try
444     {
445         if(ps!=null)
446             ps.close();
447     }
448     catch(SQLException se)
449     {
450         se.printStackTrace();
451     }
452 }
453     try
454     {
455         if(con!=null)
456             con.close();
457     }
458     catch(SQLException se1)
459     {
460         se1.printStackTrace();
461     }
462 };//finally
463 };//main
464 };//class
465 =====
466 App8 (Showing SQL Injection Problem)
467 =====
468 -----LoginApp.java-----
469 /* DB table in oracle
470      userlist
471      -----
472      uname vc2      pwd vc2
473      -----
474      raja          hyd
475      ramesh        hyd
476      */
477
478 import java.sql.*;
479 import java.util.*;
480
```

```
481 public class LoginApp
482 {
483     public static void main(String args[])throws Exception
484     {
485         //read inputs
486         Scanner sc=new Scanner(System.in);
487         System.out.println("Enter username:");
488         String user=sc.nextLine(); //gives raja
489         System.out.println("Enter Password:");
490         String pass=sc.nextLine(); //gives rao
491 // there is no need of converting input values..
492
493     // register jdbc driver and establish the connection
494     Class.forName("oracle.jdbc.driver.OracleDriver");
495     Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
496
497     //prepare SQL Query
498     String qry="select count(*) from userlist where uname='"+user+"'and pwd='"+pass+"'";
499     //create Jdbc Statement obj
500     Statement st=con.createStatement(qry);
501     //send and execute the Query in DB s/w
502     ResultSet rs=ps.executeQuery();
503     //process the ResultSet
504     int cnt=0;
505     if(rs.next())
506     {
507         cnt=rs.getInt(1);
508     }
509
510     if(cnt==0)
511         System.out.println("InValid Credentials");
512     else
513         System.out.println("Valid Credentials");
514
515     //close jdbc objs
516     rs.close();
517     ps.close();
518     con.close();
519 } //main
520 } //class
521 //>javac LoginApp.java
522 //>java LoginApp
523 enter user name:raja' --
524 enter password : hyd1 (wrong password)
525 output: Valid Credentials ( SQL Injection problem)
526 =====
527 App9 (Solving SQL Injection Problem)
528 =====
529 -----LoginApp.java-----
530 /* DB table in oracle
531     userlist
532     -----
533     uname vc2      pwd vc2
534     -----
535     raja          hyd
536     ramesh        hyd
537 */
538 import java.sql.*;
539 import java.util.*;
540
```

```
541 public class LoginApp
542 {
543     public static void main(String args[])throws Exception
544     {
545         //read inputs
546         Scanner sc=new Scanner(System.in);
547         System.out.println("Enter username:");
548         String user=sc.nextLine(); //gives raja
549         System.out.println("Enter Password:");
550         String pass=sc.nextLine(); //gives rao
551 // there is no need of converting input values..
552
553     // register jdbc driver and establish the connection
554     Class.forName("oracle.jdbc.driver.OracleDriver");
555     Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
556
557     //prepare SQL Query
558     String qry="select count(*) from userlist where uname=? and pwd=?";
559     //create Jdbc PreparedStatement obj
560     PreparedStatement ps=con.prepareStatement(qry);
561     // set values to Query params
562     ps.setString(1,user);
563     ps.setString(2,pass);
564     //send and execute the Query in DB s/w
565     ResultSet rs=ps.executeQuery();
566
567     //process the ResultSet
568     int cnt=0;
569     if(rs.next())
570     {
571         cnt=rs.getInt(1);
572     }
573
574     if(cnt==0)
575         System.out.println("InValid Credentials");
576     else
577         System.out.println("Valid Credentials");
578
579     //close jdbc objs
580     rs.close();
581     ps.close();
582     con.close();
583 } //main
584 } //class
585 //>javac LoginApp.java
586 //>java LoginApp
587 enter user name:raja' --
588 enter password : hyd1 (wrong password)
589 output: InValid Credentials ( No SQL Injection problem)
590 =====
591 App10 (prg on Inserting Dates and Retreving Dates)
592 =====
593 -----DateInsert.java-----
594 //DateInsert.java
595 import java.util.*;
596 import java.sql.*;
597
598 public class DateInsert
599 {
600     public static void main(String args[])throws Exception
```

```
601  {
602  // read input values from keyboard
603  Scanner sc=new Scanner(System.in);
604  System.out.println("Enter person no");
605  int no=sc.nextInt();
606
607  System.out.println("Enter person name:");
608  String name=sc.next();
609
610  System.out.println("Enter DOB(dd-MM-yy)");
611  String sdob=sc.next();
612
613  System.out.println("Enter DOJ(yyyy-MM-dd)");
614  String sdoj=sc.next();
615
616 //convert String Date values to java.sql.Date class objs
617 // for DOB
618  SimpleDateFormat sdf1=new SimpleDateFormat("dd-MM-yy");
619  java.util.Date udob=sdf1.parse(sdob);
620
621  java.sql.Date sqdob=new java.sql.Date(udob.getTime());
622
623 //for DOJ (yyyy-MM-dd)
624  java.sql.Date sqdobj=java.sql.Date.valueOf(sdoj);
625
626 //create jdbc con object
627  Class.forName("oracle.jdbc.driver.OracleDriver");
628  Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
629
630 //create PreparedStatement obj pointing to insert query
631  PreparedStatement ps=con.prepareStatement("insert into person_tab values(?, ?, ?, ?)");
632
633 //set values to the parameters of query
634  ps.setInt(1,no);
635  ps.setString(2,name);
636  ps.setDate(3,sqdobj);
637  ps.setDate(4,sqdobj);
638
639 //execute the query
640  int res=ps.executeUpdate();
641
642 //process the result
643 if(res==0)
644     System.out.println("record not inserted");
645 else
646     System.out.println("record inserted");
647
648 //close jdbc objs
649  ps.close();
650  con.close();
651 } //main
652 } //class
653 //>javac DateInsert.java
654 //>java DateInsert
655 -----DateRetrieve.java-----
656 //DateRetrieve.java
657 import java.sql.*;
658 import java.util.*;
659 import java.text.*;
660
```

```
661 public class DateRetrieve
662 {
663     public static void main(String args[])throws Exception
664     {
665         // create jdbc con obj
666         Class.forName("oracle.jdbc.driver.OracleDriver");
667         Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
668
669         //create jdbc Statement object
670         Statement st=con.createStatement();
671
672         // execute the query
673         ResultSet rs=st.executeQuery("select * from person_tab");
674
675         //process the ResultSet
676         while(rs.next())
677         {
678             int no=rs.getInt(1);
679             String name=rs.getString(2);
680             java.sql.Date sqdob=rs.getDate(3);
681             java.sql.Date sqdobj=rs.getDate(4);
682
683             //convert java.sql.Date class objs to java.util.Date class objs
684             java.util.Date udob=(java.util.Date)sqdob;
685             java.util.Date udoj=(java.util.Date)sqdobj;
686             //convert java.util.Date class objs to String date values
687             SimpleDateFormat sdf1=new SimpleDateFormat("MMM-yy-dd");
688             String sdob=sdf1.format(udob);
689
690             SimpleDateFormat sdf2=new SimpleDateFormat("yyyy-dd-MMM");
691             String sdoj=sdf2.format(udoj);
692
693             System.out.println(no+" "+name+" "+sdob+" "+sdoj);
694         }//while
695
696         //close jdbc objs
697         rs.close();
698         st.close();
699         con.close();
700
701     }//main
702 } //class
703 =====
704 App11 (working with Large objs)
705 =====
706 -----PhotoInsert.java-----
707 /* create table empall(eno number(4),ename varchar2(20),esalary number(7,2), ephoto blob);*/
708 import java.sql.*;
709 import java.io.*;
710 class PhotoInsert
711 {
712     public static void main(String args[])throws Exception
713     {
714         //register jdbc driver and establish the connection
715         Class.forName("oracle.jdbc.driver.OracleDriver");
716         Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
717         //create PreparedStatement obj
718         PreparedStatement ps=con.prepareStatement("insert into empall values(?, ?, ?, ?)");
719         ps.setInt(1,234);
720         ps.setString(2,"raja");
```

```
721     ps.setFloat(3,5000);
722
723     File f=new File("tips.gif");
724     FileInputStream fis=new FileInputStream(f);
725     ps.setBinaryStream(4,fis,(int)f.length());
726
727     //execute the query
728     ps.executeUpdate();
729     System.out.println("Photo Inserted");
730     //close the jdbc obj
731     fis.close();
732     ps.close();
733     con.close();
734 } //main
735 } //class
736 -----PhotoRetrieve.java-----
737 import java.sql.*;
738 import java.io.*;
739 public class PhotoRetrieve
740 {
741     public static void main(String args[])throws Exception
742     {
743         //register jdbc driver and estblish the connection
744         Class.forName("oracle.jdbc.driver.OracleDriver");
745         Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
746
747         //create JDBC Statement obj
748         Statement st=con.createStatement();
749         // execute the SQL Query
750         ResultSet rs=st.executeQuery("select * from empall");
751
752         //process the ResultSet obj
753         if(rs.next())
754         {
755             InputStream in=rs.getBinaryStream("ephoto");
756
757             FileOutputStream fos=new FileOutputStream("newpict.gif");
758
759             //Buffering based logic to complete photo retriving
760             int bytesRead=0;
761             byte [] buffer=new byte[4096];
762
763             while((bytesRead=in.read(buffer))!=-1)
764             {
765                 fos.write(buffer,0,bytesRead);
766             }
767
768             System.out.println("photo is stored in newpict.gif");
769             //close jdbc objs
770             fos.close();
771             in.close();
772             rs.close();
773             st.close();
774             con.close();
775         } //if
776     } //main
777 } //class
778 =====
779 App12(Program to call pl/sql procedure using CallableStatement)
780 =====Procedure in oracle=====
```

```
781 create or replace procedure first_pro1(x in number,y out number) as
782 begin
783 y:=x*x;
784 end;
785 -----
786 import java.sql.*;
787 class UsingProcedure
788 {
789 public static void main( String args[ ] ) throws Exception
790 {
791     //register jdbc driver
792     Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
793     //establish the connection
794     Connection con = DriverManager.getConnection("jdbc:odbc:oradsn","scott","tiger");
795     // create CallableStatement obj
796     CallableStatement cst = con.prepareCall( "{CALL first_pro1(?,?)}" );
797     //register out param with jdbc types
798     cst.registerOutParameter( 2, Types.INTEGER ) ;
799     //set value to IN param
800     cst.setInt( 1, 20);
801     // call pl/sql procedure
802     cst.execute();
803     // gather results from out params
804     int res = cst.getInt(2) ;
805
806     System.out.println( " Result is: " + res );
807     //close jdbc objs
808     cs.close();
809     con.close();
810 }
811 }
812 =====
813 App13 (App on CallableStatement obj)
814 =====
815 -----
816 /* create or replace procedure get_EmpDetails(no in number,name out varchar2, salary out number)as
817 begin
818     select ename,sal into name,salary from emp where empno=no;
819 end;
820 */
821 //CsTest1.java
822 import java.sql.*;
823 import java.util.*;
824 public class CsTest1
825 {
826     public static void main(String[] args) throws Exception
827     {
828         // read employee no from keyboard
829         Scanner sc=new Scanner(System.in);
830         System.out.println("Enter employee no:");
831         int no=sc.nextInt();
832
833         // create jdbc con object
834         Class.forName("oracle.jdbc.driver.OracleDriver");
835         Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
836         //prepare qry calling pl/sql procedure
837         String qry="{ call get_EmpDetails(?, ?, ?)}";
838         //create CallableStatement obj
839         CallableStatement cs=con.prepareCall(qry);
840         // register outparameters with jdbc types
```

```
841     cs.registerOutParameter(2,Types.VARCHAR);
842     cs.registerOutParameter(3,Types.INTEGER);
843     // set input value to IN parameter
844     cs.setInt(1,no);
845     //execute pl/sql procedure
846     cs.execute();
847     //gather results from out parameters
848     System.out.println(no+" employee salary is"+cs.getInt(3));
849     System.out.println(no+" employee name is"+cs.getString(2));
850
851     //close jdbc objects
852     cs.close();
853     con.close();
854
855 } //main
856 } //class
857
858 //>javac CsTest1.java
859 //>java CsTest1
860
861 =====
862 App14 (Application on CallableStatement obj)
863 =====
864 -----CursorsTest.java-----
865 //CursorCSTest.java
866
867 /* create or replace procedure fetch_AllEmpDetails(cond in varchar,mycur out sys_refcursor)as
868 begin
869 open mycur for
870   select * from emp where ename like cond;
871 end;
872 */
873 //CursorCsTest.java
874
875 import java.sql.*;
876 import java.util.*;
877 import oracle.jdbc.*; // for OracleTypes class ( this pkg is available in ojdbc14.jar file)
878
879 public class CursorCsTest
880 {
881   public static void main( String args[ ] ) throws Exception
882   {
883
884     // read cond character from key board
885     Scanner sc=new Scanner(System.in);
886     System.out.println("Enter characters (first letters of emp name)");
887     String cond=sc.next();
888     cond=cond+"%";
889
890     //create jdbc con obj
891     Class.forName("oracle.jdbc.driver.OracleDriver");
892     Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
893
894     CallableStatement cs = con.prepareCall("{call fetch_AllEmpDetails(?,?) }");
895
896     // register out parameter with jdbc types (Oracle corp supplied jdbc types)
897     cs.registerOutParameter(2,OracleTypes.CURSOR);
898
899     // set value to IN parameter
900     cs.setString(1,cond);
```

```
901      // execute pl/sql procudre
902      cs.execute();
903
904      // gather result from out paramter
905      ResultSet rs=(ResultSet)cs.getObject(2);
906
907      //display result
908      while(rs.next())
909      {
910          System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
911      }
912
913      // close jdbc objects
914      rs.close();
915      cs.close();
916      con.close();
917
918 =====
919 App15 (calls pl/sql fuction that contains cursor based select query and non-select query)
920 =====
921 -----CsFxTest1.java-----
922 //CsFxTest1.java (calls the above given my_fx pl/sql function)
923 /* create or replace function my_fx (no in number, cnt out number)return sys_refcursor
924 as
925   mycur sys_refcursor;
926 begin
927   open mycur for
928     select * from student where sno=no;
929
930   delete from student where sno=no;
931   cnt:=SQL%ROWCOUNT;
932   return mycur;
933 end;
934
935 */
936 import java.sql.*;
937 import java.util.*;
938 import oracle.jdbc.*;
939
940 public class CsFxTest1
941 {
942     public static void main(String args[])throws Exception
943     {
944         // read input values from keyboard
945         Scanner sc=new Scanner(System.in);
946         System.out.println("Enter student no");
947         int no=sc.nextInt();
948
949         // create jdbc con object
950         Class.forName("oracle.jdbc.driver.OracleDriver");
951         Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
952
953         //create jdbc CallableStatement obj
954         CallableStatement cs=con.prepareCall("{?= call my_fx(?,?)}");
955
956         // register out,return params with jdbc types
957         cs.registerOutParameter(1,OracleTypes.CURSOR); //return param
958         cs.registerOutParameter(3,Types.INTEGER);//out param
959
960         //set values to IN parameters
```

```
961         cs.setInt(2,no);
962
963         // call and execute pl/sql function
964         cs.execute();
965
966         // gather results from return,out params
967         ResultSet rs=(ResultSet)cs.getObject(1); // from return param(cursor)
968         //process return value result
969         while(rs.next())
970         {
971             System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
972         }
973
974         int cnt=cs.getInt(3); //out param result
975         if(cnt==0)
976             System.out.println("Record not deleted");
977         else
978             System.out.println("Record deleted");
979         //close jdbc objs
980         rs.close();
981         cs.close();
982         con.close();
983     }//main
984 } //class
985 //>javac CsFxTest1.java
986 //>java CsFxTest1
987 =====
988 App16 (AWT+CallableStatement obj)
989 =====
990 -----AuthApp.java-----
991 /* create or replace procedure auth_pro(user in varchar2, pass in varchar2, result out varchar2)
992    as
993        cnt number;
994    begin
995        select count(*) into cnt from userlist where uname=user and pwd=pass;
996        if(cnt<>0)then
997            result:='Valid Credentials';
998        else
999            result:='InValid Credentials';
1000        end if;
1001    end;
1002 */
1003 /* DB table in oracle
1004    userlist
1005    -----
1006    uname (vc2)      pwd (vc2)
1007    -----
1008    raja          hyd
1009    ramesh        hyd
1010   */
1011 //GUIApp.java
1012 import javax.swing.*;
1013 import java.awt.*;
1014 import java.awt.event.*;
1015 import java.sql.*;
1016
1017 public class GUIApp extends JFrame implements ActionListener
1018 {
1019     JLabel luname,lpwd,lres;
1020     JTextField tf1;
```

```
1021     JPasswordField tf2;
1022     JButton b1;
1023     CallableStatement cs;
1024
1025     public GUIApp()
1026     {
1027         try
1028         {
1029             setSize(400,400);
1030             setLayout(new FlowLayout());
1031             Container cp=getContentPane();
1032             luname=new JLabel("Username:");
1033             cp.add(luname);
1034             tf1=new JTextField(10);
1035             cp.add(tf1);
1036             lpwd=new JLabel("Password:");
1037             cp.add(lpwd);
1038             tf2=new JPasswordField(10);
1039             cp.add(tf2);
1040             tf2.setEchoChar('*');
1041             b1=new JButton("Login");
1042             b1.addActionListener(this);
1043             cp.add(b1);
1044             lres=new JLabel("");
1045             cp.add(lres);
1046             pack();
1047             show();
1048             setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
1049             // register jdbc driver and establish the connection
1050             Class.forName("oracle.jdbc.driver.OracleDriver");
1051             Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
1052             // create CallableStatement obj pointing to pl/sql procedure
1053             cs=con.prepareCall("{ call Auth_pro(?, ?, ?)}");
1054             //register OUT params with jdbc types
1055             cs.registerOutParameter(3,Types.VARCHAR);
1056         }//try
1057         catch(Exception e)
1058         {
1059             e.printStackTrace();
1060         }//constructor
1061
1062         public void actionPerformed(ActionEvent ae)
1063         {
1064             try
1065             {
1066                 //read text box values and set them as IN param values
1067                 cs.setString(1,tf1.getText());
1068                 cs.setString(2,tf2.getText());
1069                 // call pl/sql procedure
1070                 cs.execute();
1071                 // gather results and display results
1072                 lres.setText(cs.getString(3));
1073             }//try
1074             catch(Exception e)
1075             {
1076                 e.printStackTrace();
1077             }
1078         }//method
1079
1080         public static void main(String args[])
1081         {
1082             new GUIApp();
1083         }
1084     }
1085 }
```

```
1081 }//class
1082 //>javac GUIApp.java
1083 //>java GUIApp
1084
1085 =====
1086 App17(Using AWT+JDBC)(program using all three Statement Objects)
1087 =====Procedure in oracle=====
1088 create or replace procedure FIND_PASS_FAIL(m1 in number,m2 in number,m3 in number,res out varchar) as
1089 begin
1090     if(m1<35 or m2<35 or m3<35)then
1091         res:='fail';
1092     else
1093         res:='pass';
1094     end if;
1095 end;
1096 -----AllStmtsTest.java-----
1097 import java.sql.*;
1098 import java.awt.*;
1099 import java.awt.event.*;
1100
1101 public class AllStmtsTest extends Frame implements ActionListener
1102 {
1103     Label lno,lname,lm1,lm2,lm3,lres;
1104     Choice tno;
1105     TextField tname,tm1,tm2,tm3,tres;
1106     Button bdetails, bresult;
1107     Connection con;
1108     Statement st;
1109     PreparedStatement ps;
1110     CallableStatement cs;
1111     ResultSet rs1, rs2;
1112
1113     AllStmtsTest()
1114     {
1115         System.out.println("Constructor");
1116         setLayout(new FlowLayout());
1117         setSize(500,400);
1118         setBackground(Color.lightGray);
1119
1120         lno=new Label("Student Id");
1121         add(lno);
1122
1123         tno=new Choice();
1124         add(tno);
1125
1126         bdetails=new Button("Details");
1127         bdetails.addActionListener(this);
1128         add(bdetails);
1129
1130         lname=new Label("Student Name");
1131         add(lname);
1132
1133         tname=new TextField(20);
1134         add(tname);
1135
1136         lm1=new Label("Marks 1");
1137         add(lm1);
1138
1139         tm1=new TextField(20);
1140         add(tm1);
```

```
1141
1142     lm2=new Label("Marks 2");
1143     add(lm2);
1144
1145     tm2=new TextField(20);
1146     add(tm2);
1147
1148     lm3=new Label("Marks 3");
1149     add(lm3);
1150
1151     tm3=new TextField(20);
1152     add(tm3);
1153
1154     bresult=new Button("Result");
1155     bresult.addActionListener(this);
1156     add(bresult);
1157
1158     lres=new Label("Result is ");
1159     add(lres);
1160     tres=new TextField(20);
1161     add(tres);
1162     myinit();
1163
1164     setVisible(true);
1165
1166     addWindowListener(new WindowAdapter(){
1167         public void windowClosing(WindowEvent we)
1168         {
1169             System.exit(0);
1170         }
1171     });
1172
1173 } //----- CONSTRUCTOR -----
```

---

```
1175 void myinit()
1176 {
1177     System.out.println("myinit () method");
1178     try
1179     { //register jdbc driver
1180         Class.forName("oracle.jdbc.driver.OracleDriver");
1181         //establish the connection
1182         con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
1183
1184         //logic to perform load on startup operation
1185         st=con.createStatement();
1186         String query="select sno from All_student";
1187         rs1=st.executeQuery(query);
1188         while(rs1.next())
1189         {
1190             tno.addItem(rs1.getString("sno"));
1191         }
1192         rs1.close();
1193         st.close();
1194     // create PreparedStatement obj
1195     query="select * from All_student where sno=?";
1196     ps=con.prepareStatement(query);
1197     //create CallableStatement obj
1198     query="{call FIND_PASS_FAIL(?, ?, ?, ?)}";
1199     cs=con.prepareCall(query);
1200     cs.registerOutParameter(4,Types.VARCHAR);
```

```
1201
1202
1203     }---- TRY -----
1204     catch(SQLException se)
1205     {
1206         se.printStackTrace();
1207     }
1208 }---- INIT()
1209
1210 public void actionPerformed(ActionEvent ae)
1211 {
1212     try
1213     {
1214         if(ae.getSource() == bdetails)
1215         {
1216             System.out.println("details button is clicked");
1217             int no=Integer.parseInt(tno.getSelectedItem());
1218             ps.setInt(1,no);
1219             rs2=ps.executeQuery();
1220             while(rs2.next())
1221             {
1222                 tname.setText(rs2.getString("sname"));
1223                 tm1.setText(rs2.getString("m1"));
1224                 tm2.setText(rs2.getString("m2"));
1225                 tm3.setText(rs2.getString("m3"));
1226             }
1227             rs2.close();
1228         } //if
1229         else
1230         {
1231             System.out.println("result button is clicked");
1232             cs.setInt(1,Integer.parseInt(tm1.getText()));
1233             cs.setInt(2,Integer.parseInt(tm2.getText()));
1234             cs.setInt(3,Integer.parseInt(tm3.getText()));
1235             cs.execute(); //calls pl/sql procedure
1236             tres.setText(cs.getString(4));
1237         }
1238     }---- TRY -----
1239     catch(SQLException se)
1240     {
1241         se.printStackTrace();
1242     }
1243 }---- ACTION PERFORMED -----
1244
1245 public static void main(String[] args)
1246 {
1247     System.out.println("Main ()");
1248     new AllStmtsTest();
1249 }
1250 }
1251 =====
1252 App18( on Scrollable ResultSet obj)
1253 =====
1254 -----ScrollTest.java-----
1255 //ScrollableTest.java
1256 import java.sql.*;
1257
1258 public class ScrollableTest
1259 {
1260
```

```
1261 public static void main(String args[])throws Exception
1262 {
1263     //load jdbc driver class,create jdbc con obj
1264     Class.forName("oracle.jdbc.driver.OracleDriver");
1265     Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
1266     // create Jdbc Statement obj with type,mode values
1267     Statement st=con.createStatement	ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
1268     // create Scrollable ResultSet obj
1269     ResultSet rs=st.executeQuery("select * from student");
1270     // display records (top-bottom)
1271     System.out.println("Top-to bottom");
1272     while(rs.next())
1273     {
1274         System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
1275     }//while
1276
1277     // display records (bottom-top)
1278     System.out.println("Bottom-Top");
1279     rs.afterLast();
1280     while(rs.previous())
1281     {
1282         System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
1283     }//while
1284
1285     //display records randomly
1286     rs.first(); //gives First record
1287     System.out.println(rs.getRow()+"-->"+rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
1288
1289     rs.last(); //gives Last record
1290     System.out.println(rs.getRow()+"-->"+rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
1291     rs.relative(-2); //gives Last but 3
1292     System.out.println(rs.getRow()+"-->"+rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
1293     rs.absolute(4); //gives 4th record
1294     System.out.println(rs.getRow()+"-->"+rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
1295     rs.absolute(-2); //gives last but 1 record
1296     System.out.println(rs.getRow()+"-->"+rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
1297     rs.relative(1); //gives last record
1298     System.out.println(rs.getRow()+"-->"+rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
1299     //close jdbc objs
1300     rs.close();
1301     ps.close();
1302     con.close();
1303 }
1304 }
1305 =====
1306 App19(Using Scrollable ResultSet Object)
1307 =====
1308 -----ScrollFrame.java-----
1309 /*perform first,last,previous and next navigation buttons using scrollable ResultSet*/
1310
1311 import java.sql.*;
1312 import java.awt.*;
1313 import java.awt.event.*;
1314
1315 public class ScrollFrame extends Frame implements ActionListener
1316 {
1317     TextField tsno, tsna, tsadd;
1318     Label lno, lna, ladd;
1319     Button bfist, blast, bnext, bprevious;
1320     Connection con=null;
```

```
1321     Statement st=null;
1322     ResultSet rs=null;
1323     ScrollFrame()
1324     {
1325         System.out.println("constructor");
1326         setLayout(new FlowLayout());
1327         setBackground(Color.green);
1328         setSize(400,200);
1329
1330         //add comps to Frame window
1331         lno=new Label("SNO");
1332         add(lno);
1333
1334         tsno = new TextField(20);
1335         add(tsno);
1336
1337         lna=new Label("NAME");
1338         add(lna);
1339
1340         tsna = new TextField(20);
1341         add(tsna);
1342         ladd=new Label("ADDRESS");
1343         add(ladd);
1344
1345         tsadd = new TextField(20);
1346         add(tsadd);
1347
1348         bfirst = new Button("first");
1349         add(bfirst);
1350         bfirst.addActionListener(this);
1351
1352
1353         blast =new Button("last");
1354         add(blast);
1355         blast.addActionListener(this);
1356
1357
1358         bprevious = new Button("previous");
1359         add(bprevious);
1360         bprevious.addActionListener(this);
1361
1362         bnext = new Button("next");
1363         add(bnext);
1364         bnext.addActionListener(this);
1365         setVisible(true);
1366
1367         addWindowListener(new WindowAdapter(){
1368             public void windowClosing(WindowEvent we)
1369             {
1370                 System.exit(0);
1371
1372             }
1373         });
1374
1375         makeConnection();
1376     }
1377
1378     void makeConnection()
1379     {
1380         try
```

```
1381  {
1382      System.out.println("makeConnection()");
1383      //register jdbc driver
1384      Class.forName("oracle.jdbc.driver.OracleDriver");
1385      //estblish the connection
1386      con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
1387      //create Statement obj
1388      st = con.createStatement	ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
1389      // create Scrollable ResultSet
1390      rs = st.executeQuery("select * from student");
1391      System.out.println("scrollble resultset object is ready");
1392  }
1393  catch(Exception ce)
1394  {
1395      ce.printStackTrace();
1396  }
1397 }//make connection
1398
1399 public void actionPerformed(ActionEvent ae)
1400 {
1401     System.out.println("actionPerformed()");
1402     boolean found=false;
1403     try
1404     {
1405         System.out.println(ae.getActionCommand());
1406         if(ae.getSource()==bfirst) //when first btn is clicked
1407         {
1408             rs.first();
1409             found=true;
1410         }
1411         else if(ae.getSource()==blast) //when last btn is clicked
1412         {
1413             rs.last();
1414             found=true;
1415         }
1416         else if(ae.getSource()==bnext) //when last btn is clicked
1417         {
1418             if(!rs.isLast())
1419             {
1420                 rs.next();
1421                 found=true;
1422             }
1423         }
1424         else if(ae.getSource()==bprevious) //when previous btn is clicked
1425         {
1426             if(!rs.isFirst())
1427             {
1428                 rs.previous();
1429                 found=true;
1430             }
1431         }
1432
1433         if(found)
1434         {
1435             tsno.setText(rs.getString(1));
1436             tsna.setText(rs.getString(2));
1437             tsadd.setText(rs.getString(3));
1438         }
1439     }catch(Exception e)
```

```
1441         {
1442             e.printStackTrace();
1443         }
1444     }//actionPerformed
1445
1446     public static void main(String args[])
1447     {
1448         System.out.println("main method");
1449         new ScrollFrame();
1450     }
1451 }
1452 =====
1453 App20(Program on SensitiveResultSet Object)
1454 =====
1455 -----SensitiveTest.java-----
1456 // prg to read data from DB table
1457 import java.sql.*;
1458
1459 public class SensitiveTest
1460 {
1461     public static void main(String args[])throws Exception
1462     {
1463         //register jdbc driver and establish the connection
1464         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
1465         Connection con=DriverManager.getConnection("jdbc:odbc:oradsn","scott", "tiger");
1466         //create Scrollable and Sensitive ResultSet obj
1467         Statement st=con.createStatement	ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
1468         ResultSet rs=st.executeQuery("select * from student");
1469
1470         //logic to demonstrate sensitive behaviour
1471         int cnt=1;
1472         while(rs.next())
1473         {
1474             System.out.println(rs.getString(1)+" "+rs.getString(2)+" "+rs.getString(3));
1475
1476             if(cnt==2)
1477                 Thread.sleep(40000); //modify records from SQL prompt
1478
1479             cnt++;
1480         }//while
1481     //close jdbc objs
1482         rs.close();
1483         st.close();
1484         con.close();
1485     }//main
1486 }//class
1487 =====
1488 App21( prg on Updatable ResultSet)
1489 =====
1490 ----- UpdatableTest.java-----
1491 import java.sql.*;
1492 public class UpdatableTest
1493 {
1494     public static void main(String args[])throws Exception
1495     {
1496         //register jdbc driver and establish the connection
1497         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
1498         Connection con=DriverManager.getConnection("jdbc:odbc:oradsn","root", "root");
1499
1500         // create jdbc Statement obj and create UpdatableResultSet obj
```

```
1501 Statement st=con.createStatement	ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
1502 ResultSet rs=st.executeQuery("select * from student");
1503
1504     /*System.out.println("updates the 4th record");
1505      rs.absolute(4);
1506      rs.updateString(2,"babu");
1507      rs.updateRow();*/
1508
1509
1510     System.out.println("Inserting new row");
1511     rs.moveToInsertRow(); //creates an empty record in ResultSet obj
1512     rs.updateInt(1,131);
1513     rs.updateString(2,"keyboard");
1514     rs.updateInt(3,57);
1515     rs.insertRow();
1516
1517
1518 /* System.out.println("deleting 2 th row");
1519    rs.absolute(2);
1520    rs.deleteRow(); */
1521
1522     rs.close();
1523     st.close();
1524     con.close();
1525
1526 } //main
1527 } //class
1528 =====
1529 App22(To interact with Mysql DB s/w)
1530 =====
1531 -----MySqlSelectTest.java-----
1532 import java.sql.*;
1533 import java.util.*;
1534 import java.io.*;
1535 public class MySqlSelectTest
1536 {
1537     public static void main(String args[])throws Exception
1538     {
1539         //register jdbc driver and establish the connection
1540         Class.forName("org.gjt.mm.mysql.Driver");
1541         Connection con=DriverManager.getConnection("jdbc:mysql:///mydb1","root","root");
1542         // create Staement obj
1543         Statement st=con.createStatement();
1544         // execute the query
1545         ResultSet rs=st.executeQuery("select count(*) from product");
1546         while(rs.next())
1547         {
1548             System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
1549         }
1550         //close all jdbc stream objects
1551         rs.close();
1552         st.close();
1553         con.close();
1554     } //main
1555 } //class
1556 =====
1557 App23)prg to read data from Excel Sheet and to insert into Oracle DB table)
1558 =====
1559 ----- ExcelToOracle.java-----
1560 import java.sql.*;
```

```
1561 public class ExcelToOracle
1562 {
1563     public static void main(String args[])throws Exception
1564     {
1565         Connection excelcon=null,oracon=null;
1566         Statement st=null;
1567         PreparedStatement ps=null;
1568         ResultSet rs=null;
1569
1570         //register jdbc drivers
1571         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
1572         Class.forName("oracle.jdbc.driver.OracleDriver");
1573         System.out.println("Drivers Loaded");
1574
1575         //establish the connections
1576         excelcon=DriverManager.getConnection("jdbc:odbc:xlssdsn");
1577         oracon=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
1578         System.out.println("connections established");
1579
1580         //create Statement objs
1581         st=excelcon.createStatement();
1582         ps=oracon.prepareStatement("insert into student values(?, ?, ?)");
1583         System.out.println("statement prepared");
1584
1585         //get records from excel sheet
1586         rs=st.executeQuery("select * from [student$]");
1587
1588         while(rs.next())
1589         {
1590             //read each record from excel sheet
1591             int no=rs.getInt(1);
1592             String na=rs.getString(2);
1593             String add=rs.getString(3);
1594
1595             //insert each record into oracle db table
1596             ps.setInt(1,no);
1597             ps.setString(2,na);
1598             ps.setString(3,add);
1599
1600             ps.executeUpdate();
1601         }//while
1602
1603         System.out.println("Data has been copied");
1604         //close jdbc objs
1605         rs.close();
1606         st.close();
1607         ps.close();
1608         oracon.close();
1609         excelcon.close();
1610
1611     }//main
1612 } //class
1613
1614 =====
1615 App24)Program to read MetaData of the Database)
1616 =====
1617 -----DBcap.java-----
1618 //DatabaseMetaData program
1619 import java.io.*;
1620 import java.sql.*;
```

```
1621
1622 public class DBcap
1623 {
1624     public static void main(String args[])throws Exception
1625     {
1626         //register jdbc driver and establish the connection
1627         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
1628         Connection con=DriverManager.getConnection("jdbc:odbc:oradsn","scott","tiger");
1629
1630         DatabaseMetaData dbmd =con.getMetaData();
1631
1632         System.out.println("class that implements DatabaseMetaData(i) is"+dbmd.getClass().getName());
1633
1634         System.out.println("database name "+dbmd.getDatabaseProductName());
1635         System.out.println("database version "+dbmd.getDatabaseProductVersion());
1636         System.out.println("jdbc driver version "+dbmd.getDriverVersion());
1637         System.out.println("sql key words = "+dbmd.getSQLKeywords());
1638         System.out.println("numeric functions "+dbmd.getNumericFunctions());
1639         System.out.println("String Functions "+dbmd.getStringFunctions());
1640         System.out.println("System Functions"+dbmd.getSystemFunctions());
1641         System.out.println("Search String Escape"+dbmd.getSearchStringEscape());
1642         System.out.println(" supportsStoredProcedures = "+dbmd.supportsStoredProcedures());
1643         System.out.println(" getMaxRowSize = "+dbmd.getMaxRowSize());
1644         System.out.println(" getMaxStatementLength = "+dbmd.getMaxStatementLength());
1645         System.out.println("get Max tables in a select query="+dbmd.getMaxTablesInSelect());
1646         System.out.println("get Max Length of Table Name="+dbmd.getMaxTableNameLength());
1647         System.out.println("jdbc api version is"+dbmd.getJDBCMajorVersion()+" . "+dbmd.getJDBCMinorVersion());
1648     }
1649 }
1650 =====
1651 App25 (Application on ResultSetMetaData to print table data along with col names)
1652 =====
1653 -----RsmSelectTest.java-----
1654 import java.sql.*;
1655 import java.util.*;
1656 import java.io.*;
1657 public class RsmSelectTest
1658 {
1659     public static void main(String args[])throws Exception
1660     {
1661         Class.forName("oracle.jdbc.driver.OracleDriver");
1662         Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger")
1663         // create Staement obj
1664         Statement st=con.createStatement();
1665         // execute the query
1666         ResultSet rs=st.executeQuery("select * from student");
1667         //get ResultSetMetaData obj
1668         ResultSetMetaData rsmd=rs.getMetaData();
1669         // get col count
1670         int cnt=rsmd.getColumnCount();
1671         //print col names
1672         for(int i=0;i<=cnt;++i)
1673         {
1674             System.out.print(rsmd.getColumnLabel(i)+"\t\t\t");
1675         }//for
1676         System.out.println();
1677         //print col vlaues
1678         while(rs.next())
1679         {
1680             for(int i=0;i<cnt;++i)
```

```
1681         {
1682             System.out.print(rs.getString(i)+"\t\t\t");
1683         }//for
1684         System.out.println();
1685     }//while
1686
1687     //close all jdbc stream objects
1688     rs.close();
1689     st.close();
1690     con.close();
1691 } //main
1692 } //class
1693
1694 =====
1695 App26(Program using ParameterMetaData)
1696 =====
1697 -----PMDTest.java-----
1698 import java.sql.*;
1699
1700 public class PMDTest {
1701
1702     public static void main(java.lang.String[] args) throws Exception
1703     {
1704         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
1705         Connection con=DriverManager.getConnection("jdbc:odbc:mysqldsn","root","root");
1706
1707
1708         PreparedStatement ps = con.prepareStatement("insert into student values(?, ?, ?)");
1709
1710         ParameterMetaData pmd=ps.getParameterMetaData();
1711
1712         System.out.println("impl class name of ParameterMetaData(i)" + pmd.getClass().getName());
1713
1714         int cnt=pmd.getParameterCount();
1715
1716         for (int i = 1; i<=cnt; i++) {
1717             System.out.println("Parameter number " + i);
1718
1719             System.out.println(" Mode is " + pmd.getParameterMode(i));
1720             System.out.println(" Type is " + pmd.getParameterType(i));
1721             System.out.println(" Type name is " + pmd.getParameterTypeName(i));
1722             System.out.println(" Precision is " + pmd.getPrecision(i));
1723             System.out.println(" Scale is " + pmd.getScale(i));
1724             System.out.println(" Nullable? is " + pmd.isNullable(i));
1725             System.out.println(" Signed? is " + pmd.isSigned(i));
1726         }
1727     }
1728 }
1729
1730 =====
1731 App27) program on cached rowset)
1732 =====
1733 -----Row1.java-----
1734 import java.sql.*;
1735 import javax.sql.*;
1736 public class Row1
1737 {
1738     public static void main (String []args)throws Exception
1739     {
1740         //create RowSet obj
```

```
1741     oracle.jdbc.rowset.OracleCachedRowSet crowset= new oracle.jdbc.rowset.OracleCachedRowSet();
1742     //set jdbc properties
1743     crowset.setUrl("jdbc:oracle:thin:@localhost:1521:orcl");
1744     crowset.setUsername("scott");
1745     crowset.setPassword("tiger");
1746     crowset.setCommand("select * from student");
1747     crowset.execute();//executes Query
1748     System.out.println("query executed");
1749     //process the Rowset
1750     while (crowset.next())
1751     {
1752         System.out.println (crowset.getInt(1)+" "+crowset.getString(2)+" "+crowset.getString(3));
1753     }
1754     crowset.close();
1755 } //main
1756 }//class
1757 =====
1758 App28(program on jdbc rowset )
1759 =====
1760 -----Row2.java-----
1761 import java.sql.*;
1762 import javax.sql.*;
1763 import oracle.jdbc.rowset.*;
1764 public class Row2
1765 {
1766     public static void main (String []args)
1767     {
1768         try
1769         {
1770             //create RowsSet obj
1771             RowSet jrowset = new OracleJDCTRowSet();
1772             //set jdbc properties
1773             jrowset.setUrl("jdbc:oracle:thin:@localhost:1521:orcl");
1774             jrowset.setUsername("scott");
1775             jrowset.setPassword("tiger");
1776             jrowset.setCommand("select empno from emp");
1777             jrowset.execute();
1778             System.out.println("command executed");
1779             //process the Rowset
1780             while (jrowset.next())
1781             {
1782                 System.out.println (jrowset.getInt(1));
1783             }
1784             jrowset.close();
1785         }
1786         catch(Exception ee)
1787         {
1788             System.out.println(ee.toString());
1789         }
1790     }
1791 }
1792 =====
1793 App29)Application on web rowsets
1794 =====
1795 -----Row3.java-----
1796 import java.io.File;
1797 import java.io.FileWriter;
1798 import java.io.StringWriter;
1799 import javax.sql.rowset.WebRowSet;
1800 import oracle.jdbc.rowset.OracleWebRowSet;
```

```
1801 public class Row3 {
1802     public static void main(String[] args) {
1803         try {
1804             //create WebRowSet obj
1805             OracleWebRowSet webRS= new OracleWebRowSet();
1806             //set jdbc properties
1807             webRS.setUsername("scott");
1808             webRS.setPassword("tiger");
1809             webRS.setUrl("jdbc:oracle:thin:@localhost:1521:orcl");
1810             webRS.setCommand("SELECT * from employee");
1811             //execute the query
1812             webRS.execute();
1813             //create or locate file
1814             File myFile=new File("c:/employee1.xml");
1815             FileWriter fw = new FileWriter(myFile);
1816
1817             System.out.println("Writing db data to file " + myFile.getAbsolutePath());
1818             webRS.writeXml(fw); // writes the records of Rowset to file
1819
1820             // convert xml to a String object for display purpose
1821             StringWriter sw = new StringWriter();
1822             webRS.writeXml(sw);
1823             System.out.println(sw.toString());
1824
1825             fw.flush();
1826             fw.close();
1827         }catch(Exception e)
1828         {
1829             e.printStackTrace();
1830         }
1831     }
1832 }
1833
1834 }
1835 }
1836 =====
1837 App30) BatchProcessing/Updataion)
1838 =====
1839 -----BatchProcess.java-----
1840 import java.sql.*;
1841 public class BatchProcess
1842 {
1843     public static void main(String args[])throws Exception
1844     {
1845         //load jdbc driver and establish the connection
1846         Class.forName("oracle.jdbc.driver.OracleDriver");
1847         Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
1848         //disable the autocommit mode
1849         con.setAutoCommit(false);
1850         //create Statement obj
1851         Statement st=con.createStatement();
1852         // add queries to batch
1853         st.addBatch("update student set sname='chinnai' where sno=10070");
1854         st.addBatch("insert into student values(169,'ccc','muak');");
1855         st.addBatch("delete from student where sno=34");
1856         // execute the batch
1857         int result[]=st.executeBatch();
1858         //process the results
1859         int sum=0;
1860         for(int i=0;i<result.length;++i)
```

```
1861     {
1862         sum=sum+result[i];
1863     }
1864     System.out.println("no.of records that are effected"+sum);
1865
1866     st.close();
1867     con.close();
1868
1869 } //main
1870 } //class
1871 =====
1872 App31( Transcation Management)
1873 =====
1874 -----TxMgmtTest.java-----
1875 //TxMgmtTest.java
1876 import java.sql.*;
1877 import java.util.*;
1878
1879 public class TxMgmtTest
1880 {
1881     public static void main(String args[])
1882     {
1883         Connection con=null;
1884         Statement st=null;
1885         try
1886         {
1887             //read input value from keyboard
1888             Scanner sc=new Scanner(System.in);
1889             System.out.println("Enter src a/c no:");
1890             int srcacno=sc.nextInt();
1891             System.out.println("Enter Dest a/c no:");
1892             int destacno=sc.nextInt();
1893             System.out.println("Enter Amt to transfer:");
1894             int amt=sc.nextInt();
1895
1896             // create jdbc con obj
1897             Class.forName("oracle.jdbc.driver.OracleDriver");
1898             con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
1899
1900             //disable auto commt mode
1901             con.setAutoCommit(false); //begins Tx
1902
1903             // create jdbc statement object
1904             st=con.createStatement();
1905
1906             // add queries of transferMoney operation to batch processing
1907
1908             //query for withdraw operation
1909             st.addBatch("update jdbc_account set balance=balance-"+amt+" where acno="+srcacno);
1910                     //query for deposite operation
1911             st.addBatch("update jdbc_account set balance=balance+"+amt+" where acno="+destacno);
1912
1913             // execute the queries of batch
1914             int res[]=st.executeBatch();
1915
1916             //write the code of Tx mgmt
1917             boolean flag=true;
1918
1919             for(int i=0;i<res.length;++i)
1920             {
```

```
1921             if(res[i]==0) // if any element value is 0
1922             {
1923                 flag=false;
1924                 break;
1925             }
1926         }
1927
1928         //commit or rollback the Tx
1929         if(flag==false)
1930         {
1931             con.rollback();
1932             System.out.println("Tx is rolledback(money not transferred)");
1933         }
1934     else
1935     {
1936         con.commit();
1937         System.out.println("Tx is committed(Money transffered)");
1938     }
1939
1940     //close jdbc objs
1941     st.close();
1942     con.close();
1943 }
1944 catch(Exception e)
1945 {
1946     try
1947     {
1948         con.rollback();
1949     }
1950     catch(Exception e1){e1.printStackTrace(); }
1951 } //catch
1952 }//main
1953 }//class
1954 //>javac TxMgmtTest.java
1955 //>java TxMgmtTest
1956 =====
1957 App32 (Batch Processing by using PreparedStatement obj)
1958 =====
1959 -----psBatchTest.java-----
1960 //PsBatchTest.java
1961 import java.sql.*;
1962 public class PsBatchTest
1963 {
1964     public static void main(String args[])throws Exception
1965     {
1966         // create jdbc con obj
1967         Class.forName("oracle.jdbc.driver.OracleDriver");
1968         Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
1969
1970         //create PreparedStatement obj
1971         PreparedStatement ps=con.prepareStatement("insert into student values(?, ?, ?)");
1972         //add param values to batch
1973         ps.setInt(1,111);
1974         ps.setString(2,"raja");
1975         ps.setString(3,"hyd");
1976         ps.addBatch(); //add 1st set param values to batch
1977
1978         ps.setInt(1,222);
1979         ps.setString(2,"ravi");
1980         ps.setString(3,"hyd1");
```

```
1981     ps.addBatch();//add 2nd set param values to batch
1982
1983     ps.setInt(1,333);
1984     ps.setString(2,"rakesh");
1985     ps.setString(3,"hyd2");
1986     ps.addBatch();//add 3rd set param values to batch
1987
1988     // execute the Query
1989     int res[]={ps.executeBatch()};
1990
1991     //process the result
1992     int cnt=0;
1993     for(int i=0;i<res.length;++i)
1994     {
1995         cnt=cnt+res[i];
1996     }
1997
1998     System.out.println("no.of records inserted:"+cnt);
1999
2000     //close jdbc objs
2001     ps.close();
2002     con.close();
2003 } //main
2004 } //class
2005
2006 =====
2007 App33(program to make JDBC Application Flexible)
2008 =====
2009 -----DBDetails.properties-----
2010 #properties file
2011 driver=oracle.jdbc.driver.OracleDriver
2012 url=jdbc:oracle:oci8:@orcl
2013 user=scott
2014 password=tiger
2015 -----FlexTest.java-----
2016 //FlexTest.java
2017 import java.sql.*;
2018 import java.io.*;
2019 import java.util.*;
2020
2021 public class FlexTest
2022 {
2023     public static void main(String[] args) throws Exception
2024     {
2025         //locate properties file
2026         FileInputStream fis=new FileInputStream("DBDetails.properties");
2027         //load the data of properties file to Properties class obj
2028         Properties p=new Properties();
2029         p.load(fis);
2030         // get jdbc details from Properties class obj
2031         String s1=p.getProperty("driver");
2032         String s2=p.getProperty("url");
2033         String s3=p.getProperty("user");
2034         String s4=p.getProperty("password");
2035
2036         //write jdbc code
2037         Class.forName(s1);
2038         Connection con=DriverManager.getConnection(s2,s3,s4);
2039         Statement st=con.createStatement();
2040         ResultSet rs=st.executeQuery("select * from student ");
```

```
2041     while(rs.next())
2042     {
2043         System.out.println(rs.getString(1)+" "+rs.getString(2)+" "+rs.getString(3));
2044     }
2045     rs.close();
2046     st.close();
2047     con.close();
2048
2049 } //main
2050 } //class
2051 =====
2052 App34 (on DriverManaged Jdbc Conn Pooling )
2053 =====
2054 -----ConnPoolTest.java-----
2055 //ConnPoolTest.java (works with DriverManaged jdbc con pool)
2056 import java.sql.*;
2057 import oracle.jdbc.pool.*; // (for OracleConnectionPoolDataSource (c))
2058 public class ConnPoolTest
2059 {
2060     public static void main(String args[])throws Exception
2061     {
2062         // create jdbc datasource obj representing an empty driver managed con pool for oracle
2063         OracleConnectionPoolDataSource ds = new OracleConnectionPoolDataSource();
2064
2065         // give details to create jdbc con objects in the above empty con pool
2066         ds.setDriverType ("thin");
2067         ds.setServerName ("localhost");
2068         ds.setPortNumber (1521);
2069         ds.setServiceName ("orcl");
2070         ds.setUser ("scott");
2071         ds.setPassword("tiger"); //-->now jdbc con pool is ready with jdbc con objs
2072
2073         // get one jdbc con obj from jdbc con pool through DataSource obj
2074         Connection con=ds.getConnection();
2075         // write jdbc persistence logic
2076         Statement st=con.createStatement();
2077         ResultSet rs=st.executeQuery("select * from student");
2078         while(rs.next())
2079         {
2080             System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
2081         }
2082         //close jdbc objects....
2083         rs.close();
2084         st.close();
2085         con.close(); //releases con obj back to Jdbc Con pool
2086     } //main
2087 } //class
2088 //add ojdbc14.jar file to classpath
2089 //>javac ConnPoolTest.java
2090 //>java ConnPoolTest
2091 =====
2092 App35(SavePoint )
2093 =====
2094 -----SavePointTest.java-----
2095 //SavePointTest.java
2096 import java.sql.*;
2097 public class SavePointTest
2098 {
2099     public static void main(String args[])throws Exception
2100     {
```

```
2101 //register jdbc driver and establish the connection
2102 Class.forName("oracle.jdbc.driver.OracleDriver");
2103 Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
2104
2105 //create statement obj
2106 Statement st = con.createStatement();
2107
2108 // Begin Tx
2109 con.setAutoCommit(false);
2110     //query1 (outside the save point area)
2111     st.executeUpdate("insert into student values(567,'xyz','hyd1')");
2112         // create Named Save Point("p1")
2113         Savepoint sp=con.setSavepoint("p1");
2114             //query2( in save point area)
2115             st.executeUpdate("update student set saddr='new delhi1' where sno=111");
2116             //rollback upto savepoint(p1)
2117             con.commit(sp);
2118     // commit Tx
2119     con.commit(); //end of the Tx
2120     //query1 will be committed but query2 will be rolled back
2121     //becoz the query2 is there in savepoint area..(p1)
2122 } //main
2123 } //class
2124 //>javac SavePointTest.java
2125 //>java SavePointTest
2126 =====
2127 App36 (interacting with Postgres SQL DB s/w)
2128 =====
2129 -----PostgreSQLApp.java-----
2130 //PostgreSQLApp.java
2131 import java.sql.*;
2132 public class PostgreSQLApp
2133 {
2134     public static void main(String args[])throws Exception
2135     {
2136         //register jdbc driver with DriverManager Service
2137         //Class.forName("org.postgresql.Driver");
2138
2139         //establish the connection
2140         // Connection con=DriverManager.getConnection("jdbc:postgresql:postgres","postgres","postgres");
2141         //or
2142         Connection con=DriverManager.getConnection("jdbc:postgresql://localhost:5432/postgres","postgres","postgres");
2143
2144         // create jdbc statement obj
2145         Statement st=con.createStatement();
2146         // send and execute SQL query
2147         ResultSet rs=st.executeQuery("select * from product");
2148         //process the jdbc ResultSet obj
2149         while(rs.next())
2150         {
2151             System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getFloat(3));
2152         }
2153
2154         // close jdbc objects
2155         rs.close();
2156         st.close();
2157         con.close();
2158     } //main
2159 } //class
2160
```

```
2161 //>javac PostgreSQLApp.java
2162 //>java PostgreSQLApp
2163 =====
2164 App37( Example Application Type3 JDBC driver)
2165 =====
2166 -----Type3Test1.java-----
2167 //Type3Test1.java
2168 import java.sql.*;
2169 class Type3Test1
2170 {
2171     public static void main(String[] args) throws Exception
2172     {
2173         //register type jdbc driver
2174         Class.forName("ids.sql.IDSDriver");
2175         //Establish the connection
2176         Connection con = DriverManager.getConnection("jdbc:ids://localhost:12/conn?dsn='oradsn'&user='scott'&password='tiger'");
2177         Connection con = DriverManager.getConnection("jdbc:ids://localhost:12/conn?dsn='accdsn'");
2178
2179         //create jdbc Statement obj
2180         Statement st = con.createStatement();
2181         // execute the SQL Query
2182         ResultSet rs=st.executeQuery("Select * from student");
2183         //process the ResultSet obj
2184         while(rs.next())
2185         {
2186             System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
2187         }
2188         //close jdbc objs
2189         rs.close();
2190         st.close();
2191         con.close();
2192     }
2193 }
2194 =====
2195 App38 (Application on Type5 JDBC Driver)
2196 =====
2197 -----Type5Test.java-----
2198 //Type5Test.java
2199 import java.sql.*;
2200 public class Type5Test
2201 {   public static void main(String[] args)throws Exception {
2202
2203         //loading jdbc driver class is optional here
2204         //Class.forName("com.ddtek.jdbc.oracle.OracleDriver");
2205
2206         // Establish the Connection
2207         String url = "jdbc:datadirect:oracle://localhost:1521;ServiceName=orcl";
2208         Connection con = DriverManager.getConnection(url, "scott", "tiger");
2209
2210         //send and excute the query
2211         Statement st=con.createStatement();
2212         ResultSet rs=st.executeQuery("select * from student");
2213         while(rs.next())
2214         {
2215             System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
2216         }
2217         //close jdbc objs
2218         rs.close();
2219         st.close();
2220         con.close();
```

```
2221         }//main
2222     }//class
2223 //>javac Type5Test.java
2224 //>java TypeTest
2225
2226 =====
2227     Applications on Reflection API
2228 =====
2229 -----App1.java-----
2230 //App1.java (gives all classes of inheritance hierarchy for the given java class)
2231 public class App1
2232 {
2233     public static void main(String args[])throws Exception
2234     {
2235         // load the given class to application
2236         Class c=Class.forName(args[0]);
2237         // get super class of given class
2238         Class sc=c.getSuperclass();
2239         System.out.println("The inheritance of given class is");
2240         while(sc!=null)
2241         {
2242             System.out.println("\t\t"+sc.getName());
2243             c=sc;
2244             sc=c.getSuperclass();
2245         }//while
2246
2247     }//main
2248 }//class
2249 //>javac App1.java  ( compile time)
2250 //>java App1  java.lang.String      (runtime)
2251 //>java App1  java.awt.TextField
2252 -----App2.java-----
2253 //App2.java (gives interfaces implemented by the given java class)
2254 public class App2
2255 {
2256     public static void main(String args[])throws Exception
2257     {
2258         // load the given into Application
2259         Class c=Class.forName(args[0]);
2260         // get the interfaces implemented by given class
2261         Class inter[]=c.getInterfaces();
2262         // print interface names
2263         System.out.println("The interfaces implemented by"+args[0]);
2264         for(int i=0;i<inter.length;++i)
2265         {
2266             System.out.println("\t\t"+inter[i].getName());
2267         }//for
2268
2269     }//main
2270 }//class
2271 //>javac App2.java
2272 //>java App2 java.lang.String
2273 //>java App2 java.util.Date
2274 -----App4.java-----
2275 //App4.java (gives modifiers of the given java class)
2276 import java.lang.reflect.Modifier;
2277 public class App4
2278 {
2279     public static void main(String args[])throws Exception
2280     {
```

```
2281     // load the given in to Application
2282     Class c=Class.forName(args[0]);
2283     // get the modifiers of given class
2284     int x=c.getModifiers();
2285     System.out.println("x value is"+x);
2286     //print modifiers
2287     System.out.println("The modifiers of "+args[0]);
2288     if(Modifier.isPublic(x))
2289         System.out.println("\t\t public");
2290     if(Modifier.isFinal(x))
2291         System.out.println("\t\t final");
2292     if(Modifier.isAbstract(x))
2293         System.out.println("\t\t abstract");
2294     }//main
2295 }//class
2296 //>javac App4.java
2297 //>java App4 java.lang.String
2298 //>java App4 java.lang.Integer
2299 -----App5.java-----
2300 // App5.java (gives the modifiers of the given java class)
2301 import java.lang.reflect.*;
2302
2303 public class App5
2304 {
2305
2306     public static void main(String args[])throws Exception
2307     {
2308         //load the given class into Application
2309         Class c=Class.forName(args[0]);
2310         // get all the fields of given java class
2311         Field f[]=c.getDeclaredFields();
2312         //dipslay member variable details
2313         System.out.println("The Fields of "+args[0]);
2314         for(int i=0;i<f.length;++i)
2315         {
2316             // get modifiers of each field
2317             int x=f[i].getModifiers();
2318             if(Modifier.isPublic(x))
2319                 System.out.print("\t\t public");
2320             if(Modifier.isPrivate(x))
2321                 System.out.print("\t\t private");
2322             if(Modifier.isProtected(x))
2323                 System.out.print("\t\t protected");
2324             if(Modifier.isStatic(x))
2325                 System.out.print("\t\t static");
2326             if(Modifier.isFinal(x))
2327                 System.out.print("\t\t final");
2328             if(Modifier.isVolatile(x))
2329                 System.out.print("\t\t volatile");
2330             if(Modifier.isTransient(x))
2331                 System.out.print("\t\t transient");
2332             String ftype=f[i].getType().getName();
2333             String fname=f[i].getName();
2334             System.out.println(" "+ftype+" "+fname);
2335         }//for
2336     }//main
2337 }//class
2338 //>javac App5.java
2339 //>java App5 java.lang.String
2340 //>java App5 java.lang.Integer
```

```
2341 -----App6.java-----
2342 //App6.java (gives all the constructors of given java class)
2343 import java.lang.reflect.*;
2344
2345 public class App6
2346 {
2347     public static void main(String args[])throws Exception
2348     {
2349         //load the given class in to app
2350         Class c=Class.forName(args[0]);
2351         // get the constructors of given java class
2352         Constructor cons[]=c.getDeclaredConstructors();
2353         //print constructor details
2354         System.out.println("the constructors of "+args[0]);
2355         for(int i=0;i<cons.length;++i)
2356         {
2357             //get modifiers of each constructor
2358             int x=cons[i].getModifiers();
2359             if(Modifier.isPublic(x))
2360                 System.out.print("\t public");
2361             if(Modifier.isProtected(x))
2362                 System.out.print("\t protected");
2363             if(Modifier.isPrivate(x))
2364                 System.out.print("\t private");
2365             // get name of the each constructor
2366             String name=cons[i].getName();
2367             //get parameter types of the each constructor
2368             Class params[]=cons[i].getParameterTypes();
2369             System.out.print(" " +name+"(");
2370             for(int k=0;k<params.length;++k)
2371             {
2372                 System.out.print(params[k].getName()+" ");
2373             }//for
2374             System.out.println(")");
2375         }//for
2376     }//main
2377 };//class
2378 //>javac App6.java
2379 //>java App6 java.lang.String
2380 //>java App6.java.util.Date
2381
2382
2383
```

# Working with BLOB, CLOB, STRUCT, ARRAY

**BLOB:**

BLOB stands for Binary Large Object. BLOB is used to store large amount of binary data into database like images.

The oracle.sql.BLOB class includes the following methods:

- getBinaryOutputStream(): Returns a java.io.OutputStream to write data to the BLOB as a stream.
- getBinaryStream(): Returns the BLOB data for this Blob instance as a stream of bytes.
- getBufferSize(): Returns the ideal buffer size, according to calculations by the JDBC driver, to use in reading and writing BLOB data. This value is a multiple of the chunk size (see getChunkSize() below) and is close to 32K.
- getBytes(): Reads from the BLOB data, starting at a specified point, into a supplied buffer.
- getChunkSize(): Returns the Oracle chunking size, which can be specified by the database administrator when the LOB column is first created. This value, in Oracle blocks, determines the size of the chunks of data read or written by the LOB data layer in accessing or modifying the BLOB value. Part of each chunk stores system-related information, and the rest stores LOB data. Performance is enhanced if read and write requests use some multiple of the chunk size.
- length(): Returns the length of the BLOB in bytes.
- position(): Determines the byte position in the BLOB where a given pattern begins.
- putBytes(): Writes BLOB data, starting at a specified point, from a supplied buffer.

**Below example shows how to store images into database.**

```

package com.nit;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
public class MyBlobInsert {
public static void main(String a[]){
Connection con = null;
PreparedStatement ps = null;
InputStream fis = null;
try {
    Class.forName("oracle.jdbc.driver.OracleDriver");
    con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","manager");
    System.out.println(con);
    ps = con.prepareStatement("insert into student_profile values(?,?)");
    ps.setInt(1,19);
    File f=new File("D:\\PAWAN.jpg");
    fis = new FileInputStream(f);
    ps.setBinaryStream(2, fis,(int)f.length());
    int count = ps.executeUpdate();
    System.out.println("Count: "+count);
} catch (ClassNotFoundException e) {
    e.printStackTrace();
} catch (SQLException e) {
    System.out.println(e);
    e.printStackTrace();
} catch (FileNotFoundException e) {
    e.printStackTrace();
} finally{
    try{
        if(fis != null) fis.close();
        if(ps != null) ps.close();
        if(con != null) con.close();
    } catch(Exception ex){
        ex.printStackTrace();
    }
}}}

```

**SQL> CREATE TABLE STUDENT\_PROFILE(SNO NUMBER(9),SPHOTO BLOB);**

**Table created.**

**How to read an image from database table? or Write an example for reading BLOB from table.**

```
package com.nit;
import java.io.FileOutputStream;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
public class RetrieveImageApp {
public static void main(String[] args) throws IOException, SQLException, ClassNotFoundException {
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con = DriverManager.getConnection("jdbc:oracle:thin:@nit-
11:1521:xe","system","manager");
PreparedStatement ps=con.prepareStatement("select sphoto from student_profile where sno=19");
FileOutputStream fos=new FileOutputStream("D:\\NIT\\PAWAN.jpg");
ResultSet rs =ps.executeQuery();
while(rs.next()){
fos.write(rs.getBytes(1));
}
fos.close();
}}
```

#### **CLOB:-**

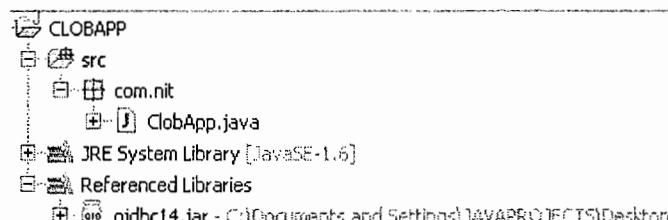
CLOB Stands for CharacterLargeObject .If a column data type is declared as a CLOB then in that column we can store any text file data.

The **oracle.sql.CLOB** class includes the following methods:

- getAsciiOutputStream(): Returns a java.io.OutputStream to write data to the CLOB as a stream.
- getAsciiStream(): Returns the CLOB value designated by the Clob object as a stream of ASCII bytes.
- getBufferSize(): Returns the ideal buffer size, according to calculations by the JDBC driver, to use in reading and writing CLOB data. This value is a multiple of the chunk size (see getChunkSize() below) and is close to 32K.
- getCharacterOutputStream(): Returns a java.io.Writer to write data to the CLOB as a stream.
- getCharacterStream(): Returns the CLOB data as a stream of Unicode characters.
- getChars(): Retrieves characters from a specified point in the CLOB data into a character array.
- getChunkSize(): Returns the Oracle chunking size, which can be specified by the database administrator when the LOB column is first created. This value, in Oracle blocks, determines the size of the chunks of data read or written by the LOB data layer in accessing or modifying the CLOB value. Part of each chunk stores system-related information and the rest stores LOB data. Performance is enhanced if you make read and write requests using some multiple of the chunk size.
- getSubString(): Retrieves a substring from a specified point in the CLOB data.
- length(): Returns the length of the CLOB in characters.
- position(): Determines the character position in the CLOB at which a given substring begins.
- putChars(): Writes characters from a character array to a specified point in the CLOB data.
- putString(): Writes a string to a specified point in the CLOB data.

**SQL> CREATE TABLE EXAMPLE(EID NUMBER(19),ENAME VARCHAR2(19),ECODE CLOB);**  
Table created.

**Write a java program to insert example eid,example ename and example ecode into above table**



```
ClobApp.java
package com.nit;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
public class ClobApp {
public static void main(String[] args) throws ClassNotFoundException,SQLException,IOException {
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@nit-11:1521:xe","system","manager");
File f=new File("D:\\NIT\\Test.java");
String sql="INSERT INTO EXAMPLE VALUES(?, ?, ?)";
PreparedStatement ps=con.prepareStatement(sql);
ps.setInt(1,99);
ps.setString(2,"NARESH");
ps.setCharacterStream(3,new FileReader(f),(int)f.length());
ps.executeUpdate();
}
}
```

**Write a java program to get example eid,example ename and example ecode from table**

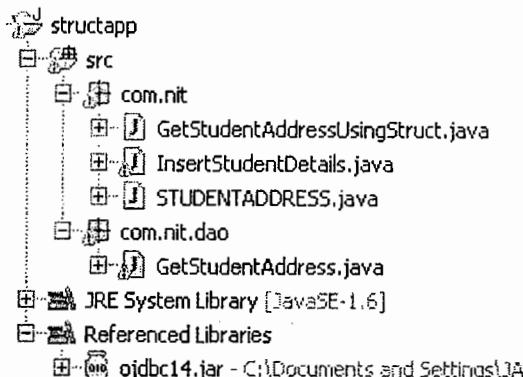
**Example**

**ClobRetrieveApp.java**

```
package com.nit;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.sql.Clob;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
public class ClobRetrieveApp {
public static void main(String[] args) throws ClassNotFoundException, SQLException, IOException {
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@nit-11:1521:xe","system","manager");
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("SELECT *FROM EXAMPLE");
while (rs.next()) {
int eid=rs.getInt(1);
String ename=rs.getString(2);
Clob blob=rs.getBlob(3);
InputStream in=blob.getAsciiStream();
FileOutputStream fos=new FileOutputStream("D:\\AdvancedJava\\"+eid+ename+".java");
int ch=in.read();
while(ch!= -1){
fos.write(ch); //storing in file
System.out.println(ch); //displaying on monitor
ch=in.read();
}
in.close();
fos.close();
}
}
```

**Struct(Object) Data Type**

This data type is required in case you want to create a user-defined type in a database. For example ,you might need to create a user defined type to represent the address of a student in a single column.

**STUDENTADDRESS.java**

```

package com.nit;
import java.sql.SQLData;
import java.sql.SQLException;
import java.sql.SQLInput;
import java.sql.SQLOutput;
public class STUDENTADDRESS implements SQLData{
private String street,city,state,typename;
private int fno,pin;
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;
}
public String getTypename() {
return typename;
}
public void setTypename(String typename) {
this.typename = typename;
}

public int getFno() {
return fno;
}

public void setFno(int fno) {
this.fno = fno;
}

public int getPin() {
return pin;
}
  
```

```

public void setPin(int pin) {
    this.pin = pin;
}

@Override
public String getSQLTypeName() throws SQLException {
    return typename;
}

@Override
public void readSQL(SQLInput stream, String name) throws SQLException {
    fno=stream.readInt();
    street=stream.readString();
    city=stream.readString();
    state=stream.readString();
    pin=stream.readInt();
    typename=name;
}

@Override
public void writeSQL(SQLOutput stream) throws SQLException {
    stream.writeInt(fno);
    stream.writeString(city);
    stream.writeString(state);
    stream.writeString(street);
    stream.writeInt(pin);
}
}

```

**InsertStudentDetails.java**

```

package com.nit;
import java.io.File;
import java.io.FileInputStream;
import java.sql.Connection;
import java.sql.Driver;
import java.sql.PreparedStatement;
import java.util.Properties;
public class InsertStudentDetails {
    public static void main(String[] args) throws Exception{
        Driver d=(Driver)(Class.forName("oracle.jdbc.driver.OracleDriver").newInstance());
        Properties p=new Properties();
        p.put("user", "system");
        p.put("password", "manager");
        Connection con=d.connect("jdbc:oracle:thin:@nit-11:1521:xe", p);
        PreparedStatement ps=con.prepareStatement("INSERT INTO
NARESHIT_STUDENTDETAILS(SNO,PHOTO,PERMENTADDRESS) VALUES(?, ?, ?)");
        ps.setInt(1, 99);
        File f=new File("C:\\\\Documents and Settings\\\\All Users\\\\Documents\\\\My Pictures\\\\Sample
Pictures\\\\Sunset.jpg");
        FileInputStream fis=new FileInputStream(f);
        ps.setBinaryStream(2,fis,(int)f.length());
        STUDENTADDRESS addr=new STUDENTADDRESS();
        addr.setFno(19);
        addr.setCity("hyd");
        addr.setStreet("Ameerpet");
        addr.setPin(500099);
        addr.setState("AP");
        addr.setTypename("NIT_STUDENTADDR2");
        ps.setObject(3,addr);
        int i=ps.executeUpdate();
        System.out.println("Personal Details of Student 99 inserted successfully");
        con.close();
    }
}

```

**GetStudentAddress.java**

```

package com.nit.dao;
import java.sql.Connection;
import java.sql.Driver;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.HashMap;
import java.util.Properties;

import com.nit.STUDENTADDRESS;

public class GetStudentAddress {
public static void main(String[] args)throws Exception {
    Driver d=(Driver)(Class.forName("oracle.jdbc.driver.OracleDriver").newInstance());
    Properties p=new Properties();
    p.put("user", "system");
    p.put("password", "manager");
    Connection con=d.connect("jdbc:oracle:thin:@nit-11:1521:xe", p);
    Statement st=con.createStatement();
    ResultSet rs=st.executeQuery("SELECT PERMENTADDRESS FROM NARESHIT_STUDENTDETAILS WHERE
SNO=99");
    if(rs.next()){
        HashMap map=new HashMap();
        map.put("NIT_STUDENTADDR2",STUDENTADDRESS.class);
        STUDENTADDRESS addr=(STUDENTADDRESS)rs.getObject(1,map);
        System.out.println("Student Address Found");
        System.out.println("Flatno:"+addr.getFno());
        System.out.println("Street:"+addr.getStreet());
        System.out.println("Pin:"+addr.getPin());
    }
    con.close();
}
}

```

**GetStudentAddressUsingStruct.java**

```

package com.nit;
import java.sql.Connection;
import java.sql.Driver;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.Struct;
import java.util.Properties;

public class GetStudentAddressUsingStruct {
public static void main(String[] args)throws Exception {
    Driver d=(Driver)(Class.forName("oracle.jdbc.driver.OracleDriver").newInstance());
    Properties p=new Properties();
    p.put("user", "system");
    p.put("password", "manager");
    Connection con=d.connect("jdbc:oracle:thin:@nit-11:1521:xe", p);
    Statement st=con.createStatement();
    ResultSet rs=st.executeQuery("SELECT PERMENTADDRESS FROM NARESHIT_STUDENTDETAILS WHERE
SNO=99");
    if(rs.next()){
        System.out.println("Student Address Found");
        Struct struct=(Struct)rs.getObject(1);
        Object addr[] = struct.getAttributes();
        System.out.println("Flatno:"+addr[0]);
        System.out.println("Street:"+addr[1]);
        System.out.println("Pin:"+addr[4]);
        System.out.println();
    }
    con.close();
}
}

```

**Output**

```

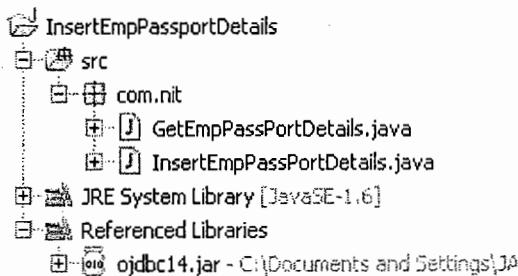
Student Address Found
Flatno:19
Street:hyd
Pin:500099
SQL> CREATE TYPE NIT_STUDENTADDR2 AS OBJECT (FLATNO NUMBER,STREET VARCHAR2(19),CITY
VARCHAR2(19),STATE VARCHAR2(19),PINCODE NUMBER)
2 /
Type created.
SQL> CREATE TABLE NARESHIT_STUDENTDETAILS(SNO NUMBER,PHOTO BLOB,PERMENTADDRESS
NIT_STUDENTADDR2,PRESENT_ADDRESS NIT_STUDENTADDR2);

```

Table created.

**The Array Data Type:**

The java.sql.Array interface of JDBC API provides an abstraction to understand the JDBC Driver object that represent a database array type. The Array object is implemented by using an SQL locator, which indicates that the array object contains a logical pointer to locate the array value in a database.

**InsertEmpPassPortDetails.java**

```

package com.nit;
import java.sql.Connection;
import java.sql.Driver;
import java.sql.PreparedStatement;
import java.util.Properties;
import oracle.sql.ARRAY;
import oracle.sql.ArrayDescriptor;
public class InsertEmpPassPortDetails {
public static void main(String[] args) throws Exception {
    Driver d=(Driver)(Class.forName("oracle.jdbc.driver.OracleDriver").newInstance());
    Properties p=new Properties();
    p.put("user", "system");
    p.put("password", "manager");
    Connection con=d.connect("jdbc:oracle:thin:@nit-11:1521:xe", p);
    PreparedStatement ps=con.prepareStatement("INSERT INTO EMPPASSPORTDETAILS VALUES(?, ?, ?)");
    ps.setInt(1, 99);
    ps.setString(2, "9989A555");
    String s1[]={ "v1", "v2", "v3", "v4", "v5" };
    ArrayDescriptor ad=ArrayDescriptor.createDescriptor("VISA_NOS", con);
    ARRAY a=new ARRAY(ad, con, s1);
    ps.setArray(3, a);
    int i=ps.executeUpdate();
    System.out.println("Rows Inserted, Count:" + i);
    con.close();
}
}

```

**GetEmpPassPortDetails.java**

```

package com.nit;
import java.sql.Array;
import java.sql.Connection;
import java.sql.Driver;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.Properties;
public class GetEmpPassPortDetails {

```

```
public static void main(String[] args) throws Exception {
    Driver d=(Driver)(Class.forName("oracle.jdbc.driver.OracleDriver").newInstance());
    Properties p=new Properties();
    p.put("user", "system");
    p.put("password", "manager");
    Connection con=d.connect("jdbc:oracle:thin:@nit-11:1521:xe", p);
    PreparedStatement ps=con.prepareStatement("SELECT PASSPORTNO,VISA_TAKEN FROM
EMPPASSPORTDETAILS WHERE EMPNO=99");
    ResultSet rs=ps.executeQuery();
    if(rs.next()){
        System.out.println("\n Employee Found,His Passport Details are:\n");
        System.out.println("PassportNo:"+rs.getString(1)+"\n");
        System.out.print("Visa's.Taken are :\n\t");
        Array a=rs.getArray(2);
        ResultSet rs1=a.getResultSet();
        boolean flag=rs1.next();
        while (flag) {
            System.out.println(rs1.getString(2));
            flag=rs1.next();
        }
        if(flag)
            System.out.print(",");
    }
    else
        System.out.println("Employee not Found");
    con.close();
}
```

**Output**

Employee Found,His Passport Details are:

PassportNo:9989A555

Visa's Taken are :

v1  
,v2  
,v3  
,v4  
,v5

SQL> CREATE TYPE VISA\_NOS AS VARRAY(5) OF VARCHAR2(19)  
2 /

Type created.

SQL> CREATE TABLE EMPPASSPORTDETAILS(EMPNO NUMBER,PASSPORTNO VARCHAR2(19),VISA\_TAKEN  
VISA\_NOS);

**FAQs****JDBC Objective Type Questions & Answers****1. Which statements about JDBC are true? (2 answers)**

- a. JDBC is an API to connect to relational-, object- and XML data sources
- b. JDBC stands for Java DataBase Connectivity
- c. JDBC is an API to access relational databases, spreadsheets and flat files
- d. JDBC is an API to bridge the object-relational mismatch between OO programs and relational databases

**Ans: b,c****2. Which packages contain the JDBC classes?**

- |                             |                                |
|-----------------------------|--------------------------------|
| a. java.jdbc and javax.jdbc | b. java.jdbc and java.jdbc.sql |
| c. java.sql and javax.sql   | d. java.rdb and javax.rdb      |

**Ans: c****3. Which type of driver provides JDBC access via one or more ODBC drivers?**

- a. Type 1 driver
- b. Type 2 driver
- c. Type 3 driver
- d. Type 4 driver

**Ans: a****4. Which type of driver converts JDBC calls into the network protocol used by the database management system directly?**

- a. Type 1 driver
- b. Type 2 driver
- c. Type 3 driver
- d. Type 4 driver

**Ans: d****5. Which type of Statement can execute parameterized queries?**

- a. PreparedStatement
- b. ParameterizedStatement
- c. ParameterizedStatement and CallableStatement
- e. All kinds of Statements (i.e. which implement a sub interface of Statement)

**Ans: a****6. How can you retrieve information from a ResultSet?**

- a. By invoking the method get(..., String type) on the ResultSet, where type is the database type
- b. By invoking the method get(..., Type type) on the ResultSet, where Type is an object which represents a database type
- c. By invoking the method getValue(...), and cast the result to the desired Java type.
- d. By invoking the special getter methods on the ResultSet: getString(...), getBoolean (...), getBlob(...),...

**Ans: d****7. How can you execute DML statements (i.e. insert, delete, update) in the database?**

- a. By making use of the InsertStatement, DeleteStatement or UpdateStatement classes
- b. By invoking the execute(...) or executeUpdate(...) method of a normal Statement object or a sub-interface object
- c. thereof
- d. By invoking the executeInsert(...), executeDelete(...) or executeUpdate(...) methods of the DataModificationStatement object
- f. By making use of the execute(...) statement of the DataModificationStatement object

**Ans:b****8. How do you know in your Java program that a SQL warning is generated as a result of executing a SQL statement in the database?**

- a. You must catch the checked SQLException which is thrown by the method which executes the statement
- b. You must catch the unchecked SQLWarningException which is thrown by the method which executes the statement
- c. You must invoke the getWarnings() method on the Statement object (or a sub interface thereof)
- d. You must query the ResultSet object about possible warnings generated by the database

**Ans:c****9. What is, in terms of JDBC, a DataSource?**

- a. A DataSource is the basic service for managing a set of JDBC drivers
- b. A DataSource is the Java representation of a physical data source
- c. A DataSource is a registry point for JNDI-services
- d. A DataSource is a factory of connections to a physical data source

**Ans:d****10. What is the meaning of ResultSet.TYPE\_SCROLL\_INSENSITIVE**

- a. This means that the ResultSet is insensitive to scrolling
- b. This means that the Resultset is sensitive to scrolling, but insensitive to updates, i.e. not updateable
- c. This means that the ResultSet is sensitive to scrolling, but insensitive to changes made by others
- d. The meaning depends on the type of data source, and the type and version of the driver you use

with this data source

**Ans:c**

**11. Are ResultSets updateable?**

- a. Yes, but only if you call the method openCursor() on the ResultSet, and if the driver and database support this option
- b. Yes, but only if you indicate a concurrency strategy when executing the statement, and if the driver and database support this option
- c. Yes, but only if the ResultSet is an object of class UpdateableResultSet, and if the driver and database support this option
- d. No, ResultSets are never updateable. You must explicitly execute DML statements (i.e. insert, delete and update) to change the data in the underlying database

**Ans: b**

**12. What statements are correct about JDBC transactions (2 correct answers)?**

- a. A transaction is a set of successfully executed statements in the database
- b. A transaction is finished when commit() or rollback() is called on the Connection object,
- c. A transaction is finished when commit() or rollback() is called on the Transaction object
- d. A transaction is finished when close() is called on the Connection object.

**Ans:b,d**

**13. How can you start a database transaction in the database?**

- a. By asking a Transaction object to your Connection, and calling the method begin() on it
- b. By asking a Transaction object to your Connection, and setting the autoCommit property of the Transaction to false
- c. By calling the method beginTransaction() on the Connection object
- d. By setting the autoCommit property of the Connection to false, and execute a statement in the database

**Ans:d**

**14. What is the meaning of the transaction isolation level**

TRANSACTION\_REPEATABLE\_READ

- a. Dirty reads, non-repeatable reads and phantom reads can occur
- b. Dirty reads are prevented; non-repeatable reads and phantom reads can occur
- c. Dirty reads and non-repeatable reads are prevented; phantom reads can occur
- d. Dirty reads, non-repeatable reads and phantom reads are prevented

**Ans:c**

**15. What statements are correct about positioned updates (i.e. cursor updates) in ResultSets? (2 correct answers)**

- a. Using the cursor technique is currently the only possible way to change the data in the current row of a ResultSet
- b. Insert statements are only supported when using scrollable cursors.
- c. Only scrollable updateable ResultSets can use this approach to change the data in the current row of a ResultSet
- d. The name of the cursor is specified by the setCursorName(String name) method the Statement object.

**Ans: b,d**

**16. How can you execute a stored procedure in the database?**

- a. Call method execute() on a CallableStatement object
- b. Call method executeProcedure() on a Statement object
- c. Call method execute() on a StoredProcedure object
- d. Call method run() on a ProcedureCommand object

**Ans: a**

**17. What happens if you call the method close() on a ResultSet object?**

- a. The method close() does not exist for a ResultSet. Only Connections can be closed.
- b. The database and JDBC resources are released
- c. You will get a SQLException, because only Statement objects can close ResultSets
- d. The ResultSet, together with the Statement which created it and the Connection from which the Statement was retrieved, will be closed and release all database and JDBC resources

**Ans: b**

**18. What happens if you call deleteRow() on a ResultSet object?**

- a. The row you are positioned on is deleted from the ResultSet, but not from the database.
- b. The row you are positioned on is deleted from the ResultSet and from the database
- c. The result depends on whether the property synchronizeWithDataSource is set to true or false
- d. You will get a compile error: the method does not exist because you can not delete rows from a ResultSet

**Ans:b**

**19. What statements are correct about batched insert and updates? (2 answers)**

- a. To create a batch of insert and update statements, you create an object of type Batch, and call the method
- b. AddStatement(String statement) for each statement you want to execute in the batch
- c. Batch insert and updates are only possible when making use of parameterized queries.
- d. To do a batched update/insert, you call addBatch(String statement) on a Statement object for each statement you want to execute in the batch
- e. To execute a batched update/insert, you call the executeBatch() method on a Statement object

**Ans: c, d****20. What is correct about DDL statements (create, grant,...)?**

- a. DDL statements are treated as normal SQL statements, and are executed by calling the execute() method on a Statement (or a sub interface thereof) object
- b. To execute DDL statements, you have to install additional support files
- c. DDL statements cannot be executed by making use of JDBC, you should use the native database tools for this.
- d. Support for DDL statements will be a feature of a future release of JDBC

**Ans: a****21. The JDBC-ODBC Bridge supports multiple concurrent open statements per connection?**

- a. True
- c. False

**Ans: a****22. Which of the following allows non repeatable read in JDBC Connection?**

- a. TRANSACTION\_READ\_UNCOMMITTED
- b. RANSACTION\_READ\_COMMITTED
- c. TRANSACTION\_SERIALIZABLE
- d. TRANSACTION\_REPEATABLE\_READ

**Ans:d****23. Which of the following statements is false as far as different type of statements is concern in JDBC?**

- |                       |                       |
|-----------------------|-----------------------|
| a. Regular Statement  | b. Prepared Statement |
| c. Callable Statement | d. Interim Statement  |

**Ans:d****24. Which of the following methods are needed for loading a database driver in JDBC?**

- a. registerDriver() method
- b. Class.forName()
- c. Both A and B
- d. getConnection()

**Ans: c****25. Which of the following is false as far as type 4 driver is concern?**

- a. Type 4 driver is "native protocol, pure java" driver
- b. Type 4 drivers are 100% Java compatible
- c. Type 4 drivers uses Socket class to connect to the database
- d. Type 4 drivers cannot be used with Netscape

**Ans:d****26. To execute a stored procedure "totalStock" in a database server, which of the following code snippet is used?**

- a. Statement stmt = connection.createStatement();stmt.execute("totalStock());
- b. CallableStatement clbstmnt = con.prepareCall("{call totalStock}");cs.executeQuery();
- c. StoreProcedureStatement  
stmt=connection.createStoreProcedure("totalStock());spstmt.executeQuery();
- b. PrepareStatement pstmt = connection.prepareStatement("totalStock());pstmt.execute();

**Ans: b****27. Which driver is efficient and always preferable for using JDBC applications?**

- a. Type - 4
- b. Type - 1
- c. Type - 3
- d. Type - 2

**Ans:a****28. JDBC facilitates to store the java objects by using which of the methods of PreparedStatement****setObject () 2. setBlob() 3. setClob()**

- a. 1, 2
- b. 1,2,3
- c. 1,3
- d. 2,3

**Ans: b****29. Which statement is static and synchronized in JDBC API?**

- a. executeQuery()
- b. executeUpdate()
- c. getConnection()
- d. prepareCall()

**Ans:c****30. The JDBC-ODBC bridge is**

- a. Three tiered
- b. Multithreaded
- c. Best for any platform
- d. All of the above

**Ans:b**

**31. All raw data types (including binary documents or images) should be read and uploaded to the database as an array of**

- a. byte
- b. int
- c. boolean
- d. char

**Ans: a**

**32. The class `java.sql.Timestamp` has its super class as**

- a. `java.sql.Time`
- b. `java.util.Date`
- c. `java.util.Time`
- d. None of the above

**Ans: b**

**33. BLOB, CLOB, ARRAY and REF type columns can be updated in**

- a. JDBC 1.0
- b. JDBC 4.0
- c. JDBC 2.0
- d. JDBC 3.0

**Ans: d**

**34. Which of the following methods finds the maximum number of connections that a specific driver can obtain?**

- a. `Database.getMaxConnections`
- b. `Connection.getMaxConnections`
- c. `DatabaseMetaData.getMaxConnections`
- d. `ResultSetMetaData.getMaxConnections`

**Ans: c**

**35. Are prepared statements actually compiled?**

- a. Yes, they compiled
- b. No, they are bound by the JDBC driver

**Ans: a**

**36. When the message "No Suitable Driver" occurs?**

- a. When the driver is not registered by `Class.forName()` method
- b. When the user name, password and the database does not match
- c. When the JDBC database URL passed is not constructed properly
- d. When the type 4 driver is used

**Ans: c**

**37. Which driver is called as thin-driver in JDBC?**

- a. Type-4 driver
- b. Type-1 driver
- c. Type-3 driver
- d. Type-2 driver

**Ans: a**

**38. How many transaction isolation levels are defined in `java.sql.Connection` interface?**

- a. 4
- b. 3
- c. 5
- d. 2

**Ans: c**

**39. Which method is used to perform DML statements in JDBC?**

- a. `execute()`
- b. `executeQuery()`
- c. `executeUpdate()`
- d. `executeResult()`

**Ans: c**

**40. What is the disadvantage of Type-4 Native-Protocol Driver?**

- a. At client side, a separate driver is needed for each database.
- b. Type-4 driver is entirely written in Java
- c. The driver converts JDBC calls into vendor-specific database protocol
- d. It does not support to read MySQL data.

**Ans:a**

### Learn more about JDBC 4.0 features

**Explain Basic Steps in writing a Java program using JDBC?**

JDBC makes the interaction with RDBMS simple and intuitive. When a Java application needs to access database :

Load the RDBMS specific JDBC driver because this driver actually communicates with the database (Incase of JDBC 4.0 this is automatically loaded).

Open the connection to database which is then used to send SQL statements and get results back.  
Create JDBC Statement object. This object contains SQL query.

Execute statement which returns resultset(s). ResultSet contains the tuples of database table as a result of SQL query.

Process the result set.

Close the connection.

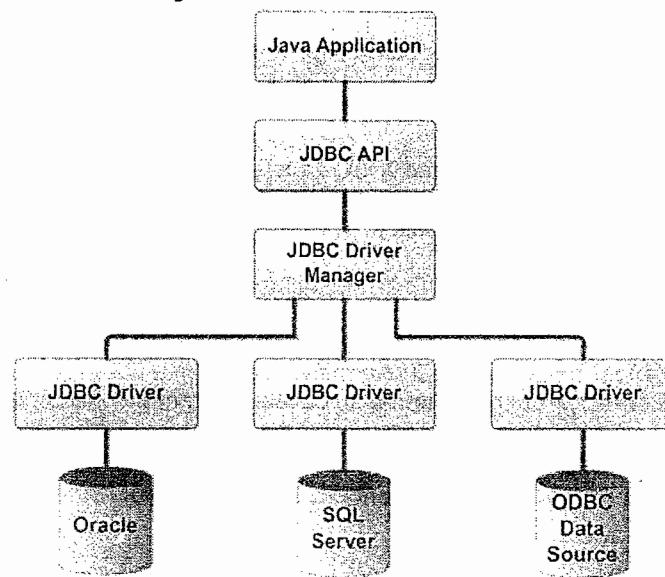
**Explain the JDBC Architecture.**

The JDBC Architecture consists of two layers:

The JDBC API, which provides the **application-to-JDBC Manager** connection.

The JDBC Driver API, which supports the **JDBC Manager-to-Driver** Connection.

The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases. The JDBC driver manager ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases. The location of the driver manager with respect to the JDBC drivers and the Java application is shown in Figure.

**What are the main components of JDBC ?**

The life cycle of a servlet consists of the following phases:

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

**Driver:** The database communications link, handling all communication with the database. Normally, once the driver is loaded, the developer need not call it explicitly.

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

**Statement :** Encapsulates an SQL statement which is passed to the database to be parsed, compiled, planned and executed.

**ResultSet:** The ResultSet represents set of rows retrieved due to query execution.

**How the JDBC application works?**

A JDBC application can be logically divided into two layers:

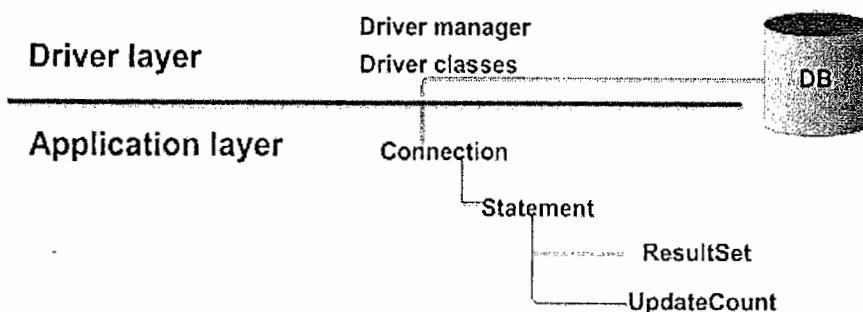
**1. Driver layer****2. Application layer**

Driver layer consists of DriverManager class and the available JDBC drivers.

The application begins with requesting the DriverManager for the connection.

An appropriate driver is chosen and is used for establishing the connection. This connection is given to the application which falls under the application layer.

The application uses this connection to create Statement kind of objects, through which SQL commands are sent to backend and obtain the results.

**Figure: JDBC Application****How do I load a database driver with JDBC 4.0 / Java 6?**

Provided the JAR file containing the driver is properly configured, just place the JAR file in the classpath. Java developers **NO** longer need to explicitly load JDBC drivers using code like Class.forName() to register a JDBC driver. The DriverManager class takes care of this by automatically locating a suitable driver when the DriverManager.getConnection() method is called. This feature is backward-compatible, so no changes are needed to the existing JDBC code.

**What is JDBC Driver interface?**

The JDBC Driver interface provides vendor-specific implementations of the abstract classes provided by the JDBC API. Each vendor driver must provide implementations of the java.sql.Connection, Statement, PreparedStatement, CallableStatement, ResultSet and Driver.

**What does the connection object represents?**

The connection object represents communication context, i.e., all communication with database is through connection object only.

**What is Statement ?**

Statement acts like a vehicle through which SQL commands can be sent. Through the connection object we create statement kind of objects.

Through the connection object we create statement kind of objects.

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

This method returns object which implements statement interface.

**What is PreparedStatement?**

A prepared statement is an SQL statement that is precompiled by the database. Through precompilation, prepared statements improve the performance of SQL commands that are executed multiple times (given that the database supports prepared statements). Once compiled, prepared statements can be customized prior to each execution by altering predefined SQL parameters.

```
PreparedStatement pstmt = conn.prepareStatement("UPDATE EMPLOYEES SET SALARY = ?  
WHERE ID = ?");  
  
pstmt.setBigDecimal(1, 153833.00);  
  
pstmt.setInt(2, 110592);
```

*Here: conn is an instance of the Connection class and "?" represents parameters. These parameters must be specified before execution.*

**What is the difference between a Statement and a PreparedStatement?**

<b>Statement</b>	<b>PreparedStatement</b>
A standard Statement is used to create a Java representation of a literal SQL statement and execute it on the database.	A PreparedStatement is a precompiled statement. This means that when the PreparedStatement is executed, the RDBMS can just run the PreparedStatement SQL statement without having to compile it first.
Statement has to verify its metadata against the database every time.	While a prepared statement has to verify its metadata against the database only once.
If you want to execute the SQL statement once go for STATEMENT	If you want to execute a single SQL statement multiple number of times, then go for PREPAREDSTATEMENT. PreparedStatement objects can be reused with passing different values to the queries

**What are callable statements ?**

Callable statements are used from JDBC application to invoke stored procedures and functions.

**How to call a stored procedure from JDBC ?**

PL/SQL stored procedures are called from within JDBC programs by means of the prepareCall() method of the Connection object created. A call to this method takes variable bind parameters as input parameters as well as output variables and creates an object instance of the CallableStatement class.

The following line of code illustrates this:

```
CallableStatement stproc_stmt = conn.prepareCall("{call procname(?,?,?)}");
```

Here conn is an instance of the Connection class.

**What are types of JDBC drivers?**

There are four types of drivers defined by JDBC as follows:

**Type 1: JDBC/ODBC**—These require an ODBC (Open Database Connectivity) driver for the database to be installed. This type of driver works by translating the submitted queries into equivalent ODBC queries and forwards them via native API calls directly to the ODBC driver. It provides no host redirection capability.

**Type2: Native API (partly-Java driver)**—This type of driver uses a vendor-specific driver or database API to interact with the database. An example of such an API is Oracle OCI (Oracle Call Interface). It also provides no host redirection.

**Type 3: Open Protocol-Net**—This is not vendor specific and works by forwarding database requests to a remote database source using a net server component. How the net server component accesses the database is transparent to the client. The client driver communicates with the net server using a database-independent protocol and the net server translates this protocol into database calls. This type of driver can access any database.

**Type 4: Proprietary Protocol-Net(pure Java driver)**—This has a same configuration as a type 3 driver but uses a wire protocol specific to a particular vendor and hence can access only that vendor's database. Again this is all transparent to the client.

**Note:** Type 4 JDBC driver is most preferred kind of approach in JDBC.

**Which type of JDBC driver is the fastest one?**

JDBC Net pure Java driver(Type IV) is the fastest driver because it converts the JDBC calls into vendor specific protocol calls and it directly interacts with the database.

**Does the JDBC-ODBC Bridge support multiple concurrent open statements per connection?**

No. You can open only one Statement object per connection when you are using the JDBC-ODBC Bridge.

**Which is the right type of driver to use and when?**

Type I driver is handy for prototyping

Type III driver adds security, caching, and connection control

Type III and Type IV drivers need no pre -installation

**What are the standard isolation levels defined by JDBC?**

The values are defined in the class `java.sql.Connection` and are:

```
TRANSACTION_NONE
TRANSACTION_READ_COMMITTED
TRANSACTION_READ_UNCOMMITTED
TRANSACTION_REPEATABLE_READ
TRANSACTION_SERIALIZABLE
```

Any given database may not support all of these levels.

**What is resultset ?**

The `ResultSet` represents set of rows retrieved due to query execution.

```
ResultSet rs = stmt.executeQuery(sqlQuery);
```

**What are the types of resultsets?**

The values are defined in the class `java.sql.Connection` and are:

`TYPE_FORWARD_ONLY` specifies that a `resultset` is not scrollable, that is, rows within it can be advanced only in the forward direction.  
`TYPE_SCROLL_INSENSITIVE` specifies that a `resultset` is scrollable in either direction but is insensitive to changes committed by other transactions or other statements in the same transaction.  
`TYPE_SCROLL_SENSITIVE` specifies that a `resultset` is scrollable in either direction and is affected by changes committed by other transactions or statements within the same transaction.

**Note:** A `TYPE_FORWARD_ONLY` *resultset* is always *insensitive*.

**What's the difference between `TYPE_SCROLL_INSENSITIVE` and `TYPE_SCROLL_SENSITIVE`?**

<code>TYPE_SCROLL_INSENSITIVE</code>	<code>TYPE_SCROLL_SENSITIVE</code>
An insensitive <code>resultset</code> is like the snapshot of the data in the database when query was executed.	A sensitive <code>resultset</code> does NOT represent a snapshot of data, rather it contains points to those rows which satisfy the query condition.
After we get the <code>resultset</code> the changes made to data are not visible through the <code>resultset</code> , and hence they are known as insensitive.	After we obtain the <code>resultset</code> if the data is modified then such modifications are visible through <code>resultset</code> .
Performance not effected with insensitive.	Since a trip is made for every 'get' operation the performance drastically get affected.

**What is rowset?**

A `RowSet` is an object that encapsulates a set of rows from either Java Database Connectivity (JDBC) result sets or tabular data sources like a file or spreadsheet. `RowSets` support component-based development models like JavaBeans, with a standard set of properties and an event notification mechanism.

**What are the different types of RowSet ?**

There are two types of `RowSet` are there. They are:

**Connected** - A connected `RowSet` object connects to the database once and remains connected until the application terminates.

**Disconnected** - A disconnected `RowSet` object connects to the database, executes a query to retrieve the data from the database and then closes the connection. A program may change the data in a disconnected `RowSet` while it is disconnected. Modified data can be updated in the database after a disconnected `RowSet` reestablishes the connection with the database.

**What is the need of BatchUpdates?**

The BatchUpdates feature allows us to group SQL statements together and send to database server in one single trip.

**What is a DataSource?**

A DataSource object is the representation of a data source in the Java programming language. In basic terms,

A DataSource is a facility for storing data.

DataSource can be referenced by JNDI.

Data Source may point to RDBMS , file System , any DBMS etc..

**What are the advantages of DataSource?**

The few advantages of data source are :

An application does not need to hardcode driver information, as it does with the DriverManager.

The DataSource implementations can easily change the properties of data sources. *For example:*

There is no need to modify the application code when making changes to the database details.

The DataSource facility allows developers to implement a DataSource class to take advantage of features like connection pooling and distributed transactions.

**What is connection pooling? what is the main advantage of using connection pooling?**

A connection pool is a mechanism to reuse connections created. Connection pooling can increase performance dramatically by reusing connections rather than creating a new physical connection each time a connection is requested..

## SERVLETS

Earlier in client-server computing, each application had its own client program and it worked as a user interface and need to be installed on each user's personal computer. Most web applications use HTML/XHTML that are mostly supported by all the browsers and web pages are displayed to the client as static documents. A web page can merely displays static content and it also lets the user navigate through the content, but a web application provides a more interactive experience.

Any computer running **Servlets** or **JSP** needs to have a container. A **container** is nothing but a piece of software responsible for loading, executing and unloading the Servlets and **JSP**. While servlets can be used to extend the functionality of any Java-enabled server. They are mostly used to extend web servers, and are efficient replacement for **CGI** scripts. CGI was one of the earliest and most prominent server side dynamic content solutions, so before going forward it is very important to know the difference between **CGI** and the **Servlets**.

### Common Gateway Interface (CGI)

The Common Gateway Interface, which is normally referred as CGI, was one of the practical technique developed for creating dynamic content. By using the CGI, a web server passes requests to an external program and after executing the program the content is sent to the client as the output. In CGI when a server receives a request it creates a new process to run the CGI program, so creating a process for each request requires significant server resources and time, which limits the number of requests that can be processed concurrently. CGI applications are platform dependent. There is no doubt that CGI played a major role in the explosion of the Internet but its performance, scalability issues make it less than optimal solutions.

### Java Servlets

Java Servlet is a generic server extension that means a java class can be loaded dynamically to expand the functionality of a server. Servlets are used with web servers and run inside a Java Virtual Machine (JVM) on the server so these are safe and portable. Unlike applets they do not require support for java in the web browser. Unlike CGI, servlets don't use multiple processes to handle separate request. Servets can be handled by separate threads within the same process. Servlets are also portable and platform independent.

### Web Server Introduction

A web server is the combination of computer and the program installed on it. Web server interacts with the client through a web browser. It delivers the web pages to the client and to an application by using the web browser and the HTTP protocols respectively. We can also define the web server as the package of large number of programs installed on a computer connected to Internet or intranet for downloading the requested files using File Transfer Protocol, serving e-mail and building and publishing web pages. A web server works on a client server model. A computer connected to the Internet or intranet must have a server program. While talking about Java language then a web server is a server that is used to support the web component like the Servlet and JSP.

Note that the web server does not support to EJB (business logic component) component.

A computer connected to the Internet for providing the services to a small company or a departmental store may contain the HTTP server (to access and store the web pages and files), SMTP server (to support mail services), FTP server (for files downloading) and NNTP server (for newsgroup). The computer containing all the above servers is called the web server. Internet service providers and large companies may have all the servers like HTTP server, SMTP server, FTP server and many more on separate machines. In case of Java, a web server can be defined as the server that only supports to the web component like servlet and jsp. Notice that it does not support to the business component like EJB.

### Servlet Container

A servlet container is nothing but a compiled, executable program. The main function of the container is to load, initialize and execute servlets. The servlet container is the official Reference Implementation for the Java Servlet and JavaServer Pages technologies. The Java Servlet and JavaServer Pages specifications are developed by Sun under the Java Community Process.

A container handles large number of requests as it can hold many active servlets, listeners etc. It is interesting to note here that the container and the objects in a container are multithreaded. So each

object must be thread safe in a container as the multiple requests are being handled by the container due to the entrance of more than one thread to an object at a time.

Note : A Servlet container may run stand alone i.e. without a web server or even on another host.

We can categorize the servlet containers as:

I. A simple servlet container is not fully functional and therefore it can only run very simple servlets and does the following :

Wait for HTTP request.

Construct a ServletRequest object and a ServletResponse object.

If the request is for a static resource, invoke the process method of the StaticResourceProcessor instance, passing the ServletRequest and ServletResponse objects.

If the request is for a servlet, load the servlet class and invoke its service method, passing the ServletRequest and ServletResponse objects. Note that in this servlet container, the servlet class is loaded every time the servlet is requested.

II. A fully functional servlet container additionally does the following for each HTTP request for a servlet:

When the servlet is called for the first time, load the servlet class and call its init method (once only).

For each request, construct an instance of javax.servlet.ServletRequest and an instance of javax.servlet.ServletResponse.

Invoke the servlet's service method, passing the ServletRequest and ServletResponse objects.

When the servlet class is shut down, call the servlet's destroy method and unload the servlet class.

**Now lets see what a servlet container does for each HTTP request for a servlet, in general :**

The servlet container **loads the servlet class and calls the init method of the servlet** as soon as the servlet is called for the first time.

Then this container makes an instance of **javax.servlet.ServletRequest** and **javax.servlet.ServletResponse** for each request.

Then it passes the **ServletRequest** and **ServletResponse** objects by invoking the servlet's **service method**.

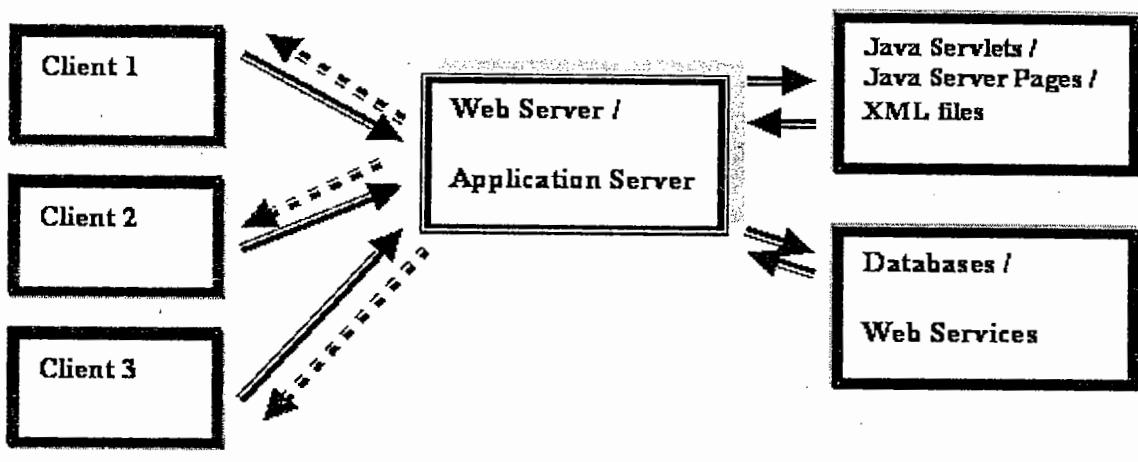
Finally, it calls the **destroy method** and unload the servlet class when the servlet class is to be shut down.

### **Introduction to Server Side Programming**

All of us (or most of us) would have started programming in Java with the ever famous "Hello World!" program. If you can recollect, we saved this file with a .java extension and later compiled the program using javac and then executed the class file with java. Apart from introducing you to the language basics, the point to be noted about this program is that - "It is a client side program". This means that you write, compile and also execute the program on a client machine (e.g. Your PC). No doubt, this is the easiest and fastest way to write, compile and execute programs. But, it has little practical significance when it comes to real world programming.

#### **1. Why Server Side Programming?**

Though it is technically feasible to implement almost any business logic using client side programs, logically or functionally it carries no ground when it comes to enterprise applications (e.g. banking, air ticketing, e-shopping etc.). To further explain, going by the client side programming logic; a bank having 10,000 customers would mean that each customer should have a copy of the program(s) in his or her PC which translates to 10,000 programs! In addition, there are issues like security, resource pooling, concurrent access and manipulations to the database which simply cannot be handled by client side programs. The answer to most of the issues cited above is ~ "Server Side Programming". Figure-1 illustrates Server side architecture in the simplest terms.

**Figure – Server Side Programming (architecture)**

## 2. Advantages of Server Side Programs

The list below highlights some of the important advantages of Server Side programs.

- i. All programs reside in one machine called the Server. Any number of remote machines (called clients) can access the server programs.
- ii. New functionalities to existing programs can be added at the server side which the clients' can advantage without having to change anything from their side.
- iii. Migrating to newer versions, architectures, design patterns, adding patches, switching to new databases can be done at the server side without having to bother about clients' hardware or software capabilities.
- iv. Issues relating to enterprise applications like resource management, concurrency, session management, security and performance are managed by service side applications.
- v. They are portable and possess the capability to generate dynamic and user-based content (e.g. displaying transaction information of credit card or debit card depending on user's choice).

## 3. Types of Server Side Programs

- i. Active Server Pages (ASP)
- ii. Java Servlets
- iii. Java Server Pages (JSPs)
- iv. Enterprise Java Beans (EJBs)
- v. PHP

To summarize, the objective of server side programs is to centrally manage all programs relating to a particular application (e.g. Banking, Insurance, e-shopping, etc). Clients with bare minimum requirement (e.g. Pentium II, Windows XP Professional, MS Internet Explorer and an internet connection) can experience the power and performance of a Server (e.g. IBM Mainframe, Unix Server, etc) from a remote location without having to compromise on security or speed. More importantly, server programs are not only portable but also possess the capability to generate dynamic responses based on user's request.

### What is Java Servlets?

Servlets are server side components that provide a powerful mechanism for developing server side programs. Servlets provide component-based, platform-independent methods for building Web-based applications, without the performance limitations of CGI programs. Unlike proprietary server extension mechanisms (such as the Netscape Server API or Apache modules), servlets are server as well as platform-independent. This leaves you free to select a "best of breed" strategy for your servers, platforms, and tools. Using servlets web developers can create fast and efficient server side application which can run on any servlet enabled web server. Servlets run entirely inside the Java Virtual Machine. Since the Servlet runs at server side so it does not checks the browser for compatibility. Servlets can access the entire family of Java APIs, including the JDBC API to access enterprise databases. Servlets can also access a library of HTTP-specific calls, receive all the benefits of the mature java language including portability, performance, reusability, and crash protection. Today servlets are the popular choice for building interactive web applications. Third-party servlet containers are available for Apache Web Server, Microsoft

IIS, and others. Servlet containers are usually the components of web and application servers, such as BEA WebLogic Application Server, IBM WebSphere, Sun Java System Web Server, Sun Java System Application Server and others.

Servlets are not designed for a specific protocols. It is different thing that they are most commonly used with the HTTP protocols. Servlets uses the classes in the `java` packages `javax.servlet` and `javax.servlet.http`. Servlets provides a way of creating the sophisticated server side extensions in a server as they follow the standard framework and use the highly portable java language.

HTTP Servlet typically used to:

- Priovide dynamic content like getting the results of a database query and returning to the client.
- Process and/or store the data submitted by the HTML.
- Manage information about the state of a stateless HTTP. e.g. an online shopping car manages request for multiple concurrent customers.

### **Features of Servlets 2.4**

In this tutorial you will learn the new features added in Servlet 2.4.

1. **Upgraded supports for Http, J2SE, and J2EE:** Servlet 2.4 depends on Http1.1 and J2SE 1.3.
2. **Additional ServletRequest methods :** In Servlet 2.4 four new methods are added in the `ServletRequest`
  - o `getRemotePort()`: It returns the IP source port of the client.
  - o `getLocalName()`: It returns the host name on which the request was received.
  - o `getLocalAddr()`: It returns the IP address on which the request was received.
  - o `getLocalPort()`: It returns the IP port number.
3. **New Support for Internationalization and charset choice:** To provide support of internationalization, Servlet 2.4 has added two new methods in the `ServletResponse` interface.
  - o **setCharacterEncoding(String encoding):** The purpose of this method is to set the response's character encoding. This method helps us to pass a charset parameter to `setContentType(String)` or passing a `Locale` to `setLocale(Locale)`. We can now avoid setting the charset in the `setContentType("text/html;charset=UTF-8")` as `setCharacterEncoding()` method pairs with the pre-existing `getCharacterEncoding()` method to manipulate and view the response's character encoding.
  - o **getContentType():** It is responsible for returning the response's content type. The content type can be dynamically set with a combination of `setContentType()`, `setLocale()`, and `setCharacterEncoding()` calls, and the method `getContentType()` provides a way to view the generated type string.
4. **New features has been added in RequestDispatcher:** In Servlet 2.4 five new request attributes has been added for providing extra information during a `RequestDispatcher.forward()` call. This features has been added in Servlet 2.4 to know the true original request URI. The following request attributes are:
  - o `javax.servlet.forward.request_uri`
  - o `javax.servlet.forward.context_path`
  - o `javax.servlet.forward.servlet_path`
  - o `javax.servlet.forward.path_info`
  - o `javax.servlet.forward.query_string`
5. **SingleThreadModel interface has been deprecated:** In Servlet 2.4 the `SingleThreadModel` interface has been deprecated.
6. **HttpSession details and interaction with logins has been clarified:** The new method `HttpSession.logout()` has been added in Servlet 2.4. Now session allows zero or negative values in the `<session-timeout>` element to indicate sessions should never time out.

If the object in the session can't be serialize in a distributed environment then it must throw an `IllegalArgumentException`.

7. **Welcome file behavior and Classloading has been clarified:** In servlet 2.4 welcome file can be a servlet.
8. **The web.xml file now uses XML Schema:** Version 2.4 `servers` must still accept the 2.2 and 2.3 deployment descriptor formats, but all new elements are solely specified in Schema.

### Features of Servlet 2.5

This version has been released on **September 26, 2005** by the **Sun MicroSystems**. It is not necessary that all web servers and application servers support the features of **Servlet 2.5**. Still most of the popular containers like **Tomcat 5.5** and **JBoss 4.0** support **Servlet 2.4**.

The list of the added features is given below:

1. **Dependency on J2SE 5.0:** The minimum platform requirement for Servlet 2.5 is JDK 1.5. Servlet 2.5 can't be used in versions below than JDK1.5. All the available features of JDK1.5 like generics, autoboxing, an improved for loop etc are guaranteed available to Servlet 2.5 programmers.
2. **Support For annotations:** Annotations provide a mechanism for decorating java code constructs (classes, methods, fields, etc.) with metadata information. Annotations are mark code in such a way that code processors may alter their behavior based on the metadata information.
3. **Several web.xml convenience:** Servlet 2.5 introduces several small changes to the web.xml file to make it more convenient to use. For example while writing a <filter-mapping>, we can now use an asterisk in a <servlet-name> which will represent all servlets as well as JSP. Previously we used to do

```
<filter-mapping>
<filter-name>FilterName</filter-name>
<servlet-name>FilterName</servlet-name>
</filter-mapping>
```

Now,

```
<filter-mapping>
<filter-name>FilterName</filter-name>
<servlet-name>*</servlet-name>
</filter-mapping>
```

Previously in <servlet-mapping> or <filter-mapping> there used to be only one <url-pattern>, but now we can have multiple <url-pattern>, like

```
<servlet-mapping>
  <servlet-name>abc</servlet-name>
  <url-pattern>/abc/*</url-pattern>
  <url-pattern>/abc/*</url-pattern>
</servlet-mapping>
```

Apart from these changes, many more facilities added in web.xml.

4. **A Handful of removed restrictions:** Servlet 2.5 removed a few restrictions around error handling and session tracking. Now it has removed the restriction that the <error-page> could not call the setStatus() method to alter the error code that triggered them. In session tracking, Servlet 2.5 eased a rule that a servlet called by RequestDispatcher include() couldn't set response headers.
5. **Some edge case clarifications:** The servlet 2.4 specification says that before calling request.getReader() we must call request.setCharacterEncoding(). However there is no such clarification given why it is so.

### Advantages of Java Servlets

1. **Portability**
2. **Powerful**
3. **Efficiency**
4. **Safety**
5. **Integration**
6. **Extensibility**
7. **Inexpensive**

Each of the points are defined below:

#### **Portability**

As we know that the servlets are written in java and follow well known standardized APIs so they are highly portable across operating systems and server implementations. We can develop a servlet on Windows machine running the tomcat server or any other server and later we can deploy that servlet effortlessly on any other operating system like Unix server running on the iPlanet/Netscape Application server. So servlets are **write once, run anywhere (WORA)** program.

#### **Powerful**

We can do several things with the servlets which were difficult or even impossible to do with CGI, for example the servlets can talk directly to the web server while the CGI programs can't do. Servlets can share data among each other, they even make the database connection pools easy to implement. They can maintain the session by using the session tracking mechanism which helps them to maintain information from request to request. It can do many other things which are difficult to implement in the CGI programs.

#### **Efficiency**

As compared to CGI the servlets invocation is highly efficient. When the servlet get loaded in the server, it remains in the server's memory as a single object instance. However with servlets there are N threads but only a single copy of the servlet class. Multiple concurrent requests are handled by separate threads so we can say that the servlets are highly scalable.

#### **Safety**

As servlets are written in java, servlets inherit the strong type safety of java language. Java's automatic garbage collection and a lack of pointers means that servlets are generally safe from memory management problems. In servlets we can easily handle the errors due to Java's exception handling mechanism. If any exception occurs then it will throw an exception.

#### **Integration**

Servlets are tightly integrated with the server. Servlet can use the server to translate the file paths, perform logging, check authorization, and MIME type mapping etc.

#### **Extensibility**

The servlet API is designed in such a way that it can be easily extensible. As it stands today, the servlet API support Http Servlets, but in later date it can be extended for another type of servlets.

#### **Inexpensive**

There are number of free web servers available for personal use or for commercial purpose. Web servers are relatively expensive. So by using the free available web servers you can add servlet support to it.

#### **What is Servlet?**

- Java™ objects which are based on servlet framework and APIs and extend the functionality of a HTTPServer.
- Mapped to URLs and managed by container with a simple architecture
- Available and running on all major web servers and app servers
- Platform and server independent

#### **CGI**

- Written in C, C++, Visual Basic and Perl
- Difficult to maintain, non-scalable, nonmanageable
- Prone to security problems of programming language
- Resource intensive and inefficient
- Platform and application-specific

#### **Servlet**

- Written in Java
- Powerful, reliable and efficient

- Improves scalability, reusability (component based)
- Leverages built-in security of Java programming language
- Platform independent and portable

### **Advantages of Servlet**

- No CGI limitations
- Abundant third-party tools and Web servers supporting Servlet
- Access to entire family of Java APIs
- Reliable, better performance and scalability
- Platform and server independent
- Secure
- Most servers allow automatic reloading Servlet's by administrative action.

### **What is JSP Technology?**

- Enables separation of business logic from presentation
- Presentation is in the form of HTML or XML/XSLT
- Business logic is implemented as JavaBeans or custom tags
- Better maintainability, reusability
- Extensible via custom tags
- Builds on Servlet technology

### **What is JSP page?**

- A text-based document capable of returning dynamic content to a client browser
- Contains both static and dynamic content
- Static content: HTML, XML
- Dynamic content: programming code, and JavaBeans, custom tags

### **Servlets and JSP – Comparison**

Servlets	JSP
<ul style="list-style-type: none"> <li>- HTML code in java</li> <li>- Any form of Data</li> <li>- Not easy to author a web page</li> </ul>	<ul style="list-style-type: none"> <li>- Java-like code in HTML</li> <li>- Structured Text</li> <li>- Very easy to author a web page</li> <li>- Code is compiled into a servlet</li> </ul>

### **JSP Benefits**

- Content and display logic are separated
- Simplify development with JSP, JavaBeans and custom tags
- Supports software reuse through the use of components
- Recompile automatically when changes are made to the source file
- Easier to author web pages
- Platform-independent

### **When to use Servlet over JSP**

- Extend the functionality of a Web server such as supporting a new file format
- Generate objects that do not contain HTML such as graphs or pie charts
- Avoid returning HTML directly from your servlets whenever possible

### **Should I Use Servlet or JSP?**

- In practice, servlet and JSP are used together
- via MVC (Model, View, Controller) architecture
- Servlet handles Controller
- JSP handles View

### **What does Servlet Do?**

- Receives client request (mostly in the form of HTTP request)
- Extract some information from the request
- Do content generation or business logic process (possibly by accessing database, invoking EJBs, etc)
- Create and send response to client (mostly in the form of HTTP response) or forward the request to another servlet or JSP page

### **Requests and Responses**

- What is a request?
  - . Information that is sent from client to a server
    - Who made the request
    - What user-entered data is sent

- Which HTTP headers are sent
- What is a response?
  - . Information that is sent to client from a server
    - Text(html, plain) or binary(image) data
    - HTTP headers, cookies, etc

**HTTP**

- HTTP request contains
  - . header
  - . a method
    - Get: Input form data is passed as part of URL
    - Post: Input form data is passed within message body
    - Put
    - Header
  - . request data

**HTTP GET and POST**

- The most common client requests
  - . HTTP GET & HTTP POST
- GET requests:
  - . User entered information is appended to the URL in a query string
  - . Can only send limited amount of data
    - . .... /servlet/ViewCourse?FirstName=Sang&LastName=Shin
- POST requests:
  - . User entered information is sent as data (not appended to URL)
  - . Can send any amount of data

**Servlet Life Cycle Methods**

- . Invoked by container
  - Container controls life cycle of a servlet
- . Defined in
  - javax.servlet.GenericServlet class or
    - . init()
    - . destroy()
    - . service() - this is an abstract method
  - javax.servlet.http.HttpServlet class
    - . doGet(), doPost(), doXxx()
    - . service() - implementation

**1. init()**

- Invoked once when the servlet is first instantiated
- Perform any set-up in this method
  - . Setting up a database connection

**2. destroy()**

- Invoked before servlet instance is removed
- Perform any clean-up
  - . Closing a previously created database connection

**3. service() & doGet() / doPost()**

- . service() methods take generic requests and responses:
  - service(ServletRequest request, ServletResponse response)
- . doGet() or doPost() take HTTP requests and responses:
  - doGet(HttpServletRequest request, HttpServletResponse response)
  - doPost(HttpServletRequest request, HttpServletResponse response)
- . Things You Do in doGet() & doPost()
  - Extract client-sent information (HTTP parameter) from HTTP request
  - Set (Save) and get (read) attributes to/from Scope objects
  - Perform some business logic or access database
  - Optionally forward the request to other Web components (Servlet/JSP)
  - Populate HTTP response message and send it to client

**Scope Objects**

- Enables sharing information among collaborating web components via attributes maintained in Scope objects
- Attributes maintained in the Scope objects are accessed with
  - . getAttribute()
  - . setAttribute()
- Four Scope objects are defined
- Web context, session, request, page

**Four Scope Objects: Accessibility**

- . Web context (ServletContext)
  - Accessible from Web components within a Web context
- . Session
  - Accessible from Web components handling a request that belongs to the session
- . Request
  - Accessible from Web components handling the request
- . Page
  - Accessible from JSP page that creates the object

**Four Scope Objects: Class**

- . Web context
  - javax.servlet.ServletContext
- . Session
  - javax.servlet.http.HttpSession
- . Request
  - subtype of javax.servlet.ServletRequest:
  - javax.servlet.http.HttpServletRequest
- . Page
  - javax.servlet.jsp.PageContext

**What is ServletContext For?**

- . Used by servlets to
  - Set and get context-wide (application-wide) object-valued attributes
  - Get request dispatcher . To forward to or include web component
  - Access Web context-wide initialization parameters set in the web.xml file
  - Access Web resources associated with the Web context
  - Log
  - Access other misc. information

**Scope of ServletContext**

- . Context-wide scope
  - o Shared by all servlets and JSP pages within a "web application"
    - Why it is called "web application scope"
  - o A "web application" is a collection of servlets and content installed under a specific subset of the server's URL namespace and possibly installed via a \*.war file
    - 1. All servlets in BookStore web application share same ServletContext object
  - o There is one ServletContext object per "web application" per Java Virtual Machine

**How to Access ServletContext Object?**

- o Within your servlet code, call getServletContext()
- o Within your servlet filter code, call getServletContext()
- o The ServletContext is contained in ServletConfig object, which the Web server provides to a servlet when the servlet is initialized
  - init (ServletConfig servletConfig) in Servlet interface

**Why HttpSession?**

- o Need a mechanism to maintain client state across a series of requests from a same user (or originating from the same browser) over some period of time
  - Example: Online shopping cart
- o Yet, HTTP is stateless
- o HttpSession maintains client state
  - Used by Servlets to set and get the values of session scope attributes

**How to Get HttpSession?**

- o via getSession() method of a Request object (HttpServletRequest)

**What is Servlet Request?**

- o Contains data passed from client to servlet
- o All servlet requests implement ServletRequest interface which defines methods for accessing
  - Client sent parameters
  - Object-valued attributes
  - Locales

- Client and server
- Input stream
- Protocol information
- Content type
- If request is made over secure channel (HTTPS)

#### **Request attributes can be set in two ways**

- Servlet container itself might set attributes to make available custom information about a request
  - o Example: javax.servlet.request.X509Certificate attribute for HTTPS
- Servlet set application-specific attribute
  - o void setAttribute(java.lang.String name, java.lang.Object o)
  - o Embedded into a request before a RequestDispatcher call

#### **Getting Client Information**

- String request.getRemoteAddr()..... Get client's IP address
- String request.getRemoteHost().....Get client's host name

#### **Getting Server Information**

String request.getServerName()..... e.g. "www.sun.com"  
 int request.getServerPort()..... e.g. Port number "8080"

#### **Getting Misc. Information**

- . Input stream
  - ServletInputStream getInputStream()
  - java.io.BufferedReader getReader()
- . Protocol
  - java.lang.String getProtocol()
- . Content type
  - java.lang.String getContentType()
- . Is secure or not (if it is HTTPS or not)
  - boolean isSecure()

#### **What is HTTP Servlet Request?**

- Contains data passed from HTTP client to HTTP servlet
- Created by servlet container and passed to servlet as a parameter of doGet() or doPost() methods
- HttpServletRequest is an extension of ServletRequest and provides additional methods for accessing
  - HTTP request URL
    - . Context, servlet, path, query information
  - Misc. HTTP Request header information
  - Authentication type & User security information
  - Cookies
  - Session

#### **HTTP Request URL**

- . Contains the following parts
  - http://[host]:[port]/[request path]?[query string]

#### **HTTP Request URL: [request path]**

- . http://[host]:[port]/[request path]?[query string]
- . [request path] is made of
  - Context: /<context of web app>
  - Servlet name: /<component alias>
  - Path information: the rest of it
- . Examples
  - http://localhost:8080/hello1/greeting
  - http://localhost:8080/hello1/greeting.jsp
  - http://daydreamer/catalog/lawn/index.html

#### **HTTP Request URL: [query string]**

http://[host]:[port]/[request path]?[query string]  
 [query string] are composed of a set of parameters and values that are user entered  
 Two ways query strings are generated
 

- A query string can explicitly appear in a web page

- . <a href="/bookstore1/catalog?Add=101">Add To Cart</a>
  - . String bookId = request.getParameter("Add");
- A query string is appended to a URL when a form with a GET HTTP method is submitted
  - . http://localhost/hello1/greeting?username=Monica+Clinton
  - . String userName=request.getParameter("username")

**Context, Path, Query, Parameter Methods**

- . String getServletContext()
- . String getQueryString()
- . String getPathInfo()
- . String getPathTranslated()

**HTTP Request Headers**

- HTTP requests include headers which provide extra information about the request
- Example of HTTP 1.1 Request:GET /search?keywords= servlets+ jsp HTTP/ 1.1
  - o Accept: image/ gif, image/ jpg, \*/\*
  - o Accept-Encoding: gzip
  - o Connection: Keep- Alive
  - o Cookie: userID= id456578
  - o Host: www.sun.com
  - o Referer: http://www.sun.com/codecamp.html
  - o User-Agent: Mozilla/ 4.7 [en] (Win98; U)
- Accept
  - o Indicates MIME types browser can handle.
- Accept-Encoding
  - o Indicates encoding (e. g., gzip or compress) browser can handle
- Authorization
  - o User identification for password- protected pages
  - o Instead of HTTP authorization, use HTML forms to send username/password and store info in session object
- Connection
  - o In HTTP 1.1, persistent connection is default
  - o Servlets should set Content-Length with setContentLength (use ByteArrayOutputStream to determine length of output) to support persistent connections.
- Cookie
  - o Gives cookies sent to client by server sometime earlier. Use getCookies, not getHeader
- Host
  - o Indicates host given in original URL.
  - o This is required in HTTP 1.1.
- If-Modified-Since
  - o Indicates client wants page only if it has been changed after specified date.
  - o Don't handle this situation directly; implement getLastModified instead.
- Referer
  - o URL of referring Web page.
  - o Useful for tracking traffic; logged by many servers.
- User-Agent
  - o String identifying the browser making the request.
  - o Use with extreme caution!

**HTTP Header Methods**

- o String getHeader(java.lang.String name)
  - value of the specified request header as String
- o java.util.Enumeration getHeaders(java.lang.String name)
  - values of the specified request header
- o java.util.Enumeration getHeaderNames()
  - names of request headers
- o int getIntHeader(java.lang.String name)
  - value of the specified request header as an int

**Authentication & User Security Information Methods**

- o String getRemoteUser()
  - name for the client user if the servlet has been password protected, null otherwise
- o String getAuthType()
  - name of the authentication scheme used to protect the servlet

- o boolean isUserInRole(java.lang.String role)
  - Is user included in the specified logical "role"?
- o String getRemoteUser()
  - login of the user making this request, if the user has been authenticated, null otherwise

**Cookie Method (in HttpServletRequest)**

- Cookie[] getCookies()
  - o an array containing all of the Cookie objects the client sent with this request

**What is Servlet Response?**

- . Contains data passed from servlet to client
- . All servlet responses implement ServletResponse interface
  - Retrieve an output stream
  - Indicate content type
  - Indicate whether to buffer output
  - Set localization information
- . HttpServletResponse extends ServletResponse
  - HTTP response status code
  - Cookies

**HTTP Response Status Codes**

- . Why do we need HTTP response status code?
  - Forward client to another page
  - Indicates resource is missing
  - Instruct browser to use cached copy

**Methods for Setting HTTPResponse Status Codes**

- . public void setStatus(int statusCode)
  - Status codes are defined in HttpServletResponse
  - Status codes are numeric fall into five general categories:
- . 100-199 Informational
- . 200-299 Successful
- . 300-399 Redirection
- . 400-499 Incomplete
- . 500-599 Server Error
  - Default status code is 200 (OK)

**Common Status Codes**

- . 200 (SC\_OK)
  - Success and document follows
  - Default for servlets
- . 204 (SC\_No\_CONTENT)
  - Success but no response body
  - Browser should keep displaying previous document
- . 301 (SC\_MOVED\_PERMANENTLY)
  - The document moved permanently (indicated in Location header)
  - Browsers go to new location automatically

**Common Status Codes**

- . 302 (SC\_MOVED\_TEMPORARILY)
  - o Note the message is "Found"
  - o Requested document temporarily moved elsewhere (indicated in Location header)
  - o Browsers go to new location automatically
  - o Servlets should use sendRedirect, not setStatus, when setting this header
- . 401 (SC\_UNAUTHORIZED)
  - o Browser tried to access password-protected page without proper Authorization header
- . 404 (SC\_NOT\_FOUND)
  - No such page

**Methods for Sending Error**

- . Error status codes (400-599) can be used in sendError methods.
- . public void sendError(int sc)
  - The server may give the error special treatment
- . public void sendError(int code, String message)
  - Wraps message inside small HTML document

**Header in Http Response****Why HTTP Response Headers?**

- o . Give forwarding location
- o . Specify cookies
- o . Supply the page modification date
- o . Instruct the browser to reload the page after a designated interval
- o . Give the file size so that persistent HTTP connections can be used
- o . Designate the type of document being generated

**Methods for Setting ArbitraryResponse Headers**

- o public void setHeader( String headerName, String headerValue)
  - Sets an arbitrary header.
- o public void setDateHeader( String name, long millisecs)
  - Converts milliseconds since 1970 to a date string in GMT format
- o public void setIntHeader( String name, int headerValue)
  - Prevents need to convert int to String before calling setHeader
- o addHeader, addDateHeader, addIntHeader
  - Adds new occurrence of header instead of replacing.

**Methods for setting Common Response Headers**

- setContentType
  - Sets the Content- Type header. Servlets almost always use this.
- setContentLength
  - Sets the Content- Length header. Used for persistent HTTP connections.
- addCookie
  - Adds a value to the Set- Cookie header.
- sendRedirect
  - Sets the Location header and changes status code.

**Common HTTP 1.1 Response Headers**

- . Location
  - o Specifies a document's new location.
  - o Use sendRedirect instead of setting this directly.
- . Refresh
  - o Specifies a delay before the browser automatically reloads a page.
- . Set-Cookie
  - o The cookies that browser should remember. Don't set this header directly.
  - o use addCookie instead.
- . Cache-Control (1.1) and Pragma (1.0)
  - A no-cache value prevents browsers from caching page. Send both headers or check HTTP version.
- . Content- Encoding
  - The way document is encoded. Browser reverses this encoding before handling document.
- . Content- Length
  - The number of bytes in the response. Used for persistent HTTP connections.
- . Content- Type
  - The MIME type of the document being returned.
  - Use setContentType to set this header.
- . Last- Modified
  - The time document was last changed
  - Don't set this header explicitly.
  - provide a getLastModified method instead.

**Body in Http Response****Writing a Response Body**

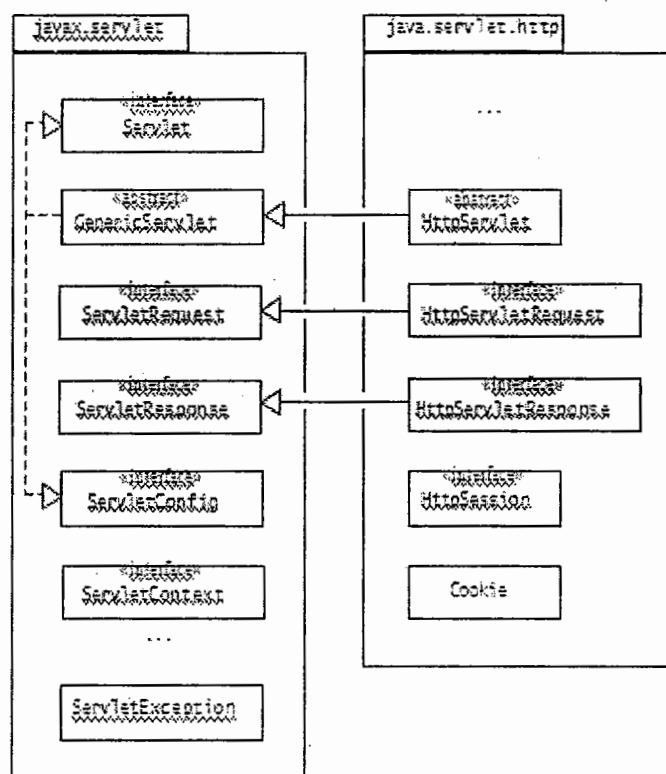
- . A servlet almost always returns a response body
- . Response body could either be a PrintWriter or a ServletOutputStream
- . PrintWriter
  - Using response.getWriter()
  - For character-based output
- . ServletOutputStream
  - Using response.getOutputStream()
  - For binary (image) data

**Handling Errors**

- . Web container generates default error page
- . You can specify custom default page to be displayed instead
- . Steps to handle errors
  - Create appropriate error html pages for error conditions
  - Modify the web.xml accordingly

**Example: Setting Error Pages in web.xml**

```
<error-page>
<exception-type>exception.BookNotFoundException</exception-type>
<location>/errorpage1.html</location>
</error-page>
<error-page>
<exception-type>exception.BooksNotFoundException</exception-type>
<location>/errorpage2.html</location>
</error-page>
<error-page>
<exception-type>exception.OrderException</exception-type>
<location>/errorpage3.html</location>
</error-page>
```

**Key Classes and Interfaces of the Servlet API**

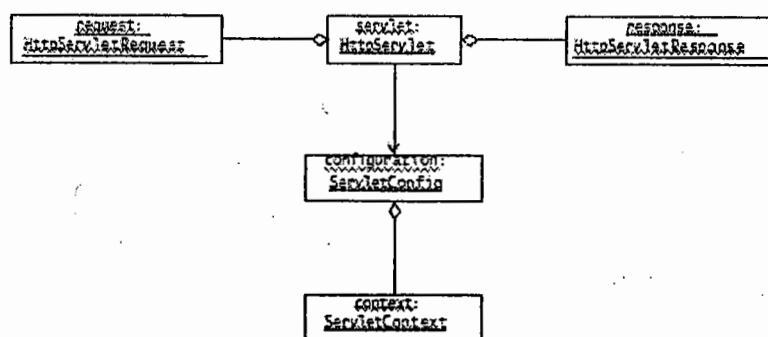
## The `javax.servlet` Package

Class/Interface	Description
<code>interface Servlet</code>	The main interface that every servlet must implement. It defines the key methods that a servlet container calls to control the servlet.
<code>abstract class GenericServlet</code>	This abstract class can be used as the starting point for implementing servlets. In particular, it implements all the methods of the <code>Servlet</code> interface except the <code>service()</code> method. This abstract class also implements the <code>ServletConfig</code> interface which allows the servlet container to pass information to the servlet.
<code>interface ServletRequest</code>	This interface provides the methods to extract information from a client request.
<code>interface ServletResponse</code>	This interface provides the methods to create and send an appropriate response to a client request.
<code>interface ServletConfig</code>	This interface which allows the servlet container to pass information to a servlet.
<code>interface ServletContext</code>	This interface allows the servlet to communicate with its container.
<code>class ServletException</code>	A general exception class to signal servlet runtime errors.

## The `javax.servlet.http` Package

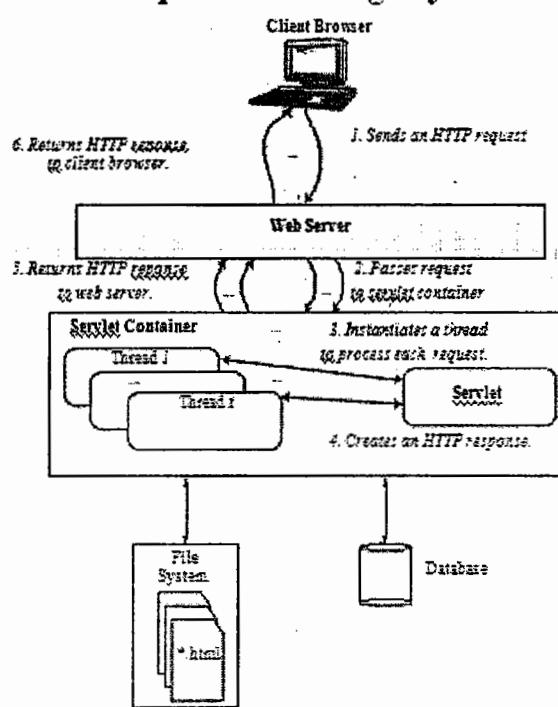
Class/Interface	Description
<code>abstract class HttpServlet</code>	This abstract class extends the <code>GenericServlet</code> class and is used for implementing HTTP servlets, i.e. servlets which use HTTP for requests and responses. In particular, it provides stubs for the <code>doHttpMethodName()</code> methods which correspond to the HTTP method used in the request (GET, POST, HEAD, etc.). A concrete servlet can override the appropriate methods to handle the different HTTP request methods.
<code>interface HttpServletRequest</code>	This interface extends the <code>ServletRequest</code> interface to handle HTTP requests.
<code>interface HttpServletResponse</code>	This interface extends the <code>ServletResponse</code> interface to create and send an appropriate HTTP response to an HTTP request.
<code>interface HttpSession</code>	This interface provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.
<code>class Cookie</code>	This class provides support for cookies to be used in requests and responses.

### Core HTTP Servlet Object Diagram



*How the objects in the above diagram function and interact is essential to developing and deploying servlets.*

### Request Handling Cycle



#### Methods used in servlet

A Generic servlet contains the following five methods:  
**init()**

**public void init(ServletConfig config) throws ServletException**

The **init() method** is called only once by the servlet container throughout the life of a servlet. By this init() method the servlet gets to know that it has been placed into service.

The servlet cannot be put into the service if:

The init() method does not return within a fixed time set by the web server.

It throws a ServletException

Parameters - The init() method takes a **ServletConfig** object that contains the initialization parameters and servlet's configuration and throws a **ServletException** if an exception has occurred.

#### **service()**

```
public void service(ServletRequest req, ServletResponse res) throws ServletException,
IOException
```

Once the servlet starts getting the requests, the service() method is called by the servlet container to respond. The servlet services the client's request with the help of two objects. These two objects **javax.servlet.ServletRequest** and **javax.servlet.ServletResponse** are passed by the servlet container.

The status code of the response always should be set for a servlet that throws or sends an error.

Parameters - The service() method takes **the ServletRequest** object that contains the client's request and the object **ServletResponse** contains the servlet's response. The service() method throws **ServletException** and **IOExceptions** exception.

#### **getServletConfig()**

```
public ServletConfig getServletConfig()
```

This method contains parameters for initialization and startup of the servlet and returns a -----

**ServletConfig object.** This object is then passed to the init method. When this interface is implemented then it stores **the ServletConfig object** in order to return it. It is done by the generic class which implements this interface.

Returns - the ServletConfig object

#### **getServletInfo()**

```
public String getServletInfo()
```

The information about the servlet is returned by this method like version, author etc. This method returns a string which should be in the form of plain text and not any kind of markup.

Returns - a string that contains the information about the servlet

#### **destroy()**

```
public void destroy()
```

This method is called when we need to close the servlet. That is before removing a servlet instance from service, the servlet container calls the destroy() method. Once the servlet container calls the destroy() method, no service methods will then be called . That is after the exit of all the threads running in the servlet, the destroy() method is called. Hence, the servlet gets a chance to clean up all the resources like memory, threads etc which are being held.

#### **Life Cycle of a Servlet**

The life cycle of a servlet can be categorized into four parts:

1. **Loading and Instantiation:** The servlet container loads the servlet during startup or when the first request is made. The loading of the servlet depends on the attribute `<load-on-startup>` of web.xml file. If the attribute `<load-on-startup>` has a positive value then the servlet is loaded with loading of the container otherwise it loads when the first request comes for service. After loading of the servlet, the container creates the instances of the servlet.
2. **Initialization:** After creating the instances, the servlet container calls the init() method and passes the servlet initialization parameters to the init() method. The init() must be called by the servlet container before the servlet can service any request. The initialization parameters persist until the servlet is destroyed. The init() method is called only once throughout the life cycle of the servlet. The servlet will be available for service if it is loaded successfully otherwise the servlet container unloads the servlet.
3. **Servicing the Request:** After successfully completing the initialization process, the servlet will be available for service. Servlet creates separate threads for each request. The servlet container calls the service() method for servicing any request. The service() method determines the kind of

request and calls the appropriate method (`doGet()` or `doPost()`) for handling the request and sends response to the client using the methods of the response object.

4. **Destroying the Servlet:** If the servlet is no longer needed for servicing any request, the servlet container calls the `destroy()` method. Like the `init()` method this method is also called only once throughout the life cycle of the servlet. Calling the `destroy()` method indicates to the servlet container not to send the any request for service and the servlet releases all the resources associated with it. Java Virtual Machine claims for the memory associated with the resources for garbage collection.

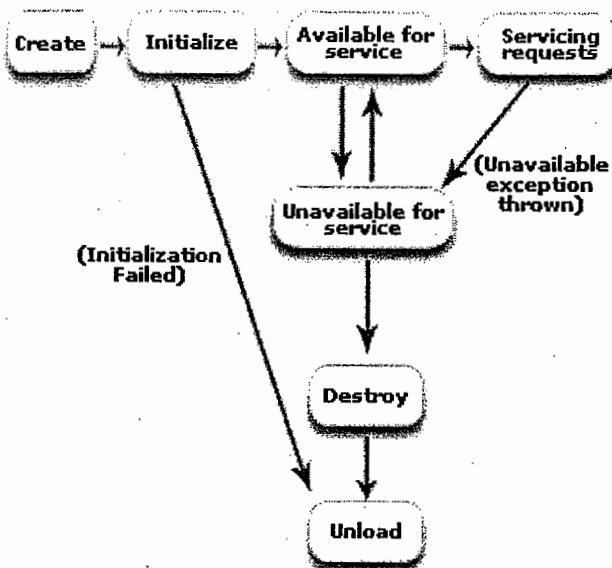


Figure:Life Cycle of a Servlet

**LifeCycle:-** The procedure followed by the technology to execute an application. The various stages that arise at the runtime when the application is under execution can be called as life cycle.

**Note:**

Servlets has three lifecycle methods and they are defined in Servlet Interface.

The three lifecycle methods of servlet are:-

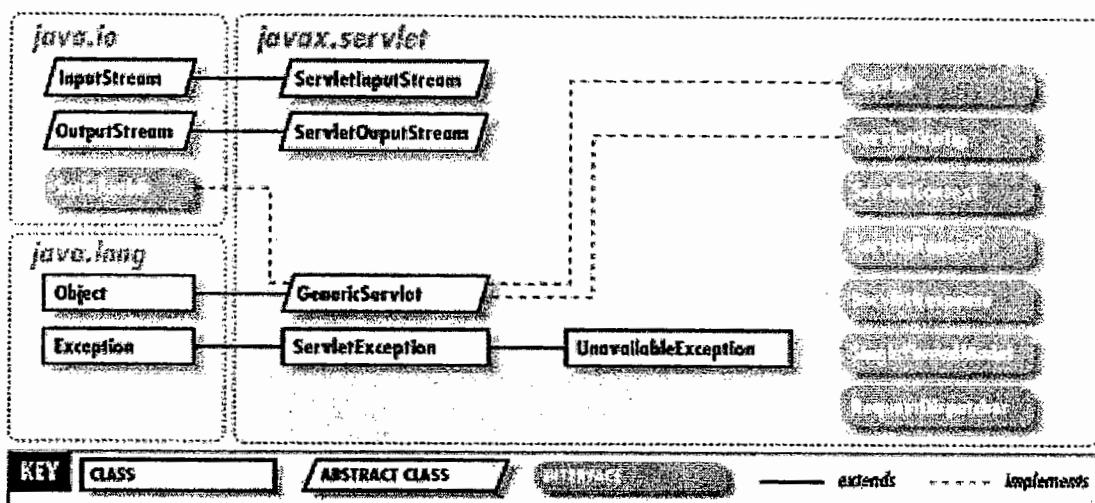
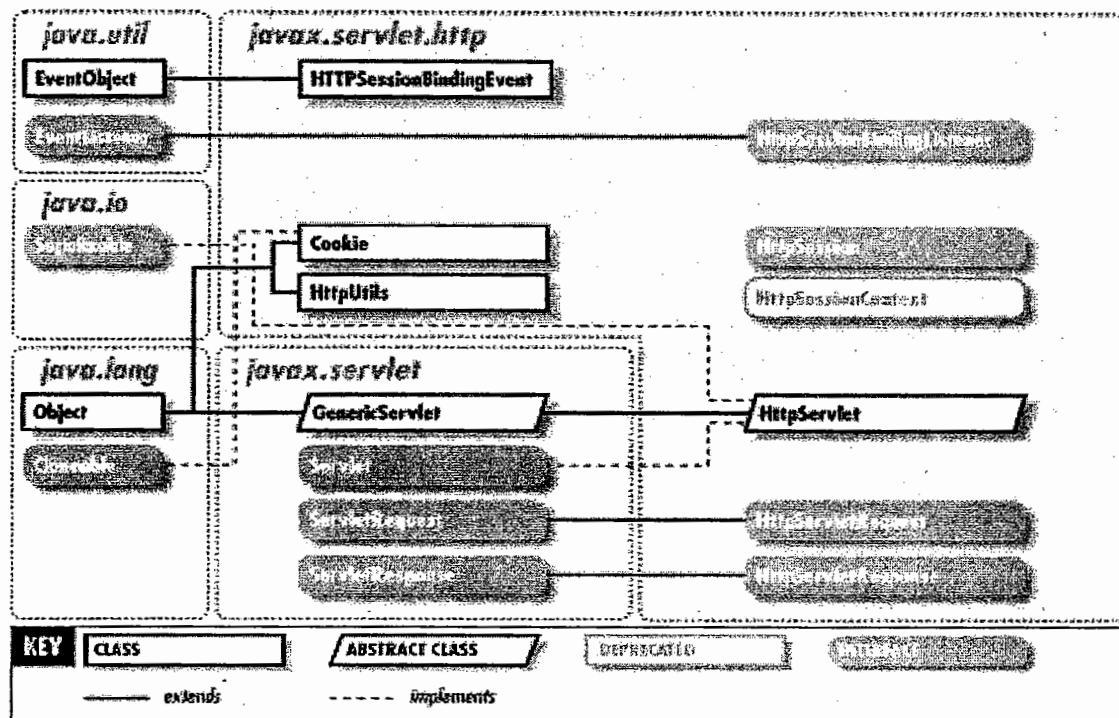
- 1.`init( )`
- 2.`service( )`
- 3.`destroy( )`

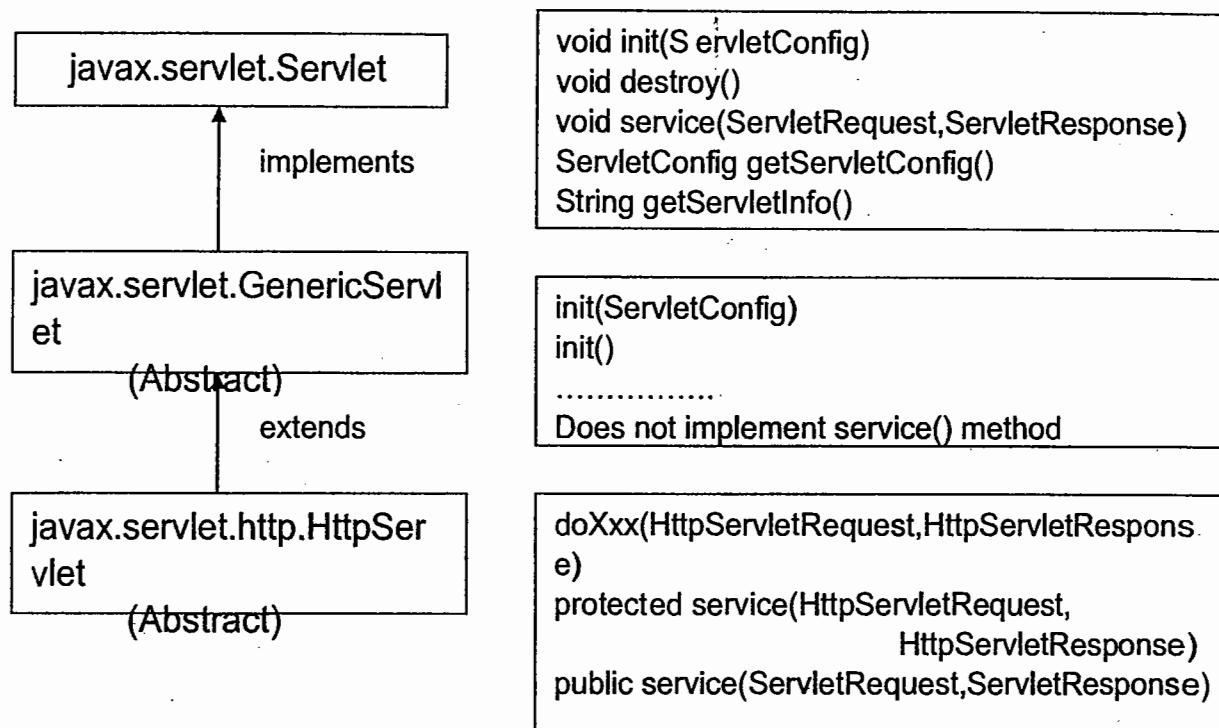
**1. `init():-`** `init()` method would be executed by the container automatically as soon as an object of the servlet is created. object of the servlet would be created only once. Thus `init()` method would be executed only once i.e when the object of the servlet is created for the first time.

**2.`service():-`** `service()` method would be executed by the container automatically as and when the request is coming to a servlet. container always calls `service()` method by passing the data (i.e coming from the client) in the form of `ServletRequest` object and address of the client (from where the request is receive) in the form of `ServletResponse` object as arguments.

**3.`destroy():-`** `destroy()` method would be executed before object of the servlet is deleted. container maintains every servlet object for a certain period of time even if no request is coming to that servlet. After certain period of time container deletes object of the servlet before deleting or destroying the servlet object.container automatically calls `destroy()` method and executes `destroy()` method completely and after the execution of the `destroy()` method, the servlet object would be completely deleted.

since `init()`,`service()` and `destroy()` methods are automatically executed by the container based on certain conditions representing different stages of a web application or servlet application we call them as LifeCycle methods of a servlet.

**The javax.servlet package :****The javax.servlet.http package :**



#### Understanding two init( ) methods of Servlet API:

1) public void init(servletConfig cg) throws ServletException

Life cycle method

2) public void init() throws ServletException

Not a life cycle method. It is convenience method given to programmers.

The javax.servlet.GenericServlet class of servlet API contains both init(ServletConfig cg) method & init() method. In javax.servlet.http.HttpServlet class there are no init() method definitions. When instantiation event is raised the servlet container calls init(ServletConfig) method as life cycle method, but it does not call init() method as life cycle method.

#### **Understanding flow of execution related to both init method:**

##### **Scenario 1:**

```

//GenericServlet.java (pre-defined class)
public abstract class GenericServlet implements Servlet{
    ServletConfig config;
    public void init(ServletConfig cg) throws ServletException
        { (5)
            config=cg; //initialization logic of servlet config object
            init();
        }
    public void init() throws ServletException {
        //null body method
    }
    public servletConfig getServletConfig() {
        return config;
    }
    .....
    .....
    //other methods of GenericServlet class
    .....
}
  
```

##### TestSrv.java (Our Servlet class)

(1) (2) (3) (4)

```

public class TestSrv extends GenericServlet/HttpServlet {
    public void init()
    {(6)
        .....
        //our servlet program related initialization logic
    }
}
  
```

```

.....
}//init()
public void service(,-)throws ServletException.IOException
{ (7)
    ServletConfig cg=getServletConfig();
.....
.....
}

```

**With respect to above scenario1:**

- 1) End user gives first request to our servlet program (TestSrv) (let us assume no <load-on-startup> is enabled on this servlet program).
- 2) Servlet container creates our servlet class object using 0-param constructor.
- 3) Servlet container creates ServletConfig object as right hand object to our servlet class object and also raises **instantiation event**.
- 4) To process instantiation event servlet container calls init(-) life cycle method on our servlet class object having ServletConfig object as argument value. Since that method is not available in our servlet class the super class init(-) method(GenericServlet) will be executed.
- 5) This super class init(-) method contains logic to initialize the ServletConfig obj and also calls init() method at the end .
- 6) init() method of our servlet class(TestSrv) executes and this completes the initialization process of our servlet program .
- 7) Servlet container creates ServletRequest,ServletResponse objs for current request raises **request arrival event** and it calls public service(,-) method on our servlet class object. This method execution will process the request and sends the response to browser window.

In scenario1 our servlet program can get access to ServletConfig object by calling getServletConfig() method of predefined GenericServlet class in the life cycle methods or other methods of our servlet program.

**ServletConfig cg= getServletConfig();** → Inherited method of GenericServlet class

Note: public methods of super class can be called in sub class methods without objects.

**Scenario 2:**

GenericServlet.java (pre-defined class)

```

public abstract class GenericServlet implements servlet
{
    Servletconfig config;
    public void init(ServletConfig cg) {
        config=cg;
        init();
    }
    public void init() {
        //null body method
    }
    public ServletConfig getServletConfig() {
        return config;
    }
    .....//other methods of GenericServlet class
    .....
}
//class

```

TestSrv.java(Our servlet class)

(1) (2) (3) (4)

```

public class Testsrvc extends GenericServlet/HttpServlet
{
    ServletConfig cg;
    public void init(ServletConfig cg)
    {
        this.cg = cg; //(5) explicit initialization logic of ServletConfig object
        ..... //our servlet program related initilaztin logic
    }
    public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException {
        use cg here
        .....(6)
    }
}

```

```
.....  
}
```

**In scenario 2** OurServlet program contains init(-) method directly so the control will not go to init(-) method of the GenericServlet class. So programmer must initialize ServletConfig object in its init(-) method of servlet program to make it visible to other methods of these servlet program that means in scenario 2 programmer must not forget the explicit initialization of ServletConfig object in the init(-) method of his servlet prg.

### Scenario 3:

#### GenericServlet.java (pre-defined class)

```
public abstract class GenericServlet implements Servlet
{
    ServletConfig config;
    public void init(ServletConfig cg) throws ServletException {
        (6)
        config= cg; //initialization logic of ServletConfig object
        init();
    }
    public void init() throws ServletException {
        (7) //null method
    }
    public ServletConfig getServletConfig() {
        return config;
    }
    ..... //other methods of GenericServlet class
}
//class
```

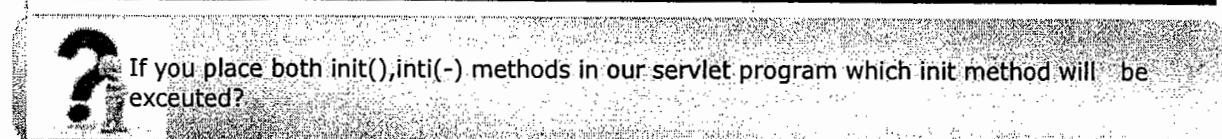
#### TestSrv.java (Ourservlet class)

```
(1) (2) (3) (4)
public class TestSrv extends GenericServlet/HttpServlet
{
    public void init(ServletConfig cg) {
        (5)
        super.init(cg);
        ..... (8)
        .....//our servlet program realted initialization logic
    }
    public void service(ServletRequest, ServletResponse res) throws
    ServletException,IOException {
        (9)
        ServletConfig cg=getServletConfig();
        .....
    }
}
//method
//class
```

In scenario 3 programmer can use getServletConfig() to get access to ServletConfig object to its servlet program.

- % In scenario 1 use getServletConfig() method to get access to ServletConfig object.
- % In scenario 2 initialize ServletConfig object explicitly in init(-) method to use that object.
- % In scenario 3 call super.init( -) method in our init(-) method and also call getServletConfig() method to get access to ServletConfig object.

Always give chance to init(-)method of javax.servlet.GenericServlet class to execute once during instantiation and initialization process of our servlet program because it initializes ServletConfig object and makes programmer free from that process. Due to this scenario 1 is most recommended approach to place init() methods in our servlet program.



**A** Since init(-) method is the life cycle method the Servlet container calls init(-) method and init() method will not be executed.

When init(-) method is life cycle method of servlet why servlet API is providing init() method as additional method?

init() method is a convenience method provided in javax.servlet.GenericServlet class. Generally, it is suggested that init(javax.servlet.ServletConfig) method present in javax.servlet.GenericServlet class should be given a change to get executed, during servlet's initialization. This method, apart from performing initialization of ServletConfig obj, invokes init() method at the end.

But if our servlet class contains init(javax.servlet.ServletConfig) method, then due to method over-riding, container invokes our version of the method. If our method calls the javax.servlet.GenericServlet's init(javax.servlet.ServletConfig) method, by using super keyword then no problem. But if the developer forgets this, then initialization is not complete, which may lead to problems.

To solve the above problem, we place init() method in our servlet class so servlet container calls init(ServletConfig) method of GenericServlet class as life cycle method and this method internally calls init(). Due to this our servlet class init() executes as shown in scenario1.so programmer need not to initialize ServletConfig object manually

A good programmer never keeps init(-) method in his servlet program to place the initialization logic of servlet program like creating JDBC connection object. He always keeps this logic in init() Method even though it is not life cycle method.

If init() method is not given in servlet API programmer should follow scenario 2 or scenario 3 to place init methods and he needs to do some explicit work to see the initialization of ServletConfig object. To overcome this problem servlet api gives init() method as convince method to the programmer and programmer can use that method as discussed in scenario 1.

If we don't place any init() methods then super class(GenericServlet) init(-) method will be executed which it internally also calls init() method of same class.

#### Understanding two service (--) methods and 7 doXxx() methods of pre-defined HttpServlet class:

while we develop our servlet program by extending it from javax.servlet.http.HttpServlet class we can place request processing logic in our servlet program either by using one of the 2 service(--) methods or by using 7 doXxx(--) methods

- 1) public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException  
public service(--) or service(--) method1 (Life cycle method)
- 2) protected void service(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException  
protected service(--) or service(--) method2 (Not a Life cycle method)
- 3) public void doXxx(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException (Not Life cycle methods)  
doXxx- total 7 number of doXxx(--) methods are available like doGet(--), doPost(--), doDelete(--), doPut(--), doHead(--), doTrace(--), doOption(--)

Even though 7 doXxx(--) methods are there the regularly used doXxx(--) methods are doGet(--) & doPost(--). Only public service(--) / service(--) method1 is the life cycle method Servlet container calls this public service(--) method as life cycle method when the **request arrival event** is raised.

#### **Scenario 1 to understand flow of execution related to service (--) , doXxx(--) methods:**

```

//HttpServlet.java(pre-defined class)
public abstract class HttpServlet extends GenericServlet implements Serializable
{
    public void service(ServletRequest req, ServletResponse res)
        throws ServletException, IOException {
        (4)
        HttpServletRequest request= (HttpServletRequest)req;
        HttpServletResponse response= (HttpServletResponse)res;
        service(request,response);
    }
    protected void service(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
        { (5)
        String method=req.getMethod();
        If(method.equals("GET"))
            doGet(req,res);
        elseif(method.equals("POST"))
            doPost(req,res);
        elseif(method.equals("HEAD"))
            doHead(req,res);
        elseif(method.equals("DELETE"))
            doDelete(req,res);
        elseif(method.equals("PUT"))
            doPut(req,res);
        elseif(method.equals("OPTIONS"))
            doOption(req,res);
        elseif(method.equals("TRACE"))
            doTrace(req,res);
        else
            res.sendError("Invalid request method");
    }
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException
    {
        //send 405 error response to browser window.
    }
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException
    {
        //send 405 error response to browser window
    }

    protected void doXxx(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException
    {
        //send 405 error response to browser window
    }
}//class

```

**Our servlet program:**

```

//TestSrv.java
public class TestSrv extends HttpServlet
{
    (1)(GET) (2) (3)
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException,
    IOException
    {
        .....
        .....//some logic (6)
    }
}

```

**With respect to above scenario code**

- (1) End user gives request to our servlet program having request method "GET" from client(browser window).
- (2) Servlet container creates or locates our servlet class object ,if created the servlet container completes the instantiation and initialization related life cycle operations.

- (3) Servlet container raises request arrival event and calls public service(-,-) method on our servlet class object as life cycle method having ServletRequest,ServletResponse objects as arguments. Since public service(-,-) method is not available in our servlet program, the super class (pre-defined HttpServlet class) public service(-,-) method executes.
- (4) The public service(-,-) method of pre-defined HttpServlet class converts simple Servlet Request obj to HttpServletRequest obj, simple ServletResponse obj to HttpServletResponse obj & calls protected service(-,-) method having these two objects. Since protected service(-,-) method is not available in our servlet class the super class (pre-defined HttpServlet class) protected service(-,-) method will be executed.
- (5) The protected service(-,-) method of pre-defined HttpServlet class reads request method of client generated request and calls an appropriate doXxx(-,-) method i.e., doGet (-,-) method. Since doGet (-,-) method is available in our servlet program that method will be executed.
- (6) The doGet(-,-) method of TestSrv program will process the request & sends generated response to browser window as web page.

**Assume the client makes a HTTP request based on GET method in the following situations**

1. If our class contains both doGet() and doPost()
  - a. public service(-,-) of HttpServlet
  - b. protected service(-,-) of HttpServlet
  - c. doGet(-,-) of our class
2. If our class contains only doPost()
  - a. public service(-,-) of HttpServlet
  - b. protected service(-,-) of HttpServlet
  - c. protected doGet(-,-) of HttpServlet(as our class does not contain doGet(-,-))
  - d. 405 response back to client
3. If our class overrides public service(-,-) method, and contains doGet(-,-), doPost(-,-).
  - a. public service(-,-) of our class[doGet(-,-) of our class will not get invoked, because the control did not pass on to protected service(-,-) of HttpServlet].
4. If our class overrides public service(-,-) method, and does not contain doGet(-,-)
  - a. public service() of our class  
No 405 response back to client.
5. If our class overrides public service(-,-) method and it makes a call to super.service(-,-) method, and contains doGet(-,-), doPost(-,-)
  - a. public service(-,-) of our class
  - b. public service(-,-) of HttpServlet(because of super.service(-,-))
  - c. protected service(-,-) of HttpServlet
  - d. doGet(-,-) of our class
6. If our class overrides public service() method, and it makes a call to super.service(-,-) method, and contains only doPost(-,-)
  - a. public service(-,-) of our class
  - b. public service(-,-) of HttpServlet(because of super.service(-,-))
  - c. protected service(-,-) of HttpServlet
  - d. protected doGet(-,-) of HttpServlet
  - e. 405 response back to client

In the execution of our servlet program don't let the control going to doXxx(-,-) methods of javax.servlet.http.HttpServlet class because they always generate 405 error response page indicating our servlet is totally incomplete to process the request.



**If both service (-,-) methods are placed in our servlet program then which method will be executed?**

**A** Since public service(-,-) method is life cycle method of our servlet program only public service(-,-) method will be executed. Even though public service(-,-) method is life cycle method it is recommended to place request processing logic of our servlet program by using doXxx(-,-) methods because they are defined based on protocol Http standards moreover if they are not placed properly the super class doXxx(-,-) methods simply send 405 error to client indicating the problem.

**When all the methods of pre-defined HttpServlet class, are concrete methods why the class itself is given as an abstract class?**

note: In java abstract class can have only abstract methods or only concrete methods or mix of both.

javax.servlet.http.HttpServlet class is abstract, even though none of the methods within it are abstract it is because, it contains seven doXxx() methods, to match seven ways of making HTTP request(GET/POST/DELETE/OPTIONS/TRACE/PUT/HEAD). These methods are the request processing methods of a HttpServlet, just like service(-,-) method for a GenericServlet.

Since there is only one request-processing method for GenericServlet, it is defined as abstract in javax.servlet.GenericServlet class, which makes the developer making his class to extend javax.servlet.GenericServlet class, to provide implementation only from one method.

But in the case of javax.servlet.http.HttpServlet class, if all 9 request-processing methods are defined as abstract, then every developer who creates a child class for it, has to provide implementations for all the seven methods, which is quite an issue. So the specification has made javax.servlet.http.HttpServlet class to contain implementations for all the 9 methods, but they made the javax.servlet.http.HttpServlet class itself as abstract, which means no developer can create an instance of it directly.

### **The seven http request methods/methodologies**

The client can send request to web resource program of the web application in seven different ways by using seven different Http methods. To process these methods based request in our servlet program we can override and use 7 different doXxx(-,-) methods.

- ¾ GET (default)
- ¾ POST
- ¾ HEAD
- ¾ PUT
- ¾ DELETE
- ¾ TRACE
- ¾ OPTIONS

#### **GET:**

Default request method designed to get data from server by generating request without data or with limited amount of data(max. of 256 kb).

**POST:** Can send request with unlimited amount of data and gathers data from server as response.

 The response of GET based or POST based request contains everything including response body like response headers, miscellaneous information,etc.,

#### **HEAD:**

Same as GET but the HEAD based request related response does not contain response body.

- 
1. HEAD based requests are useful to test whether web resource program is present or not.
  2. Even though there are 7 request methods the most regularly used request methods in real world while developing java websites are GET,POST.

#### **PUT:**

This is capable of allowing client to place new file or web resource program in already deployed web application of web server. In real projects after placing websites in the web server of ISP(Internet Service Providers) machine we use FTP(File Transfer Protocol) application from our computers to maintain that website. This FTP application uses PUT method request to add new file or new web resource program in that ISP machine website.

#### **DELETE:**

Allows client to send a request having the capability to delete file or web resource program of web application in the server. FTP application uses this delete method to delete web page or document or anything from the hosted web application of the ISP machine based web server.

#### **TRACE:**

This trace method request returns all the debugging messages and flow of execution details regarding the request and response of certain web resource programs.

**OPTIONS:**

The options method based request given to web resource program determines using which Http request methods that this servlet can be request from client.

**For example:**

If our servlet program overrides doGet(-,-) method as shown below then the OPTIONS method based request given to the servlet program returns the following response

Allow: HEAD, GET, OPTIONS, TRACE

**Eg:**

```
public class TestSrv extends HttpServlet/GenericServlet
{
    public void doGet(HttpServletRequest, HttpServletResponse res) throws
        ServletException, IOException
    {
        .....
        .....
    }
}
```

Note: "POST","PUT" are non-idempotent. "GET","HEAD","OPTIONS","TRACE","DELETE" are idempotent.

```
1 =====
2 App0(first application)
3 =====
4 ----- DateSrv.java-----
5 //DateSrv.java
6 package com.nt;
7
8 import javax.servlet.*;
9 import java.io.*;
10 import java.util.*;
11
12 public class DateSrv extends GenericServlet
13 {
14     public void service(ServletRequest req,ServletResponse res) throws ServletException,IOException
15     {
16         // set response content type
17         res.setContentType("text/html");
18         //Get PrintWriter obj
19         PrintWriter pw = res.getWriter();
20         // write response
21         Date d = new Date();
22         pw.println("<b><i><center> Date and Time is "+d+" </b></i></center>");
23
24         //close stream
25         pw.close();
26     }//service(-,-)
27 } //class
28
29 -----web.xml-----
30 <web-app>
31     <servlet>
32         <servlet-name>abc</servlet-name>
33         <servlet-class>com.nt.DateSrv</servlet-class>
34     </servlet>
35     <servlet-mapping>
36         <servlet-name>abc</servlet-name>
37         <url-pattern>/test1</url-pattern>
38     </servlet-mapping>
39 </web-app>
40
41
42 =====
43 App1 (webapplication having multipe servlet prgs)
44 =====
45 -----HtmISrv.java-----
46 //HtmISrv.java
47 package com.nt;
48 import javax.servlet.*;
49 import javax.servlet.http.*;
50 import java.io.*;
51
52 public class HtmISrv extends HttpServlet
53 {
54
55     public void service(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
56     {
57         // getPrintWriter obj
58         PrintWriter pw = res.getWriter();
59         // set response content type
60         res.setContentType("text/html");
```

```
61      //write logic to generate output(webpage)
62      pw.println("<table border='1'>");
63      pw.println("<tr><th>Player</th><th>Role</th></tr>");
64      pw.println("<tr><td>Dhoni</td><td>Captain</td></tr>");
65      pw.println("<tr><td>Sachin</td><td>All Rounder</td></tr>");
66      pw.println("</table>");
67
68      //close stream obj
69      pw.close();
70  } //service(-,-)
71 } //class
73 -----WordSrv.java-----
74 //WordSrv.java
75 package com.nt;
76 import javax.servlet.*;
77 import javax.servlet.http.*;
78 import java.io.*;
79
80 public class WordSrv extends HttpServlet
81 {
82
83     public void service(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
84     {
85         // getPrintWriter obj
86         PrintWriter pw = res.getWriter();
87         // set response content type
88         res.setContentType("application/msword");
89
90         //write logic to generate output(webpage)
91         pw.println("<table border='0'>");
92         pw.println("<tr><th>Player</th><th>Role</th></tr>");
93         pw.println("<tr><td>Dhoni</td><td>Captain</td></tr>");
94         pw.println("<tr><td>Sachin</td><td>All Rounder</td></tr>");
95         pw.println("</table>");
96
97         //close stream obj
98         pw.close();
99     } //service(-,-)
100 } //class
101 -----XmlSrv.java-----
102 package com.nt;
103 import javax.servlet.*;
104 import javax.servlet.http.*;
105 import java.io.*;
106
107 public class XmlSrv extends HttpServlet
108 {
109
110     public void service(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
111     {
112         // getPrintWriter obj
113         PrintWriter pw = res.getWriter();
114         // set response content type
115         res.setContentType("text/xml");
116
117         //write logic to generate output(webpage)
118         pw.println("<table border='0'>");
119         pw.println("<tr><th>Player</th><th>Role</th></tr>");
120         pw.println("<tr><td>Dhoni</td><td>Captain</td></tr>");
```

```
121     pw.println("<tr><td>Sachin</td><td>All Rounder</td></tr>");  
122     pw.println("</table>");  
123  
124     //close stream obj  
125     pw.close();  
126 } //service(-,-)  
127 } //class  
128 -----ExcelSrv.java-----  
129 //ExcelSrv.java  
130 package com.nt;  
131 import javax.servlet.*;  
132 import javax.servlet.http.*;  
133 import java.io.*;  
134  
135 public class ExcelSrv extends HttpServlet  
136 {  
137  
138     public void service(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException  
139     {  
140         // getPrintWriter obj  
141         PrintWriter pw = res.getWriter();  
142         // set response content type  
143         res.setContentType("application/vnd.ms-excel");  
144  
145         //write logic to generate output(webpage)  
146         pw.println("<table border='0'>");  
147         pw.println("<tr><th>Player</th><th>Role</th></tr>");  
148         pw.println("<tr><td>Dhoni</td><td>Captain</td></tr>");  
149         pw.println("<tr><td>Sachin</td><td>All Rounder</td></tr>");  
150         pw.println("</table>");  
151  
152         //close stream obj  
153         pw.close();  
154     } //service(-,-)  
155 } //class  
156 -----web.xml-----  
157 <web-app>  
158     <servlet>  
159         <servlet-name>abc</servlet-name>  
160         <servlet-class>com.nt.HtmlSrv</servlet-class>  
161     </servlet>  
162  
163     <servlet-mapping>  
164         <servlet-name>abc</servlet-name>  
165         <url-pattern>/hturl</url-pattern>  
166     </servlet-mapping>  
167  
168     <servlet>  
169         <servlet-name>mno</servlet-name>  
170         <servlet-class>com.nt.WordSrv</servlet-class>  
171     </servlet>  
172  
173     <servlet-mapping>  
174         <servlet-name>mno</servlet-name>  
175         <url-pattern>/wdurl</url-pattern>  
176     </servlet-mapping>  
177  
178     <servlet>  
179         <servlet-name>pqr</servlet-name>  
180         <servlet-class>com.nt.ExcelSrv</servlet-class>
```

```
181     </servlet>
182
183     <servlet-mapping>
184         <servlet-name>pqr</servlet-name>
185         <url-pattern>/xlsurl</url-pattern>
186     </servlet-mapping>
187
188     <servlet>
189         <servlet-name>xyz</servlet-name>
190         <servlet-class>com.nt.XmlSrv</servlet-class>
191     </servlet>
192
193     <servlet-mapping>
194         <servlet-name>xyz</servlet-name>
195         <url-pattern>/xmlurl</url-pattern>
196     </servlet-mapping>
197
198 </web-app>
199 -----
200 =====
201 App2(Html-Servlet Communication)
202 =====
203 -----sum.html-----
204 <html>
205     <head>
206         <title>some html</title>
207     </head>
208     <body>
209         <form action="sumurl" method="post">
210             First Number
211             <input type="text" name="fno" size="5">
212             <br>
213             Second Number
214             <input type="text" name="sno"
215             size="5">
216             <br>
217             <input type="SUBMIT"
218             value="SUM">
219         </form>
220     </body>
221 </html>
222 -----web.xml-----
223 <?xml version="1.0" encoding="UTF-8"?>
224
225 <!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
226           "http://java.sun.com/dtd/web-app_2_3.dtd">
227 <web-app>
228
229     <servlet>
230         <servlet-name>sum</servlet-name>
231         <servlet-class>SumServlet</servlet-class>
232     </servlet>
233     <servlet-mapping>
234         <servlet-name>sum</servlet-name>
235         <url-pattern>/sumurl</url-pattern>
236     </servlet-mapping>
237
238     <welcome-file-list>
239         <welcome-file>sum.html</welcome-file>
240     </welcome-file-list>
```

```
241
242 </web-app>
243 -----SumServlet.java-----
244 package com.nt;
245 import javax.servlet.*;
246 import javax.servlet.http.*;
247 import java.io.*;
248
249 public class SumServlet extends HttpServlet {
250
251     public SumServlet() {
252         System.out.println("Inside constructor");
253     }
254
255     public void init(ServletConfig sc) {
256         System.out.println("Inside init() method");
257     }
258
259     public void doGet(HttpServletRequest req, HttpServletResponse res) {
260         System.out.println("Inside doGet() ");
261         try {
262             int x, y, sum;
263             x = Integer.parseInt(req.getParameter("fno"));
264             y = Integer.parseInt(req.getParameter("sno"));
265             sum = x+ y;
266
267             PrintWriter pw;
268             pw = res.getWriter();
269
270             pw.println("<html>");
271             pw.println("<body>");
272             pw.println("<center> <h2> <br> <br>"); 
273             pw.println("Result is " + sum);
274             pw.println("</h2> </center>"); 
275             pw.println("</body> </html>"); 
276         } // try
277         catch(Exception e) {
278             e.printStackTrace();
279         }
280     } // doGet()
281
282     public void doPost(HttpServletRequest req, HttpServletResponse res) {
283         System.out.println("Inside doPost()");
284         try {
285             doGet(req, res);
286         }
287         catch(Exception e) {
288             e.printStackTrace();
289         }
290     } // doPost()
291
292 } //class
293 =====
294 App3(Working with diff types form comps)
295 =====
296 -----Form.html-----
297 <form action="formurl">
298     <table border="1">
299         <tr>
300             <td>Name</td>
```

```
301      <td><input type="text" name="tname" value="" /></td>
302  </tr>
303  <tr>
304    <td>Age</td>
305    <td><input type="password" name="tage" value="" /> </td>
306  </tr>
307  <tr>
308    <td>Gender</td>
309    <td>
310      <input type="radio" name="gen" value="M" checked="checked" /> Male &nbsp;
311      <input type="radio" name="gen" value="F" /> FeMale &nbsp;
312    </td>
313  </tr>
314  <tr>
315    <td>Address: </td>
316    <td>
317      <textarea name="taddress" rows="4" cols="20">
318        enter address
319      </textarea>
320    </td>
321  </tr>
322  <tr>
323    <td>Marital Status</td>
324    <td><input type="checkbox" name="ms" value="married" /> Married</td>
325  </tr>
326  <tr>
327    <td>Qualification</td>
328    <td>
329      <select name="qlfy">
330        <option value="B.Tech">Engg</option>
331        <option value="MBBS">Medical</option>
332        <option value="B.A">Arts</option>
333      </select>
334    </td>
335  </tr>
336  <tr>
337    <td>Courses:</td>
338    <td><select name="crs" size="3" multiple="multiple">
339      <option value="java">JAVA pkg</option>
340      <option value=".net">.NET pkg</option>
341      <option value="testing">Testing pkg</option>
342    </select>
343    </td>
344  </tr>
345  <tr>
346    <td>Hobbies</td>
347    <td>
348      <input type="checkbox" name="hb" value="read" checked/> Reading
349      <input type="checkbox" name="hb" value="stamps"/> Stamp Collection
350      <input type="checkbox" name="hb" value="roaming" /> Travelling
351    </td>
352  </tr>
353  <tr>
354    <td colspan="2">
355      <input type="submit" value="submit" />
356      <input type="reset" value="cancel" />
357    </td>
358  </tr>
359 </table>
360 </form>
```

```
361 -----web.xml-----
362 <servlet>
363   <servlet-name>abc</servlet-name>
364   <servlet-class>com.nt.FormSrv</servlet-class>
365 </servlet>
366 <servlet-mapping>
367   <servlet-name>abc</servlet-name>
368   <url-pattern>/formurl</url-pattern>
369 </servlet-mapping>
370 <welcome-file-list>
371   <welcome-file>Form.html</welcome-file>
372 </welcome-file-list>
373 </web-app>
374 -----FormSrv.java-----
375 package com.nt;
376 import java.io.IOException;
377 import java.io.PrintWriter;
378 import javax.servlet.ServletException;
379 import javax.servlet.http.HttpServlet;
380 import javax.servlet.http.HttpServletRequest;
381 import javax.servlet.http.HttpServletResponse;
382
383 public class FormSrv extends HttpServlet {
384
385     protected void processRequest(HttpServletRequest request, HttpServletResponse response)
386     throws ServletException, IOException {
387         //get PrintWriter obj
388         PrintWriter pw=response.getWriter();
389         // set response content type
390         response.setContentType("text/html");
391         // read from data
392         String name=request.getParameter("tname");
393         int age=Integer.parseInt(request.getParameter("tage"));
394         String gender=request.getParameter("gen");
395         String ms=request.getParameter("ms");
396         String addrs=request.getParameter("taddress");
397         String qlfy=request.getParameter("qlfy");
398         String crs[]=request.getParameterValues("crs");
399         String hb[]=request.getParameterValues("hb");
400         // write request processing logic
401         if(gender.equalsIgnoreCase("M"))
402         {
403             if(age<=5)
404                 pw.println(name+"u r baby boy");
405             else if (age<=12)
406                 pw.println(name+"u r smalll boy");
407             else if(age<=19)
408                 pw.println(name+"u r teenage boy");
409             else if(age<=35)
410                 pw.println(name+"u r young man ");
411             else if(age<=50)
412                 pw.println(name+"u r middle aged man ");
413             else
414                 pw.println(name+" u r Budda");
415         }//if
416         else if(gender.equalsIgnoreCase("F"))
417         {
418             if(age<=5)
419                 pw.println(name+"u r baby girl");
420             else if (age<=12)
```

```

421         pw.println(name+"u r small girl");
422     else if(age<=19)
423         pw.println(name+"u r teenage girl");
424     else if(age<=35)
425         pw.println(name+"u r young woman ");
426     else if(age<=50)
427         pw.println(name+"u r middle aged woman ");
428     else
429         pw.println(name+" u r old woman");
430     }
431
432     pw.println("<br>name="+name);
433     pw.println("<br>age="+age);
434     pw.println("<br>Gender="+gender);
435     pw.println("<br>Marital Status="+ms);
436     pw.println("<br>Address="+addrs);
437     pw.println("<br> Qualification="+qf);
438     pw.println("<br> Courses");
439     if(crs!=null)
440     {
441         for(int i=0;i<crs.length;++i)
442             pw.println(crs[i]+"... ");
443     }
444
445     pw.println("<br> Hobies");
446     if(hb!=null)
447     {
448         for(int i=0;i<hb.length;++i)
449             pw.println(hb[i]+"... ");
450     }
451
452 }
453
454 protected void doGet(HttpServletRequest request, HttpServletResponse response)
455 throws ServletException, IOException {
456     processRequest(request, response);
457 }
458
459 protected void doPost(HttpServletRequest request, HttpServletResponse response)
460 throws ServletException, IOException {
461     processRequest(request, response);
462 }
463 }
464 =====
465 App4(VoterApp web application with form both client side and server side form validations)
466 =====
467 -----
468 -----input.html-----
469 <html>
470 <head>
471 <script language="JavaScript" src="Validation.js">
472 </script>
473 </head>
474 <form action="vturl" onsubmit="return validate(this)" >
475     Person name : <input type="text" name="pname"/><br>
476     person age : <input type="text" name="page"/><br>
477     <!-- hidden box -->
478     <input type="hidden" value="no" name="vflag"/> <br>
479
480     <input type="submit" value="check Voting Eligibility">

```

+ Array.toString(crs)

```
481 </form>
482 -----Validation.js-----
483 function validate(frm)
484 {
485     frm.vflag.value="yes";
486     //read form data
487     var un=frm.pname.value;
488     var age=frm.page.value;
489     // perform client side form validations
490     if(un=="") //required rule
491     {
492         alert(" person name is mandatory");
493         frm.pname.focus();
494         return false;
495     }//if
496     if(age=="") //required rule
497     {
498         alert("person age is mandatory");
499         frm.page.focus();
500         return false;
501     }//if
502     else
503     {
504         if(isNaN(age)) // checks weather age is numeric value or not
505         {
506             alert("page age must be a numeric value");
507             frm.page.focus();
508             frm.page.value="";
509             return false;
510         }//if
511     }//else
512     return true;
513 } //function
514 -----web.xml-----
515 <web-app>
516     <servlet>
517         <servlet-name>abc</servlet-name>
518         <servlet-class>com.nt.VoterSrv</servlet-class>
519     </servlet>
520     <servlet-mapping>
521         <servlet-name>abc</servlet-name>
522         <url-pattern>/vturl</url-pattern>
523     </servlet-mapping>
524 </web-app>
525 -----VoterSrv.java-----
526 package com.nt;
527 import javax.servlet.*;
528 import javax.servlet.http.*;
529 import java.io.*;
530
531 public class VoterSrv extends HttpServlet
532 {
533     public void xyz(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
534     {
535         System.out.println("VoterSrv:xyz(-,-)");
536         // get PrintWriter obj
537         PrintWriter pw=res.getWriter();
538         // set response content Type
539         res.setContentType("text/html");
540         // read form data from the form comps of form page
```

```

541     String name=req.getParameter("pname");
542     String tage=req.getParameter("page");
543     int age=0;
544     String vstatus=req.getParameter("vflag"); // get Client Side Validation status
545     if(vstatus.equals("no")) // if Client side validations are not done
546     {
547         //Server Form validation logic
548         if(name.equals("") || name==null || name.length()<=0) // required rule
549         {
550             pw.println("<font color=red> Person name is mandatory </font>");  

551             return;
552         }
553         if(tage.equals("") || tage==null || tage.length()==0) //required rule
554         {
555             pw.println("<font color=red> Person age is mandatory</font>");  

556             return;
557         }
558         else // to check weather age is numeric value or not
559         {
560             try{
561                 // convert given age value to numeric value.
562                 age=Integer.parseInt(tage);
563             }
564             catch(NumberFormatException nfe)
565             {
566                 pw.println("<font color=red> Age must be numeric value </font>");  

567                 return ;
568             }
569         } //else
570         System.out.println("Server Side validations completed");
571     } //if
572
573     if(vstatus.equals("yes")) //when client form validation are done
574     {
575         age=Integer.parseInt(tage);
576     }
577
578
579     // write request processing logic / B.logic
580     if(age>=18)
581         pw.println("<h1><font color='green'>" +name+ " u r eligible to vote </font></h1>");
582     else
583         pw.println("<h1><font color='red'>" +name+ " u r eligible not to vote </font></h1>");  

584
585     pw.println("<br><br><a href='input.html'><img src='tips.gif' width='100' height='100'> </a>");  

586
587         //close stream
588         pw.close();
589     } //doGet(-,-)
590
591     public void doGet(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
592     {
593         System.out.println("VoterSrv:doGet(-,-)");
594         xyz(req,res);
595     }
596
597
598     public void doPost(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
599     {
600

```

```
601         System.out.println("VoterSrv:doPost(-,-)");
602         xyz(req,res);
603     }
604 } //class
605 //>javac -d . VoterSrv.java
606
607 =====
608 App5>>>>>>> Example App on servlet to DB s/w communication>>>>>.
609 -----input.html-----
610
611 <form action="dburl" method="get">
612   Employee no: <input type="text" name="teno"/><br>
613   <input type="submit" value="GetDetails">
614 </form>
615 -----web.xml-----
616
617 <web-app>
618   <servlet>
619     <servlet-name>abc</servlet-name>
620     <servlet-class>com.nt.DBsrv</servlet-class>
621     <load-on-startup>1</load-on-startup>
622   </servlet>
623
624   <servlet-mapping>
625     <servlet-name>abc</servlet-name>
626     <url-pattern>/dburl</url-pattern>
627   </servlet-mapping>
628
629   <welcome-file-list>
630     <welcome-file>input.html</welcome-file>
631   </welcome-file-list>
632 </web-app>
633 -----DBsrv.java-----
634 //DBsrv.java
635 package com.nt;
636 import javax.servlet.*;
637 import javax.servlet.http.*;
638 import java.io.*;
639
640 import java.sql.*;
641
642 public class DBsrv extends HttpServlet
643 { Connection con;
644   PreparedStatement ps;
645   public void init(){
646     try{
647       //register jdbc driver
648       Class.forName("oracle.jdbc.driver.OracleDriver");
649       //Establish the connection
650       con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","scott","tiger","System","manager");
651       //create Jdbc PreparedStatement obj
652       ps=con.prepareStatement("select empno,ename,job,sal from emp where empno=?");
653     } //try
654     catch(Exception e)
655     { e.printStackTrace(); }
656   } //init()
657
658   public void doGet(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
659   {
660     try{
```

```
661 //General settings
662     PrintWriter pw=res.getWriter();
663     res.setContentType("text/html");
664     // read form data
665     int no=Integer.parseInt(req.getParameter("teno"));
666     //set param value to SQL Query
667     ps.setInt(1,no);
668     // execute the SQL Query
669     ResultSet rs=ps.executeQuery();
670     //process the ResultSet
671     if(rs.next())
672     {
673         pw.println("<br> Employee no"+rs.getInt(1));
674         pw.println("<br> Employee name"+rs.getString(2));
675         pw.println("<br> Employee Desg :" +rs.getString(3));
676         pw.println("<br> Employee Salary:" +rs.getFloat(4));
677     }//if
678     else
679     {
680         pw.println("<br> No Employee Found ");
681     }
682     } //try
683     catch(Exception e)
684     { e.printStackTrace(); }
685 } //doGet(-,-)
686
687 public void doPost(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
688 {
689     doGet(req,res);
690 } //doPost(-,-)
691
692 public void destroy(){
693 //close jdbc objs
694     try{
695         if(ps!=null)
696             ps.close();
697     } //try
698     catch(Exception e)
699     { e.printStackTrace(); }
700
701     try{
702         if(con!=null)
703             con.close();
704     } //try
705     catch(Exception e)
706     { e.printStackTrace(); }
707 } //destroy()
708 } //class
709 //>javac -d . DBSrv.java
710
711
712 =====
713 App6)Servlet-DB s/w communication (Using init param values)
714 -----
715 -----input.html-----
716 <form action="dburl" method="get">
717     Employee no: <input type="text" name="teno"><br>
718     <input type="submit" value="getEmpDetails">
719 </form>
720 -----web.xml-----
```

```
721 <web-app>
722
723
724     <servlet>
725         <servlet-name>db</servlet-name>
726         <servlet-class>DBSrv</servlet-class>
727         <init-param>
728             <param-name>driver</param-name>
729             <param-value>oracle.jdbc.driver.OracleDriver</param-value>
730         </init-param>
731
732         <init-param>
733             <param-name>dburl</param-name>
734             <param-value>jdbc:oracle:oci8:@orcl</param-value>
735         </init-param>
736
737         <init-param>
738             <param-name>dbuser</param-name>
739             <param-value>scott</param-value>
740         </init-param>
741
742         <init-param>
743             <param-name>dbpwd</param-name>
744             <param-value>tiger</param-value>
745         </init-param>
746     <load-on-startup>1</load-on-startup>
747 </servlet>
748
749     <servlet-mapping>
750         <servlet-name>db</servlet-name>
751         <url-pattern>/dburl</url-pattern>
752     </servlet-mapping>
753 </web-app>
754 -----DBSrv.java-----
755 //DBSrv.java (show servlet to DB s/w communication based on approach1)
756 package com.nt;
757 import javax.servlet.*;
758 import javax.servlet.http.*;
759 import java.io.*;
760 import java.sql.*;
761
762 public class DBSrv extends HttpServlet
763 {
764     Connection con;
765     PreparedStatement ps;
766     public void init()
767     {
768         try
769         {
770             //get Access to ServletConfig obj
771             ServletConfig cg=getServletConfig();
772             // read init param values from web.xml
773             String s1=cg.getInitParameter("driver");
774             String s2=cg.getInitParameter("dburl");
775             String s3=cg.getInitParameter("dbuser");
776             String s4=cg.getInitParameter("dbpwd");
777
778             // create jdbc con obj
779             Class.forName(s1);
780             con=DriverManager.getConnection(s2,s3,s4);
```

```
781
782     // create jdbc PreparedStatement obj.....
783     ps=con.prepareStatement("select ename,sal from emp where empno=?");
784     }//try
785     catch(Exception e)
786     {
787         e.printStackTrace();
788     }
789 }//init
790
791 public void doGet(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
792 {
793     try
794     {
795         // read form data from form page
796         int no=Integer.parseInt(req.getParameter("teno"));
797
798         //get PrintWriter obj
799         PrintWriter pw=res.getWriter();
800         // set response content type
801         res.setContentType("text/html");
802
803         // write jdbc code.....
804         //set value to parameter of the qry
805         ps.setInt(1,no);
806
807         // execute the query
808         ResultSet rs=ps.executeQuery();
809
810         //process ResultSet obj and display emp details....
811         if(rs.next())
812         {
813             pw.println("<br><b><i> Emp name is: </i></b>" +rs.getString(1));
814             pw.println("<br><b><i> Emp Salary is: </i></b>" +rs.getFloat(2));
815         }//if
816
817
818         //close ResultSet obj
819         rs.close();
820
821         //close stream obj
822         pw.close();
823     }//try
824     catch(Exception e)
825     {
826         e.printStackTrace();
827     }
828 }//doGet(--)
829 public void doPost(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
830 {
831     doGet(req,res);
832 }
833
834 public void destroy()
835 {
836     try
837     {
838         if(ps!=null)
839         ps.close();
840     }
```

```

841         catch(Exception e1)
842         {
843             e1.printStackTrace();
844         }
845         try
846         {
847             if(con!=null)
848                 con.close();
849         }
850         catch(Exception e2)
851         {
852             e2.printStackTrace();
853         }
854     }//destroy()
855 }//class
856 =====
857 App7) Servlet-DB s/w communication (using context param values)
858 =====
859 -----input.html-----
860 <form action="dburl" method="get">
861   Employee no: <input type="text" name="teno"><br>
862   <input type="submit" value="getEmpDetails">
863 </form>
864 -----web.xml-----
865 <web-app>
866   <context-param>
867     <param-name>driver</param-name>
868     <param-value>oracle.jdbc.driver.OracleDriver</param-value>
869   </context-param>
870   <context-param>
871     <param-name>dburl</param-name>
872     <param-value>jdbc:oracle:thin:@localhost:1521:orcl</param-value>
873   </context-param>
874   <context-param>
875     <param-name>dbuser</param-name>
876     <param-value>scott</param-value>
877   </context-param>
878   <context-param>
879     <param-name>dbpwd</param-name>
880     <param-value>tiger</param-value>
881   </context-param>
882
883   <servlet>
884     <servlet-name>db</servlet-name>
885     <servlet-class>com.nt.DBSrv</servlet-class>
886     <load-on-startup>1</load-on-startup>
887   </servlet>
888
889   <servlet-mapping>
890     <servlet-name>db</servlet-name>
891     <url-pattern>/dburl</url-pattern>
892   </servlet-mapping>
893 </web-app>
894 -----DBSrv.java-----
895 //DBSrv.java (show servlet to DB s/w communication based on approach1)
896 package com.nt;
897 import javax.servlet.*;
898 import javax.servlet.http.*;
899 import java.io.*;
900 import java.sql.*;

```

```
901
902 public class DBSrv extends HttpServlet
903 {
904     Connection con;
905     PreparedStatement ps;
906     public void init()
907     {
908         try
909         {
910             //get Access to ServletContext obj
911             ServletConfig sc=getServletContext();
912             // read context param values from web.xml
913             String s1=sc.getInitParameter("driver");
914             String s2=sc.getInitParameter("dburl");
915             String s3=sc.getInitParameter("dbuser");
916             String s4=sc.getInitParameter("dbpwd");
917
918             // create jdbc con obj
919             Class.forName(s1);
920             con=DriverManager.getConnection(s2,s3,s4);
921
922             // create jdbc PreparedStatement obj.....
923             ps=con.prepareStatement("select ename,sal from emp where empno=?");
924         }//try
925         catch(Exception e)
926         {
927             e.printStackTrace();
928         }
929     }//init
930
931     public void doGet(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
932     {
933         try
934         {
935             // read form data from form page
936             int no=Integer.parseInt(req.getParameter("teno"));
937
938             //get PrintWriter obj
939             PrintWriter pw=res.getWriter();
940             // set response content type
941             res.setContentType("text/html");
942
943             // write jdbc code.....
944             //set value to parameter of the qry
945             ps.setInt(1,no);
946
947             // execute the query
948             ResultSet rs=ps.executeQuery();
949
950             //process ResultSet obj and display emp details....
951             if(rs.next())
952             {
953                 pw.println("<br><b><i> Emp name is: </i></b>" +rs.getString(1));
954                 pw.println("<br><b><i> Emp Salary is: </i></b>" +rs.getFloat(2));
955             } //if
956
957
958             //close ResultSet obj
959             rs.close();
960         }
961     }
962 }
```

```
961         //close stream obj
962         pw.close();
963     } //try
964     catch(Exception e)
965     {
966         e.printStackTrace();
967     }
968 } //doGet(-,-)
969 public void doPost(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
970 {
971     doGet(req,res);
972 }
973
974 public void destroy()
975 {
976     try
977     {
978         if(ps!=null)
979             ps.close();
980     }
981     catch(Exception e1)
982     {
983         e1.printStackTrace();
984     }
985     try
986     {
987         if(con!=null)
988             con.close();
989     }
990     catch(Exception e2)
991     {
992         e2.printStackTrace();
993     }
994 } //destroy()
995 } //class
996 -----
997 =====
998 App9) Applet to Servlet Communiuation
999 =====
1000 -----Main.html-----
1001 <frameset rows="30%,*>
1002   <frame src="Form.html" name="f1">
1003   <frame name="f2">
1004 </frameset>
1005 -----Form.html-----
1006 <applet code="WishApplet.class" width="400" height="400"/>
1007 -----
1008 -----WishApplet.java-----
1009 //WishApplet.java
1010 package com.nt;
1011 import java.applet.*;
1012 import java.awt.event.*;
1013 import java.awt.*;
1014 import java.net.*;
1015
1016
1017 public class WishApplet extends Applet implements ActionListener
1018 {
1019     Label l1;
1020     TextField tf1;
```

```
1021     Button btn;
1022
1023     public WishApplet()
1024     {
1025         setSize(300,400);
1026         setBackground(Color.RED);
1027         //add comps
1028         l1=new Label("name");
1029         add(l1);
1030         tf1=new TextField(20);
1031         add(tf1);
1032         btn=new Button("send");
1033         add(btn);
1034         //register listener
1035         btn.addActionListener(this);
1036         setLayout(new FlowLayout());
1037     }
1038
1039     public void actionPerformed(ActionEvent ae)
1040     {
1041         try
1042         {
1043             //frame request url
1044             String qstr="?prname="+tf1.getText().replace(' ','+');
1045             URL myurl =new URL("http://localhost:2020/AppletServletApp/wurl"+qstr);
1046             // get AppletContext obj
1047             AppletContext apc = getAppletContext();
1048             //send request
1049             apc.showDocument(myurl,"f2");
1050         }
1051         catch(Exception e)
1052         {
1053             e.printStackTrace();
1054         }
1055     }//method
1056 } //class
1057 -----web.xml-----
1058 <web-app>
1059     <servlet>
1060         <servlet-name>XYZ</servlet-name>
1061         <servlet-class>WishSrv</servlet-class>
1062     </servlet>
1063     <servlet-mapping>
1064         <servlet-name>XYZ</servlet-name>
1065         <url-pattern>/wurl</url-pattern>
1066     </servlet-mapping>
1067     <welcome-file-list>
1068         <welcome-file>Main.html</welcome-file>
1069     </welcome-file-list>
1070 </web-app>
1071 -----WishSrv.java-----
1072 //WishSrv.java
1073 package com.nt;
1074 import javax.servlet.*;
1075 import java.io.*;
1076 import javax.servlet.http.*;
1077 import java.util.*;
1078
1079 public class WishSrv extends HttpServlet
1080 {
```

```
1081 public void service(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
1082 {
1083     // get PrintWriter
1084     PrintWriter pw = res.getWriter();
1085     // set content type
1086     res.setContentType("text/html");
1087     //read form data (applet data)
1088     String name = req.getParameter("pname");
1089     // process request and generate response
1090     Calendar cl = Calendar.getInstance(); //gives current date and time
1091     int h = cl.get(Calendar.HOUR_OF_DAY);
1092     if(h <= 12)
1093         pw.println("<b>Good morning:</b>" + name);
1094     else if(h <= 16)
1095         pw.println("<b>Good Afternoon:</b>" + name);
1096     else if(h <= 20)
1097         pw.println("<b>Good Evening:</b>" + name);
1098     else
1099         pw.println("<b> Good Night </b>");
1100
1101     //close stream obj
1102     pw.close();
1103 } //service(-,-)
1104 } //class
1105 =====
1106 App10(Servlet-DB Communication with the support of Server Managed JDBCConnectionPool)
1107 =====
1108 -----index.htm-----
1109 <html>
1110 <head><title>Displays the table from database</title></head>
1111 <body bgcolor="#ffffa9">
1112 <form action="poolurl" method="GET">
1113     <center>
1114         <h2>Display Table</h2>
1115         <br><br><br>
1116         <table>
1117             <tr>
1118                 <td><b>Enter the table name</b>&nbsp;&nbsp;</td>
1119                 <td><input type="text" name="table"></td>
1120             </tr>
1121             <tr>
1122                 <td align="right">
1123                     <input type="submit" value="Display">&nbsp;&nbsp;
1124                 </td>
1125                 <td>
1126                     <input type="reset" value="cancel">
1127                 </td>
1128             </tr>
1129             </table>
1130         </center>
1131     </form>
1132 </body>
1133 </html>
1134 -----web.xml-----
1135 <web-app>
1136     <welcome-file-list>
1137         <welcome-file>index.htm</welcome-file>
1138     </welcome-file-list>
1139     <servlet>
1140         <servlet-name>conPool</servlet-name>
```

```
1141      <servlet-class>com.nt.ConnPoolServlet</servlet-class>
1142  </servlet>
1143  <servlet-mapping>
1144      <servlet-name>conPool</servlet-name>
1145      <url-pattern>/poolurl</url-pattern>
1146  </servlet-mapping>
1147 </web-app>
1148 ----- ConnPoolServlet.java -----
1149 package com.nt;
1150 import java.io.PrintWriter;
1151 import java.io.IOException;
1152
1153 import javax.servlet.ServletException;
1154 import javax.servlet.http.HttpServlet;
1155 import javax.servlet.http.HttpServletRequest;
1156 import javax.servlet.http.HttpServletResponse;
1157
1158 import java.sql.Connection;
1159 import java.sql.Statement;
1160 import java.sql.DriverManager;
1161 import java.sql.ResultSet;
1162 import java.sql.ResultSetMetaData;
1163 import javax.sql.DataSource;
1164
1165 import javax.naming.InitialContext;
1166
1167
1168 public class ConnPoolServlet extends HttpServlet
1169 {
1170
1171     public void service(HttpServletRequest req,HttpServletResponse res)
1172             throws ServletException,IOException
1173     {
1174         res.setContentType("text/html");
1175         PrintWriter out;
1176         out=res.getWriter();
1177         String tableName=req.getParameter("table");
1178         try
1179         {
1180             Connection con=makeConnection();
1181
1182
1183             Statement st=con.createStatement();
1184             ResultSet rs=st.executeQuery("select * from "+tableName);
1185
1186             ResultSetMetaData rsmd=rs.getMetaData();
1187             int columnCount=rsmd.getColumnCount();
1188
1189             out.println("<html><body bgcolor=#ffffa9>");
1190             out.println("<center><h3>The Details of the table "+tableName);
1191             out.println("<br><br><table cellspacing=1 bgcolor=#00ff00>");
1192             out.println("<tr bgcolor=#ffffa9>");
1193
1194             for(int col=1;col<=columnCount;col++)
1195             {
1196                 out.println("<th>"+rsmd.getColumnLabel(col)+"&ampnbsp&ampnbsp</th>");
1197
1198             } //for
1199             out.println("</tr>");
1200 }
```

```

1201         while(rs.next())
1202     {
1203         out.println("<tr bgcolor=#ffffa9>");
1204
1205         for(int i=1;i<=columnCount;i++)
1206             out.println("<td>" + rs.getString(i) + "&nbsp;&nbsp;</td>");
1207     }//for
1208
1209     out.println("</tr>");
1210 } //while
1211
1212     out.println("</table><br>");
1213     out.println("To view another table <b><a href=index.htm>Click here</a></b>"); 
1214     out.println("</body></html>"); 
1215 } //try
1216 catch(Exception e)
1217 {
1218     out.println("<html><body bgcolor=#ffffa9><center><br><br>"); 
1219     out.println("<h3>Table Doesn't Exist in Database</h3>"); 
1220     out.println("<br>To view another table <a href=index.htm>Click here</a>"); 
1221     out.println("</body></html>"); 
1222 } //catch
1223 } //service
1224
1225 public Connection makeConnection()
1226 {
1227     Connection con=null;
1228     try
1229     { //get InitialContext obj
1230         InitialContext ic=new InitialContext();
1231         // get DataSource obj from registry
1232         DataSource ds=(DataSource)ic.lookup("MyDsJndi");
1233         // get jdbc con obj from jdbc con pool
1234         con=ds.getConnection();
1235     } //try
1236     catch(Exception e)
1237     {
1238         System.out.println(e);
1239     } //catch
1240     return con;
1241 } //makeConnection()
1242 } //class
1243 =====
1244 App1( Login Application)
1245 =====
1246 -----Login.html-----
1247 <form action="loginurl" method="get">
1248   user name : <input type="text" name="uname"/><br>
1249   password : <input type="password" name="pwd"/><br>
1250   <input type="submit" value="Login"/>
1251 </form>
1252 -----web.xml-----
1253 <servlet>
1254   <servlet-name>Login</servlet-name>
1255   <servlet-class>com.nt.LoginSrv</servlet-class>
1256 </servlet>
1257
1258 <servlet-mapping>
1259   <servlet-name>Login</servlet-name>
1260   <url-pattern>/loginurl</url-pattern>

```

```
1261 </servlet-mapping>
1262
1263 <welcome-file-list>
1264   <welcome-file>index.jsp</welcome-file>
1265 </welcome-file-list>
1266 </web-app>
1267 -----LoginSrv.java-----
1268 package com.nt;
1269 import java.io.IOException;
1270 import java.io.PrintWriter;
1271 import java.sql.CallableStatement;
1272 import java.sql.Connection;
1273 import java.sql.DriverManager;
1274 import java.sql.Types;
1275
1276 import javax.servlet.ServletException;
1277 import javax.servlet.http.HttpServlet;
1278 import javax.servlet.http.HttpServletRequest;
1279 import javax.servlet.http.HttpServletResponse;
1280
1281
1282 public class LoginSrv extends HttpServlet {
1283     Connection con=null;
1284     CallableStatement cs=null;
1285
1286     public void init(){
1287         try
1288         {
1289             // register jdbc driver and establish the connection
1290             Class.forName("oracle.jdbc.driver.OracleDriver");
1291             con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
1292             // create CallableStatement obj
1293             cs=con.prepareCall("{call Auth_pro(?, ?, ?)}");
1294             //register OUT parameter with jdbc types
1295             cs.registerOutParameter(3,Types.VARCHAR);
1296         }
1297         catch(Exception e)
1298         {
1299             e.printStackTrace();
1300         }
1301     }
1302
1303     public void doGet(HttpServletRequest request, HttpServletResponse response)
1304         throws ServletException, IOException {
1305         //read form data
1306         String uname=request.getParameter("uname");
1307         String pass=request.getParameter("pwd");
1308         try
1309         {
1310             // call pl/sql procedure by setting values to IN params
1311             cs.setString(1,uname);
1312             cs.setString(2,pass);
1313             //call pl/sql procedure
1314             cs.execute();
1315             //gather results from OUT params
1316             String result=cs.getString(3);
1317             // display results
1318             PrintWriter pw=response.getWriter();
1319             response.setContentType("text/html");
1320             pw.println("Result is"+result);
1321             pw.println("<br>req obj class name"+request.getClass());
```

```
1321         pw.println("<br>res obj class name"+response.getClass());
1322     }
1323     catch(Exception e)
1324     { e.printStackTrace(); }
1325 } //doGet(-,-)
1326
1327 public void doPost(HttpServletRequest request, HttpServletResponse response)
1328     throws ServletException, IOException {
1329     doGet(request, response);
1330 }
1331
1332
1333 public void destroy() {
1334     //close jdbc objs
1335     try
1336     {
1337         if(cs!=null)
1338             cs.close();
1339         catch(Exception e)
1340             { e.printStackTrace(); }
1341
1342     try
1343     {
1344         if(con!=null)
1345             con.close();
1346         catch(Exception e)
1347             { e.printStackTrace(); }
1348
1349     } //try
1350     } //destroy()
1351 } //class
1352 =====
1353 App12>>>Application on Servlet to Db s/w(LoginApp) (Approach4 (DAO) >>>
1355 -----Login.html-----
1356
1357 <form action="loginurl">
1358     Username: <input type="text" name="user"><br>
1359     Password: <input type="text" name="pwd"><br>
1360     <input type="submit" value="Login">
1361 </form>
1362 -----web.xml-----
1363 <web-app>
1364     <servlet>
1365         <servlet-name>Login</servlet-name>
1366         <servlet-class>com.nt.LoginSrv</servlet-class>
1367     </servlet>
1368     <servlet-mapping>
1369         <servlet-name>Login</servlet-name>
1370         <url-pattern>/loginurl</url-pattern>
1371     </servlet-mapping>
1372 </web-app>
1373 -----LoginSrv.java-----
1374 //LoginSrv.java
1375 package com.nt;
1376
1377 import java.io.IOException;
1378 import java.io.PrintWriter;
1379
1380 import javax.servlet.ServletException;
```

```
1381 import javax.servlet.http.HttpServlet;
1382 import javax.servlet.http.HttpServletRequest;
1383 import javax.servlet.http.HttpServletResponse;
1384 import com.dao.AuthenticateDAO;
1385
1386 public class LoginSrv extends HttpServlet {
1387     protected void doGet(HttpServletRequest req, HttpServletResponse res)
1388         throws ServletException, IOException {
1389         //general settings
1390         res.setContentType("text/html");
1391         PrintWriter pw=res.getWriter();
1392         //read form data
1393         String un=req.getParameter("user");
1394         String pass=req.getParameter("pwd");
1395         // use DAO class persistence logic
1396         String result=AuthenticateDAO.validate(un,pass);
1397         // Display results
1398         pw.println("Result is:"+result);
1399         //close stream
1400         pw.close();
1401     }//doGet(-,-)
1402     protected void doPost(HttpServletRequest req, HttpServletResponse res)
1403         throws ServletException, IOException {
1404         doGet(req,res);
1405     }//doPost(-,-)
1406 } //class
1407 -----AuthenticateDAO.java-----
1408 //AuthenticateDAO.java
1409 package com.dao;
1410
1411 import java.sql.Connection;
1412 import java.sql.DriverManager;
1413 import java.sql.PreparedStatement;
1414 import java.sql.ResultSet;
1415
1416 public class AuthenticateDAO {
1417     private static final String qry="select count(*) from userlist where uname=? and pwd=?";
1418
1419     public static String validate(String un,String pass)
1420     {
1421         Connection con=null; PreparedStatement ps=null;
1422         ResultSet rs=null;
1423         try{
1424             //register driver and create con obj
1425             Class.forName("oracle.jdbc.driver.OracleDriver");
1426             con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","manager");
1427             //create PreparedStatement obj
1428             ps=con.prepareStatement(qry);
1429             //set param values
1430             ps.setString(1,un);
1431             ps.setString(2,pass);
1432             // execute Query
1433             rs=ps.executeQuery();
1434             //process the ResultSet obj
1435             int cnt=0;
1436             if(rs.next())
1437                 cnt=rs.getInt(1);
1438
1439             if(cnt==0)
1440                 return "InValid Credentials";
```

```
1441     else
1442         return "Valid Credentials";
1443     } //try
1444     catch(Exception e)
1445     {
1446         e.printStackTrace();
1447         return "Internal Problem";
1448     }
1449     finally
1450     {
1451         try{
1452             if(rs!=null)
1453                 rs.close();
1454         }
1455         catch(Exception e)
1456         { e.printStackTrace();}
1457
1458         try{
1459             if(ps!=null)
1460                 ps.close();
1461         }
1462         catch(Exception e)
1463         { e.printStackTrace();}
1464
1465         try{
1466             if(con!=null)
1467                 con.close();
1468         }
1469         catch(Exception e)
1470         { e.printStackTrace();}
1471     }
1472 } //validate(-,-)
1473 } //class
1474
1475 =====
1476 App13)Example App on rd.forward(-,-) and rd.include(-,-)
1477 =====
1478 -----input.html-----
1479 <form action="dburl" method="get">
1480     <b>Employee no </b><input type="text" name="tno"><br>
1481     <input type="submit" value="GetEmpDetails">
1482 </form>
1483 -----Footer.html-----
1484 <hr><br><br><br>
1485 <center><i><b>&copy; copy rights are reserved 2010-11</b></i></center>
1486 -----web.xml-----
1487 <web-app>
1488     <context-param>
1489         <param-name>driver</param-name>
1490         <param-value>oracle.jdbc.driver.OracleDriver</param-value>
1491     </context-param>
1492     <context-param>
1493         <param-name>url</param-name>
1494         <param-value>jdbc:oracle:thin:@localhost:1521:orcl</param-value>
1495     </context-param>
1496     <context-param>
1497         <param-name>dbuser</param-name>
1498             <param-value>scott</param-value>
1499     </context-param>
1500     <context-param>
```

```
1501      <param-name>dbpwd</param-name>
1502      <param-value>tiger</param-value>
1503  </context-param>
1504
1505  <servlet>
1506    <servlet-name>ABC</servlet-name>
1507    <servlet-class>com.nt.DBSrv</servlet-class>
1508  </servlet>
1509
1510 <servlet-mapping>
1511   <servlet-name>ABC</servlet-name>
1512   <url-pattern>/dburl</url-pattern>
1513 </servlet-mapping>
1514
1515 <servlet>
1516   <servlet-name>XYZ</servlet-name>
1517   <servlet-class>com.nt.ErrSrv</servlet-class>
1518 </servlet>
1519
1520 <servlet-mapping>
1521   <servlet-name>XYZ</servlet-name>
1522   <url-pattern>/eurl</url-pattern>
1523 </servlet-mapping>
1524
1525 <servlet>
1526   <servlet-name>H</servlet-name>
1527   <servlet-class>com.nt.HeaderSrv</servlet-class>
1528 </servlet>
1529
1530 <servlet-mapping>
1531   <servlet-name>H</servlet-name>
1532   <url-pattern>/Headurl</url-pattern>
1533 </servlet-mapping>
1534
1535
1536 <welcome-file-list>
1537   <welcome-file>input.html</welcome-file>
1538 </welcome-file-list>
1539
1540 </web-app>
1541 ----- DBSrv.java -----
1542 //DBSrv.java(Main Servlet prg)
1543 package com.nt;
1544 import javax.servlet.*;
1545 import javax.servlet.http.*;
1546 import java.io.*;
1547 import java.sql.*;
1548
1549 public class DBSrv extends HttpServlet
1550 {
1551     Connection con=null;
1552     PreparedStatement ps1;
1553     public void init()
1554     {
1555         try
1556         {
1557             //get Access to ServletConfig obj
1558             ServletContext sc=getServletContext();
1559             // reading context parameter values of web.xml
1560             String s1=sc.getInitParameter("driver");
```

```
1561     String s2=sc.getInitParameter("url");
1562     String s3=sc.getInitParameter("dbuser");
1563     String s4=sc.getInitParameter("dbpwd");
1564     //create jdbc con object.....  

1565     Class.forName(s1);
1566     con=DriverManager.getConnection(s2,s3,s4);
1567 }//try
1568 catch(SQLException se)
1569 {
1570     se.printStackTrace();
1571 }
1572 catch(Exception e)
1573 {
1574     e.printStackTrace();
1575 }
1576 }//init()
1577
1578
1579 public void doGet(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
1580 {
1581     // get PrintWriter obj
1582     PrintWriter pw=res.getWriter();
1583     // set response content type
1584     res.setContentType("text/html");
1585     System.out.println("start of req processing logic in DBSrv prg");
1586
1587     try
1588     {
1589         //include header content from HeaderSrv
1590         RequestDispatcher rd1=req.getRequestDispatcher("Headurl");//pointing to HeadSrv servlet prg
1591         rd1.include(req,res);//includes Header logic's Html output
1592
1593         // read form data from form page
1594         int no=Integer.parseInt(req.getParameter("tno"));
1595
1596         //write jdbc code to manipulate DB table data...
1597         ps1=con.prepareStatement("select ename,sal from emp where empno=?");
1598         ps1.setInt(1,no);
1599
1600         ResultSet rs=ps1.executeQuery();
1601
1602         if(rs.next())
1603         {
1604             pw.println("Employee name is:"+rs.getString(1));
1605             pw.println("Employee Salary is:"+rs.getInt(2));
1606         }
1607
1608         //close ResultSet obj
1609         rs.close();
1610         System.out.println("end of req processing logic in DBSrv prg");
1611         //include FooterContent from Footer.html
1612         RequestDispatcher rd2=req.getRequestDispatcher("Footer.html");
1613         rd2.include(req,res); //includes Footer logic's Html output
1614
1615     } //try
1616     catch(Exception e)
1617     {
1618         RequestDispatcher rd=req.getRequestDispatcher("eurl"); //pointing ErrSrv servlet prg
1619         rd.forward(req,res); //forwards the request
1620     }
}
```

```
1621 } //doGet(-,-)
1622
1623 public void doPost(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
1624 {
1625     try
1626     {
1627         doGet(req,res);
1628     }
1629     catch(Exception e)
1630     {
1631         e.printStackTrace();
1632     }
1633 } //doPost(-,-)
1634
1635
1636 public void destroy()
1637 {
1638     try
1639     {
1640         if(ps1!=null)
1641             ps1.close();
1642     }
1643     catch(SQLException se)
1644     {
1645         se.printStackTrace();
1646     }
1647     try
1648     {
1649         if(con!=null)
1650             con.close();
1651     }
1652     catch(SQLException se)
1653     {
1654         se.printStackTrace();
1655     }
1656
1657 } //destroy
1658 }
1659 -----ErrSrv.java-----
1660 package com.nt;
1661 import javax.servlet.*;
1662 import javax.servlet.http.*;
1663 import java.io.*;
1664
1665 public class ErrSrv extends HttpServlet
1666 {
1667
1668     public void service(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
1669     {
1670         PrintWriter pw = res.getWriter();
1671         res.setContentType("text/html");
1672         //include header content from HeaderSrv
1673         RequestDispatcher rd1 = req.getRequestDispatcher("Headurl"); //pointing to HeadSrv servlet prg
1674         rd1.include(req,res); //includes Header logic's Html output
1675
1676         pw.println("<font color=red size=2>Internal problem</font>");
1677         pw.println("<br><br><a href='input.html'>home</a>");
1678         System.out.println("Service(-,-) in ErrSrv prg");
1679
1680 }
```

```
1681     //include FooterContent from Footer.html
1682     RequestDispatcher rd2=req.getRequestDispatcher("Footer.html");
1683     rd2.include(req,res); //includes Footer logic's Html output
1684
1685 }
1687 }//class
1688 -----HeaderSrv.java-----
1689 package com.nt;
1690 import javax.servlet.*;
1691 import javax.servlet.http.*;
1692 import java.io.*;
1693
1694 public class HeaderSrv extends HttpServlet
1695 {
1696
1697     public void service(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
1698     {
1699         PrintWriter pw=res.getWriter();
1700         res.setContentType("text/html");
1701
1702         // writer Header logic
1703         pw.println("<center><font color=red size=7>HCL Technologies </font></center><br>");
1704         pw.println("<br><br><br><hr>");
1705     }//service(-,-)
1706 }//class
1707 -----
1708 App14(Using rd.forward(-,-) & rd.include(-,-) )
1709 =====
1710 -----Main.html-----
1711 <html>
1712     <head>
1713         <title> Forward - Include Demonstration </title>
1714     </head>
1715     <frameset rows="30%, *">
1716         <frame src="Search.html" name="SearchFrame" >
1717         <frame name="ResultsFrame">
1718     </frameset>
1719 </html>
1720 -----Search.html-----
1721 <html>
1722     <center>
1723         <form target="ResultsFrame" action="mainurl" method="GET" >
1724             Select Designation(s)
1725             <select name="job" multiple>
1726                 <option value=CLERK>CLERKS</option>
1727                 <option value=SALEMEN>SALEMEN</option>
1728                 <option value=MANAGER>MANAGERS</option>
1729                 <option value=ANALYST>ANALYSTS</option>
1730                 <option value=PRESIDENT>PRESIDENTS</option>
1731             </select>
1732             <input type=submit value=execute>
1733         </form>
1734     </center>
1735 </html>
1736 -----web.xml-----
1737 <?xml version="1.0" encoding="ISO-8859-1"?>
1738
1739 <!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
1740           "http://java.sun.com/dtd/web-app_2_3.dtd">
```

```
1741 <web-app>
1742     <servlet>
1743         <servlet-name> servlet1 </servlet-name>
1744         <servlet-class> com.nt.DummyServlet </servlet-class>
1745     </servlet>
1746     <servlet>
1747         <servlet-name> servlet2 </servlet-name>
1748         <servlet-class> com.nt.ProcessServlet </servlet-class>
1749     </servlet>
1750     <servlet>
1751         <servlet-name> servlet3 </servlet-name>
1752         <servlet-class> com.nt.DBServlet </servlet-class>
1753     </servlet>
1754
1755
1756     <servlet-mapping>
1757         <servlet-name> servlet1 </servlet-name>
1758         <url-pattern> /mainurl </url-pattern>
1759     </servlet-mapping>
1760     <servlet-mapping>
1761         <servlet-name> servlet2 </servlet-name>
1762         <url-pattern> /HTMLServlet </url-pattern>
1763     </servlet-mapping>
1764     <servlet-mapping>
1765         <servlet-name> servlet3 </servlet-name>
1766         <url-pattern> /QueryServlet </url-pattern>
1767     </servlet-mapping>
1768
1769     <welcome-file-list>
1770         <welcome-file> Main.html </welcome-file>
1771     </welcome-file-list>
1772 </web-app>
1773 ----- DummyServlet.java -----
1774 // This servlet just forwards the request to ProcessServlet.
1775 // This is used just for demonstrating forward concept.
1776 // Any HTML response, from this servlet will not be received by the client.
1777 // If response is committed, using flush() method, then doing a forward / include
1778 // will give InvalidStateException.
1779 package com.nt;
1780 import javax.servlet.*;
1781 import javax.servlet.http.*;
1782 import java.io.*;
1783
1784 public class DummyServlet extends HttpServlet
1785 {
1786     public void doGet(HttpServletRequest req, HttpServletResponse res)
1787             throws ServletException, IOException
1788     {
1789         ServletContext sc = getServletContext();
1790         RequestDispatcher rd = sc.getRequestDispatcher("/HTMLServlet");
1791         rd.forward(req, res);
1792     }
1793 }
1794 ----- ProcessServlet.java -----
1795 // This servlet includes response of DBServlet in between Table heading and closing.
1796 package com.nt;
1797 import javax.servlet.*;
1798 import javax.servlet.http.*;
1799 import java.io.*;
1800
```

```
1801 public class ProcessServlet extends HttpServlet
1802 {
1803     public void doGet(HttpServletRequest req, HttpServletResponse res)
1804         throws ServletException, IOException
1805     {
1806         res.setContentType("text/html");
1807         PrintWriter pw = res.getWriter();
1808
1809         pw.println("<HTML><BODY>");
1810         pw.println("<CENTER><TABLE BORDER=1>");
1811         pw.println("<tr><th>EMPNO</th>");
1812         pw.println("<th>ENAME</th>");
1813         pw.println("<th>JOB</th>");
1814         pw.println("<th>MGR</th>");
1815         pw.println("<th>HIREDATE</th>");
1816         pw.println("<th>SAL</th>");
1817         pw.println("<th>COMM</th>");
1818         pw.println("<th>DEPTNO</th></tr>");
1819
1820         ServletContext sc = getServletContext();
1821         RequestDispatcher rd = sc.getRequestDispatcher("/QueryServlet");
1822         rd.include(req, res);
1823
1824         pw.println("</TABLE></CENTER>");
1825         pw.println("</HTML></BODY>");
1826     }
1827 }
1828 -----DBServlet.java-----
1829 package com.nt;
1830 import javax.servlet.*;
1831 import javax.servlet.http.*;
1832 import java.io.*;
1833 import java.sql.*;
1834 import java.math.*;
1835
1836
1837 public class DBServlet extends HttpServlet
1838 {
1839     public void doGet(HttpServletRequest req, HttpServletResponse res)
1840         throws ServletException, IOException
1841     {
1842         Connection con = null;
1843         Statement stmt = null;
1844         ResultSet rs = null;
1845         PrintWriter pw = res.getWriter();
1846
1847         String jobs[] = req.getParameterValues("job");
1848
1849         StringBuffer sb = new StringBuffer();
1850
1851         sb.append("(");
1852
1853         for (int i = 0; i < jobs.length; i++)
1854         {
1855             if (i == jobs.length - 1)
1856                 sb.append(" " + jobs[i] + " ");
1857             else
1858                 sb.append(" " + jobs[i] + " ,");
1859         }
1860     }
```

```
1861 sb.append(")");
1862
1863 String condition=sb.toString();
1864
1865 try
1866 {
1867     Class.forName("oracle.jdbc.driver.OracleDriver");
1868     con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott", "tiger");
1869
1870     stmt=con.createStatement();
1871     rs=stmt.executeQuery("SELECT * FROM EMP WHERE JOB IN " + condition+ " ORDER BY JOB ");
1872
1873     while(rs.next())
1874     {
1875         int empno=rs.getInt(1);
1876         String ename=rs.getString(2);
1877         String job=rs.getString(3);
1878         int mgr=rs.getInt(4);
1879         Date hiredate=rs.getDate(5);
1880         float sal=rs.getFloat(6);
1881         float comm=rs.getFloat(7);
1882         int deptno=rs.getInt(8);
1883
1884         BigDecimal bd=new BigDecimal(sal);
1885         pw.println("<tr>");
1886         pw.println("<td>" +empno+ "</td>");
1887         pw.println("<td>" +ename+ "</td>");
1888         pw.println("<td>" +job+ "</td>");
1889         pw.println("<td>" +mgr+ "</td>");
1890         pw.println("<td>" +hiredate+ "</td>");
1891         pw.println("<td>" +bd.setScale(2,BigDecimal.ROUND_HALF_UP)+ "</td>");
1892         pw.println("<td>" +comm+ "</td>");
1893         pw.println("<td>" +deptno+ "</td>");
1894         pw.println("</tr>");
1895     } // while
1896
1897     rs.close();
1898     con.close();
1899 } // try
1900 catch(Exception e)
1901 {
1902     e.printStackTrace();
1903 }
1904 } // doGet()
1905 } // class
1906
1907 App15>>>Example App on rd.include(-,-) to get communication b/w two web applications of same server(weblogic)
1908 -----
1909 WaOne (First App)
1910 -----Form.html-----
1911
1912 <form action="furl" method="get">
1913     Value: <input type="text" name="t1"><br>
1914         <input type="submit" value="GetSquare&Cube">
1915 </form>
1916
1917 -----web.xml-----
1918 <web-app>
1919     <servlet>
1920         <servlet-name>abc</servlet-name>
```

```
1921      <servlet-class>com.nt.FSrv</servlet-class>
1922      </servlet>
1923      <servlet-mapping>
1924          <servlet-name>abc</servlet-name>
1925          <url-pattern>/furl</url-pattern>
1926      </servlet-mapping>
1927  </web-app>
1928 -----FSrv.java-----
1929 //FSrv.java (Servlet prg)
1930 package com.nt;
1931 import javax.servlet.*;
1932 import javax.servlet.http.*;
1933 import java.io.*;
1934 public class FSrv extends HttpServlet
1935 {
1936     public void doGet(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
1937     {
1938         // General settings
1939         res.setContentType("text/html");
1940         PrintWriter pw = res.getWriter();
1941         //read form data
1942         int val = Integer.parseInt(req.getParameter("t1"));
1943         //Display Square value
1944         pw.println("<br>FSrv: square value is " + (val * val));
1945         // include the response of SSrv prg belonging to WaTwo web application
1946         // get WaOne web application's ServletContext obj
1947         ServletContext sc1 = getServletContext();
1948         // get WaTwo web application's ServletContext obj
1949         ServletContext sc2 = sc1.getServletContext("WaTwo");
1950         // get RequestDispatcher obj pointing to SSrv prg of WaTwo
1951         RequestDispatcher rd = sc2.getRequestDispatcher("/surl");
1952         //include response
1953         rd.include(req, res);
1954         //close stream
1955         pw.close();
1956     } //doGet(-,-)
1957
1958     public void doPost(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
1959     {
1960         doGet(req, res);
1961     } //doPost(-,-)
1962 } //class
1963 -----
1964 WaTwo (Second web application)
1965 -----SSrv.java-----
1966 // SSrv.java
1967 package com.nt;
1968 import javax.servlet.*;
1969 import javax.servlet.http.*;
1970 import java.io.*;
1971
1972 public class SSrv extends HttpServlet
1973 {
1974     public void doGet(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
1975     {
1976         // General settings
1977         PrintWriter pw = res.getWriter();
1978         res.setContentType("text/html");
1979         // read form data
1980         int val = Integer.parseInt(req.getParameter("t1"));
```

```
1981     pw.println("<br> Cube value is"+(val*val*val));
1982
1983     //do not call pw.close();
1984 }//doGet(-,-)
1985 public void doPost(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
1986 {
1987     doGet(req,res);
1988 }//doPost(-,-)
1989 }//class
1990 -----web.xml-----
1991 <web-app>
1992   <servlet>
1993     <servlet-name>abc</servlet-name>
1994     <servlet-class>com.nt.SSrv</servlet-class>
1995   </servlet>
1996   <servlet-mapping>
1997     <servlet-name>abc</servlet-name>
1998     <url-pattern>/surl</url-pattern>
1999   </servlet-mapping>
2000 </web-app>
2001 -----
2002 -----
2003 App16>>>>>>>>Example App res.sendRedirect(-)
2004 -----
2005 TestApp (First App)
2006 -----Form.html-----
2007
2008 <form action="billurl" >
2009   Item Name: <input type="text" name="t1"><br>
2010   Item Price : <input type="text" name="t2"><br>
2011   Item Qty : <input type="text" name="t3"><br>
2012   <input type="submit" value="Get Bill Details">
2013 </form>
2014
2015 -----web.xml-----
2016
2017 <web-app>
2018   <servlet>
2019     <servlet-name>abc</servlet-name>
2020     <servlet-class>com.nt.BillSrv</servlet-class>
2021   </servlet>
2022
2023   <servlet-mapping>
2024     <servlet-name>abc</servlet-name>
2025     <url-pattern>/billurl</url-pattern>
2026   </servlet-mapping>
2027
2028 </web-app>
2029
2030 -----BillSrv.java-----
2031 //BillSrv.java
2032 package com.nt;
2033
2034 import javax.servlet.*;
2035 import javax.servlet.http.*;
2036 import java.io.*;
2037
2038 public class BillSrv extends HttpServlet
2039 {
2040   public void doGet(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
```

```
2041 {
2042     //General settings
2043     PrintWriter pw=res.getWriter();
2044     res.setContentType("text/html");
2045
2046     //read form data
2047     String name=req.getParameter("t1");
2048     float price=Float.parseFloat(req.getParameter("t2"));
2049     float qty=Float.parseFloat(req.getParameter("t3"));
2050     //calc Bill Amt
2051     float bamt=price*qty;
2052
2053     if(bamt>=50000)
2054     {
2055         System.out.println("before res.sendRedirect(-) in BillSrv");
2056         res.sendRedirect("http://localhost:7879/TestApp2/discounturl?bill="+bamt+"&iname="+name);
2057         System.out.println("after res.sendRedirect(-) in BillSrv");
2058     }//if
2059     else
2060     {
2061         pw.println("<br> From BillSrv prg <br>");
2062         pw.println("<br> Item name:"+name+" Price:"+price+" Qty :" +qty+"Bill Amt:"+bamt);
2063     }//else
2064
2065     //close stream
2066     pw.close();
2067 } //doGet(-,-)
2068
2069 public void doPost(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
2070 {
2071     doGet(req,res);
2072 } //doPost(-,-)
2073 } //class
2074 -----
2075 TestApp2
2076 -----DiscountSrv.java-----
2077 //DiscountSrv.java
2078 package com.nt;
2079 import javax.servlet.*;
2080 import javax.servlet.http.*;
2081 import java.io.*;
2082
2083 public class DiscountSrv extends HttpServlet
2084 {
2085     public void doGet(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
2086     {
2087         // General settings
2088         PrintWriter pw=res.getWriter();
2089         res.setContentType("text/html");
2090
2091         //read additional request params given by BillSrv prg (source prg)
2092         float amt=Float.parseFloat(req.getParameter("bill"));
2093         String name=req.getParameter("iname");
2094
2095         System.out.println("From DiscountSrv: doGet(-,-) method");
2096
2097         //Calc discount Amt
2098         float discount= amt * 0.3f;
2099         float finalamt=amt-discount;
2100 }
```

```
2101 // Display Details
2102 pw.println("<br> Item name :" +name);
2103 pw.println("<br> Bill Amt :" +amt);
2104 pw.println("<br> Duscount :" +discount+ " Final Amount:" +finalamt);
2105 pw.println("<br> From DiscountSrv prg");
2106 //close stream
2107 pw.close();
2108 } //doGet(-,-)
2109 public void doPost(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
2110 {
2111 doGet(req,res);
2112 }
2113
2114 }//class
2115 -----web.xml-----
2116
2117 <web-app>
2118 <servlet>
2119 <servlet-name>xyz</servlet-name>
2120 <servlet-class>com.nt.DiscountSrv</servlet-class>
2121 </servlet>
2122
2123 <servlet-mapping>
2124 <servlet-name>xyz</servlet-name>
2125 <url-pattern>/discounturl</url-pattern>
2126 </servlet-mapping>
2127 </web-app>
2128
2129 =====
2130 App17 (Using res.sendRedirect(-) to redirect the request to internet websites)
2131 =====
2132 -----input.html-----
2133 <form action="searchurl" method="get">
2134 Search String: <input type="text" name="tsearch"><br>
2135 SearchEngg : <input type="radio" name="engg" value="google" checked>Google
2136 <input type="radio" name="engg" value="bing" >Bing
2137 <input type="radio" name="engg" value="yahoo" >YahooSearch
2138 <br>
2139 <input type="submit" value="search">
2140 </form>
2141 -----web.xml-----
2142 <web-app>
2143 <servlet>
2144 <servlet-name>abc</servlet-name>
2145 <servlet-class>com.nt.SearchSrv</servlet-class>
2146 </servlet>
2147
2148 <servlet-mapping>
2149 <servlet-name>abc</servlet-name>
2150 <url-pattern>/searchurl</url-pattern>
2151 </servlet-mapping>
2152 </web-app>
2153 -----SearchSrv.java-----
2154 //SeachSrv.java
2155 package com.nt;
2156 import javax.servlet.*;
2157 import javax.servlet.http.*;
2158 import java.io.*;
2159 import java.util.*;
2160
```



```
2221     ServletContext sc=getServletContext();
2222     sc.setAttribute("attr3","val3");
2223
2224     //forward request to Srv2 prg
2225     RequestDispatcher rd=request.getRequestDispatcher("/Srv2");
2226     rd.forward(request,response);
2227 }
2228 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
2229 {
2230     doGet(request,response);
2231 }
2232 }
2233 -----Srv2.java-----
2234 package com.nt;
2235
2236 import java.io.IOException;
2237 import java.io.PrintWriter;
2238
2239 import javax.servlet.RequestDispatcher;
2240 import javax.servlet.ServletContext;
2241 import javax.servlet.ServletException;
2242 import javax.servlet.http.HttpServlet;
2243 import javax.servlet.http.HttpServletRequest;
2244 import javax.servlet.http.HttpServletResponse;
2245 import javax.servlet.http.HttpSession;
2246
2247 public class Srv2 extends HttpServlet {
2248
2249     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
2250         // General settings
2251         PrintWriter pw=response.getWriter();
2252         response.setContentType("text/html");
2253
2254         //read request attribute value
2255         pw.println("Srv2:attr1 (request) value"+request.getAttribute("attr1"));
2256         //read session attribute value
2257         HttpSession ses=request.getSession();
2258         pw.println("<br>Srv2:attr2 (session) value"+ses.getAttribute("attr2"));
2259         //read ServletContext attribuite value
2260         ServletContext sc=getServletContext();
2261         pw.println("<br>Srv2:attr3 (sc) value"+sc.getAttribute("attr3"));
2262
2263         //forward request to Srv3 prg
2264         RequestDispatcher rd=request.getRequestDispatcher("/Srv3");
2265         rd.forward(request,response);
2266     }
2267     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
2268         doGet(request,response);
2269     }
2270 }
2271 -----Srv3.java-----
2272 package com.nt;
2273 import java.io.IOException;
2274 import java.io.PrintWriter;
2275
2276 import javax.servlet.RequestDispatcher;
2277 import javax.servlet.ServletContext;
2278 import javax.servlet.ServletException;
2279 import javax.servlet.annotation.WebServlet;
2280 import javax.servlet.http.HttpServlet;
```

```
2281 import javax.servlet.http.HttpServlet;
2282 import javax.servlet.http.HttpServletRequest;
2283 import javax.servlet.http.HttpSession;
2284
2285 public class Srv3 extends HttpServlet {
2286
2287     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
2288
2289         // General settings
2290         PrintWriter pw=response.getWriter();
2291         response.setContentType("text/html");
2292
2293         //read request attribute value
2294         pw.println("Srv3:attr1 (request) value"+request.getAttribute("attr1"));
2295
2296
2297         //read session attribute value
2298         HttpSession ses=request.getSession();
2299         pw.println("<br>Srv3:attr2 (session) value"+ses.getAttribute("attr2"));
2300
2301         //read ServletContext attribuite value
2302         ServletContext sc=getServletContext();
2303         pw.println("<br>Srv3:attr3 (sc) value"+sc.getAttribute("attr3"));
2304
2305     }
2306     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
2307         doGet(request,response);
2308     }
2309 }
2310 -----Srv4.java-----
2311 package com.nt;
2312 import java.io.IOException;
2313 import java.io.PrintWriter;
2314
2315 import javax.servlet.RequestDispatcher;
2316 import javax.servlet.ServletContext;
2317 import javax.servlet.ServletException;
2318 import javax.servlet.annotation.WebServlet;
2319 import javax.servlet.http.HttpServlet;
2320 import javax.servlet.http.HttpServletRequest;
2321 import javax.servlet.http.HttpServletResponse;
2322 import javax.servlet.http.HttpSession;
2323
2324 public class Srv4 extends HttpServlet {
2325
2326     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
2327
2328         // General settings
2329         PrintWriter pw=response.getWriter();
2330         response.setContentType("text/html");
2331
2332         //read request attribute value
2333         pw.println("Srv4:attr1 (request) value"+request.getAttribute("attr1"));
2334
2335         //read session attribute value
2336         HttpSession ses=request.getSession();
2337         pw.println("<br>Srv4:attr2 (session) value"+ses.getAttribute("attr2"));
2338
2339         //read ServletContext attribuite value
2340         ServletContext sc=getServletContext();
```

```
2341     pw.println("<br>Srv4:attr3 (sc) value"+sc.getAttribute("attr3"));
2342
2343 }
2344 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
2345
2346     doGet(request,response);
2347 }
2348 }
2349 -----web.xml-----
2350 <web-app>
2351     <servlet>
2352         <servlet-name>ABC</servlet-name>
2353         <servlet-class>com.nt.Srv1</servlet-class>
2354     </servlet>
2355
2356     <servlet-mapping>
2357         <servlet-name>ABC</servlet-name>
2358         <url-pattern>/s1url</url-pattern>
2359     </servlet-mapping>
2360
2361     <servlet>
2362         <servlet-name>XYZ</servlet-name>
2363         <servlet-class>com.nt.Srv2</servlet-class>
2364     </servlet>
2365
2366     <servlet-mapping>
2367         <servlet-name>XYZ</servlet-name>
2368         <url-pattern>/s2url</url-pattern>
2369     </servlet-mapping>
2370
2371     <servlet>
2372         <servlet-name>mno</servlet-name>
2373         <servlet-class>com.nt.Srv3</servlet-class>
2374     </servlet>
2375
2376     <servlet-mapping>
2377         <servlet-name>mno</servlet-name>
2378         <url-pattern>/s3url</url-pattern>
2379     </servlet-mapping>
2380     <servlet>
2381         <servlet-name>jkr</servlet-name>
2382         <servlet-class>com.nt.Srv4</servlet-class>
2383     </servlet>
2384
2385     <servlet-mapping>
2386         <servlet-name>jkr</servlet-name>
2387         <url-pattern>/s4url</url-pattern>
2388     </servlet-mapping>
2389
2390 </web-app>
2391 -----
2392
2393
2394 -----
2395 App19>>>>>>>>>Example App on Annotations based servlet
2396 -----
2397 -----input.html-----
2398
2399 <form action="test1" method="get">
2400     Name :<input type="text" name="tname"><br>
```

```
2401 <input type="submit" value="send">
2402 </form>
2403 -----FormSrv.java-----
2404 package com.nt;
2405 import java.io.IOException;
2406 import java.io.PrintWriter;
2407
2408 import javax.servlet.ServletException;
2409 import javax.servlet.annotation.WebServlet;
2410 import javax.servlet.http.HttpServlet;
2411 import javax.servlet.http.HttpServletRequest;
2412 import javax.servlet.http.HttpServletResponse;
2413
2414 /**
2415 * Servlet implementation class FormSrv
2416 */
2417 @WebServlet(
2418     name = "abc",
2419     urlPatterns = {
2420         "/test2",
2421         "/test1"
2422     })
2423 public class FormSrv extends HttpServlet {
2424     @Override
2425     public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
2426     {
2427         // get PrintWriter obj
2428         PrintWriter pw = res.getWriter();
2429         // set response content type
2430         res.setContentType("text/html");
2431
2432         // read form data
2433         String name = req.getParameter("tname");
2434
2435         //display response
2436         pw.println("Hello Mr./Miss./Mrs" + name + " Good morning ");
2437
2438         //close stream
2439         pw.close();
2440     }//doGet(-,-)
2441     protected void doPost(HttpServletRequest request, HttpServletResponse response)
2442         throws ServletException, IOException {
2443         doGet(request, response);
2444     }
2445 }
2446
2447 -----
2448 =====
2449 App20 (App to demonstrate Stateless Behaviour of WebApplication)
2450 =====
2451 -----Details.html-----
2452 <form action="s1url" method="get">
2453
2454     <b> Name: </b><input type="text" name="pname"><br>
2455     <b> Age :</b><input type="text" name="page"><br>
2456     <b> MaritalStatus: </b><input type="checkbox" value="married" name="ms"> Married
2457
2458     <input type="submit" value="continue">
2459 </form>
2460 -----web.xml-----
```

```
2461 <web-app>
2462   <servlet>
2463     <servlet-name>ABC</servlet-name>
2464     <servlet-class>com.nt.Srv1</servlet-class>
2465   </servlet>
2466
2467   <servlet-mapping>
2468     <servlet-name>ABC</servlet-name>
2469     <url-pattern>/s1url</url-pattern>
2470   </servlet-mapping>
2471
2472   <servlet>
2473     <servlet-name>XYZ</servlet-name>
2474     <servlet-class>com.nt.Srv2</servlet-class>
2475   </servlet>
2476
2477   <servlet-mapping>
2478     <servlet-name>XYZ</servlet-name>
2479     <url-pattern>/s2url</url-pattern>
2480   </servlet-mapping>
2481
2482   <welcome-file-list>
2483     <welcome-file>Details.html</welcome-file>
2484   </welcome-file-list>
2485
2486 </web-app>
2487 -----Srv1.java-----
2488 /Srv1.java (Generates Second Form Dynamically Based on first Form Data)
2489 package com.nt;
2490 import javax.servlet.*;
2491 import javax.servlet.http.*;
2492 import java.io.*;
2493
2494 public class Srv1 extends HttpServlet
2495 {
2496
2497   public void doGet(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
2498   {
2499     // read form1 data
2500     String name=req.getParameter("pname");
2501     String age=req.getParameter("page");
2502     String mstatus=req.getParameter("ms");
2503
2504     if(mstatus==null)
2505       mstatus="single";
2506
2507
2508     // Get PrintWriter obj
2509     PrintWriter pw=res.getWriter();
2510     res.setContentType("text/html");
2511
2512     // Generate Secohd Form Dynamically Form here
2513     if(mstatus.equals("married"))
2514     {
2515       pw.println("<form action='s2url' >");
2516       pw.println("<b> Spouse Name </b><input type='text' name='st1'><br>");
2517       pw.println("<b> No.of Children </b><input type='text' name='st2'><br>");
2518       pw.println("<input type='submit' value='sumbit'>");
2519       pw.println("</form>");
2520     }
  }
```

```

2521     else
2522     {
2523         pw.println("<form action='s2url' >");
2524         pw.println("<b> when do u want marry </b><input type='text' name='st1'><br>");
2525         pw.println("<b> why do u want marry </b><input type='text' name='st2'><br>");
2526         pw.println("<input type='submit' value='sumbit'>");
2527         pw.println("</form>");
2528     }
2529
2530     pw.close();
2531 } //doGet(-,-)
2532 public void doPost(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
2533 {
2534     doGet(req,res);
2535 }
2536
2537 }//class
2538 -----Srv2.java-----
2539 //Srv2.java (takes the request from Second form (Dynamic Form))
2540 package com.nat;
2541 import javax.servlet.*;
2542 import javax.servlet.http.*;
2543 import java.io.*;
2544
2545 public class Srv2 extends HttpServlet
2546 {
2547
2548     public void doGet(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
2549     {
2550         // get PrintWriter object
2551         PrintWriter pw = res.getWriter();
2552         res.setContentType("text/html");
2553
2554         // read 1st form , second form data
2555         pw.println("<br>First Form data is ");
2556         pw.println("<br>Name is "+req.getParameter("pname"));
2557         pw.println("<br>Age is "+req.getParameter("page"));
2558         pw.println("<br>Marital Status is "+req.getParameter("ms"));
2559
2560         pw.println("<br><br>Second form data is "+req.getParameter("st1")+"<br> "+req.getParameter("st2"));
2561
2562         pw.close();
2563 } //doGet(-,-)
2564 public void doPost(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
2565 {
2566     doGet(req,res);
2567 }
2568
2569 }//class
2570 -----
2571 =====
2572 App21(SessionTracking using HiddenForm Fields)
2573 =====
2574 -----input.html-----
2575
2576 <form action="t1url" method="get">
2577     Name: <input type="text" name="tname"><br>
2578     Father name: <input type="text" name="tfname"><br>
2579     Marital Status <input type="radio" name="ms" value="married">Married
2580     &nbsp;&nbsp;&nbsp;

```

```
2581     <input type="radio" name="ms" value="single">Single <br>
2582
2583     <input type="submit" value="continue">
2584 </form>
2585
2586 -----web.xml-----
2587 <web-app>
2588     <servlet>
2589         <servlet-name>abc1</servlet-name>
2590         <servlet-class>com.nt.TestSrv1</servlet-class>
2591     </servlet>
2592
2593     <servlet-mapping>
2594         <servlet-name>abc1</servlet-name>
2595         <url-pattern>/t1url</url-pattern>
2596     </servlet-mapping>
2597
2598     <servlet>
2599         <servlet-name>xyz1</servlet-name>
2600         <servlet-class>com.nt.TestSrv2</servlet-class>
2601     </servlet>
2602
2603     <servlet-mapping>
2604         <servlet-name>xyz1</servlet-name>
2605         <url-pattern>/t2url</url-pattern>
2606     </servlet-mapping>
2607
2608 </web-app>
2609 -----TestSrv1.java-----
2610 //TestSrv1.java (generates dynamic form page)
2611 package com.nt;
2612 import javax.servlet.*;
2613 import javax.servlet.http.*;
2614 import java.io.*;
2615
2616 public class TestSrv1 extends HttpServlet
2617 {
2618
2619     public void service(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
2620     {
2621         // general code
2622         PrintWriter pw = res.getWriter();
2623         res.setContentType("text/html");
2624
2625         // read form1 data
2626         String pname = req.getParameter("tname");
2627         String pfname = req.getParameter("tfname");
2628         String pms = req.getParameter("ms");
2629
2630         // design and send dynamic form page to browser window
2631         // based on the marital status....
2632         if(pms.equals("single"))
2633         {
2634             pw.println("<form action='t2url' method='get'>");
2635             pw.println("why do u want to marry <input type='text' name='f2t1'><br>");
2636             pw.println("when do u want to marry <input type='text' name='f2t2'><br>");
2637             // add form1/req1 data to dynamic form page as hidden box values
2638             pw.println("<input type='hidden' name='hname' value='"+pname+">");
2639             pw.println("<input type='hidden' name='hfname' value='"+pfname+">");
2640             pw.println("<input type='hidden' name='hms' value='"+pms+">");
```

```

2641
2642         pw.println("<input type='submit' value='submit'>");
2643         pw.println("</form>");
2644     }
2645     else
2646     {
2647         pw.println("<form action='t2url' method='get'>");
2648         pw.println("Spouse name<input type='text' name='f2t1'><br>");
2649         pw.println("no.of children <input type='text' name='f2t2'><br>");
2650         // add form1/req1 data to dynamic form page as hidden box values
2651         pw.println("<input type='hidden' name='hname' value='"+pname+">");
2652         pw.println("<input type='hidden' name='hfname' value='"+pfname+">");
2653         pw.println("<input type='hidden' name='hms' value='"+pms+">");
2654
2655         pw.println("<input type='submit' value='submit'>");
2656         pw.println("</form>");
2657     }
2658
2659     //close the stream obj
2660     pw.close();
2661 } //service
2662 } //class
2663 -----TestSrv2.java-----
2664 //TestSrv2.java
2665 package com.nt;
2666 import javax.servlet.*;
2667 import javax.servlet.http.*;
2668 import java.io.*;
2669 import java.sql.*;
2670
2671 public class TestSrv2 extends HttpServlet
2672 {
2673
2674     public void service(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
2675     {
2676         // general code
2677         PrintWriter pw=res.getWriter();
2678         res.setContentType("text/html");
2679
2680         // read form1 data
2681         String name=req.getParameter("hname");
2682         String fname=req.getParameter("hfname");
2683         String ms=req.getParameter("hms");
2684         // read form2 data
2685         String f2val1=req.getParameter("f2t1");
2686         String f2val2=req.getParameter("f2t2");
2687
2688         // write form1/req1,from2/req2 values to DB table as record
2689         try
2690         {
2691             Class.forName("oracle.jdbc.driver.OracleDriver");
2692             Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:ordl","scott","tiger");
2693             PreparedStatement ps=con.prepareStatement("insert into person_tab values(?,?,?,?,?)");
2694             ps.setString(1,name);
2695             ps.setString(2,fname);
2696             ps.setString(3,ms);
2697             ps.setString(4,f2val1);
2698             ps.setString(5,f2val2);
2699
2700             ps.executeUpdate();

```

```
2701                               pw.println("<br>Record Inserted<br>");
2702 }//try
2703 catch(Exception e)
2704 {e.printStackTrace(); }
2705
2706 // display form1 ,form2 data on the browser window
2707 pw.println("<br> form1 data is"+name+"..."+fname+"...."+ms);
2708 pw.println("<br> form2 data is"+f2val1+"..."+f2val2);
2709
2710 //close stream obj
2711 pw.close();
2712 }//service
2713 }//class
2714 -----
2715 =====
2716 App22(Basic Example on Cookies)
2717 =====
2718 -----
2719 -----SetCookiesSrv.java-----
2720 package com.nt;
2721 import java.io.*;
2722 import javax.servlet.*;
2723 import javax.servlet.http.*;
2724
2725 public class SetCookiesSrv extends HttpServlet {
2726
2727     public void doGet(HttpServletRequest request,HttpServletResponse response)
2728             throws ServletException,IOException {
2729
2730         Cookie cookie1,cookie2,cookie3,cookie4;
2731         // default maxage is -1, indicating cookie
2732         //2 in-memory cookies applies only to current browsing session
2733         cookie1=new Cookie("ap","Hyderabad");
2734         cookie2=new Cookie("tn","Chennai");
2735         response.addCookie(cookie1);
2736         response.addCookie(cookie2);
2737
2738         // Cookie is valid for an hour, regardless of whether
2739         //user quits browser, reboots computer or whatever
2740         //2 persistent cookies
2741         cookie3=new Cookie("kr","Bangalore");
2742         cookie4=new Cookie("mh","mumbai");
2743         cookie3.setMaxAge(3600);
2744         cookie4.setMaxAge(3600);
2745
2746         response.addCookie(cookie3);
2747         response.addCookie(cookie4);
2748
2749         //get PrintWriter obj
2750         PrintWriter pw=res.getWriter();
2751         response.setContentType("text/html");
2752
2753         pw.println("Successful in setting cookies");
2754         System.out.println("Successful in setting cookies....");
2755     } //doGet()
2756 } //class
2757 -----ShowCookiesSrv.java-----
2758 // program to work with cookies
2759 package com.nt;
2760 import java.io.*;
```

```
2761 import javax.servlet.*;
2762 import javax.servlet.http.*;
2763 public class ShowCookiesSrv extends HttpServlet {
2764
2765     public void doGet(HttpServletRequest request,HttpServletResponse response)
2766             throws ServletException,IOException {
2767         response.setContentType("text/html");
2768         PrintWriter out=response.getWriter();
2769         String title="Active cookies";
2770
2771         out.println("<html> <head> <title>" + title);
2772         out.println("</title> </head> <body>");
2773         out.println("<table border=1 align='center'>");
2774         out.println("<tr> <td> Cookie Name</td>");
2775         out.println("<td> Cookie Value</td></tr>");
2776
2777         Cookie[] ck=request.getCookies();
2778         if(ck!=null) {
2779             for (int i=0;i<ck.length;i++) {
2780                 out.println("<tr><td>" + ck[i].getName() + "</td>" +
2781                         "<td>" + ck[i].getValue() + "</td></tr>");
2782             }//for
2783             out.println("</table></body></html>");
2784         }//if
2785         else {
2786             System.out.println("No cookies present....");
2787             pw.println("<br><b>no cookies present.....</b>");
2788         }//else
2789
2790     }//doGet()
2791 } //class
2793 -----web.xml-----
2794 <web-app>
2795     <servlet>
2796         <servlet-name>s1</servlet-name>
2797         <servlet-class>com.nt.SetCookiesSrv</servlet-class>
2798     </servlet>
2799     <servlet-mapping>
2800         <servlet-name>s1</servlet-name>
2801         <url-pattern>/test1</url-pattern>
2802     </servlet-mapping>
2803
2804     <servlet>
2805         <servlet-name>s2</servlet-name>
2806         <servlet-class>com.nt.ShowCookiesSrv</servlet-class>
2807     </servlet>
2808     <servlet-mapping>
2809         <servlet-name>s2</servlet-name>
2810         <url-pattern>/test2</url-pattern>
2811     </servlet-mapping>
2812 </web-app>
2813 =====
2814 App23(SessionTracking using Cookies)
2815 =====
2816 -----Tax.html-----
2817 <html>
2818     <body bgcolor="#ffffff">
2819         <form action="furl" method=GET>
2820             Name <input type=text name=name> <BR>
```

```
2821      Father name <input type=text name=Fname>
2822      <input type=submit value = continue>
2823    </form>
2824  </font>
2825 </body>
2826 </html>
2827 -----FirstServlet.java-----
2828 package com.nt;
2829 import java.io.*;
2830 import javax.servlet.*;
2831 import javax.servlet.http.*;
2832
2833 public class FirstServlet extends HttpServlet
2834 {
2835     public void doGet (HttpServletRequest request,HttpServletResponse response)
2836         throws ServletException, IOException
2837     {
2838         // set response content type
2839         response.setContentType("text/html");
2840         // get PrintWriter obj
2841         PrintWriter out = response.getWriter();
2842
2843         //get values submitted by the form1
2844         String s1 = request.getParameter("name");
2845         String s2 = request.getParameter("Fname");
2846
2847         // create cookies to store the form1 data
2848         Cookie ck1 = new Cookie("name",s1);
2849         Cookie ck2 = new Cookie("fname",s2);
2850
2851         response.addCookie(ck1);
2852         response.addCookie(ck2);
2853
2854 // now we need to generate the second form dynamically from here
2855         out.print("<form action='surl' method=get>");
2856         out.print("income for this year <input type=text name=income> <BR>");
2857         out.print(" Tax<input type=text name=tax>");
2858         out.print("<br><br> <BR><BR><input type=submit value = submit>");
2859         out.print("</form>");
2860         //close streams
2861         out.close();
2862     }
2863     public void doPost (HttpServletRequest request,HttpServletResponse response)
2864         throws ServletException, IOException
2865         doGet(request,response);
2866     }
2867 }
2868 -----SecondServlet.java-----
2869 import java.io.*;
2870 import javax.servlet.*;
2871 import javax.servlet.http.*;
2872 import java.sql.*;
2873
2874 public class SecondServlet extends HttpServlet
2875 {
2876     public void doGet (HttpServletRequest request,HttpServletResponse response)
2877         throws ServletException, IOException
2878     {
2879         // set content type
2880         response.setContentType("text/html");
```

```
2881 // get PrintWriter obj
2882 PrintWriter out = response.getWriter();
2883
2884 //get values submitted by the form2
2885 String income = request.getParameter("income");
2886 String tax = request.getParameter("tax");
2887
2888 //read form1/req1 data from cookie
2889 Cookie ck[] = request.getCookies();
2890 String name=null, fname=null;
2891 if(ck!=null) {
2892     name=ck[0].getValue();
2893     fname=ck[1].getValue();
2894 } //if
2895
2896 // use jdbc code to store form1/req1 and form2/req2 values in Db table
2897 try{
2898     //register jdbc driver and establish the connection
2899     Class.forName("oracle.jdbc.driver.OracleDriver");
2900     Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
2901     //create PreparedStatement obj pointing to SQL Query
2902     PreparedStatement ps=con.prepareStatement("insert into tax_tab values(?, ?, ?, ?)");
2903     //set values to Query Params
2904     ps.setString(1, name);
2905     ps.setString(2, fname);
2906     ps.setString(3, income);
2907     ps.setString(4, tax);
2908     //execute the Query
2909     ps.executeUpdate();
2910     out.println("<br>Record has been Inserted");
2911 } //try
2912 catch(Exception e)
2913 {
2914     e.printStackTrace();
2915 }
2916 //display both form1,form2 data
2917 out.print("Form2 data "+income+".... "+tax+"<br>");
2918 out.print("Form1 data "+name+".... "+fname+"<br>");
2919 //close streams
2920 out.close();
2921 } //doGet(-,-)
2922 public void doPost(HttpServletRequest request, HttpServletResponse response)
2923         throws ServletException, IOException
2924 {
2925     doGet(request, response);
2926 }
2927
2928
2929
2930 } //class
2931
2932 -----web.xml-----
2933 <web-app>
2934
2935 <servlet>
2936     <servlet-name>FF1</servlet-name>
2937     <servlet-class>com.nt.FirstServlet</servlet-class>
2938 </servlet>
2939
2940 <servlet-mapping>
```

```
2941 <servlet-name>FF1</servlet-name>
2942 <url-pattern>/furl</url-pattern>
2943 </servlet-mapping>
2944
2945 <servlet>
2946   <servlet-name>SF2</servlet-name>
2947   <servlet-class>com.nt.SecondServlet</servlet-class>
2948 </servlet>
2949 <servlet-mapping>
2950   <servlet-name>SF2</servlet-name>
2951   <url-pattern>/surl</url-pattern>
2952 </servlet-mapping>
2953 </web-app>
2954 -----
2955 =====
2956 App24(Session Tracking using HttpSession with Cookies)
2957 =====
2958 -----Create info table in oracle database-----
2959 CREATE TABLE INFO(NAME VARCHAR2(20),
2960   ADDR VARCHAR2(20),
2961   AGE VARCHAR2(3),
2962   EXP VARCHAR2(2),
2963   SKILLS VARCHAR2(20),
2964   CITY VARCHAR2(20),
2965   SALARY VARCHAR2(10));
2966 -----personal.html-----
2967 <HTML>
2968   <HEAD>
2969     <TITLE> HttpSession with cookies </TITLE>
2970   </HEAD>
2971
2972   <BODY bgcolor="lightblue" >
2973     <form action ="furl" method="Post" >
2974
2975     <h1><center><FONT COLOR="#FF0033">PERSONAL DETAILS</FONT></center><h1>
2976     <br><br>
2977
2978     <table align="center">
2979       <center>
2980         <tr>
2981           <td>Enter Name:</td>
2982           <td><input type="text" name="name" ></td>
2983         </tr>
2984         <tr>
2985           <td>Enter Address:</td>
2986           <td><input type="text" name="address" ></td>
2987         </tr>
2988         <tr>
2989           <td>Enter Age:</td>
2990           <td><input type="text" name="age" ></td>
2991         </tr>
2992         <tr>
2993           <td><input type="Submit" value="Continue"></td>
2994         </tr>
2995     </table>
2996   </BODY>
2997 </HTML>
2998 -----web.xml-----
2999 <web-app>
3000   <servlet>
```

```
3001 <servlet-name>f</servlet-name>
3002 <servlet-class>com.nt.FirstServlet</servlet-class>
3003 </servlet>
3004 <servlet-mapping>
3005 <servlet-name>f</servlet-name>
3006 <url-pattern>/furl</url-pattern>
3007 </servlet-mapping>
3008
3009 <servlet>
3010 <servlet-name>s</servlet-name>
3011 <servlet-class>com.nt.SecondServlet</servlet-class>
3012 </servlet>
3013 <servlet-mapping>
3014 <servlet-name>s</servlet-name>
3015 <url-pattern>/surl</url-pattern>
3016 </servlet-mapping>
3017 <servlet>
3018 <servlet-name>t</servlet-name>
3019 <servlet-class>com.nt.ThirdServlet</servlet-class>
3020 </servlet>
3021 <servlet-mapping>
3022 <servlet-name>t</servlet-name>
3023 <url-pattern>/turl</url-pattern>
3024 </servlet-mapping>
3025
3026 </web-app>
3027 -----FirstServlet.java-----
3028 package com.nt;
3029 import java.io.* ;
3030 import javax.servlet.*;
3031 import javax.servlet.http.*;
3032 import java.net.*;
3033 import java.util.*;
3034
3035 public class FirstServlet extends HttpServlet
3036 {
3037     public void service(HttpServletRequest req,
3038                         HttpServletResponse res) throws ServletException, IOException
3039     {
3040         //general settings
3041         res.setContentType( "text/html" );
3042         PrintWriter pw = res.getWriter();
3043         //read form1/request1 data
3044         String name=req.getParameter("name");
3045         String address=req.getParameter("address");
3046         String age=req.getParameter("age");
3047         //create Session for browser window
3048         HttpSession session = req.getSession(true);
3049         //keep form1/req1 data in session attributes
3050         session.setAttribute("name", name);
3051         session.setAttribute("Addr", address);
3052         session.setAttribute("age", age);
3053
3054         // generate form2 dynamically
3055         pw.println("<BODY BGCOLOR=cyan>");
3056         pw.println("<CENTER><H1><FONT COLOR=red>Provide Your Exp
3057             & Skils</FONT></H1></CENTER>");
3058         pw.println("<FORM ACTION='surl' METHOD=GET>");
3059         pw.println("<TABLE ALIGN= CENTER>");
3060         pw.println("<TR>");
```



```
3121     pw.println("</TD></TR>");  
3122  
3123     pw.println("<TR>");  
3124     pw.println("<TD>");  
3125     pw.println("<H2><FONT COLOR=BLUE>Enter Expected Salary :");  
3126     pw.println("<INPUT TYPE=TEXT NAME=sal SIZE=16>");  
3127     pw.println("</TD></TR>");  
3128  
3129     pw.println("<TR><TD>");  
3130     pw.println("<INPUT TYPE=SUBMIT VALUE=Submit >");  
3131     pw.println("</TD></TR>");  
3132     pw.println("</TABLE></FORM></BODY>");  
3133     //close stream  
3134     pw.close();  
3135 } // service  
3136 } // class  
3137 -----ThirdServlet.java-----  
3138 package com.nt;  
3139 import java.io.* ;  
3140 import javax.servlet.* ;  
3141 import javax.servlet.http.* ;  
3142 import java.net.* ;  
3143 import java.util.* ;  
3144 import java.sql.*;  
3145  
3146 public class ThirdServlet extends HttpServlet  
3147 {  
3148     public void doGet(HttpServletRequest req, HttpServletResponse res)  
3149             throws ServletException, IOException  
3150     {  
3151         //general settings  
3152         res.setContentType( "text/html" ) ;  
3153         PrintWriter pw = res.getWriter( ) ;  
3154         // read form3/request3 data  
3155         String city=req.getParameter("city");  
3156         String sal=req.getParameter("sal");  
3157         // Get access to Session obj  
3158         HttpSession session = req.getSession(false) ;  
3159         // read form1/req1 and form2/req2 data from session.attributes  
3160         String name = (String)session.getAttribute("name");  
3161         String addr = (String)session.getAttribute("Addr");  
3162         String age = (String)session.getAttribute("age");  
3163         String exp = (String)session.getAttribute("exp");  
3164         String skils = (String)session.getAttribute("skils");  
3165  
3166         // insert form1/req1 , form2/req2 and form3 and req3 values as record  
3167         try  
3168         {  
3169             Class.forName("oracle.jdbc.driver.OracleDriver");  
3170             Connection con=DriverManager.getConnection  
3171                     ("jdbc:oracle:thin:@localhost:1521:orcl", "scott", "tiger");  
3172             PreparedStatement pst=  
3173                     con.prepareStatement("INSERT INTO INFO VALUES(?,?,?,?,?,?)");  
3174  
3175             pst.setString(1,name);  
3176             pst.setString(2,addr);  
3177             pst.setString(3,age);  
3178             pst.setString(4,exp);  
3179             pst.setString(5,skils);  
3180         }
```

```
1

3181     pst.setString(6,city);
3182     pst.setString(7,sal);
3183
3184     int i = pst.executeUpdate();
3185
3186 //invalidate the session
3187     session.invalidate();
3188
3189     if(i > 0)
3190     {
3191         pw.println("<BODY BGCOLOR=cyan>");
3192         pw.println("<CENTER><H1><FONT COLOR=red>Successfully Inserted</FONT></H1></CENTER>"); 
3193         pw.println("<a href= personal.html>Home</a>"); 
3194         pw.println("</table></body>"); 
3195     }
3196     else
3197     {
3198         pw.println("<BODY BGCOLOR=cyan>"); 
3199         pw.println("<CENTER><H1><FONT COLOR=red>Try Again</FONT></H1></CENTER>"); 
3200         pw.println("<a href= personal.html>Home</a>"); 
3201     }
3202 } // try
3203 catch(Exception e)
3204 {
3205     e.printStackTrace();
3206     pw.println("<BODY BGCOLOR=cyan>"); 
3207     pw.println("<CENTER><H1><FONT COLOR=red>Try Again</FONT></H1></CENTER>"); 
3208     pw.println("<a href= personal.html>Home</a>"); 
3209 }
3210 } // doGet(-,-)
3211 public void doPost(HttpServletRequest req, HttpServletResponse res)
3212             throws ServletException, IOException
3213     doGet(req,res);
3214 }
3215
3216 } // class
3217 =====
3218 App25(Session Tracking using HttpSession with URLReWriting)
3219 =====
3220 -----Create info table in oracle database-----
3221 CREATE TABLE INFO(NAME VARCHAR2(20),
3222                     ADDR VARCHAR2(20),
3223                     AGE VARCHAR2(3),
3224                     EXP VARCHAR2(2),
3225                     SKILLS VARCHAR2(20),
3226                     CITY VARCHAR2(20),
3227                     SALARY VARCHAR2(10));
3228 -----personal.html-----
3229 <HTML>
3230     <HEAD>
3231         <TITLE> HttpSession with URL Rewriting Example </TITLE>
3232     </HEAD>
3233
3234     <BODY bgcolor="lightblue" >
3235         <form action ="furl" method="Post" >
3236
3237             <h1><center><FONT COLOR="#FF0033">PERSONAL DETAILS</FONT></center><h1>
3238             <br><br>
3239             <table align="center">
```

```
3241             <center>
3242             <tr>
3243                 <td>Enter Name:</td>
3244                 <td><input type="text" name="name" ></td>
3245             </tr>
3246             <tr>
3247                 <td>Enter Address:</td>
3248                 <td><input type="text" name="address" ></td>
3249             </tr>
3250             <tr>
3251                 <td>Enter Age:</td>
3252                 <td><input type="text" name="age" ></td>
3253             </tr>
3254             <tr>
3255                 <td><input type="Submit" value="Continue"></td>
3256             </tr>
3257         </table>
3258     </BODY>
3259 </HTML>
3260 -----web.xml-----
3261 <?xml version="1.0" encoding="UTF-8"?>
3262 <!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
3263           "http://java.sun.com/dtd/web-app_2_3.dtd">
3264
3265 <web-app>
3266   <servlet>
3267     <servlet-name>f</servlet-name>
3268     <servlet-class>com.nt.FirstServlet</servlet-class>
3269   </servlet>
3270   <servlet-mapping>
3271     <servlet-name>f</servlet-name>
3272     <url-pattern>/furl</url-pattern>
3273   </servlet-mapping>
3274
3275   <servlet>
3276     <servlet-name>s</servlet-name>
3277     <servlet-class>com.nt.SecondServlet</servlet-class>
3278   </servlet>
3279   <servlet-mapping>
3280     <servlet-name>s</servlet-name>
3281     <url-pattern>/surl</url-pattern>
3282   </servlet-mapping>
3283   <servlet>
3284     <servlet-name>t</servlet-name>
3285     <servlet-class>com.nt.ThirdServlet</servlet-class>
3286   </servlet>
3287   <servlet-mapping>
3288     <servlet-name>t</servlet-name>
3289     <url-pattern>/turl</url-pattern>
3290   </servlet-mapping>
3291
3292 </web-app>
3293 -----FirstServlet.java-----
3294 package com.nt;
3295 import java.io.*;
3296 import javax.servlet.*;
3297 import javax.servlet.http.*;
3298 import java.net.*;
3299 import java.util.*;
3300
```

```
3301 public class FirstServlet extends HttpServlet
3302 {
3303     public void doGet(HttpServletRequest req,
3304                         HttpServletResponse res) throws ServletException, IOException
3305     {
3306         res.setContentType( "text/html" );
3307         PrintWriter pw = res.getWriter( );
3308
3309         String name=req.getParameter("name");
3310         String address=req.getParameter("address");
3311         String age=req.getParameter("age");
3312
3313         HttpSession session = req.getSession(true);
3314         session.setAttribute("name", name);
3315         session.setAttribute("Addr", address);
3316         session.setAttribute("age", age);
3317
3318
3319         pw.println("<BODY BGCOLOR=cyan>");
3320         pw.println("<CENTER><H1><FONT COLOR=red>Provide Your Exp
3321             & Skils</FONT></H1></CENTER>");
```

3322 pw.println("<FORM ACTION="+res.encodeURL("surl")+" METHOD=GET>");

3323 pw.println("<TABLE ALIGN=CENTER>");

3324 pw.println("<TR>");

3325 pw.println("<TD>");

3326 pw.println("<H2><FONT COLOR=BLUE>Enter Number of Years Exp :");

3327 pw.println("<INPUT TYPE=TEXT NAME=exp SIZE=6>");

3328 pw.println("</TD></TR>");

3329
3330 pw.println("<TR>");

3331 pw.println("<TD>");

3332 pw.println(" <H2><FONT COLOR=blue><B>Select Skils:</B>");

3333 pw.print("&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;");

3334 pw.print("<SELECT NAME=skils>");

3335 pw.print("<OPTION VALUE=JAVA>JAVA/J2EE </OPTION>");

3336 pw.print("<OPTION VALUE=.NET>.Net </OPTION>");

3337 pw.print("<OPTION VALUE=ORACLE>ORACLE 10G </OPTION>");

3338 pw.print("<OPTION VALUE=XML>XML & Web Services </OPTION>");

3339 pw.print("</SELECT>");

3340 pw.println("</TD></TR>");

3341
3342 pw.println("<TR><TD>");

3343 pw.println("<INPUT TYPE=Submit value=Continue>");

3344 pw.println("</TD></TR></TABLE></BODY>");

3345 } // doGet(--)

3346 public void doPost(HttpServletRequest req,
3347 HttpServletResponse res) throws ServletException, IOException
3348 doGet(req,res);
3349 }
3350
3351 } // class
3352 -----SecondServlet.java-----
3353 package com.nt;
3354 import java.io.\*;
3355 import javax.servlet.\*;
3356 import javax.servlet.http.\*;
3357 import java.net.\*;
3358 import java.util.\*;
3359
3360 public class SecondServlet extends HttpServlet

```
3361 {
3362     public void doGet(HttpServletRequest req, HttpServletResponse res)
3363             throws ServletException, IOException
3364     {
3365         res.setContentType( "text/html" );
3366         PrintWriter pw = res.getWriter( );
3367
3368         String exp=req.getParameter("exp");
3369         String skils=req.getParameter("skils");
3370
3371         HttpSession session = req.getSession(false);
3372
3373         session.setAttribute("exp", exp );
3374         session.setAttribute("skils", skils );
3375
3376
3377         pw.println("<BODY BGCOLOR=cyan>");
3378         pw.println("<CENTER><H1><FONT COLOR=red>Provide City & Salary</FONT></H1></CENTER>");
3379
3380         pw.println("<FORM ACTION="+res.encodeURL("turl")+" METHOD=GET>");
3381         pw.println("<TABLE ALIGN= CENTER>");
3382         pw.println("<TR>");
3383         pw.println("<TD>");
3384         pw.println("<H2><FONT COLOR=BLUE>Enter Preference City :");
3385         pw.println("<INPUT TYPE=TEXT NAME=city SIZE=6>");
3386         pw.println("</TD></TR>");
3387
3388         pw.println("<TR>");
3389         pw.println("<TD>");
3390         pw.println("<H2><FONT COLOR=BLUE>Enter Expected Salary :");
3391         pw.println("<INPUT TYPE=TEXT NAME=sal SIZE=16>");
3392         pw.println("</TD></TR>");
3393
3394         pw.println("<TR><TD>");
3395         pw.println("<INPUT TYPE=SUBMIT VALUE=Submit >");
3396         pw.println("</TD></TR>");
3397         pw.println("</TABLE></FORM></BODY>");
3398 } // service
3399
3400     public void doPost(HttpServletRequest req, HttpServletResponse res)
3401             throws ServletException, IOException
3402     {
3403         doGet(req,res);
3404     }
3405 } // class
3406 -----ThirdServlet.java-----
3407 package com.nt;
3408 import java.io.*;
3409 import javax.servlet.*;
3410 import javax.servlet.http.*;
3411 import java.net.*;
3412 import java.util.*;
3413 import java.sql.*;
3414 public class ThirdServlet extends HttpServlet
3415 {
3416     public void doGet(HttpServletRequest req, HttpServletResponse res)
3417             throws ServletException, IOException
3418     {
3419         res.setContentType( "text/html" );
3420         PrintWriter pw = res.getWriter( );
```

```
3421
3422     String city=req.getParameter("city");
3423     String sal=req.getParameter("sal");
3424
3425     HttpSession session = req.getSession(false) ;
3426
3427     String name = (String)session.getAttribute("name");
3428     String addr = (String)session.getAttribute("Addr");
3429     String age = (String)session.getAttribute("age");
3430     String exp = (String)session.getAttribute("exp");
3431     String skils = (String)session.getAttribute("skils");
3432
3433     try
3434     {
3435         Class.forName("oracle.jdbc.driver.OracleDriver");
3436         Connection con=DriverManager.getConnection
3437             ("jdbc:oracle:thin:@localhost:1521:orcl", "scott", "tiger");
3438         PreparedStatement pst=
3439             con.prepareStatement("INSERT INTO INFO VALUES(?,?,?,?,?,?)");
3440
3441         pst.setString(1,name);
3442         pst.setString(2,addr);
3443         pst.setString(3,age);
3444         pst.setString(4,exp);
3445         pst.setString(5,skils);
3446         pst.setString(6,city);
3447         pst.setString(7,sal);
3448
3449         int i = pst.executeUpdate();
3450
3451         session.invalidate();
3452
3453     if(i > 0)
3454     {
3455         pw.println("<BODY BGCOLOR=cyan>");
3456         pw.println("<CENTER><H1><FONT COLOR=red>Successfully Inserted</FONT></H1></CENTER>");
3457         pw.println("<a href= personal.html>Home</a>");
3458         pw.println("</table></body>");
3459     }
3460     else
3461     {
3462         pw.println("<BODY BGCOLOR=cyan>");
3463         pw.println("<CENTER><H1><FONT COLOR=red>Try Again</FONT></H1></CENTER>");
3464         pw.println("<a href= personal.html>Home</a>");
3465     }
3466     } // try
3467     catch(Exception e)
3468     {
3469         e.printStackTrace();
3470         pw.println("<BODY BGCOLOR=cyan>");
3471         pw.println("<CENTER><H1><FONT COLOR=red>Try Again</FONT></H1></CENTER>");
3472         pw.println("<a href= personal.html>Home</a>");
3473     }
3474 } // doGet(-,-)
3475 public void doPost(HttpServletRequest req, HttpServletResponse res)
3476     throws ServletException, IOException
3477     doGet(req,res);
3478 }
3479 } // class
3480 =====
```

```
3481 App26( Basic App on Filters)
3482 =====
3483 -----MyFilter.java-----
3484 package com.nt;
3485 import javax.servlet.*;
3486 import java.io.IOException;
3487 import java.io.PrintWriter;
3488 public class MyFilter implements Filter
3489 {
3490     public MyFilter()
3491     {
3492         System.out.println("Inside constructor of Filter class");
3493     }
3494
3495     public void init (FilterConfig fcon) throws ServletException
3496     {
3497         System.out.println("Inside init() method of Filter class");
3498     }
3499
3500     public void doFilter(ServletRequest req,ServletResponse res,FilterChain fc)
3501             throws IOException,ServletException
3502     {
3503         System.out.println("inside dofilter() of Filter class");
3504         int x=Integer.parseInt(req.getParameter("first"));
3505         int y=Integer.parseInt(req.getParameter("second"));
3506
3507         if ((x<0 ||y<0))
3508         {
3509             PrintWriter pw=res.getWriter();
3510             pw.println("<html> <head>");
3511             pw.println("<head><title>From Filter !!!</title></head>");
3512             pw.println("<body> sorry !!! ur input should be only positive numbers ! ! . . ");
3513             pw.println("</body></html>");
3514         }//if
3515         else
3516         {
3517             fc.doFilter(req,res);
3518         }//else
3519     }//doFilter
3520
3521     public void destroy()
3522     {
3523         System.out.println("inside destroy() of Filter class");
3524     }
3525 } //class
3526 -----web.xml-----
3527 <web-app>
3528     <filter>
3529         <filter-name>CheckFilter</filter-name>
3530         <filter-class>com.nt.MyFilter</filter-class>
3531     </filter>
3532     <filter-mapping>
3533         <filter-name>CheckFilter</filter-name>
3534         <url-pattern>/ser/testurl</url-pattern>
3535     </filter-mapping>
3536
3537     <servlet>
3538         <servlet-name>myservlet</servlet-name>
3539         <servlet-class>MyServlet</servlet-class>
3540     </servlet>
```

```
3541     <servlet-mapping>
3542         <servlet-name>myser</servlet-name>
3543         <url-pattern>/ser/testurl</url-pattern>
3544     </servlet-mapping>
3545 </web-app>
3546 -----MyServlet.java-----
3547 import javax.servlet.*;
3548 import javax.servlet.http.*;
3549 import java.io.IOException;
3550 import java.io.PrintWriter;
3551
3552 public class MyServlet extends HttpServlet
3553 {
3554     public void doGet(HttpServletRequest req,HttpServletResponse res)
3555             throws ServletException, IOException
3556     {
3557         System.out.println("inside doGet() of Servlet class");
3558         int x=Integer.parseInt(req.getParameter("first"));
3559         int y=Integer.parseInt(req.getParameter("second"));
3560         int z=x+y;
3561         PrintWriter pw=res.getWriter();
3562         pw.println("<html>");
3563         pw.println("<head><title>From servlet !!!</title></head>");
3564         pw.println("<body> result is"+z+"</body></html>");
3565         pw.close();
3566     }//doGet
3567 } //class
3568 =====
3569 App27(App to apply multipe Filters on one Servlet prg)
3570 =====
3571 -----index.html-----
3572 <html>
3573
3574     <body>
3575         <center>
3576             <form name=f1 action="loginurl">
3577                 <b> Enter usern name </b> <input type=text name=uname size=10 > <br>
3578                 <b> Enter password </b> <input type=password name=pwd size=10 > <br>
3579                 </b> <input type=submit value=send size=10> <br>
3580             </form>
3581             This is the index page of the web application.
3582         </body>
3583     </html>
3584 -----web.xml-----
3585 <web-app>
3586     <filter>
3587         <filter-name>Auth</filter-name>
3588         <filter-class>com.nt.AuthFilter</filter-class>
3589     </filter>
3590     <filter-mapping>
3591         <filter-name>Auth</filter-name>
3592         <url-pattern>/loginurl</url-pattern>
3593     </filter-mapping>
3594     <filter>
3595         <filter-name>pTest</filter-name>
3596         <filter-class>com.nt.PerfTestFilter</filter-class>
3597     </filter>
3598     <filter-mapping>
3599         <filter-name>pTest</filter-name>
3600         <url-pattern>/loginurl</url-pattern>
```

```
3601    </filter-mapping>
3602    <servlet>
3603        <servlet-name>main</servlet-name>
3604        <servlet-class>com.nt.MainSrv</servlet-class>
3605    </servlet>
3606    <servlet-mapping>
3607        <servlet-name>main</servlet-name>
3608        <url-pattern>/loginurl</url-pattern>
3609    </servlet-mapping>
3610 </web-app>
3611 -----AuthFilter.java-----
3612 package com.nt;
3613 import java.sql.*;
3614 import javax.servlet.*;
3615 import javax.servlet.http.*;
3616 import java.io.IOException;
3617
3618 public class AuthFilter implements Filter
3619 {
3620     private FilterConfig f = null;
3621     private ServletContext sc = null;
3622
3623     public void init(FilterConfig filterConfig)
3624     {
3625         f = filterConfig;
3626         sc = f.getServletContext();
3627     }//init()
3628
3629     public void doFilter(ServletRequest request,
3630                         ServletResponse response,
3631                         FilterChain chain)
3632     throws IOException, ServletException
3633     {
3634         HttpServletRequest hreq = (HttpServletRequest) request;
3635         String login_page="/one.html";
3636         String un = hreq.getParameter("uname");
3637         String pwd = hreq.getParameter("pwd");
3638
3639         sc.log("in filter :" + hreq.getRequestURI());
3640         sc.log("in filter :" + hreq.getServletPath());
3641         sc.log("uname = " + un + "password = " + pwd);
3642
3643         try
3644         {
3645             DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
3646             Connection con=DriverManager.getConnection
3647                 ("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
3648             PreparedStatement ps=con.prepareStatement
3649                 ("SELECT * FROM USERLIST WHERE USERNAME=? AND PASSWORD=?");
3650             ps.setString(1,un);
3651             ps.setString(2,pwd);
3652             ResultSet rs=ps.executeQuery();
3653             if(rs.next())
3654             {
3655                 /* here is the code to check for uname/pwd */
3656                 sc.log("user name and passwords are given and they are correct");
3657                 chain.doFilter(request,response);
3658                 return;
3659             }//if
3660         } //else
```

```

3661         {
3662             sc.log("Trying to dispaly the same login page");
3663             RequestDispatcher rd=sc.getRequestDispatcher(login_page);
3664             rd.forward(request,response);
3665             sc.log("after forwarding request to login page");
3666             return;
3667         }//else
3668     }//try
3669     catch(SQLException se)
3670     {
3671         se.printStackTrace();
3672     }
3673     catch(Exception e)
3674     {
3675         e.printStackTrace();
3676     }
3677 }//doFilter()
3678
3679     public void destroy()
3680     {
3681         f=null;
3682     }//destroy()
3683 }//class
3684 -----one.html-----
3685 <html>
3686     <body>
3687         <center><b> u must enter valid uname && password</b> </b>
3688             <form name=f1 action="loginurl">
3689                 <b> Enter usern name </b> <input type=text name=uname size=10 > <br>
3690                 <b> Enter password </b> <input type=password name=pwd size=10 > <br>
3691                 </b> <input type=submit value=send size=10> <br>
3692             </form>
3693         </body>
3694     </html>
3695 -----PerfTestFilter.java-----
3696 //Filter to check the time taken to process a request. We can use such a filter
3697 //while optimizing the code to improve the speed.
3698 package com.nt;
3699 import java.io.*;
3700 import javax.servlet.*;
3701 import javax.servlet.http.*;
3702
3703 public final class PerfTestFilter implements Filter
3704 {
3705     FilterConfig fg=null;
3706     public void init(FilterConfig filterConfig) throws ServletException
3707     {
3708         fg= filterConfig;
3709     }//init()
3710
3711     public void doFilter(ServletRequest request,ServletResponse response,FilterChain chain)
3712     throws IOException, ServletException
3713     {
3714         // get request trapping time
3715         long beforeProcess = System.currentTimeMillis();
3716
3717         chain.doFilter(request, response);
3718         // get response trapping time
3719         long afterProcess = System.currentTimeMillis();
3720         // find out the difference

```

```
3721     long diff = afterProcess - beforeProcess;
3722     //get HttpServletRequest obj
3723     HttpServletRequest hreq = (HttpServletRequest) request;
3724     // write msg to log file
3725     fg.getServletContext().log("Total Time for processing "+ hreq.getRequestURI()+" : "+ diff);
3726 } //doFilter()
3727
3728     public void destroy()
3729     {
3730         fg=null;
3731     } //destroy()
3732
3733
3734 } //class
3735 -----MainSrv.java-----
3736 package com.nt;
3737 import javax.servlet.*;
3738 import javax.servlet.http.*;
3739 import java.io.*;
3740
3741 public class MainSrv extends HttpServlet
3742 {
3743     public void doGet(HttpServletRequest request, HttpServletResponse response)
3744     throws ServletException, java.io.IOException
3745     {
3746         PrintWriter out;
3747         response.setContentType("text/html");
3748         out = response.getWriter();
3749
3750         out.println("<HTML><HEAD><TITLE>");
3751         out.println("</TITLE>MainSrv</HEAD><BODY>");
3752         try
3753         {
3754             Thread.sleep(60000);
3755         }
3756         catch(Exception e)
3757         {
3758             e.printStackTrace();
3759         }
3760
3761         out.println(" <h1><center> Login Successfull <h1>");
3762         out.println(" </BODY> ");
3763
3764         out.close();
3765     } //doGet()
3766 } //class
3767 =====
3768 App28(File Uploading using java zoom api)
3769 =====
3770 -----upload1.html-----
3771 <form method="post" action="test1" enctype="multipart/form-data">
3772     Select File1: <input type="file" name="f1"><br><br>
3773     Select File2: <input type="file" name="f2"><br><br>
3774     <input type="Submit" value="Upload">
3775 </form>
3776 -----web.xml-----
3777 <web-app>
3778     <servlet>
3779         <servlet-name>upl</servlet-name>
3780         <servlet-class>com.nt.UplSrv1</servlet-class>
```

```
3781 </servlet>
3782 <servlet-mapping>
3783   <servlet-name>upl</servlet-name>
3784   <url-pattern>/test1</url-pattern>
3785 </servlet-mapping>
3786 </web-app>
3792 -----UpISrv1.java-----
3793 package com.nt;
3794 import java.io.*;
3795 import javax.servlet.*;
3796 import javax.servlet.http.*;
3797 import javazoom.upload.MultipartFormDataRequest;
3798 import java.util.Hashtable;
3799 import java.util.Enumeration;
3800 import javazoom.upload.UploadFile;
3801 import javazoom.upload.UploadBean;
3802
3803 public class UpISrv1 extends HttpServlet
3804 {
3805     public void doPost (HttpServletRequest req, HttpServletResponse res)
3806         throws ServletException, IOException
3807     {
3808         PrintWriter out = res.getWriter();
3809         try
3810         {
3811             UploadBean upb = new UploadBean();
3812             upb.setFolderstore("c:/store");
3813             upb.setOverwrite(false);
3814
3815             /* create a parser for parsing form data */
3816             MultipartFormDataRequest nreq = new MultipartFormDataRequest(req);
3817             upb.store(nreq); //completes file uploading
3818
3819             //Display the names of uploaded files
3820             out.println("<br>The names of Uploaded files are <br>");
3821
3822             Hashtable ht = nreq.getFiles();
3823             Enumeration e =ht.elements();
3824             while(e.hasMoreElements())
3825             {
3826                 UploadFile f1=(UploadFile)e.nextElement();
3827                 out.println(f1.getFileName() +"  
");
3828             } //while
3829         } //try
3830         catch(Exception e)
3831         {
3832             out.println(e);
3833         } //catch
3834         out.println("<h1> your files are saved/ Uploaded</h1></body></html>");
3835         out.close();
3836     } //doPost ()
3837     public void doGet (HttpServletRequest req, HttpServletResponse res)
3838         throws ServletException, IOException{
3839         doPost(req,res);
3840     }
```

```
3841
3842
3843 }//class
3844 =====
3845 >>>>>>>>>>>Examples on WebApplication Security>>>>>>>>>
3846 =====
3847 App29[On Programmatic Security]
3848 -----login.html-----
3849 <center>
3850   <form action="adminurl">
3851     Username : <input type=text name="tuname"> <br>
3852     Password : <input type=password name="tpwd"><br>
3853     <input type=submit value="login">
3854   </form>
3855 </center>
3856 -----web.xml-----
3857 <web-app>
3858   <servlet>
3859     <servlet-name>s1</servlet-name>
3860     <servlet-class>com.nt.AdminServlet</servlet-class>
3861   </servlet>
3862   <servlet-mapping>
3863     <servlet-name>s1</servlet-name>
3864     <url-pattern>/adminurl</url-pattern>
3865   </servlet-mapping>
3866
3867   <servlet>
3868     <servlet-name>s2 </servlet-name>
3869     <servlet-class>com.nt.LoginServlet</servlet-class>
3870   </servlet>
3871   <servlet-mapping>
3872     <servlet-name>s2</servlet-name>
3873     <url-pattern>/loginurl</url-pattern>
3874   </servlet-mapping>
3875 </web-app>
3876 -----LoginServlet.java-----
3877 package com.nt;
3878 import javax.servlet.*;
3879 import javax.servlet.http.*;
3880 import java.io.*;
3881 public class LoginServlet extends HttpServlet
3882 {
3883   public void doGet(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
3884   {
3885     HttpSession ses = req.getSession(true);
3886     String un = req.getParameter("tuname");
3887     String pw = req.getParameter("tpwd");
3888     if(un.equals("admin") && pw.equals("admin"))
3889     {
3890       ses.setAttribute("loggedin","ok");
3891       String up = (String)ses.getAttribute("target");
3892       RequestDispatcher rd = req.getRequestDispatcher(up);
3893       rd.forward(req,res);
3894     }
3895     else
3896     {
3897       RequestDispatcher rd = req.getRequestDispatcher("login.html");
3898       rd.forward(req,res);
3899     }
3900   }
}
```

```

3901 }
3902 -----AdminServlet.java-----
3903 package com.nt;
3904 import javax.servlet.*;
3905 import javax.servlet.http.*;
3906 import java.io.*;
3907 public class AdminServlet extends HttpServlet
3908 {
3909 public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
3910 {
3911     HttpSession ses=req.getSession(true);
3912     String str =(String) ses.getAttribute("loggedin");
3913     if(str==null)
3914     {
3915         ses.setAttribute("target","/adminurl");
3916         RequestDispatcher rd = req.getRequestDispatcher("/loginurl");
3917         rd.forward(req,res);
3918     }
3919     else
3920     {
3921         PrintWriter pw = res.getWriter();
3922         pw.println("<font color=red size=7> From AdminServlet prg </font>");
3923         pw.close();
3924     }
3925 }
3926 }
3927 =====
3928 Applications Declarative Web Security
3929 App30: (ON BASIC/DIGEST Model)
3930 -----FirstSrv.java-----
3931 package com.nt;
3932 import javax.servlet.*;
3933 import javax.servlet.http.*;
3934 import java.io.*;
3935 public class FirstSrv extends HttpServlet
3936 {
3937 public void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException
3938 {
3939     PrintWriter pw=resp.getWriter();
3940     pw.println("page successfully displayed");
3941     pw.println("<center><b>the user name is "+req.getRemoteUser()+"</center></b>");
3942     pw.println("<center><b>the auth type is "+req.getAuthType()+"</center></b>");
3943     pw.close();
3944 } // service()
3945 } // class
3946 -----web.xml-----
3947 <web-app>
3948     <servlet>
3949         <servlet-name> servlet1 </servlet-name>
3950         <servlet-class>com.nt.FirstSrv</servlet-class>
3951     </servlet>
3952
3953     <servlet-mapping>
3954         <servlet-name> servlet1 </servlet-name>
3955         <url-pattern>/srv1</url-pattern>
3956     </servlet-mapping>
3957
3958     <security-constraint>
3959         <web-resource-collection>
3960             <web-resource-name>Authentication</web-resource-name>

```

```
3961      <url-pattern>/srv1</url-pattern>
3962      <http-method>GET</http-method>
3963  </web-resource-collection>
3964
3965  <auth-constraint>
3966    <role-name>manager</role-name>
3967  </auth-constraint>
3968
3969  </security-constraint>
3970  <login-config>
3971    <auth-method>BASIC</auth-method>
3972    <realm-name>myrealm</realm-name>
3973  </login-config>
3974 </web-app>
3975 =====
3976 WebApplication on Declarative Security
3977 App31: (ON FORM Model)
3978 -----login.jsp-----
3979 <html>
3980   <head>
3981     <title>Security WebApp login page</title>
3982   </head>
3983   <body bgcolor="#cccccc">
3984     <blockquote>
3985       <h2>Please enter your user name and password:</h2>
3986       <p>
3987       <form method="POST" action="j_security_check">
3988         <table border=1>
3989           <tr>
3990             <td>Username:</td>
3991             <td><input type="text" name="j_username"></td>
3992           </tr>
3993           <tr>
3994             <td>Password:</td>
3995             <td><input type="password" name="j_password"></td>
3996           </tr>
3997           <tr>
3998             <td colspan=2 align=right><input type=submit
3999                           value="Submit"></td>
4000           </tr>
4001         </table>
4002       </form>
4003     </blockquote>
4004   </body>
4005 </html>
4006 -----login_fail.jsp-----
4007 <html>
4008   <head>
4009     <title>Login failed</title>
4010   </head>
4011   <body bgcolor="#ffffff">
4012     <blockquote>
4013       <h2>Sorry, your user name and password were not recognized.</h2>
4014       <p><b>
4015         <a href="login.jsp">Return to welcome page</a> </b>
4016       </blockquote>
4017     </body>
4018   </html>
4019 -----edit.jsp-----
4020 <html>
```

```
4021 <body>
4022 <%
4023 out.println("<center><b>the user name is "+request.getRemoteUser()+"</center></b>");
4024 %>
4025 <center><b>login successfull... welcome to home page</center></b>
4026 </body>
4027 -----web.xml-----
4028 <!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
4029 "http://java.sun.com/dtd/web-app_2_3.dtd">
4030 <web-app>
4031
4032 <servlet>
4033   <servlet-name> servlet2 </servlet-name>
4034   <jsp-file>/edit.jsp</jsp-file>
4035 </servlet>
4036
4037 <servlet-mapping>
4038   <servlet-name> servlet2 </servlet-name>
4039   <url-pattern>/test2 </url-pattern>
4040 </servlet-mapping>
4041
4042 <security-constraint>
4043   <web-resource-collection>
4044     <web-resource-name> Authentication1 </web-resource-name>
4045     <url-pattern>/test2 </url-pattern>
4046     <http-method>GET </http-method>
4047   </web-resource-collection>
4048   <auth-constraint>
4049     <role-name> manager </role-name>
4050   </auth-constraint>
4051 </security-constraint>
4052
4053 <login-config>
4054   <auth-method> FORM </auth-method>
4055     <form-login-config>
4056       <form-login-page>/login.jsp </form-login-page>
4057       <form-error-page>/login_fail.jsp </form-error-page>
4058     </form-login-config>
4059   <realm-name> myrealm </realm-name>
4060 </login-config>
4061 </web-app>
4062
4063
```

# Understanding Http

## **javax.servlet.GenericServlet**

Signature: public abstract class GenericServlet extends java.lang.Object implements Servlet, ServletConfig, java.io.Serializable

- 9 GenericServlet defines a generic, protocol-independent servlet.
- 9 GenericServlet gives a blueprint and makes writing servlet easier.
- 9 GenericServlet provides simple versions of the lifecycle methods init and destroy and of the methods in the ServletConfig interface.
- 9 GenericServlet implements the log method, declared in the ServletContext interface.
- 9 To write a generic servlet, it is sufficient to override the abstract service method.

## **javax.servlet.http.HttpServlet**

Signature: public abstract class HttpServlet extends GenericServlet implements java.io.Serializable

- 9 HttpServlet defines a HTTP protocol specific servlet.
- 9 HttpServlet gives a blueprint for Http servlet and makes writing them easier.
- HttpServlet extends the GenericServlet and hence inherits the properties GenericServlet

Provides an abstract class to be subclassed to create an HTTP servlet suitable for a Web site. A subclass of HttpServlet must override at least one method, usually one of these:

- 9 doGet, if the servlet supports HTTP GET requests
- 9 doPost, for HTTP POST requests
- 9 doPut, for HTTP PUT requests
- 9 doDelete, for HTTP DELETE requests
- 9 init and destroy, to manage resources that are held for the life of the servlet
- 9 getServletInfo, which the servlet uses to provide information about itself

There's almost no reason to override the service method. service handles standard HTTP requests by dispatching them to the handler methods for each HTTP request type (the doXXX methods listed above). Likewise, there's almost no reason to override the doOptions and doTrace methods.

Servlets typically run on multithreaded servers, so be aware that a servlet must handle concurrent requests and be careful to synchronize access to shared resources. Shared resources include in-memory data such as instance or class variables and external objects such as files, database connections, and network connections.

### **Constructor:**

HttpServlet() : Does nothing, because this is an abstract class.

Method Summary	
protected void <a href="#">doDelete(HttpServletRequest req, HttpServletResponse resp)</a>	Called by the server (via the service method) to allow a servlet to handle a DELETE request.
protected void <a href="#">doGet(HttpServletRequest req, HttpServletResponse resp)</a>	Called by the server (via the service method) to allow a servlet to handle a GET request.
protected void <a href="#">doHead(HttpServletRequest req, HttpServletResponse resp)</a>	Receives an HTTP HEAD request from the protected service method and handles the request.
protected void <a href="#">doOptions(HttpServletRequest req, HttpServletResponse resp)</a>	Called by the server (via the service method) to allow a servlet to handle a OPTIONS request.
protected void <a href="#">doPost(HttpServletRequest req, HttpServletResponse resp)</a>	Called by the server (via the service method) to allow a servlet to handle a POST request.
protected void <a href="#">doPut(HttpServletRequest req, HttpServletResponse resp)</a>	Called by the server (via the service method) to allow a servlet to handle a PUT request.
protected void <a href="#">doTrace(HttpServletRequest req, HttpServletResponse resp)</a>	Called by the server (via the service method) to allow a servlet to handle a TRACE request.

<b>protected long</b>	<b>getLastModified(HttpServletRequest req)</b> Returns the time the HttpServletRequest object was last modified, in milliseconds since midnight January 1, 1970 GMT.
<b>protected void</b>	<b>service(HttpServletRequest req, HttpServletResponse resp)</b> Receives standard HTTP requests from the public service method and dispatches them to the doXXX methods defined in this class.
<b>void</b>	<b>service(ServletRequest req, ServletResponse res)</b> Dispatches client requests to the protected service method.

### Skeleton of a HTTP Servlet

```

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class ServletSkeleton extends HttpServlet {
    public void init() throws ServletException { /* implementation */ } // (1)

    public void doPost(HttpServletRequest req,
                       HttpServletResponse resp)
        throws ServletException, IOException
    { /* implementation */ }

    public void doGet(HttpServletRequest req,
                      HttpServletResponse resp)
        throws ServletException, IOException
    { /* implementation */ }

    public void destroy() { /* implementation */ } // (4)

    public String getServletInfo() { /* implementation */ } // (5)
}

```

### HTTP headers

HTTP headers allow the client and the server to pass additional information with the request or the response. A request header consists of its case-insensitive name followed by a colon ':', then by its value (without CRLF in it). Leading white space before the value is ignored. Headers are grouped according the context in which they may appear:

#### General headers

These headers apply to both requests and responses but are unrelated to the data eventually transmitted in the body. They therefore apply only to the message being transmitted. There are only a few of them and new ones cannot be added without increasing the version number of the HTTP protocol. The exhaustive list for HTTP/1.1 is Cache-Control:, Connection:, Date:, Pragma:, Trailer:, Transfer -Encoding:, Upgrade:, Via: and Warning:.

#### Request headers

These headers give more precise information about the resource to be fetched or about the client itself. Among them one can find cache-related headers, transforming a GET method in a conditional GET, like If-Modified-Since:, user-preference information like Accept-Language: or Accept-Charset: or plain client information like User-Agent:. New request headers cannot officially be added without increasing the version number of the HTTP protocol. But, it is common for new request headers to be added if both the server and the client agree on their meaning. In that case, a client should not assume that they will be handled adequately by the server; unknown request headers are handled as entity headers.

**Response headers**

These headers give more information about the resource sent back, like its real location (Location:) or about the server itself, like its name and version (Server:). New response headers cannot be added without increasing the version number of the HTTP protocol. But, it is common for new response headers to be added if both the server and the client agree on their meaning. In that case, a server should not assume that they will be handled adequately by the client ; unknown response headers are handled as entity headers.

**Entity headers**

These headers give more information about the body of the entity, like its length (Content-Length:), an identifying hash (Content-MD5:), or its MIME-type (Content-Type:). New entity headers can be added without increasing the version number of the HTTP protocol.

Headers can also be grouped according to how caching and non-caching proxies handle them:

**End-to-end headers**

These headers must be transmitted to the final recipient of the message; that is, the server for a request message or the client for a response message. Such a header means that intermediate proxies must retransmit it unmodified and also that caches must store it.

**Hop-by-hop headers**

These headers are meaningful only for a single transport-level connection and must not be retransmitted by proxies or cached. Such headers are: Connection:, Keep-Alive:, Proxy-Authenticate:, Proxy-Authorization:, TE:, Trailers:, Transfer-Encoding: and Upgrade:. Note that only hop-by-hop headers may be set using the Connection: general header.

In order to learn about the specific semantic of each header, see its entry in the comprehensive list of HTTP headers.

**Useful request headers**

Among the numerous HTTP request headers, several are especially useful when set correctly. If you are building your own requests, by using XMLHttpRequest or when writing an extension and sending custom HTTP requests via XPCOM, then it is important to ensure the presence of headers that are often set by browsers based on the preferences of the user. Controlling the language of the resource. Most user-agents, like Firefox, allow the user to set a preference for the language for receiving a resource. The browser translate this into an Accept-Language: header. It is good practice for web developers, when building specific HTTP requests, to include such a header too.

Using conditional GET

Caching is a major tool to accelerate the display of web pages. Even when parts of a webpage are refreshed via an XMLHttpRequest:, it is a good idea to use the If-Modified-Since: header (and other similar ones) in order to fetch the new content only if it has changed. This approach lowers the burden on the network.

**Useful response headers**

The configuration of a web server is a critical part to ensure good performance and optimal security of a web site. Among the numerous HTTP response headers, several are of specific importance and should be configured on the server

**Restricting framing**

Several cross-site scripting (XSS) attacks take advantage of the ability to put third-party content inside an <frame> or <iframe>. In order to mitigate that risk, modern browsers have introduced the X-Frame-Options: response header. By setting it with the value DENY, it prevents the browser from displaying this resource inside of a frame. Using it on critical resources (like those containing a formularies or critical information) will reduce the risk caused by XSS attacks. Note that this specific HTTP response header is not the only way to mitigate XSS risks; other techniques, like setting some Content Security Policies, may be helpful too.

**Compression**

Minimizing the amount of data transferred accelerates the display of a web page. Though most techniques, like CSS Sprites, should be applied on the site itself, compression of data must be set at the web server level. If set, resources requested by the client with an Accept-Encoding: request header are

compressed using the appropriate method and sent back with a Content-Encoding: response header. Setting these in Apache 2 servers is done by using the mod\_deflate module.

Note: Be aware that not all data formats can be efficiently compressed. Already-compressed media data, like JPEG images or most audio and video formats, do not shrink using another pass of compression. In fact, they often become larger due to the overhead of the compression method. It is important not to try to compress these resource types any further; there is no advantage in size and the compression/decompression mechanism is resource-intensive.

### Controlling cache

HTTP Caching is a technique that prevents the same resource from being fetched several times if it hasn't changed. Configuring the server with the correct response headers allows the user-agent to adequately cache the data. In order to do that, be sure that:

Any static resource provides an Expires: response header that is set to far in the future. That way, the resource may stay in the cache until the user-agent flushes it for its own reasons (like reaching its cache size limit).

**Note:** On Apache, use the ExpiresDefault directive in your .htaccess to define a relative expires: ExpiresDefault "access plus 1 month".

Any dynamic resource provides a Cache-control: response header. Theoretically, any HTTP request done through a safe method (GET or HEAD) or even through a solely idempotent one (DELETE, PUT) may be cached; but in practice careful study is needed to determine if the caching of the response may lead to inappropriate side-effects.

### Setting the correct MIME types

The MIME type is the mechanism to tell the client the kind of document transmitted: the extension of a file name has no meaning on the web. It is therefore important that the server is correctly set up so that the correct MIME type is transmitted with each document: user-agents often use this MIME-type to determine what default action to do when a resource is fetched.

### Examples of Request Message

Now let's put it all together to form an HTTP request to fetch **hello.htm** page from the web server running on [tutorialspoint.com](http://tutorialspoint.com)

```
GET /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

Here we are not sending any request data to the server because we are fetching a plain HTML page from the server. Connection is a general-header, and the rest of the headers are request headers. The following example shows how to send form data to the server using request message body:

```
POST /cgi-bin/process.cgi HTTP/1.1
```

```
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
```

```
Host: www.tutorialspoint.com
```

```
Content-Type: application/x-www-form-urlencoded
```

```
Content-Length: length
```

```
Accept-Language: en-us
```

```
Accept-Encoding: gzip, deflate
```

```
Connection: Keep-Alive
```

```
licenseID=string&content=string&/paramsXML=string
```

Here the given URL `/cgi-bin/process.cgi` will be used to process the passed data and accordingly, a response will be returned. Here **content-type** tells the server that the passed data is a simple web form data and **length** will be the actual length of the data put in the message body. The following example shows how you can pass plain XML to your web server:

```
POST /cgi-bin/process.cgi HTTP/1.1
```

```
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
```

```
Host: www.tutorialspoint.com
```

```
Content-Type: text/xml; charset=utf-8
```

```
Content-Length: length
```

```
Accept-Language: en-us
```

```
Accept-Encoding: gzip, deflate
```

```
Connection: Keep-Alive
```

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<string xmlns="http://clearforest.com/">string</string>
```

#### Examples of Response Message

Now let's put it all together to form an HTTP response for a request to fetch the `hello.htm` page from the web server running on `tutorialspoint.com`

```
HTTP/1.1 200 OK
```

```
Date: Mon, 27 Jul 2009 12:28:53 GMT
```

```
Server: Apache/2.2.14 (Win32)
```

```
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
```

```
Content-Length: 88
```

```
Content-Type: text/html
```

```
Connection: Closed
```

```
<html>
```

```
<body>
```

```
<h1>Hello, World!</h1>
```

```
</body>
```

```
</html>
```

The following example shows an HTTP response message displaying error condition when the web server could not find the requested page:

```
HTTP/1.1 404 Not Found
```

```
Date: Sun, 18 Oct 2012 10:36:20 GMT
```

```
Server: Apache/2.2.14 (Win32)
```

```
Content-Length: 230
```

```
Connection: Closed
```

```
Content-Type: text/html; charset=iso-8859-1
```

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
```

```
<html>
```

```
<head>
```

```
<title>404 Not Found</title>
```

```
</head>
```

```
<body>
```

```
<h1>Not Found</h1>
```

```
<p>The requested URL /t.html was not found on this server.</p>
```

```
</body>
```

```
</html>
```

Following is an example of HTTP response message showing error condition when the web server encountered a wrong HTTP version in the given HTTP request:

HTTP/1.1 400 Bad Request

Date: Sun, 18 Oct 2012 10:36:20 GMT

Server: Apache/2.2.14 (Win32)

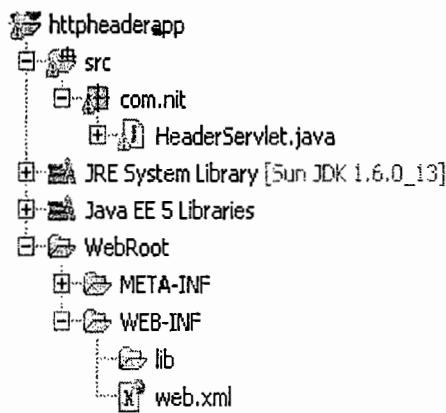
Content-Length: 230

Content-Type: text/html; charset=iso-8859-1

Connection: Closed

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html>
<head>
<title>400 Bad Request</title>
</head>
<body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not understand.</p>
<p>The request line contained invalid characters following the protocol string.</p>
</body>
</html>
```

#### Write a servlet program to display the HTTP request header information.



In this program we are going to make a servlet which will retrieve all the Http request header. To make a program over this firstly we need to make one class named HeaderServlet. In **HttpServletRequest** there are too many headers. To retrieve all the headers firstly we need to call the **getWriter()** which returns **PrintWriter** object and helps us to display all the headers. To get a header names call the method **getHeaderNames()** of the request object which will return the Enumeration of the headers. Now to retrieve all the headers from the Enumeration use the method **hasMoreElements()**. This method checks whether there are more headers or not. To display the output on your browser use the **PrintWriter** object.

## HeaderServlet.java

```
package com.nit;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HeaderServlet extends HttpServlet{
protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
response.setContentType("text/html");
PrintWriter pw = response.getWriter();
pw.println("<h1><font color=orange>Request Headers are</font></h1>");
Enumeration enumeration = request.getHeaderNames();
while(enumeration.hasMoreElements()){
String headerName = (String)enumeration.nextElement();
Enumeration headerValues = request.getHeaders(headerName);
if (headerValues != null){
while (headerValues.hasMoreElements()){
String values = (String) headerValues.nextElement();
pw.println("<h1>" +headerName + "</h1>" + ":" + values);
}
}
}
}
```

## web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <servlet>
    <servlet-name>nit</servlet-name>
    <servlet-class>com.nit.HeaderServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>nit</servlet-name>
    <url-pattern>/header</url-pattern>
  </servlet-mapping>
</web-app>
```

**Request Headers are**

host

: mit-11:8099

## **user-agent**

: Mozilla/5.0 (W

### **accept**

:text/html,application/xhtml+xml,application/xml,application/xslt+xml

### accept-*b*

: en-US,en;q=0.5

accept-e

: gap, delicate

comme

**Examples of Http response messages****Http status codes****Http status codes**

<b>1xx</b>	<b>Informational</b>
<u>100</u>	Client should continue with request
<u>101</u>	Server is switching protocols
<u>102</u>	Server has received and is processing the request
<u>103</u>	Resume aborted PUT or POST requests
<u>122</u>	URI is longer than a maximum of 2083 characters
<b>2xx</b>	<b>Success</b>
<u>200</u>	standard response for successful HTTP requests
<u>201</u>	request has been fulfilled, new resource created
<u>202</u>	request accepted, processing pending
<u>203</u>	request processed, information may be from another source
<u>204</u>	request processed, no content returned
<u>205</u>	request processed, no content returned, reset document view
<u>206</u>	partial resource return due to request header
<u>207</u>	XML, can contain multiple separate responses
<u>208</u>	results previously returned
<u>226</u>	request fulfilled, response is instance-manipulations
<b>3xx</b>	<b>Redirection</b>
<u>300</u>	multiple options for the resource delivered
<u>301</u>	this and all future requests directed to the given URI
<u>302</u>	temporary response to request found via alternative URI
<u>303</u>	permanent response to request found via alternative URI
<u>305</u>	resource has not been modified since last requested
<u>306</u>	content located elsewhere, retrieve from there
<u>307</u>	subsequent requests should use the specified proxy
<u>308</u>	connect again to different uri as provided
<b>4xx</b>	resumable HTTP Requests
<u>400</u>	<b>Client Error</b>
<u>401</u>	request cannot be fulfilled due to bad syntax
<u>402</u>	authentication is possible but has failed

<u>403</u>	payment required, reserved for future use
<u>404</u>	server refuses to respond to request
<u>405</u>	requested resource could not be found
<u>406</u>	request method not supported by that resource
<u>407</u>	content not acceptable according to the Accept headers
<u>408</u>	client must first authenticate itself with the proxy
<u>409</u>	server timed out waiting for the request
<u>410</u>	request could not be processed because of conflict
<u>411</u>	resource is no longer available and will not be available again
<u>412</u>	request did not specify the length of its content
<u>413</u>	server does not meet request preconditions
<u>414</u>	request is larger than the server is willing or able to process
<u>415</u>	URI provided was too long for the server to process
<u>416</u>	server does not support media type
<u>417</u>	client has asked for unprovidable portion of the file server cannot meet requirements of Expect request-header field
<u>420</u>	I'm a teapot Twitter rate limiting
<u>422</u>	request unable to be followed due to semantic errors
<u>423</u>	resource that is being accessed is locked
<u>424</u>	request failed due to failure of a previous request
<u>426</u>	client should switch to a different protocol
<u>428</u>	origin server requires the request to be conditional
<u>429</u>	user has sent too many requests in a given amount of time
<u>431</u>	server is unwilling to process the request
<u>444</u>	server returns no information and closes the connection
<u>449</u>	request should be retried after performing action
<u>450</u>	Windows Parental Controls blocking access to webpage
<u>451</u>	The server cannot reach the client's mailbox
<u>499</u>	connection closed by client while HTTP server is processing
<b>5xx</b>	<b>Server Error</b>
<b>500</b>	generic error message
<b>501</b>	

<u>502</u>	server does not recognise method or lacks ability to fulfill
<u>503</u>	server received an invalid response from upstream server
<u>504</u>	server is currently unavailable
<u>504</u>	gateway did not receive response from upstream server
<u>505</u>	server does not support the HTTP protocol version
<u>506</u>	content negotiation for the request results in a circular reference
<u>507</u>	server is unable to store the representation
<u>509</u>	server detected an infinite loop while processing the request
<u>510</u>	bandwidth limit exceeded
<u>511</u>	further extensions to the request are required
<u>598</u>	client needs to authenticate to gain network access
<u>599</u>	network read timeout behind the proxy
	network connect timeout behind the proxy

## SESSION TRACKING

### Understanding dynamic form pages:

.html files based form pages are called "**static form pages**". The form page that comes as response of dynamic web resource programs like servlet/JSP having dynamic components and content is called as "**dynamic form page**".

To generate dynamic form page from servlet program place <form>, <input> tags in pw.println(). To read huge amount of data from end user, it is not recommended to take single static form page, where end user will be forced to read and ignore some unrelated questions.

Details.html

Name :	Raaj
Age :	23
Father's Name:	Ramesh
Marital Status:	married
Spouse Name:	
No. of Children:	
When to Marry:	After 2 Years
Why to Marry :	To Get Personal Friend

**Submit**

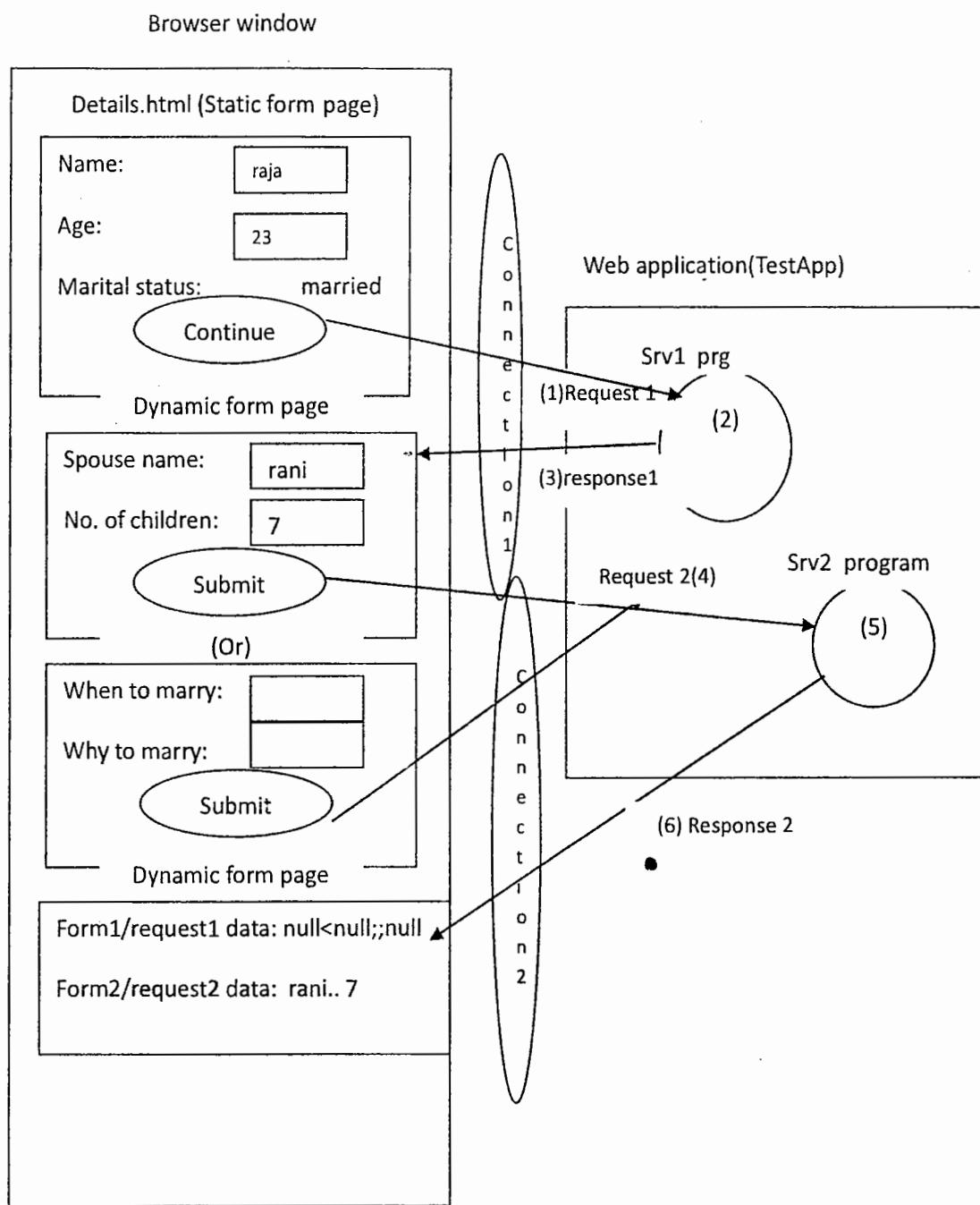
In the above form page, end user is forced to read and ignore marriage life related questions even though he has not selected that marital status checkbox.

To overcome this problem ask common question in first form page which is static form page and generate second form page dynamically based on the content given in first form page, then end user will not be forced to read and ignore certain questions of form pages.

Set of continues and related operations performed on web application by a client (browser window) with the support of multiple requests and responses is called as "**session**".

Eg: login to logout in gmail.com

Start of advanced java class to end of advanced java on a particular day.



In the above diagram Srv1 is reading content of form1 form page (Details.html) and uses the value given by maritalstatus checkbox to generate form2 as dynamic form page with dynamic content.

**Example App**


---

```
=====
App20 (App to demonstrate Stateless Behaviour of WebApplication)
=====
```

-----*Details.html*-----

```
<form action="s1url" method="get">

<b> Name: </b><input type="text" name="pname"><br>
<b> Age :</b><input type="text" name="page"><br>
<b> MaritalStatus: </b><input type="checkbox" value="married" name="ms"> Married

<input type="submit" value="continue">
</form>
```

---

*web.xml*-----

```
<web-app>
  <servlet>
    <servlet-name>ABC</servlet-name>
    <servlet-class>com.nt.Srv1</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>ABC</servlet-name>
    <url-pattern>/s1url</url-pattern>
  </servlet-mapping>

  <servlet>
    <servlet-name>XYZ</servlet-name>
    <servlet-class>com.nt.Srv2</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>XYZ</servlet-name>
    <url-pattern>/s2url</url-pattern>
  </servlet-mapping>

  <welcome-file-list>
    <welcome-file>Details.html</welcome-file>
  </welcome-file-list>
```

*</web-app>**Srv1.java*-----*/Srv1.java (Generates Second Form Dynamically Based on first Form Data)*

```
package com.nt;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class Srv1 extends HttpServlet
{
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws
    ServletException, IOException
    {
```

```

// read form1 data
String name=req.getParameter("pname");
String age=req.getParameter("page");
String mstatus=req.getParameter("ms");

if(mstatus==null)
    mstatus="single";

// Get PrintWriter obj
PrintWriter pw=res.getWriter();
res.setContentType("text/html");

// Generate Second Form Dynamically Form here
if(mstatus.equals("married"))
{
    pw.println("<form action='s2url' >");
    pw.println("<b> Spouse Name </b><input type='text' name='st1'><br>");
    pw.println("<b> No.of Children </b><input type='text' name='st2'><br>");
    pw.println("<input type='submit' value='sumbit'>");
    pw.println("</form>");
}
else
{
    pw.println("<form action='s2url' >");
    pw.println("<b> when do u want marry </b><input type='text' name='st1'><br>");
    pw.println("<b> why do u want marry </b><input type='text' name='st2'><br>");
    pw.println("<input type='submit' value='sumbit'>");
    pw.println("</form>");
}

pw.close();
}//doGet(--)

public void doPost(HttpServletRequest req,HttpServletResponse res) throws
ServletException,IOException
{
    doGet(req,res);
}

}//class
-----Srv2.java-----
//Srv2.java (takes the request from Second form (Dynamic Form))
package com.nat;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class Srv2 extends HttpServlet
{

    public void doGet(HttpServletRequest req,HttpServletResponse res) throws
    ServletException,IOException
    {
        // get PrintWriter object

```

```
PrintWriter pw=res.getWriter();
res.setContentType("text/html");

// read 1st form , second form data
pw.println("<br>First Form data is ");
pw.println("<br>Name is "+req.getParameter("pname"));
pw.println("<br>Age is "+req.getParameter("page"));
pw.println("<br>Marital Status is "+req.getParameter("ms"));

pw.println("<br><br>Second form data is "+req.getParameter("st1")+"<br>
"+req.getParameter("st2"));

pw.close();
}//doGet(-,-)

public void doPost(HttpServletRequest req,HttpServletResponse res) throws
ServletException,IOException
{
    doGet(req,res);
}

}//class
```

During a session if web application is not capable of remembering client data across the multiple requests then it is called as "**stateless behavior**" of web application. That means while processing current request in web application, we cannot use previous request related information.

An example scenario of stateless behavior of web application is nothing but the above web application where Srv2 is getting second request from client but failing to read and use request1 data, that means the web application failing to use previous request data while processing current request.



By default all web applications are stateless web applications.

During a session if web application is able to remember client data across the multiple requests then it is called as "**statefull behavior**" of web application. That means, while processing current request in web application, we can use previous request related information.

To make web applications as stateful web applications we need to work with "session tracking /session management techniques".



Why web applications are stateless web applications by default?

Web applications are stateless by default because the protocol http is given as stateless protocol. According to this stateless protocol for every request given by the web application from browser window, one new connection will be created between browser window and web server and that connection will be closed automatically once the request related response goes to browser window from web server. Due to this one connection related request data is not visible and it receives a file processing another connection related request given by client. The above whole discussion shows web server always treats browser window as new client for each request even though same browser window is giving multiple requests to same server.

#### **Protocol HTTP is designed as stateless and it is not given as stateful protocol:**

**The reason is:**

If protocol http is given as stateful protocol, browser window uses single connection to communicate with web server for multiple requests given to web application. This gives chance to browser window/ end user to engage the connections between browser window and web server for long time and to keep them in idle state for long time. This may create the situation of reaching to maximum connections of web server even though most of the connections are idle.

To overcome this problem the protocol http is designed as stateless protocol. Due to this browser window creates new connection with web server for every request and closes that connection once the request related response comes back to browser window. Due to this there is no chance/possibility for client (browser window) engaging or keeping connection with server in idle state for longtime. So web server never reaches to maximum connections and never maintains idle connections. For this benefit the protocol http is designed as stateless protocol.

The stateless behavior of protocol http makes web applications as stateless web applications that means no web resource program can use previous request data while processing current request during a session.

#### **Session tracking/Management techniques:**

To make a web application as statefull web application even though protocol http is stateless we need to work with session tracking techniques. They are

1. Hidden form fields
  2. Http cookies
  3. Http session with cookies
  4. Http session with url rewriting
- # session tracking makes java web application to keep track of (to remember) client data across the multiple request during a session.
  - # Statefull web application is nothing but a web application whose web resource programs can use previous request data while processing current request of a session.

**Technique1:****2.34.0 Hidden form fields (hidden boxes)**

Hidden Box is an invisible text box of form page, hidden box value goes to server as request parameter when form is submitted.

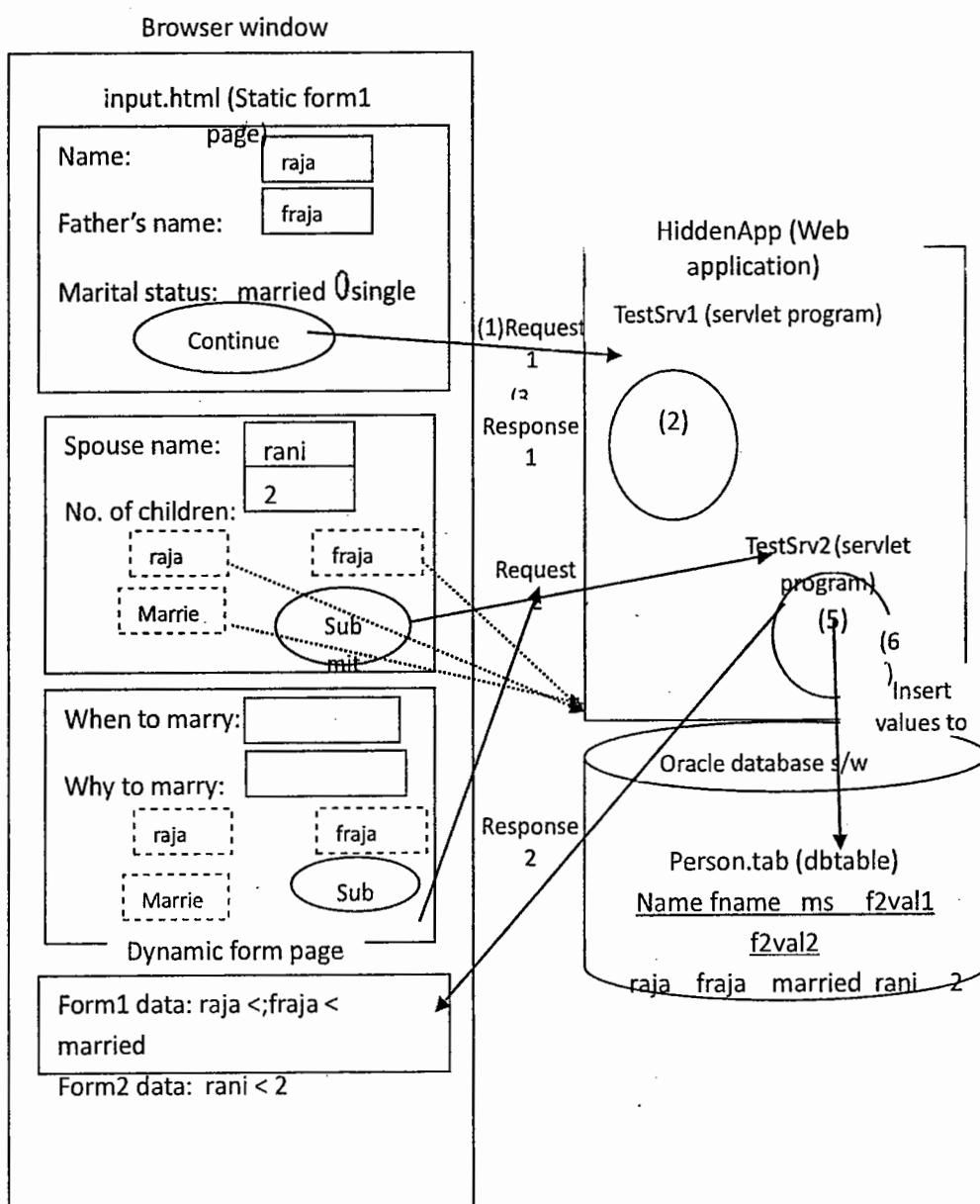
**In form page:**

```
<input type="hidden" name="h1" value="hello">
    Hidden box name → hidden box value/
    Or                               request parameter value
    Request parameter Name
```

**In servlet program:**

```
String val1=req.getParameter("h1"); //gives "hello"
```

Session management or session tracking both are same.



In the above diagram TestSrv1 servlet program reading form1/ request1 data and adding that data to the dynamically generated form page/Form2 as hidden box values(look at step 2 and step3).Due to this when form2 sends its request2 to TestSrv2 program the TestSrv2 program can read form1/request1, form2/request2 data. This is nothing but accessing previous request data (request1) while processing current request (request2). Technically this is called "session tracking"(look at step 4 & step 5).

#### **Advantages and disadvantages of hidden form fields based session tracking technique:**

##### **Advantages:**

1. Basic html knowledge is sufficient to work with this technique.
2. Hidden boxes resides in the form page holding the client data across the multiple requests that means they do not allocate memory on server and don't give burden to server.
3. This technique works with all server side technologies like servlets, JSP, ASP.net, PHP, etc.,
4. This technique works with both java and non-java servers.

##### **Disadvantages:**

1. The hidden box values of form page can be viewed using source code of web page. That means there is no security (data secrecy is not there).
2. Hidden boxes travel over the network along with the request and response. This indicates more network traffic.
3. We cannot store all kinds of java objects in hidden boxes except text/string values.
4. If more hidden boxes should be added in each dynamic form page to preserve client data across the multiple requests.

#### **Real-world application examples of SessionTracking:**

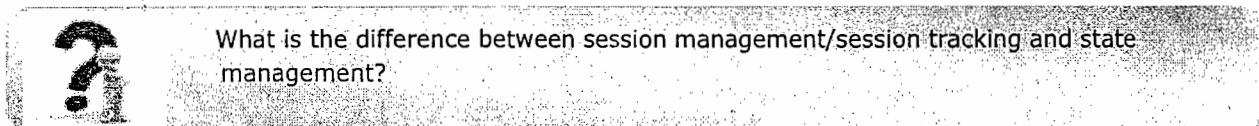
- # To remember user identity during session of email -operations.
- # While developing shopping cart applications in online shopping websites where items selected by end user by generating multiple requests will be remembered.
- # To remember user identity during an e-commerce session.
- # To remember credit card/debit card details during a session while performing the web based online transactions.
- # To remember player identity while playing online games.
- # To remember customer identity while performing online stock brokerage.
- # To remember end user choices and interests towards look & appearance of web pages if website allows to customize the look and appearance.
- # To render direct advertisement in websites.

#### **Point to Remember:**

The advertisement that comes based on the operations performed by end user is called "direct advertisement".

**Eg:** If end user is searching for new 7 wonders in google search engine the google website displays direct advertisements talking about tours and travels.

In session tracking, web application stores and remembers data across the multiple requests only during a session. Once the session between client and web application is completed the web application forgets client data. Storing client data in database table or in servletContext attributes across the multiple request doesnot come under session tracking or session management. Because they remember client data even after session completion and they do not store data and specific to one client.





In state management, web application remember client's data irrespective of the session that is started and completed using database table and ServletContext attributes support. In session management/tracking, web application remember client's data only during the session. Once session is completed this data will go off. For this hidden form fields, cookies, HttpSession, uri rewriting kind of techniques will be utilized.

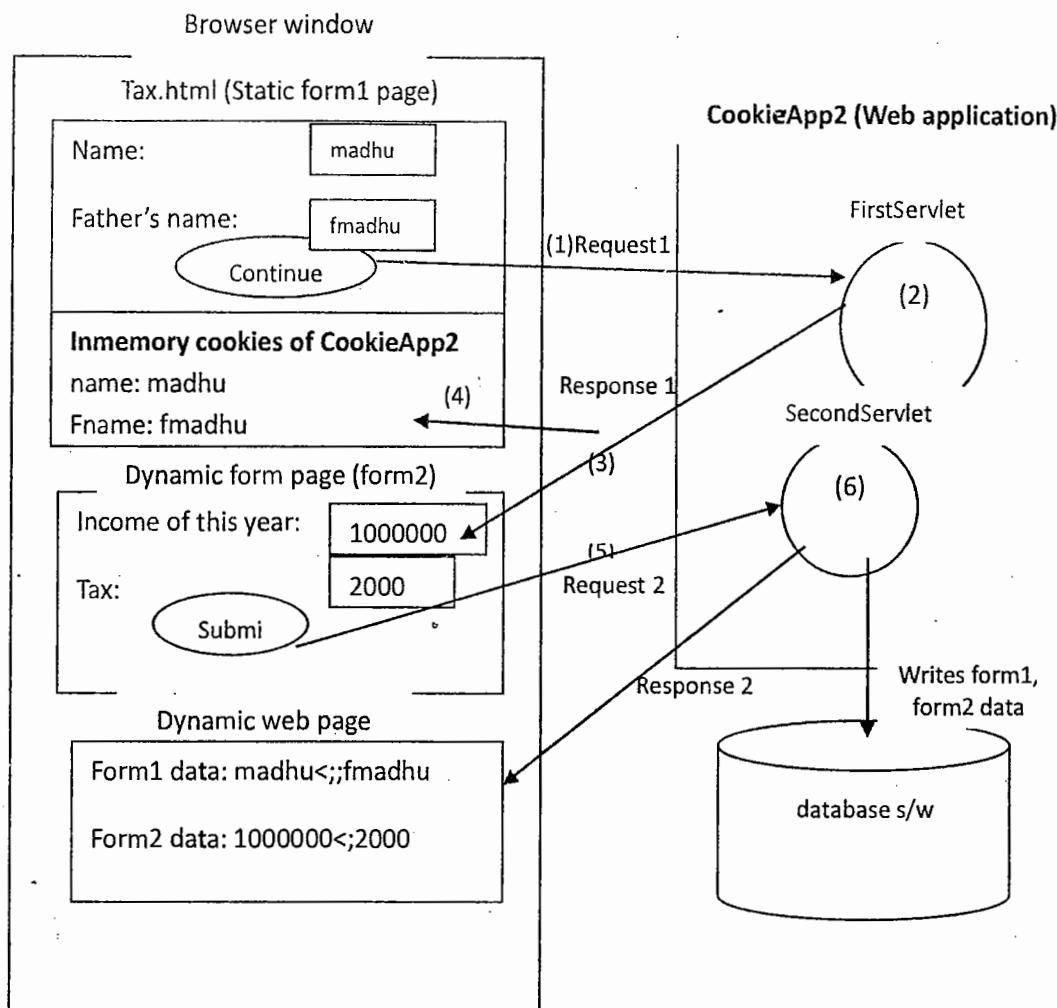
The session started between browser window and web application is specific to each browser window (client). Multiple clients start multiple sessions with single web application on one per client basis. Session and session related data always specific to one client so only that client can use session data across the multiple requests during that session.

### Technique2 :Http cookies

Cookies are small textual information which allocate memory at client side (browser window or client machine) having the capability to remember client data across the multiple requests during a session. To work with cookies we need the protocol Http and the servlets of type javax.servlet.http.HttpServlet. Cookies travel over the network along with request and response.

There are **two** types of cookies:

- 1) In memory/per session cookies
    - # No expiry time
    - # Allocates memory on browser window
    - # Once browser window is closed, these cookies will be destroyed automatically
  - 2) Persistent cookies
    - # Contains expiry time
    - # Allocates memory in the files of client machine file system
    - # Will be destroyed automatically once expiry time has reached
- # Every cookie contains a name and allows one string value.
  - # Every cookie contains cookie id and also remembers the web application/domain name for which it belongs to.
  - # Serverside web resource programs of web application create cookies and adds them to response to send to client. So cookies allocate memory as String information at client side
  - # Cookies go back to web application along with the request when browser window gives request back to web application for which these cookies belong to.
  - # Cookies come to client from server/web application along with the response ,as the values of special response header called "**set-cookie**".
  - # Cookies go back to the web application along with the request given by the browser window, as the values of special request header called "**cookie**".
  - # At client side or browser window, we can see multiple cookies belonging to different domains.



With respect to the diagram,

1. The form1 page gives request1 data to FirstServlet program of CookieApp2 web application having form1/request1 data.
2. FirstServlet program creates two in-memory cookies having form1/request1 data & adds them to response.
3. FirstServlet generates response, having 2 inmemory cookies and form2 as dynamic form page these inmemory cookies becomes value of **set-cookie** response header.
4. The two inmemory cookies of response1 allocates memory on browser window representing form1/request1 data and they also remember CookieApp2 as their domain name/web application name.
5. Form2 generates request2 to the SecondServlet program of CookieApp2 web application. This request carries form2 data and also carries the two cookie values of CookieApp2 application (form1/request1 data) as the values of RequestHeader "cookie".
6. SecondServlet reads form2 data directly from request2 and also reads form1/request1 data from the cookies of request2. That means SecondServlet program is able to use request1/form1 data while processing request2 (which is nothing but session tracking).
7. SecondServlet program writes form1/request1 data, form2/request2 data as record to database table.
8. SecondServlet program generates dynamic web page having form1/request1 data and form2/request2 data.

**Cookie API (working with methods of javax.Servlet.http.Cookie class):**

- # To create cookies and to add them to responseIn servlet program

1) `Cookie ck1=new Cookie("ap","hyd");`

Cookie name must be String cookie value must be string

`res.addCookie(ck1); //ck1 acts as inmemory cookie`

2) `Cookie ck2=new Cookie("mh","Mumbai");  
ck2.setMaxAge(1800);  
res.addCookie(ck2); //ck2 is persistent cookie having 1800 secs as expiry time.`



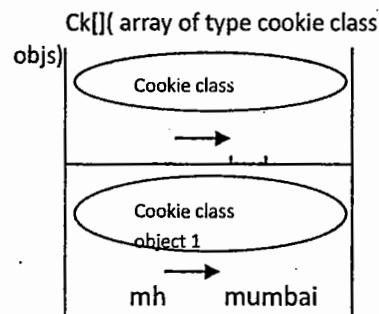
Every cookie must be added to response.

- # To modify cookie value:

```
ck1.setValue("hyd1");
ck2.setValue("navi Mumbai");
```

- # To read cookie values:

```
Cookie ck[]=req.getCookies();
if(ck!=null)
{
    for(int i=0;i<ck.length;i++)
    {
        pw.println(ck[i].getName()+" "+ck[i].getValue());
    }
} //if -> gives ap hyd
      mh Mumbai values
```



- # To delete cookies:

We cannot delete cookies programmatically being from servlet programs by using API because they allocate memory at client side.



In memory cookies will be destroyed once browser window is closed. Persistent cookies will be destroyed once their expiry time is reached or their related files are deleted or modified explicitly.

- # To know maximum age of cookies:

```
int time=ck1.getMaxAge(); // -1 will come which is the default max age of inmemory cookie
int time=ck2.getMaxAge(); //1800 sec will come
```

- # To know domain domainname:

```
String s=ck1.getDomain(); //gives domain/web application name of cookie
String s=ck2.getDomain();
```

**Point to remember:**

- # Cookie is a direct concrete class

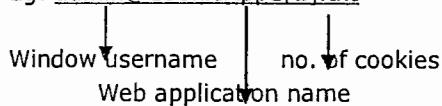
- # In windows Xp operating system the Internet Explorer browser window related persistent cookies allocates memory in a special text file of following directory:

C:\Documents and settings\<win-login user>\cookies folder

The notation of special text file is :

<windows user name>@<domain/web application name> [count]

Eg: admin@CookieApp1[2].txt



The Internet Explorer related persistence cookies will be destroyed automatically if end user try to modify content in the file where the persistent cookies reside (the above file).

In Netscape navigator browser window persistent cookies, in memory cookies allocate memory in browser window but persistent cookies contain expiry time.

To see all the cookies use:

Tools menu → cookie manager → manage stored cookies

- # In yahoo, gmail websites based login page there is a check box to remember username and password in the computer. When that checkbox is selected it uses persistence cookies to remember username and password

On that computer.

### **Advantages and Disadvantages of http cookies:**

#### **Advantages :**

- # Cookies allocate memory at clientside that means, they do not give burden to the server.
- # Cookies work with all serverside technologies and all servers.
- # Persistent cookies can remember client data even after session having expiry time.

#### **Disadvantages:**

- # To work with cookies our servlet type must be HttpServlet (cookies do not work with GenericServlet).
- # Cookies can be deleted through browser settings. This may fail session tracking.
  - In Internet Explorer: Tools menu → internet options → delete cookies option.
  - In Netscape: Tools menu → cookie manager → manage stored cookies → remove all cookies option.
- # Cookies can be restricted coming to client from server through browser settings. This may fail session tracking.
  - In Internet Explorer:
    - Tools menu → internet options → privacy tab (to block cookies).
    - In Netscape, use tools menu → cookie manager → block cookies from this site to block the cookies.
- # The values placed in cookies can be viewed through browser settings that means there is no data secrecy.
- # There is a restriction on no. of cookies that can be there that is per browser window, per domain maximum of 20 cookies and all domains together per browser window maximum of 300 cookies (may vary browser to browser).

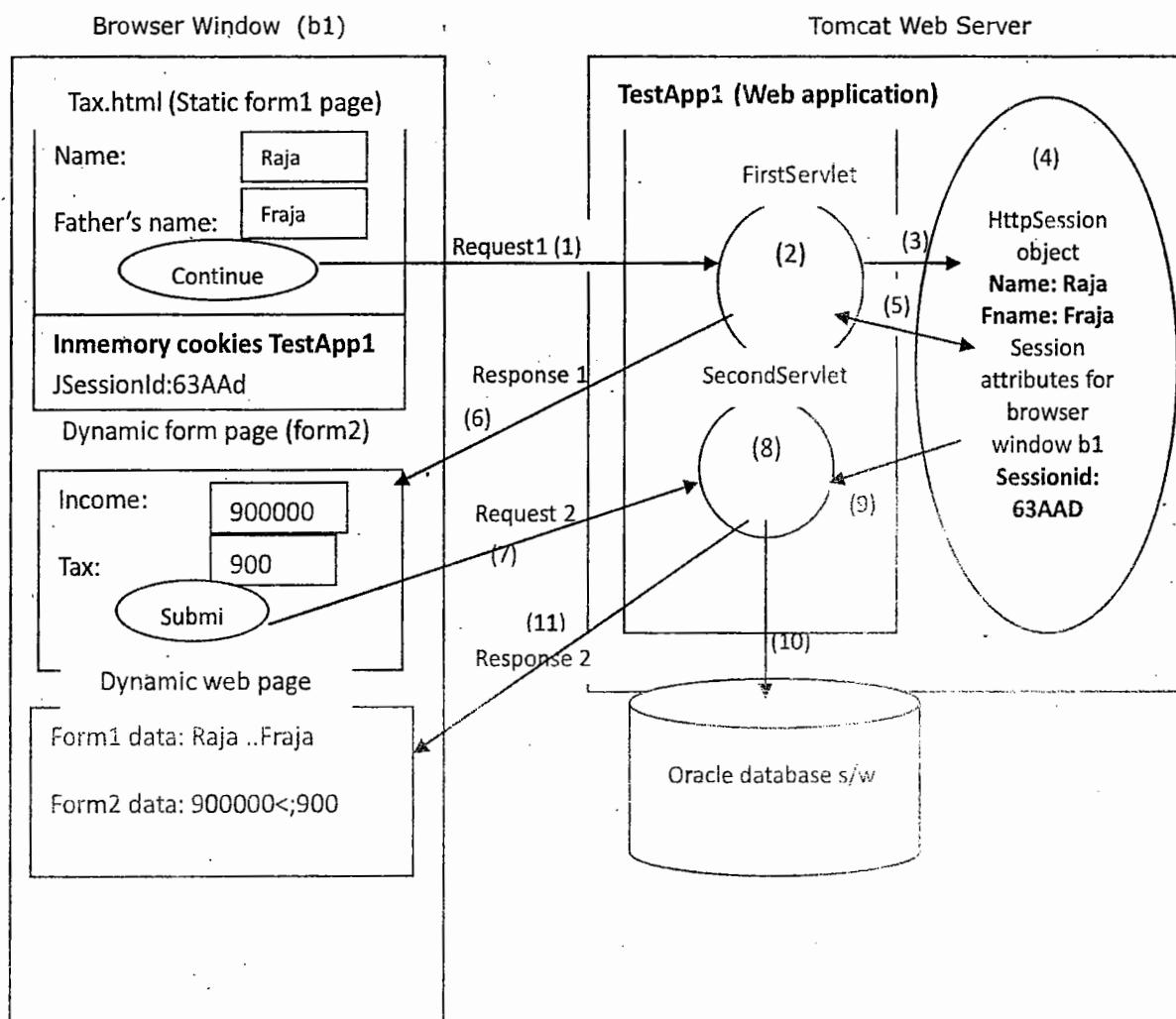
**Point to remember:**

Cookies can store only string values that means they cannot store other java objects as values.

**Technique3: HttpSession with Cookies:**

HttpSession object allocates memory on server on one per browser window (client basis). This HttpSession object remembers client data across the multiple requests during a session in the form of session attribute values.

If session object is created on the server for a client then we can say, session is started between browser window and web application and we can say session is completed when HttpSession object is closed or invalidated (made inactive). Every HttpSession object contains session ID and this session ID goes to browser window(client), comes back to server/web application in the form of cookie values.

**With respect to the diagram:**

1. The static form page Tax.html gives request 1 from browser window b1.
2. FirstServlet program of TestApp1 web application receives request1 and reads Tax.html/request1 data.
3. This FirstServlet program creates HttpSession object on the server for browser window b1 to begin session between browser window b1 and TestApp1 web application.
4. FirstServlet program keeps form1/request1 data in HttpSession object as session attribute value.
5. FirstServlet program gets the server generated session id of the HttpSession object.
6. FirstServlet program generates response having form2 as dynamic form page. This response also

- contains the session id of the session object(b1) as in-memory cookie value so that session id related cookie allocates memory on browser window as shown in the above diagram.
7. Form2 generates request2 to SecondServlet program of TestApp1 web application. This request contains cookie that holds session id.
  8. SecondServlet program reads the form2 data and session id from cookie.
  9. SecondServlet program uses the session id collected from cookie of request2 to search, access HttpSession object of browser window b1 on server and also reads its session attribute values which is nothing but form1/request1 data. That means SecondServlet is able to use form1/request1 data while processing form2 generated request2(Session Tracking).
  10. SecondServlet now can write form1/request1, form2/request2 data to database table as record.
  11. SecondServlet generates dynamic webpage as response2 having form1 and form2 data, if necessary it can also invalidate session by calling session.invalidate().



session object and its attributes are visible in all the web resource programs of java web application but they are specific to a client. That means the web resource programs must get request from that browser window for which session object is created in order to access that session object and its attributes.

#### Points to remember:

- # While working with this technique the browser window is identified as a client during the session based on its session id. For this the server sends the session id of session object to browser window along with the response once session object is created for that browser window on the server.
- # In Http cookies technique the client data during a session is preserved/remembered in the form of cookie values at client side.
- # In the HttpSession with cookies technique the client data is remembered/preserved during a session in the form of session attribute values of HttpSession object. Here cookie is just used to carry session id during a session.
- # In HttpSession with cookies technique, the cookie that is created holding session id is automatically generated inmemory cookie and programmer cannot make that cookie as persistent cookie.

#### Session API (working with javax.servlet.http.HttpSession):

To create HttpSession object (or) to get access to an existing HttpSession object.

##### # HttpSession ses=req.getSession();

This method creates new HttpSession object on the server for browser window if HttpSession object is not already available for that browser window on the server otherwise this method provides access to existing session object of browser window.

This method can create new session between browser window and web application if session is not already there otherwise it makes current request participating in the existing session i.e., already started between browser window and web application.

##### # HttpSession ses=req.getSession(false);

This method gives access to existing HttpSession object of browser window if HttpSession object is already available on the server for that browser widow otherwise returns null (i.e no new HttpSession object will be created on the server for browser window).

This method makes the browser generated request (client) to participate in the existing session that is already started between browser window and web application but it cannot create new session between browser window and web application.

##### # HttpSession ses=req.getSession(true);

This method creates new HttpSession object on the server for browser window if HttpSession object is not already available for that browser window on the server otherwise this method

provides access to existing session object of browser window.

This method can create new session between browser window and web application if session is not already there otherwise it makes current request participating in the existing session i.e., already started between browser window and web application.

#### **2.34.2.1 To invalidate the session/HttpSession object:**

- # Invalidating the session is nothing but closing the session between browser window and web application (like sign-out operation).
- # In this process the HttpSession object of browser window on the server becomes inactive object and ready for garbage collection object.
- 1. When browser window is closed.
  - # When browser window is closed the in-memory cookie that holds session-id will be destroyed. Due to this the HttpSession object on the server for that browser window becomes invalidated object.
- 2. When web resource program servlet/JSP program calls ses.invalidate();
- 3. When HttpSession object/session completes maximum inactive interval/session idle timeout period.
  - # Programmatic approach (java code)  
ses.setMaxInactiveInterval(1800); //1800 secs
  - # Declarative approach (xml code)
 

```
<web-app>
    <session-config>
        <session-timeout>20</session-timeout> // 20 mins
    </session-config>
    .....
    .....
</web-app>
```



Maximum Inactive Interval period/idle timeout period of session is nothing but how much maximum time that the session can be there in idle state continuously.

- # In most of the web servers the default maximum inactive intervals idle time out period of session is 30mins by default.

#### **Point to remember:**

- # Tomcat manager itself is a web application.



If programmer specifies session idle timeout period in a web application in both programmatic & declarative approaches having two different values. Can you tell me which one will be effected?

The configurations done in programmatic approach will be effected because the code in web resource program executes after web.xml file and overrides the settings done in web.xml file.

**To know maxInactiveInterval/session idle timeout period of session**

```
int time =ses.getMaxInactiveInterval();→ default is 30 minutes
```

**To know session ID**

```
String id = ses.getId( );
```

**To get access to servlet context object**

```
ServletContext sc=ses.getServletContext( );
```

**To know session/HttpSession object creation time**

```
long ms=ses.getCreationTime( );
```

Here "ms" represents number of milliseconds that are there between the creation date and time of session object and 00:00 hrs of Jan 1<sup>st</sup> 1970.

These milliseconds can be converted into java.util.Date class object using **Date d=new Date(ms);**

**To know the last accessed Date and time of HttpSession**

```
Long ms= ses.getLastAccessedTime( );
```

```
Date d=new Date (ms);
```

This method gives date and time representing when the web resource programs of web application lastly accessed this session object.

**To know whether this session is new/old:**

```
boolean b=ses.isNew( );
```

This method returns true when new session is started between browser window and web application otherwise this method returns false (when existing session is accessed). If this method is called in webresource program that creates new HttpSession object on the server for browser window that before the session id of that session object goes to browser window then this method returns true otherwise this method returns false.

In previous diagram if this method is called before 6 step then it returns true otherwise it returns false even though it is called in any servlet program. This method is useful for programmer to know current request given by client has created new session (or) has participated in existing session. In HttpSession object the client data during a session will be remembered as session attribute values.

**To create new session Attributes:**

Use ses.setAttribute (-,-) or ses.putValue(-,-) methods.

```
ses.setAttribute("name","raja");
ses.setAttribute("age",newInteger(22));
or
ses.put value("name","raja");
```

**To modify existing session attribute values**

Use ses.setAttribute(-,-) or ses.putValue(-,-) methods

```
ses.setAttribute("name","madhu");
ses.setAttribute("name",newInteger(21));
or
ses.putValue("name","madhu");
```

**To read existing session attribute values**

ses.getAttribute(-) or ses.getValue(-)methods

```
Integer il=(Integer)ses.getAttribute("age");
String name=(String)ses.getValue("name");
```

**To remove existing session attributes:**

ses.removeValue(-) or ses.removeAttribute(-) methods.

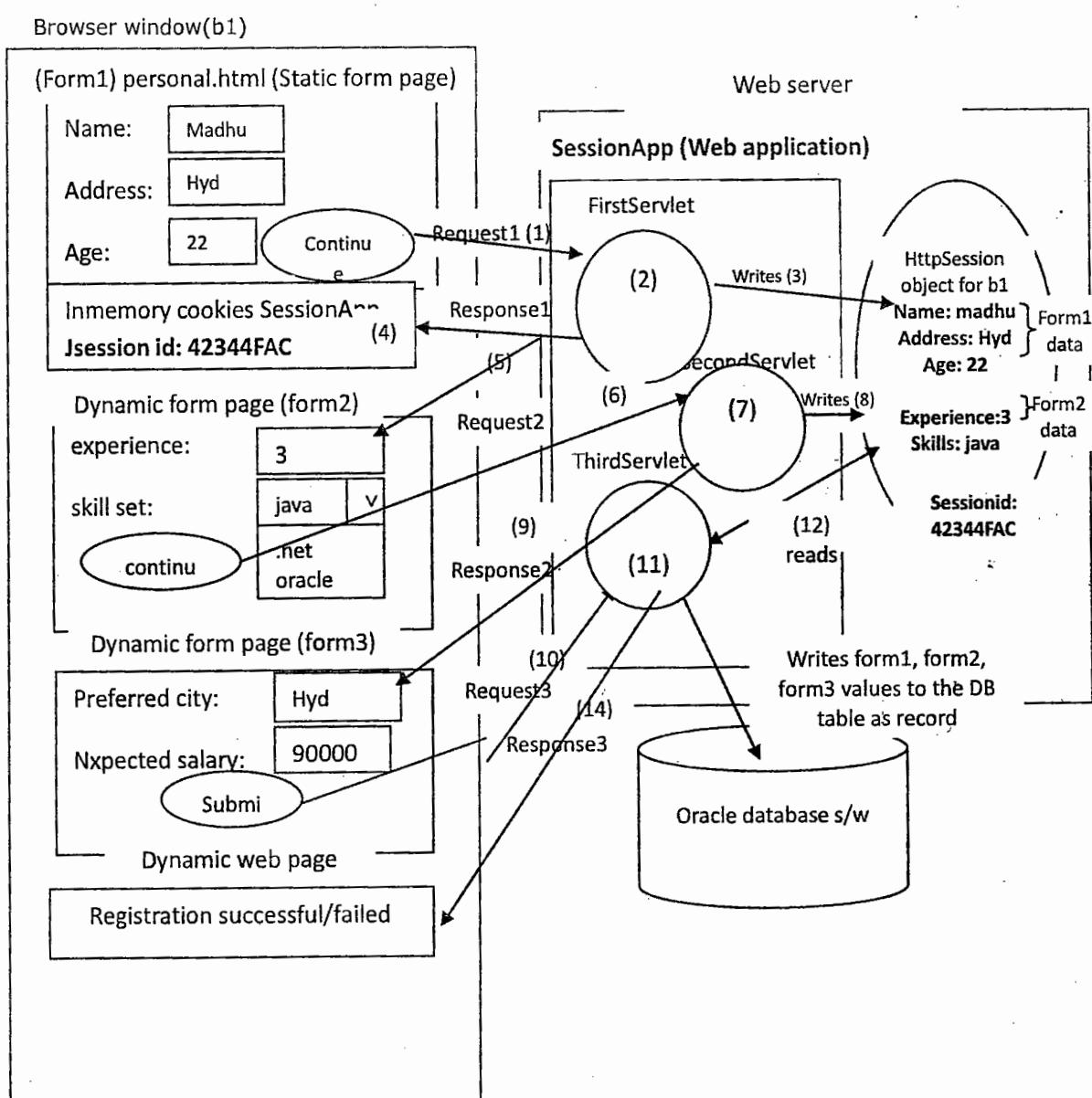
```
ses.removeAttribute("name");
ses.removeAttribute("age");or ses.removeValue("name");
```

**Point to remember:**

`ses.putValue()`, `ses.getValue()`, `ses.removeValue()` methods are deprecated from servlet-API 2.2 that means in future versions of servlet.API may not be available

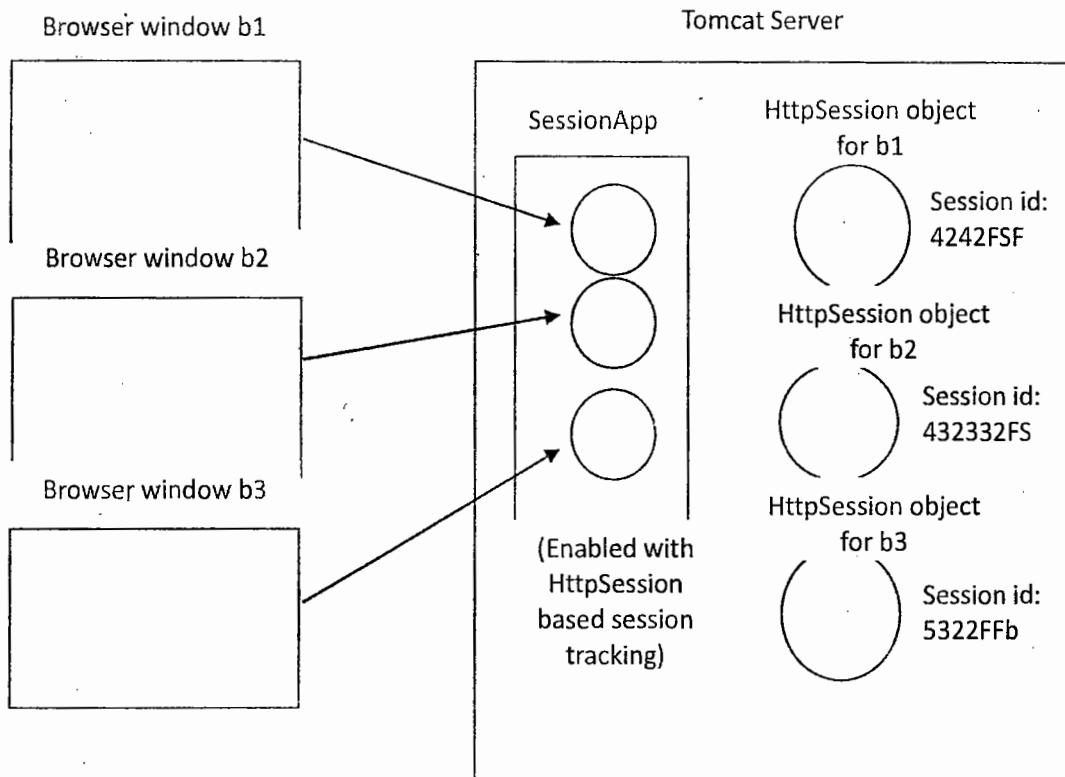
In the below application, form1/request1 data, form2/request2 data received by FirstServlet, SecondServlet is placed in HttpSession object as session attribute values. The ThirdServlet that is getting request3 from form3 is reading form1/request1, form2/request2 data from session object as session attribute values. That means ThirdServlet is able to use request1/form1, request2/form2 data while processing request3. This is nothing but "session tracking".

In the below diagram FirstServlet program begins the session between browser window b1 and SessionApp web application by creating new HttpSession object on server for browser window b1 and SecondServlet, ThirdServlet joins in that session for request2, request3, but ThirdServlet invalidate that session by calling `ses.invalidate()` method before response3.



If multiple browser windows are giving to a java web application on which HttpSession based session tracking is enabled then multiple HttpSession objects will be created on the server on one per browser window/client basis.

Web application treats every browser window as a client but web application does not treat tab of browser window as a separate client.



 What happens if browser window is closed and new browser window is opened in the middle of the session that is there with web application?

 When browser window is closed, the in memory cookie that contains sessionId will be destroyed and that sessionId is not visible for new browser window. So, the session of old browser window will be invalidated when that browser window is closed and this session will not be continued with new browser window.

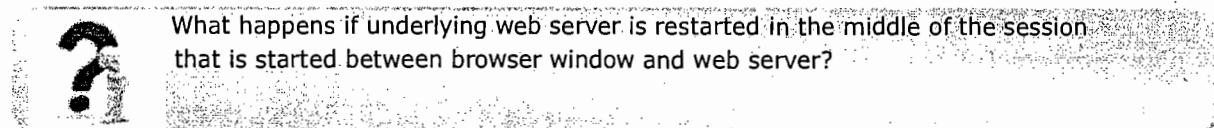
#### Advantages and disadvantages of HttpSession with cookies technique:

##### Advantages:

1. Client data across the multiple requests will be stored in the session attributes of session on the server. So, client data will not travel along with request and response over the network during session tracking, this gives data secrecy.
2. In session attributes, programmer can keep any kind of java objects as values.
3. Allows specifying session idle timeout period.
4. Server can create and manage any number of sessions by creating any number of HttpSession objects
5. Gives full control to begin the session and to wind up the session.
6. Through HttpSessionListener we can keep track of various operations related to this session tracking.
7. This technique works with all java based server side technologies and servers.

**Disadvantages:**

1. HttpSession object allocates memory on the server it may give burden to the server.
2. If cookies are restricted to coming to browser window or if cookies are deleted in the middle of the session, then this session tracking fails.
3. To work with this technique our servlet program should be of type HttpServlet class.



**A** Surprisingly, session will be continued after restarting the server. Reason is, when server is restarted/shut down, the server writes data and sessionIds of all HttpSession objects to a file of server machine file system through "**serialization**" process. So when the server is restarted these session objects will be from the file through "**deserialization**" process. We are not responsible for this serialization and deserialization server will take care.

**Point to Remember:** The above process takes place only in Tomcat server. In other servers like weblogic the session will be expired when you restart the server.

**Technique4 :HttpSession with URL rewriting:**

The third session tracking technique which is nothing but HttpSession with cookies uses in-memory cookies to send sessionId to browser window along with the response from web resource program and to send sessionId back to web application along with the request from browser window.

Due to this the session tracking of HttpSession with cookies techniques fails. If cookies are restricted coming to browser window, from web applications by using browser settings.

To overcome above problem do not depend upon cookies to send and receive sessionId along with the request and response to/from browser window respectively.

For this we can append sessionId to a url that goes to browser window along with the response and comes back to web application from browser window along with the request. This process is nothing but "URL rewriting" since HttpSession object is also involved in the url rewriting ,this process is technically called as "**HttpSession with URL rewriting**" session tracking technique.

If web resource program of web application generates dynamic form pages then the url placed in action attribute of <form> tag comes to browser window along with the response and goes back to web application along with the request submitted by dynamic form page. So, in HttpSession with url rewriting technique we can append sessionId to this url.

```
response.encodeURL("surl")
```

- response.encodeURL(-) method can append sessionId of current HttpSession object to the given url as shown below.
- response.encodeURL("surl"); → gives output like  
**surl;jsessionid=3ECFCC2136E**

**Key points:**

- # The difference between HttpSession with cookies, HttpSession with url rewriting techniques is the way they deal with the session id of the HttpSession object.
- # The HttpSession with cookies technique uses in- memory cookie to send sessionId to browser window along with the response and to bring sessionId back to web application from browser along with the request.
- # The HttpSession with URL rewriting, appends sessionId to a url that goes to browser window along with the response from web application and comes back to web application along with the request from browser window.

- # If web application is enabled with HttpSession with url rewriting and if cookies are not blocked through browser settings then web application uses cookies internally to deal with session ids by removing the effect of URL rewriting once the web application knows that the browser window has not blocked the cookies.

For Example Application on HttpSession with URL-Rewriting refer **App15** of page no:

**Example:**

```
=====
=====
App25(Session Tracking using HttpSession with URLReWriting)
=====
=====
-----Create info table in oracle database-----
CREATE TABLE INFO(NAME VARCHAR2(20),
                  ADDR VARCHAR2(20),
                  AGE VARCHAR2(3),
                  EXP VARCHAR2(2),
                  SKILLS VARCHAR2(20),
                  CITY VARCHAR2(20),
                  SALARY VARCHAR2(10));
-----personal.html-----
<HTML>
  <HEAD>
    <TITLE> HttpSession with URL Rewriting Example </TITLE>
  </HEAD>

  <BODY bgcolor="lightblue" >
    <form action ="furl" method="Post" >

      <h1><center><FONT COLOR="#FF0033">PERSONAL
      DETAILS</FONT></center><h1>
      <br><br>

      <table align="center">
        <center>
        <tr>
          <td>Enter Name:</td>
          <td><input type="text" name="name" ></td>
        </tr>
        <tr>
          <td>Enter Address:</td>
          <td><input type="text" name="address" ></td>
        </tr>
        <tr>
          <td>Enter Age:</td>
          <td><input type="text" name="age" ></td>
        </tr>
        <tr>
          <td><input type="Submit" value="Continue"></td>
        </tr>
      </table>
    </BODY>
  </HTML>
-----web.xml-----
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd">
```

```

<web-app>
  <servlet>
    <servlet-name>f</servlet-name>
    <servlet-class>com.nt.FirstServlet</servlet-class>
  </servlet>
    <servlet-mapping>
      <servlet-name>f</servlet-name>
      <url-pattern>/furl</url-pattern>
    </servlet-mapping>
  <servlet>
    <servlet-name>s</servlet-name>
    <servlet-class>com.nt.SecondServlet</servlet-class>
  </servlet>
    <servlet-mapping>
      <servlet-name>s</servlet-name>
      <url-pattern>/surl</url-pattern>
    </servlet-mapping>
  <servlet>
    <servlet-name>t</servlet-name>
    <servlet-class>com.nt.ThirdServlet</servlet-class>
  </servlet>
    <servlet-mapping>
      <servlet-name>t</servlet-name>
      <url-pattern>/turl</url-pattern>
    </servlet-mapping>

```

```
</web-app>
```

-----FirstServlet.java-----

```

package com.nt;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.net.*;
import java.util.*;

public class FirstServlet extends HttpServlet
{
  public void doGet(HttpServletRequest req,
                     HttpServletResponse res) throws ServletException, IOException
  {
    res.setContentType("text/html");
    PrintWriter pw = res.getWriter();

    String name=req.getParameter("name");
    String address=req.getParameter("address");
    String age=req.getParameter("age");

    HttpSession session = req.getSession(true);
    session.setAttribute("name", name);
    session.setAttribute("Addr", address);
    session.setAttribute("age", age);

    pw.println("<BODY BGCOLOR=cyan>");
    pw.println("<CENTER><H1><FONT COLOR=red>Provide Your Exp
              & Skills</FONT></H1></CENTER>");
    pw.println("<FORM ACTION="+res.encodeURL("surl")+" METHOD=GET>");

```

### -SecondServlet.java-

```
package com.nt;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.net.*;
import java.util.*;

public class SecondServlet extends HttpServlet
{
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        res.setContentType( "text/html" );
        PrintWriter pw = res.getWriter();

        String exp=req.getParameter("exp");
        String skils=req.getParameter("skils");

        HttpSession session = req.getSession(false) ;

        session.setAttribute("exp", exp );
        session.setAttribute("skils", skils );

        pw.println("<BODY BGCOLOR=cyan>");
    }
}
```

```

pw.println("<CENTER><H1><FONT COLOR=red>Provide City &
Salary</FONT></H1></CENTER>");

pw.println("<FORM ACTION="+res.encodeURL("turl")+" METHOD=GET>");
pw.println("<TABLE ALIGN=CENTER>");
pw.println("<TR>");
pw.println("<TD>");
pw.println("<H2><FONT COLOR=BLUE>Enter Preference City :");
pw.println("<INPUT TYPE=TEXT NAME=city SIZE=6>");
pw.println("</TD></TR>");

pw.println("<TR>");
pw.println("<TD>");
pw.println("<H2><FONT COLOR=BLUE>Enter Expected Salary :");
pw.println("<INPUT TYPE=TEXT NAME=sal SIZE=16>");
pw.println("</TD></TR>");

pw.println("<TR><TD>");
pw.println("<INPUT TYPE=SUBMIT VALUE=Submit >");
pw.println("</TD></TR>");
pw.println("</TABLE></FORM></BODY>");
} // service
}
} // class

```

---

```

-----ThirdServlet.java-----
package com.nt;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.net.*;
import java.util.*;
import java.sql.*;

public class ThirdServlet extends HttpServlet
{
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        doGet(req,res);
    }
}

```

```

-----ThirdServlet.java-----
package com.nt;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.net.*;
import java.util.*;
import java.sql.*;

public class ThirdServlet extends HttpServlet
{
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        res.setContentType( "text/html" );
        PrintWriter pw = res.getWriter( );

        String city=req.getParameter("city");
        String sal=req.getParameter("sal");

        HttpSession session = req.getSession(false) ;

        String name = (String)session.getAttribute("name");
        String addr = (String)session.getAttribute("Addr");
        String age = (String)session.getAttribute("age");
        String exp = (String)session.getAttribute("exp");
        String skils = (String)session.getAttribute("skils");
    }
}

```

```

try
{
    Class.forName("oracle.jdbc.driver.OracleDriver");
    Connection con=DriverManager.getConnection
        ("jdbc:oracle:thin:@localhost:1521:orcl", "scott",
    "tiger");
    PreparedStatement pst=
        con.prepareStatement("INSERT INTO INFO
VALUES(?,?,?,?,?,?)");

    pst.setString(1,name);
    pst.setString(2,addr);
    pst.setString(3,age);
    pst.setString(4,exp);
    pst.setString(5,skills);
    pst.setString(6,city);
    pst.setString(7,sal);

    int i = pst.executeUpdate();

    session.invalidate();

    if(i > 0)
    {
        pw.println("<BODY BGCOLOR=cyan>");
        pw.println("<CENTER><H1><FONT COLOR=red>Successfully
Inserted</FONT></H1></CENTER>");
        pw.println("<a href= personal.html>Home</a>");
        pw.println("</table></body>");
    }
    else
    {
        pw.println("<BODY BGCOLOR=cyan>");
        pw.println("<CENTER><H1><FONT COLOR=red>Try
Again</FONT></H1></CENTER>");
        pw.println("<a href= personal.html>Home</a>");
    }
} // try
catch(Exception e)
{
    e.printStackTrace();
    pw.println("<BODY BGCOLOR=cyan>");
    pw.println("<CENTER><H1><FONT COLOR=red>Try
Again</FONT></H1></CENTER>");
    pw.println("<a href= personal.html>Home</a>");
}
} // doGet(-,-)
public void doPost(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
    doGet(req,res);
} // class

```

**Advantages and disadvantages of HttpSession with URL rewriting:**

- # Same as the third technique i.e., "http session with cookies", But this technique continues session tracking even though cookies are restricted coming to browser window.

- # In order to append sessionID automatically to the url that is placed in response sendRedirect(-) method then use response.encodeRedirectURL(-) method.

In source servlet program

```
.....  
.....  
res.sendRedirect(res.encodeRedirectURL("/surl"));  
.....
```

- # If res.encodeRedirectURL("/s2url") gives /s2url;jsessionid=234242FAFBR2343 as output, then the output becomes the argument value of res.sendRedirect(-) to perform send redirection. It is useful to perform HttpSession with URL rewriting along with sendRedirection concept .

**Conclusion on session tracking:**

- \* While developing website for huge customers/clients then use Http cookies based session tracking because cookies allocates memory at client side and does not give burden to server.  
Eg: gmail.com, yahoo.com, online-shopping
- \* While developing websites related to certain private organization which is having limited no. of customers then use HttpSession with URL rewriting. Eg: [www.citibank.com](http://www.citibank.com), [www.sbilife.com](http://www.sbilife.com)
- \* If necessary we can apply multiple session tracking techniques in a single web application like
  - >use cookies for in-sensitive data based session tracking.(to remember the selected items of online shopping)
  - >Use HttpSession with URL rewriting in the same application for sensitive data based session tracking like username and password.



**What is the use of request, session, ServletContext attributes in real projects?**



Use servletcontext attributes to maintain global data in the web application(Eg: To remember request count). While working with 3<sup>rd</sup> & 4<sup>th</sup> techniques of session tracking use session attributes. In servlet to servlet communication or in servlet to JSP communication through rd.forward(-) method, the source servlet program uses request attributes to send data to destination servlet program/JSP program.

---

## Servlet Objective type Questions and Answers

**1. The method getWriter returns an object of type PrintWriter. This class has println methods to generate output. Which of these classes define the getWriter method? Select one correct answer.**

- a. HttpServletRequest b. HttpServletResponse c. ServletConfig d. ServletContext

**2. Name the method defined in the HttpServletResponse class that may be used to set the content type. Select one correct answer.**

- a. setType b. setContent c. setContentType d. setResponseContentType

**3. Which of the following statement is correct. Select one correct answer.**

- a. The response from the dedicated server to a HEAD request consists of status line, content type and the document.
- b. The response from the server to a GET request does not contain a document.
- c. The setStatus method defined in the HttpServletRequest class takes an int as an argument and sets the status of Http response
- d. The HttpServletResponse defines constants like SC\_NOT\_FOUND that may be used as a parameter to setStatus method.

**4. The sendError method defined in the HttpServlet class is equivalent to invoking the setStatus method with the following parameter. Select one correct answer.**

- a. SC\_OK b. SC\_MOVED\_TEMPORARILY c. SC\_NOT\_FOUND  
d. SC\_INTERNAL\_SERVER\_ERROR e. ESC\_BAD\_REQUEST

**5. The sendRedirect method defined in the HttpServlet class is equivalent to invoking the setStatus method with the following parameter and a Location header in the URL. (Select the one correct answer.)**

- a. SC\_OK b. SC\_MOVED\_TEMPORARILY c. SC\_NOT\_FOUND  
d. SC\_INTERNAL\_SERVER\_ERROR e. ESC\_BAD\_REQUEST

**6. Which of the following statements are correct about the status of the Http response. (Select the one correct answer.)**

- a. A status of 200 to 299 signifies that the request was successful.
- b. A status of 300 to 399 are informational messages.
- c. A status of 400 to 499 indicates an error in the server.
- d. A status of 500 to 599 indicates an error in the client.

**7. To send binary output in a response, the following method of HttpServletResponse may be used to get the appropriate Writer/Stream object. (Select the one correct answer.)**

- a. getStream b. getOutputStream c. getBinaryStream d. getWriter

**8. To send text output in a response, the following method of HttpServletResponse may be used to get the appropriate Writer/Stream object.**

- a. getStream b. getOutputStream c. getBinaryStream d. getWriter

**9. Is the following statement true or false. URL rewriting may be used when a browser is disabled. In URL encoding the session id is included as part of the URL.**

**10. Name the class that includes the getSession method that is used to get the HttpSession object.**

- a. HttpServletRequest b. HttpServletResponse c. SessionContext d. SessionConfig

**11. Which of the following are correct statements? Select two correct answers.**

- a. The getRequestDispatcher method of ServletContext class takes the full path of the servlet, whereas the getRequestDispatcher method of HttpServletRequest class takes the path of the servlet relative to the ServletContext.
- b. The include method defined in the RequestDispatcher class can be used to access one servlet from another. But it can be invoked only if no output has been sent to the server.
- c. The getRequestDispatcher(String URL) is defined in both ServletContext and HttpServletRequest method
- d. The getNamedDispatcher(String) defined in HttpServletRequest class takes the name of the servlet and returns an object of RequestDispatcher class.

**12. A user types the URL <http://www.javaprepare.com/scwd/index.html>. Which HTTP request gets generated. Select one correct answer.**

- a. GET method b. POST method c. HEAD method d. PUT method

**13. Which HTTP method gets invoked when a user clicks on a link? Select one correct answer.**

- a. GET method b. POST method c. HEAD method d. PUT method

**14. When using HTML forms which of the following is true for POST method?**

- a. POST allows users to bookmark URLs with parameters.
- b. The POST method should not be used when large amount of data needs to be transferred.
- c. POST allows secure data transmission over the http method.
- d. POST method sends data in the body of the request.

- 15. Which of the following is not a valid HTTP/1.1 method. Select one correct answer.**  
 a. CONNECT method      b. COMPARE method      c. OPTIONS method      d. TRACE method
- 16. Name the http method used to send resources to the server. Select one correct answer.**  
 a. FTP method      b. PUT method      c. WRITE method      d. COPY method
- 17. Name the http method that sends the same response as the request.**  
 a. DEBUG method      b. TRACE method      c. OPTIONS method      d. HEAD method
- 18. Which three digit error codes represent an error in request from client?**  
 a. Codes starting from 200      b. Codes starting from 300  
 c. Codes starting from 400      d. Codes starting from 500
- 19. Name the location of compiled class files within a war file? Select one correct answer.**  
 a. /META-INF/classes      b. /classes      c. /WEB-INF/classes      d. /root/classes

**Answers**

1. B. The class HttpServletResponse defines the getWriter method.
2. B. setContentType sets the content type of the response being sent to the client.
3. D. The response from the server to a HEAD request does not contain the document, whereas the response to GET request does contain a document. So A and B are incorrect. C is incorrect because setStatus is defined in HttpServletResponse.
4. C. sendError(String URL) is equivalent to sending SC\_NOT\_FOUND (404) response code.
5. B. sendRedirect(String URL) is equivalent to sending SC\_MOVED\_TEMPORARILY (302) response code and a location header in the URL.
6. A. The following table specifies the specific the status code of Http response.

Status Code	Purpose
100-199	Informational
200-299	Request was successful
300-399	Request file has moved.
400-499	Client error
500-599	Server error.

7. B. The getOutputStream method is used to get an output stream to send binary data. The getWriter method is used to get a PrintWriter object that can be used to send text data.
8. D
9. true. The statement about URL encoding is correct.
10. A. The class HttpServletRequest defines the getSession method.
11. A, C.
12. A. GET method gets invoked when a URL is typed.
13. A. GET method gets invoked when user clicks on a link.
14. D. Since POST does not have attributes in the URL, it cannot be used to bookmark the URL. Since arguments are present in the body of the request, using POST method does not guarantee security.
15. B. COMPARE is not a valid HTTP method.
16. B. PUT method is used to send resources from client to server.
17. B. TRACE method is used for debugging. It sends the same response as request.
18. C. A status code of 4XX represents a client error.
19. C. Classes are stored in /WEB-INF/classes.

**Servlet Interview Questions****1. What is the Servlet?**

A servlet is a Java programming language class that is used to extend the capabilities of servers that host applications accessed by means of a request-response programming model.

**2. What are the new features added to Servlet 2.5?**

Following are the changes introduced in Servlet 2.5:

- A new dependency on J2SE 5.0
- Support for annotations
- Loading the class
- Several web.xml conveniences
- A handful of removed restrictions
- Some edge case clarifications

**3. What are the uses of Servlet?**

Typical uses for HTTP Servlets include:

- Processing and/or storing data submitted by an HTML form.

Providing dynamic content, e.g. returning the results of a database query to the client.

A Servlet can handle multiple request concurrently and can be used to develop high performance system

Managing state information on top of the stateless HTTP, e.g. for an online shopping cart system which manages shopping carts for many concurrent customers and maps every request to the right customer.

#### 4. What are the advantages of Servlet over CGI?

Servlets have several advantages over CGI:

A Servlet does not run in a separate process. This removes the overhead of creating a new process for each request.

A Servlet stays in memory between requests. A CGI program (and probably also an extensive runtime system or interpreter) needs to be loaded and started for each CGI request.

There is only a single instance which answers all requests concurrently. This saves memory and allows a Servlet to easily manage persistent data.

Several web.xml conveniences

A handful of removed restrictions

Some edge case clarifications

#### 5. What are the phases of the servlet life cycle?

The life cycle of a servlet consists of the following phases:

**Servlet class loading** : For each servlet defined in the deployment descriptor of the Web application, the servlet container locates and loads a class of the type of the servlet. This can happen when the servlet engine itself is started, or later when a client request is actually delegated to the servlet.

**Servlet instantiation** : After loading, it instantiates one or more object instances of the servlet class to service the client requests.

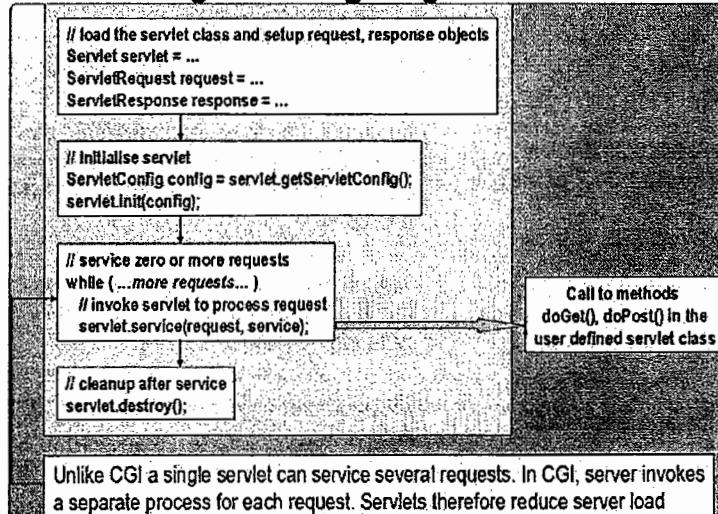
**Initialization (call the init method)** : After instantiation, the container initializes a servlet before it is ready to handle client requests. The container initializes the servlet by invoking its init() method, passing an object implementing the ServletConfig interface. In the init() method, the servlet can read configuration parameters from the deployment descriptor or perform any other one-time activities, so the init() method is invoked once and only once by the servlet container.

**Request handling (call the service method)** : After the servlet is initialized, the container may keep it ready for handling client requests. When client requests arrive, they are delegated to the servlet through the service() method, passing the request and response objects as parameters. In the case of HTTP requests, the request and response objects are implementations of HttpServletRequest and HttpServletResponse respectively. In the HttpServlet class, the service() method invokes a different handler method for each type of HTTP request, doGet() method for GET requests, doPost() method for POST requests, and so on.

**Removal from service (call the destroy method)** : A servlet container may decide to remove a servlet from service for various reasons, such as to conserve memory resources. To do this, the servlet container calls the destroy() method on the servlet. Once the destroy() method has been called, the servlet may not service any more client requests. Now the servlet instance is eligible for garbage collection

The life cycle of a servlet is controlled by the container in which the servlet has been deployed.

### Servlet Life Cycle Managed by the Server and Container



### 6. Why do we need a constructor in a servlet if we use the init method?

Even though there is an init method in a servlet which gets called to initialize it, a constructor is still required to instantiate the servlet. Even though you as the developer would never need to explicitly call the servlet's constructor, it is still being used by the container (the container still uses the constructor to create an instance of the servlet). Just like a normal POJO (plain old java object) that might have an init method, it is no use calling the init method if you haven't constructed an object to call it on yet.

### 7. How the servlet is loaded?

A servlet can be loaded when:

First request is made.

Server starts up (auto-load).

There is only a single instance which answers all requests concurrently. This saves memory and allows a Servlet to easily manage persistent data.

Administrator manually loads.

### 8. How a Servlet is unloaded?

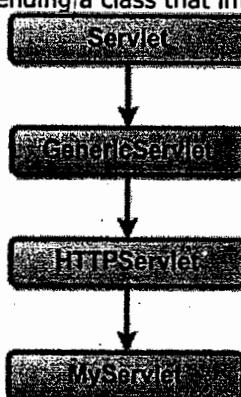
A servlet is unloaded when:

Server shuts down.

Administrator manually unloads.

### 9. What is Servlet interface?

The central abstraction in the Servlet API is the Servlet interface. All servlets implement this interface, either directly or, more commonly by extending a class that implements it.



*Note: Most Servlets, however, extend one of the standard implementations of that interface, namely javax.servlet.GenericServlet and javax.servlet.http.HttpServlet.*

### 10. What Is the GenericServlet class?

GenericServlet is an abstract class that implements the Servlet interface and the ServletConfig interface. In addition to the methods declared in these two interfaces, this class also provides simple versions of the lifecycle methods init and destroy, and implements the log method declared in the ServletContext interface.

*Note: This class is known as generic servlet, since it is not specific to any protocol.*

### 11. What's the difference between GenericServlet and HttpServlet?

GenericServlet	HttpServlet
The GenericServlet is an abstract class that is extended by HttpServlet to provide HTTP protocol-specific methods.	An abstract class that simplifies writing HTTP servlets. It extends the GenericServlet base class and provides a framework for handling the HTTP protocol.
The GenericServlet does not include protocol-specific methods for handling request parameters, cookies, sessions and setting response headers.	The HttpServlet subclass passes generic service method requests to the relevant doGet() or doPost() method.
GenericServlet is not specific to any protocol.	HttpServlet only supports HTTP and HTTPS protocol.

**12. Why is HttpServlet declared abstract?**

The HttpServlet class is declared abstract because the default implementations of the main service methods do nothing and must be overridden. This is a convenience implementation of the Servlet interface, which means that developers do not need to implement all service methods. If your servlet is required to handle doGet() requests for example, there is no need to write a doPost() method too.

**13. Can servlet have a constructor ?**

One can definitely have constructor in servlet. Even you can use the constructor in servlet for initialization purpose, but this type of approach is not so common. You can perform common operations with the constructor as you normally do. The only thing is that you cannot call that constructor explicitly by the new keyword as we normally do. In the case of servlet, servlet container is responsible for instantiating the servlet, so the constructor is also called by servlet container only.

**14. What are the types of protocols supported by HttpServlet ?**

It extends the GenericServlet base class and provides a framework for handling the HTTP protocol. So, HttpServlet only supports HTTP and HTTPS protocol.

**15. What is the difference between doGet() and doPost()?**

#	doGet()	doPost()
1	In doGet() the parameters are appended to the URL and sent along with header information.	In doPost(), on the other hand will (typically) send the information through a socket back to the webserver and it won't show up in the URL bar.
2	The amount of information you can send back using a GET is restricted as URLs can only be 1024 characters.	You can send much more information to the server this way - and it's not restricted to textual data either. It is possible to send files and even binary data such as serialized Java objects!
3	doGet() is a request for information; it does not (or should not) change anything on the server. (doGet() should be idempotent)	doPost() provides information (such as placing an order for merchandise) that the server is expected to remember
4	Parameters are not encrypted	Parameters are encrypted
5	doGet() is faster if we set the response content length since the same connection is used. Thus, increasing the performance	doPost() is generally used to update or post some information to the server. doPost is slower compared to doGet since doPost does not write the content length
6	doGet() should be idempotent. i.e. doGet should be able to be repeated safely many times	This method doesn't need to be idempotent. Operations requested through XMLHttpRequest can have side effects for which the user can be held accountable.
7	doGet() should be safe without any side effects for which user is held responsible	This method does not need to be either safe
8	It allows bookmarks.	It disallows bookmarks.

**16. When to use doGet() and when doPost()?**

Always prefer to use GET (As because GET is faster than POST), except mentioned in the following reason:

If data is sensitive

Data is greater than 1024 characters

If your application don't need bookmarks.

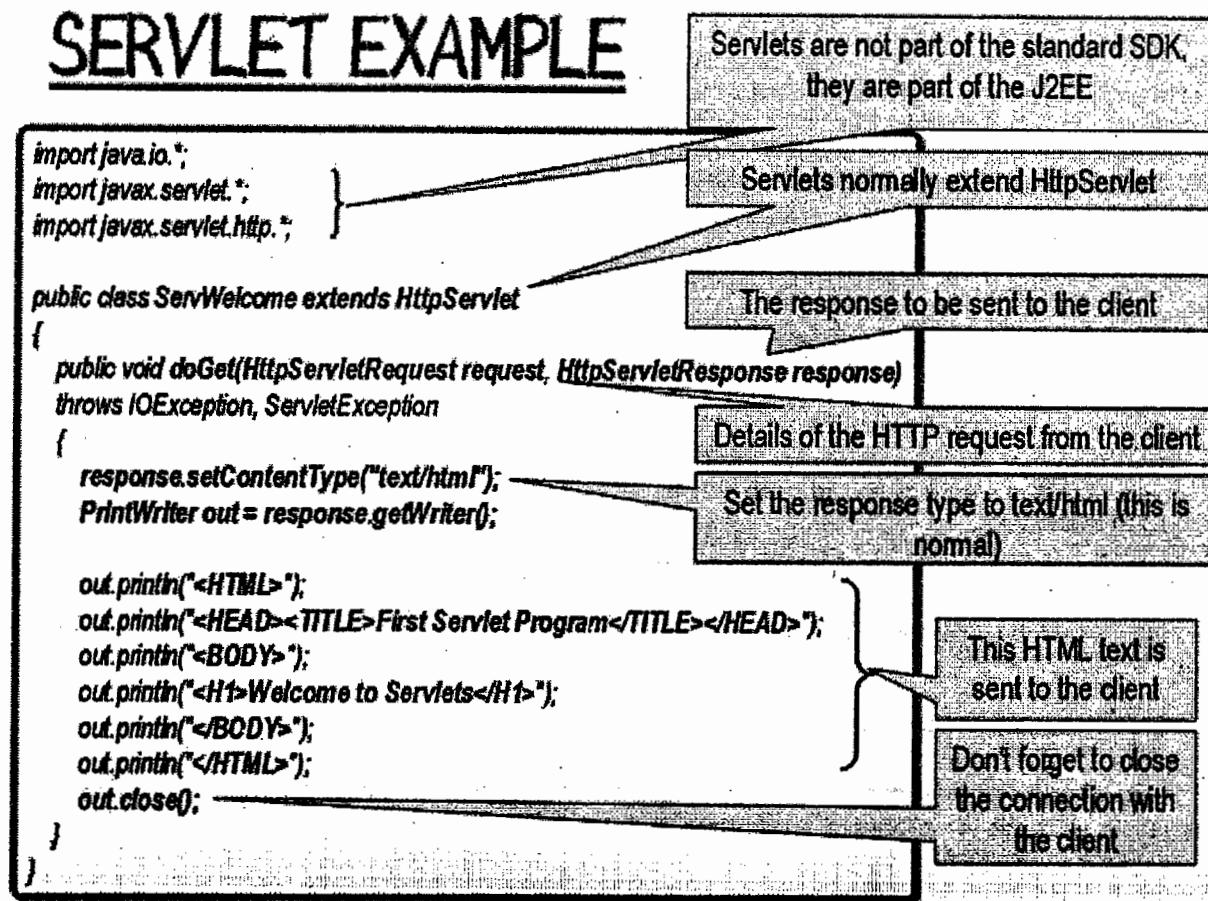
**17. How do I support both GET and POST from the same Servlet?**

The easy way is, just support POST, then have your doGet method call your doPost method:

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
                  throws ServletException, IOException
{
    doPost(request, response);
}
```

**18. Should I override the service() method?**

We never override the service method, since the HTTP Servlets have already taken care of it. The default service function invokes the doXXX() method corresponding to the method of the HTTP request. For example, if the HTTP request method is GET, doGet() method is called by default. A servlet should override the doXXX() method for the HTTP methods that servlet supports. Because HTTP service method checks the request method and calls the appropriate handler method, it is not necessary to override the service method itself, Only override the appropriate doXXX() method.

**19. How the typical servlet code look like ?****20. What Is a servlet context object?**

A servlet context object contains the information about the Web application of which the servlet is a part. It also provides access to the resources common to all the servlets in the application. Each Web application in a container has a single servlet context associated with it.

**21. What are the differences between the ServletConfig Interface and the ServletContext Interface?**

ServletConfig	ServletContext
The ServletConfig interface is implemented by the servlet container in order to pass configuration information to a servlet. The server passes an object that implements the ServletConfig interface to the servlet's init() method.	A ServletContext defines a set of methods that a servlet uses to communicate with its servlet container.
There is one ServletConfig parameter per servlet.	There is one ServletContext for the entire webapp and all the servlets in a webapp share it.
The param-value pairs for ServletConfig object are specified in the <init-param> within the <servlet> tags in the web.xml file	The param-value pairs for ServletContext object are specified in the <context-param> tags in the web.xml file.

## 22. What's the difference between forward() and sendRedirect() methods?

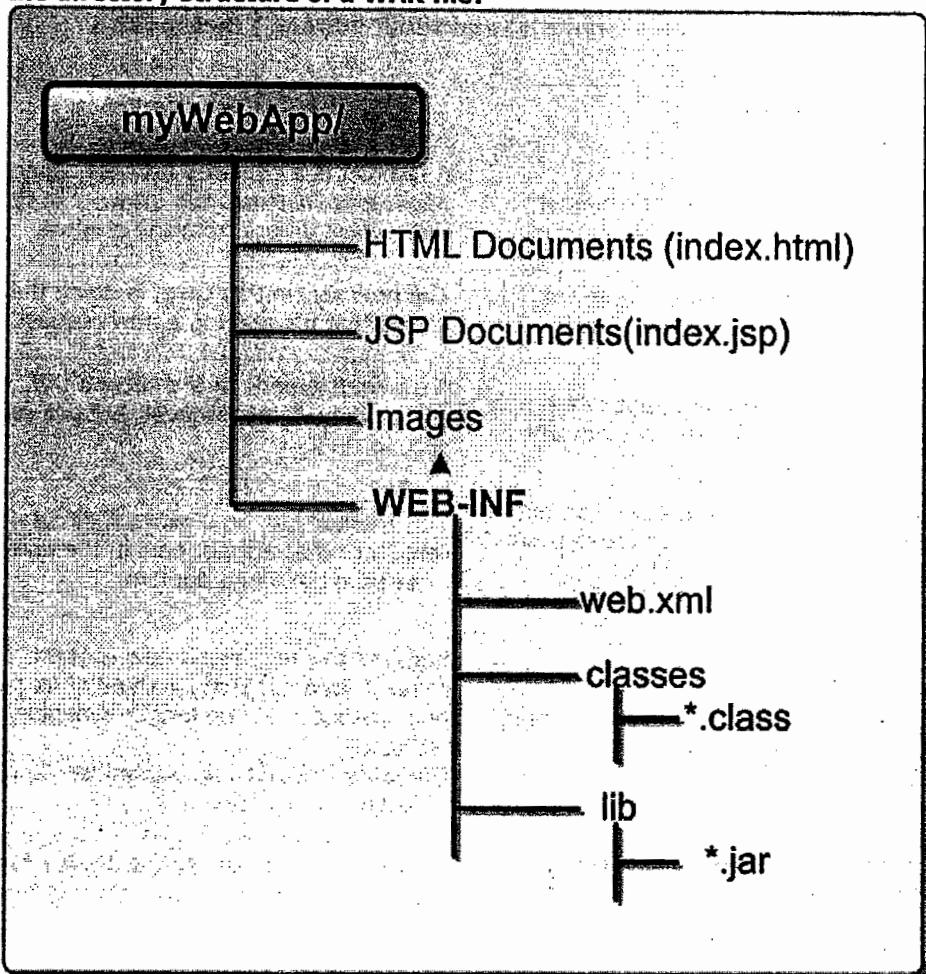
forward()	sendRedirect()
A forward is performed internally by the servlet.	A redirect is a two step process, where the web application instructs the browser to fetch a second URL, which differs from the original.
The browser is completely unaware that it has taken place, so its original URL remains intact.	The browser, in this case, is doing the work and knows that it's making a new request.
Any browser reload of the resulting page will simply repeat the original request, with the original URL	A browser reloads of the second URL ,will not repeat the original request, but will rather fetch the second URL.
Both resources must be part of the same context (Some containers make provisions for cross-context communication but this tends not to be very portable)	This method can be used to redirect users to resources that are not part of the current context, or even in the same domain.
Since both resources are part of the same context, the original request context is retained	Because this involves a new request, the previous request scope objects, with all of its parameters and attributes are no longer available after a redirect. (Variables need to be passed by via the session object).
Forward is marginally faster than redirect.	redirect is marginally slower than a forward, since it requires two browser requests, not one.

## 23. What is the difference between the include() and forward() methods?

Include()	forward()
The RequestDispatcher include() method inserts the contents of the specified resource directly in the flow of the servlet response, as if it were a part of the calling servlet.	The RequestDispatcher forward() method is used to show a different resource in place of the servlet that was originally called.
If you include a servlet or JSP document, the included resource must not attempt to change the response status code or HTTP headers, any such request will be ignored.	The forwarded resource may be another servlet, JSP or static HTML document, but the response is issued under the same URL that was originally requested. In other words, it is not the same as a redirection.
The include() method is often used to include common "boilerplate" text or template markup that may be included by many servlets.	The forward() method is often used where a servlet is taking a controller role; processing some input and deciding the outcome by returning a particular response page.

## 24. What's the use of the servlet wrapper classes??

The HttpServletRequestWrapper and HttpServletResponseWrapper classes are designed to make it easy for developers to create custom implementations of the servlet request and response types. The classes are constructed with the standard HttpServletRequest and HttpServletResponse instances respectively and their default behaviour is to pass all method calls directly to the underlying objects.

**25. What is the directory structure of a WAR file?****26. What is a deployment descriptor?**

A deployment descriptor is an XML document with an .xml extension. It defines a component's deployment settings. It declares transaction attributes and security authorization for an enterprise bean. The information provided by a deployment descriptor is declarative and therefore it can be modified without changing the source code of a bean.

The JavaEE server reads the deployment descriptor at run time and acts upon the component accordingly.

**27. What is the difference between the getRequestDispatcher(String path) method of javax.servlet.ServletRequest Interface and javax.servlet.ServletContext Interface?****ServletRequest.getRequestDispatcher(String path)**

The getRequestDispatcher(String path) method of javax.servlet.ServletRequest interface accepts parameter the path to the resource to be included or forwarded to, which can be relative to the request of the calling servlet. If the path begins with a "/" it is interpreted as relative to the current context root.

**ServletContext.getRequestDispatcher(String path)**

The getRequestDispatcher(String path) method of javax.servlet.ServletContext interface cannot accept relative paths. All path must start with a "/" and are interpreted as relative to current context root.

**28. What is preinitialization of a servlet?**

A container does not initialize the servlets as soon as it starts up, it initializes a servlet when it receives a request for that servlet first time. This is called lazy loading. The servlet specification defines the element, which can be specified in the deployment descriptor to make the servlet container load and initialize the servlet as soon as it starts up. The process of loading a servlet before any request comes in is called preloading or preinitializing a servlet.

**29. What is the <load-on-startup> element?**

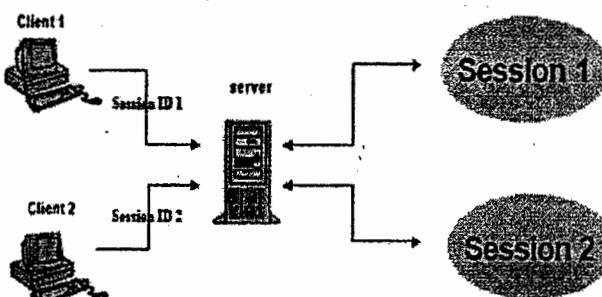
The <load-on-startup> element of a deployment descriptor is used to load a servlet file when the server starts instead of waiting for the first request. It is also used to specify the order in which the files are to be loaded. The <load-on-startup> element is written in the deployment descriptor as follows:

```
<servlet>
  <servlet-name>ServletName</servlet-name>
  <servlet-class>ClassName</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
```

*Note: The container loads the servlets in the order specified in the <load-on-startup> element.*

**30. What is session?**

A session refers to all the requests that a single client might make to a server in the course of viewing any pages associated with a given application. Sessions are specific to both the individual user and the application. As a result, every user of an application has a separate session and has access to a separate set of session variables.

**31. What is Session Tracking?**

Session tracking is a mechanism that servlets use to maintain state about a series of requests from the same user (that is, requests originating from the same browser) across some period of time.

**32. What is the need of Session Tracking in web application?**

HTTP is a stateless protocol i.e., every request is treated as new request. For web applications to be more realistic they have to retain information across multiple requests. Such information which is part of the application is referred as "state". To keep track of this state we need session tracking.

*Typical example:* Putting things one at a time into a shopping cart, then checking out--each page request must somehow be associated with previous requests.

**33. What are the types of Session Tracking ?**

Sessions need to work with all web browsers and take into account the users security preferences.

Therefore there are a variety of ways to send and receive the identifier:

**URL rewriting :** URL rewriting is a method of session tracking in which some extra data (session ID) is appended at the end of each URL. This extra data identifies the session. The server can associate this session identifier with the data it has stored about that session. This method is used with browsers that do not support cookies or where the user has disabled the cookies.

**Hidden Form Fields :** Similar to URL rewriting. The server embeds new hidden fields in every dynamically generated form page for the client. When the client submits the form to the server the hidden fields identify the client.

**Cookies :** Cookie is a small amount of information sent by a servlet to a Web browser. Saved by the browser, and later sent back to the server in subsequent requests. A cookie has a name, a single value, and optional attributes. A cookie's value can uniquely identify a client.

**Secure Socket Layer (SSL) Sessions :** Web browsers that support Secure Socket Layer communication can use SSL's support via HTTPS for generating a unique session key as part of the encrypted conversation.

**34. How do I use cookies to store session state on the client?**

In a servlet, the `HttpServletResponse` and `HttpServletRequest` objects passed to method `HttpServlet.service()` can be used to create cookies on the client and use cookie information transmitted during client requests. JSPs can also use cookies, in scriptlet code or, preferably, from within custom tag code.

To set a cookie on the client, use the `addCookie()` method in class `HttpServletResponse`. Multiple cookies may be set for the same request, and a single cookie name may have multiple values.

To get all of the cookies associated with a single HTTP request, use the `getCookies()` method of class `HttpServletRequest`.

**35. What are some advantages of storing session state in cookies?**

Cookies are usually persistent, so for low-security sites, user data that needs to be stored long-term (such as a user ID, historical information, etc.) can be maintained easily with no server interaction.

For small- and medium-sized session data, the entire session data (instead of just the session ID) can be kept in the cookie.

**36. What are some disadvantages of storing session state in cookies?**

Cookies are controlled by programming a low-level API, which is more difficult to implement than some other approaches.

All data for a session are kept on the client. Corruption, expiration or purging of cookie files can all result in incomplete, inconsistent, or missing information.

Cookies may not be available for many reasons: the user may have disabled them, the browser version may not support them, the browser may be behind a firewall that filters cookies, and so on. Servlets and JSP pages that rely exclusively on cookies for client-side session state will not operate properly for all clients. Using cookies, and then switching to an alternate client-side session state strategy in cases where cookies aren't available, complicates development and maintenance.

Browser instances share cookies, so users cannot have multiple simultaneous sessions.

Cookie-based solutions work only for HTTP clients. This is because cookies are a feature of the HTTP protocol. Notice that the while package javax.servlet.http supports session management (via class HttpSession), package javax.servlet has no such support.

**37. What is URL rewriting?**

URL rewriting is a method of session tracking in which some extra data is appended at the end of each URL. This extra data identifies the session. The server can associate this session identifier with the data it has stored about that session.

Every URL on the page must be encoded using method `HttpServletResponse.encodeURL()`. Each time a URL is output, the servlet passes the URL to `encodeURL()`, which encodes session ID in the URL if the browser isn't accepting cookies, or if the session tracking is turned off.

E.g., `http://abc/path/index.jsp;jsessionid=123465hfhs`

**Advantages**

URL rewriting works just about everywhere, especially when cookies are turned off.

Multiple simultaneous sessions are possible for a single user. Session information is local to each browser instance, since it's stored in URLs in each page being displayed. This scheme isn't foolproof, though, since users can start a new browser instance using a URL for an active session, and confuse the server by interacting with the same session through two instances.

Entirely static pages cannot be used with URL rewriting, since every link must be dynamically written with the session state. It is possible to combine static and dynamic content, using (for example) templating or server-side includes. This limitation is also a barrier to integrating legacy web pages with newer, servlet-based pages.

**DisAdvantages**

Every URL on a page which needs the session information must be rewritten each time a page is served. Not only is this expensive computationally, but it can greatly increase communication overhead.

URL rewriting limits the client's interaction with the server to HTTP GETs, which can result in awkward restrictions on the page.

URL rewriting does not work well with JSP technology.

If a client workstation crashes, all of the URLs (and therefore all of the data for that session) are lost.

**38. How can an existing session be invalidated?**

An existing session can be invalidated in the following two ways:

Setting timeout in the deployment descriptor: This can be done by specifying timeout between the `<session-timeout>` tags as follows:

```
<session-config>
  <session-timeout>10</session-timeout>
</session-config>
```

This will set the time for session timeout to be ten minutes.

Setting timeout programmatically: This will set the timeout for a specific session. The syntax for setting the timeout programmatically is as follows:

```
public void setMaxInactiveInterval(int interval)
```

The `setMaxInactiveInterval()` method sets the maximum time in seconds before a session becomes invalid.

Note : Setting the inactive period as negative(-1), makes the container stop tracking session, i.e., session never expires.

### 39. How can the session in Servlet can be destroyed?

An existing session can be destroyed in the following two ways:

Programmatically : Using session.invalidate() method, which makes the container abandon the session on which the method is called.

When the server itself is shutdown.

### 40. A client sends requests to two different web components. Both of the components access the session. Will they end up using the same session object or different session ?

Creates only one session i.e., they end up with using same session .

Sessions is specific to the client but not the web components. And there is a 1-1 mapping between client and a session.

### 41. What is servlet lazy loading?

A container doesnot initialize the servlets as soon as it starts up, it initializes a servlet when it receives a request for that servlet first time. This is called lazy loading.

The servlet specification defines the <load-on-startup> element, which can be specified in the deployment descriptor to make the servlet container load and initialize the servlet as soon as it starts up.

The process of loading a servlet before any request comes in is called preloading or preinitializing a servlet.

### 42. What is Servlet Chaining?

Servlet Chaining is a method where the output of one servlet is piped into a second servlet. The output of the second servlet could be piped into a third servlet, and so on. The last servlet in the chain returns the output to the Web browser.

### 43. How are filters?

Filters are Java components that are used to intercept an incoming request to a Web resource and a response sent back from the resource. It is used to abstract any useful information contained in the request or response. Some of the important functions performed by filters are as follows:

Security checks

Modifying the request or response

Data compression

Logging and auditing

Response compression

Filters are configured in the deployment descriptor of a Web application. Hence, a user is not required to recompile anything to change the input or output of the Web application.

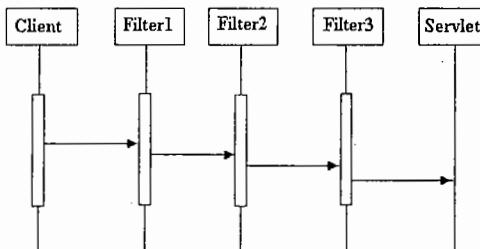
### 44. What are the functions of an intercepting filter?

The functions of an intercepting filter are as follows:

It intercepts the request from a client before it reaches the servlet and modifies the request if required.

It intercepts the response from the servlet back to the client and modifies the request if required.

There can be many filters forming a chain, in which case the output of one filter becomes an input to the next filter. Hence, various modifications can be performed on a single request and response.



### 45. What are the functions of the Servlet container?

The functions of the Servlet container are as follows:

**Lifecycle management** : It manages the life and death of a servlet, such as class loading, instantiation, initialization, service, and making servlet instances eligible for garbage collection.

**Communication support** : It handles the communication between the servlet and the Web server.

**Multithreading support** : It automatically creates a new thread for every servlet request received. When the Servlet service() method completes, the thread dies.

**Declarative security** : It manages the security inside the XML deployment descriptor file.

**JSP support** : The container is responsible for converting JSPs to servlets and for maintaining them.

## **JSP**

### **JavaServer Pages**

JavaServer Pages (JSP) technology is the Java platform technology for delivering dynamic content to web clients in a portable, secure and well-defined way. The JavaServer Pages specification extends the Java Servlet API to provide web application developers with a robust framework for creating dynamic web content on the server using HTML, and XML templates, and Java code, which is secure, fast, and independent of server platforms. JSP has been built on top of the Servlet API and utilizes Servlet semantics. JSP has become the preferred request handler and response mechanism. Although JSP technology is going to be a powerful successor to basic Servlets, they have an evolutionary relationship and can be used in a cooperative and complementary manner.

Servlets are powerful and sometimes they are a bit cumbersome when it comes to generating complex HTML. Most servlets contain a little code that handles application logic and a lot more code that handles output formatting. This can make it difficult to separate and reuse portions of the code when a different output format is needed. For these reasons, web application developers turn towards JSP as their preferred servlet environment.

#### **Evolution of Web Applications**

Over the last few years, web server applications have evolved from static to dynamic applications. This evolution became necessary due to some deficiencies in earlier web site design. For example, to put more of business processes on the web, whether in business-to-consumer (B2C) or business-to-business (B2B) markets, conventional web site design technologies are not enough. The main issues, every developer faces when developing web applications, are:

1. Scalability - a successful site will have more users and as the number of users is increasing fastly, the web applications have to scale correspondingly.
2. Integration of data and business logic - the web is just another way to conduct business, and so it should be able to use the same middle-tier and data-access code.
3. Manageability - web sites just keep getting bigger and we need some viable mechanism to manage the ever-increasing content and its interaction with business systems.
4. Personalization - adding a personal touch to the web page becomes an essential factor to keep our customer coming back again. Knowing their preferences, allowing them to configure the information they view, remembering their past transactions or frequent search keywords are all important in providing feedback and interaction from what is otherwise a fairly one-sided conversation.

Apart from these general needs for a business-oriented web site, the necessity for new technologies to create robust, dynamic and compact server-side web applications has been realized. The main characteristics of today's dynamic web server applications are as follows:

1. Serve HTML and XML, and stream data to the web client
2. Separate presentation, logic and data
3. Interface to databases, other Java applications, CORBA, directory and mail services
4. Make use of application server middleware to provide transactional support.
5. Track client sessions

Now let us have a look on the role of Java technology and platform in this regard.

#### **Java's Role for Server Applications**

Sun Microsystems, having consulted many expert partners from other related IT industries, has come out with a number of open APIs for the technologies and services on server side. This collection of APIs is named as Java 2 Enterprise Edition ([J2EE](#)). The J2EE specification provides a platform for enterprise applications, with full API support for enterprise code and guarantees of portability between server implementations. Also it brings a clear division between code which deals with presentation, business logic and data.

The J2EE specification meets the needs of web applications because it provides:

1. Rich interaction with a web server via servlets and built-in support for sessions available in both servlets and EJBs.
2. The use of EJBs to mirror the user interaction with data by providing automatic session and transaction support to EJBs operating in the EJB server.
3. Entity EJBs to represent data as an object and seamless integration with the Java data access APIs
4. Flexible template-based output using JSP and XML

This family of APIs mean that the final web page can be generated from a user input request, which was processed by a servlet or JSP and a session EJB, which represents the user's session with the server, using data extracted from a database and put into an entity EJB. Thus, the Java revolution of portable code and open APIs is married with an evolution in existing products such as database, application, mail and web servers. The wide availability of products to run Java applications on the server has made this a fast-moving and very competitive market, but the essential compatibility through specifications, standard APIs and class libraries has held. This makes server-side Java a very exciting area.

### **JavaServer Pages - An Overview**

The JavaServer Pages 1.2 specification provides web developers with a framework to build applications containing dynamic web content such as HTML, DHTML, XHTML and XML. A JSP page is a text based document containing static HTML and dynamic actions which describe how to process a response to the client in a more powerful and flexible manner. Most of a JSP file is plain HTML but it also has, interspersed with it, special JSP tags.

Click here for a simple [JSP file](#). There are many JSP tags such as:

1. `<%@ page language="java" %>` - this is a JSP directive denoted by `<%@`,
2. scriptlets indicated by `<% ... %>` tags and
3. `<%@ include file="sample.html" %>` -this directive includes the contents of the file sample.html in the response at that point.

To process a JSP file, we need a JSP engine that can be connected with a web server or can be accommodated inside a web server. Firstly when a web browser seeks a JSP file through an URL from the web server, the web server recognizes the .jsp file extension in the URL requested by the browser and understands that the requested resource is a JavaServer Page. Then the web server passes the request to the JSP engine. The JSP page is then translated into a Java class, which is then compiled into a servlet.

This translation and compilation phase occurs only when the JSP file is requested for the first time, or if it undergoes any changes to the extent of getting retranslated and recompiled. For each additional request of the JSP page thereafter, the request directly goes to the servlet byte code, which is already in memory. Thus when a request comes for a servlet, an `init()` method is called when the Servlet is first loaded into the virtual machine, to perform any global initialization that every request of the servlet will need. Then the individual requests are sent to a `service()` method, where the response is put together. The servlet creates a new thread to run `service()` method for each request. The request from the browser is converted into a Java object of type `HttpServletRequest`, which is passed to the Servlet along with an `HttpServletResponse` object that is used to send the response back to the browser. The servlet code performs the operations specified by the JSP elements in the .jsp file.

### **The Components of JSPs**

JSP syntax is almost similar to XML syntax. The following general rules are applicable to all JSP tags.

1. Tags have either a start tag with optional attributes, an optional body, and a matching end tag or they have an empty tag possibly with attributes.
2. Attribute values in the tag always appear quoted. The special strings ' and " can be used if quotes are a part of the attribute value itself.

Any whitespace within the body text of a document is not significant, but is preserved, which means that any whitespace in the JSP being translated is read and preserved during translation into a servlet.

The character \ can be used as an escape character in a tag, for instance, to use the % character, \% can be used.

JavaServer Pages are text files that combine standard HTML and new scripting tags. JSPs look like HTML, but they get compiled into Java servlets the first time they are invoked. The resulting servlet is a combination of HTML from the JSP file and embedded dynamic content specified by the new tags. Everything in a JSP page can be divided into two categories:

1. Elements that are processed on the server
2. Template data or everything other than elements, that the engine processing the JSP engines.

Element data or that part of the JSP which is processed on the server, can be classified into the following categories:

1. Directives
2. Scripting elements
3. Standard actions

JSP directives serve as messages to the JSP container from the JSP. They are used to set global values such as class declaration, methods to be implemented, output content type, etc. They do not produce any output to the client. All directives have scope of the entire JSP file. That is, a directive affects the whole JSP file, and only that JSP file. Directives are characterized by the @ character within the tag and the general syntax is:

```
<%@ directivename attribute="value" attribute="value" %>
```

The three directives are page, include and taglib.

Scripting elements are used to include scripting code (Java code) within the JSP. They allow to declare variables and methods, include arbitrary scripting code and evaluate an expression. The three types of scripting element are: Declaration, Scriptlets and Expressions.

A declaration is a block of Java code in a JSP that is used to define class-wide variables and methods in the generated class file. Declarations are initialized when the JSP page is initialized and have class scope. Anything defined in a declaration is available throughout the JSP, to other declarations, expressions or code.

A scriptlet consists of one or more valid Java statements. A scriptlet is a block of Java code that is executed at request-processing time. A scriptlet is enclosed between <% and %>. What the scriptlet actually does depends on the code, and it can produce output into the output stream to the client. Multiple scriptlets are combined in the compiled class in the order in which they appear in the JSP. Scriptlets like any other Java code block or method, can modify objects inside them as a result of method invocations.

An expression is a shorthand notation for a scriptlet that outputs a value in the response stream back to the client. When the expression is evaluated, the result is converted to a string and displayed. An expression is enclosed within <%= and %>. If any part of expression is an object, the conversion is done using the `toString()` method of the object.

Standard actions are specific tags that affect the runtime behavior of the JSP and affect the response sent back to the client. The JSP specification lists some standard action types to be provided by all containers, irrespective of the implementation. Standard actions provide page authors with some basic functionality to exploit; the vendor is free to provide other actions to enhance behavior.

### How JSP and JSP Container function

A JSP page is executed in a JSP container or a JSP engine, which is installed in a web server or in a application server. When a client asks for a JSP page the engine wraps up the request and delivers it to the JSP page along with a response object. The JSP page processes the request and modifies the response object to incorporate the communication with the client. The container or the engine, on getting the response, wraps up the responses from the JSP page and delivers it to the client. The underlying layer for a JSP is actually a servlet implementation. The abstractions of the request and response are the same as

the HttpServletRequest and HttpServletResponse respectively. If the protocol used is HTTP, then the corresponding objects are HttpServletRequest and HttpServletResponse.

The first time the engine intercepts a request for a JSP, it compiles this translation unit (the JSP page and other dependent files) into a class file that implements the servlet protocol. If the dependent files are other JSPs they are compiled into their own classes. The servlet class generated at the end of the translation process must extend a superclass that is either

1. specified by the JSP author through the use of the extends attribute in the page directive or
2. is a JSP container specific implementation class that implements javax.servlet.jsp.JspPage interface and provides some basic page specific behavior.

Since most JSP pages use HTTP, their implementation classes must actually implement the javax.servlet.jsp.HttpJspPage interface, which is a sub interface of javax.servlet.jsp.JspPage.

The javax.servlet.jsp.JspPage interface contains two methods:

1. public void jspInit() - This method is invoked when the JSP is initialized and the page authors are free to provide initialization of the JSP by implementing this method in their JSPs.
2. public void jspDestroy() - This method is invoked when the JSP is about to be destroyed by the container. Similar to above, page authors can provide their own implementation.

The javax.servlet.jsp.HttpJspPage interface contains one method:

```
public void _jspService(HttpServletRequest request, HttpServletResponse response) throws  
ServletException, IOException
```

This method generated by the JSP container is invoked, every time a request comes to the JSP. The request is processed and the JSP generates appropriate response. This response is taken by the container and passed back to the client.

### **JSP Architecture**

There are two basic ways of using the JSP technology. They are the client/server (page-centric) 2-tier approach and the N-tier approach (dispatcher).

#### **The Page-Centric Approach**

Applications built using a client-server (2-tier) approach consist of one or more application programs running on client machines and connecting to a server-based application to work. With the arrival of Servlets technology, 2-tier applications could also be developed using Java programming language. This model allows JSPs or Servlets direct access to some resource such as database or legacy application to service a client's request. The JSP page is where the incoming request is intercepted, processed and the response sent back to the client. JSPs differ from Servlets in this scenario by providing clean code, separating code from the content by placing data access in EJBs. Even though this model makes application development easier, it does not scale up well for a large number of simultaneous clients as it entails a significant amount of request processing to be performed and each request must establish or share a potentially scarce/expensive connection to the resource in question.

**Page-view** - This basic architecture involves direct request invocations to a server page with embedded Java code, and markup tags which dynamically generate output for substitution within the HTML. This approach has been blessed a number of benefits. It is very straightforward and is a low-overhead approach from a developer perspective. All the Java code may be embedded within the HTML, so changes are confined to a very limited area, reducing complexity drastically.

The big trade-off here is in the level of sophistication. As the scale of the system grows, some limitations begin to surface, such as bloating of business logic code in the page instead of factoring forward to a mediating Servlet or factoring back to a worker bean. It is a fact that utilizing a Servlet and helper beans helps to separate developer roles more cleanly and improves the potential for code reuse.

Page-view with bean - This pattern is used when the above architecture becomes too cluttered with business-related code and data access code. The Java code representing the business logic and simple data storage implementation in the previous model moves from the JSP to the JavaBean worker. This refactoring leaves a much cleaner JSP with limited Java code, which can be comfortably owned by an individual in a web-production role, since it encapsulates mostly markup tags.

### **The Dispatcher Approach**

In this approach, a Servlet or JSP acts as a mediator or controller, delegating requests to JSP pages and JavaBeans. There are three different architectures. They are mediator-view, mediator-composite view and service to workers.

In an N-tier application, the server side of the architecture is broken up into multiple tiers. In this case, the application is composed of multiple tiers, where the middle tier, the JSP, interacts with the back end resources via another object or EJBs component. The Enterprise JavaBeans server and the EJB provide managed access to resources, support transactions and access to underlying security mechanisms, thus addressing the resource sharing and performance issues of the 2-tier approach.

The first step in N-tiered application design should be identifying the correct objects and their interaction and the second step is identifying the JSPs or Servlets. These are divided into two categories.

Front end JSPs or Servlets manage application flow and business logic evaluation. They act as a point to intercept the HTTP requests coming from the users. They provide a single entry point to an application, simplifying security management and making application state easier to maintain.

Presentation JSPs or Servlets generate HTML or XML with their main purpose in life being presentation of dynamic content. They contain only presentation and rendering logic.

These categories resemble to the Modal-View design pattern, where the front-end components is the model and the presentation component the view. In this approach, JSPs are used to generate the presentation layer and either JSPs or Servlets to perform process-intensive tasks. The front-end component acts as the controller and is in charge of the request processing and the creation of any beans or objects used by the presentation JSP, as well as deciding, depending on the user's actions, which JSP to forward this request to. There is no processing logic within the presentation JSP itself and it simply responsible for retrieving any objects or beans that may have been previously created by the Servlet and extracting the dynamic content for insertion within static templates.

### **Benefits of JSP**

One of the main reasons why the JavaServer Pages technology has evolved into what it is today and it is still evolving is the overwhelming technical need to simplify application design by separating dynamic content from static template display data. Another benefit of utilizing JSP is that it allows to more cleanly separate the roles of web application/HTML designer from a software developer. The JSP technology is blessed with a number of exciting benefits, which are chronicled as follows:

1. The JSP technology is platform independent, in its dynamic web pages, its web servers, and its underlying server components. That is, JSP pages perform perfectly without any hassle on any platform, run on any web server, and web-enabled application server. The JSP pages can be accessed from any web server.
2. The JSP technology emphasizes the use of reusable components. These components can be combined or manipulated towards developing more purposeful components and page design. This definitely reduces development time apart from the At development time, JSPs are very different from Servlets, however, they are precompiled into Servlets at run time and executed by a JSP engine which is installed on a Web-enabled application server such as BEA WebLogic and IBM WebSphere.

### **Conclusion**

JSP and Servlets are gaining rapid acceptance as means to provide dynamic content on the Internet. With full access to the Java platform, running from the server in a secure manner, the application possibilities are almost limitless. When JSPs are used with Enterprise JavaBeans technology, e-commerce and

database resources can be further enhanced to meet an enterprise's needs for web applications providing secure transactions in an open platform. J2EE technology as a whole makes it easy to develop, deploy and use web server applications instead of mingling with other technologies such as CGI and ASP. There are many tools for facilitating quick web software development and to easily convert existing server-side technologies to JSP and Servlets.

Many application server vendors are aggressively deploying JSP within their products. This results in developing robust e-commerce applications as JSP provides XML functionality and scalability. By providing a clear separation between content and coding, JSP solves many problems attached with existing server-side applications.

### **JSP ARCHITECTURE**

JSP pages are high level extension of servlet and it enable the developers to embed java code in html pages. JSP files are finally compiled into a servlet by the JSP engine. Compiled servlet is used by the engine to serve the requests.

*javax.servlet.jsp* package defines two interfaces:

*JSPPage*  
*HttpJspPage*

These interfaces defines the three methods for the compiled JSP page. These methods are:

*jspInit()*  
*jspDestroy()*  
*\_jspService(HttpServletRequest request, HttpServletResponse response)*

In the compiled JSP file these methods are present. Programmer can define *jspInit()* and *jspDestroy()* methods, but the *\_jspService(HttpServletRequest request, HttpServletResponse response)* method is generated by the JSP engine.

### **Introduction To JSP Tags**

In this lesson we will learn about the various tags available in JSP with suitable examples. In JSP tags can be divided into 4 different types. These are:

#### **1. Directives**

In the directives we can import packages, define error handling pages or the session information of the JSP page.

#### **2. Declarations**

This tag is used for defining the functions and variables to be used in the JSP.

#### **3. Scriptlets**

In this tag we can insert any amount of valid java code and these codes are placed in *\_jspService* method by the JSP engine.

#### **4. Expressions**

We can use this tag to output any data on the generated page. These data are automatically converted to string and printed on the output stream.

Now we will examine each tags in details with examples. **DIRECTIVES**

Syntax of JSP directives is:

<%@directive attribute="value" %>

---

Where directive may be:

1. page: page is used to provide the information about it.  
Example: <%@page language="java" %>
2. include: include is used to include a file in the JSP page.  
Example: <%@ include file="/header.jsp" %>
3. taglib: taglib is used to use the custom tags in the JSP pages (custom tags allows us to define our own tags).  
Example: <%@ taglib uri="tlds/taglib.tld" prefix="mytag" %>

and attribute may be:

1. language="java"  
This tells the server that the page is using the java language. Current JSP specification supports only java language.  
Example: <%@page language="java" %>
2. extends="mypackage.myclass"  
This attribute is used when we want to extend any class. We can use comma(,) to import more than one packages.  
Example: <%@page language="java" import="java.sql.\* , mypackage.myclass" %>
3. session="true"  
When this value is true session data is available to the JSP page otherwise not. By default this value is true.  
Example: <%@page language="java" session="true" %>
4. errorPage="error.jsp"  
errorPage is used to handle the un-handled exceptions in the page.  
Example: <%@page language="java" session="true" errorPage="error.jsp" %>
5. contentType="text/html; charset=ISO-8859-1"  
Use this attribute to set the mime type and character set of the JSP.  
Example: <%@page language="java" session="true" contentType="text/html; charset=ISO-8859-1" %>

## JSP Actions

### What is JSP Actions?

Servlet container provides many built in functionality to ease the development of the applications. Programmers can use these functions in JSP applications. The JSP Actions tags enables the programmer to use these functions. The JSP Actions are XML tags that can be used in the JSP page.

### Here is the list of JSP Actions:

#### **jsp:include**

The **jsp:include** action work as a subroutine, the Java servlet temporarily passes the request and response to the specified JSP/Servlet. Control is then returned back to the current JSP page.

#### **jsp:param**

The **jsp:param** action is used to add the specific parameter to current request. The **jsp:param** tag can be used inside a **jsp:include**, **jsp:forward** or **jsp:params** block.

#### **jsp:forward**

The **jsp:forward** tag is used to hand off the request and response to another JSP or servlet. In this case the request never return to the calling JSP page.

#### **jsp:plugin**

In older versions of Netscape Navigator and Internet Explorer; different tags is used to embed

applet. The **jsp:plugin** tag actually generates the appropriate HTML code to embed the Applets correctly.

#### **jsp:fallback**

The **jsp:fallback** tag is used to specify the message to be shown on the browser if applets is not supported by browser.

Example:

```
<jsp:fallback>
    <p>Unable to load applet</p>
</jsp:fallback>
```

#### **jsp:getProperty**

The **jsp:getProperty** is used to get specified property from the JavaBean object.

#### **jsp:setProperty**

The **jsp:setProperty** tag is used to set a property in the JavaBean object.

#### **jsp:useBean**

The **jsp:useBean** tag is used to instantiate an object of Java Bean or it can re-use existing java bean object.

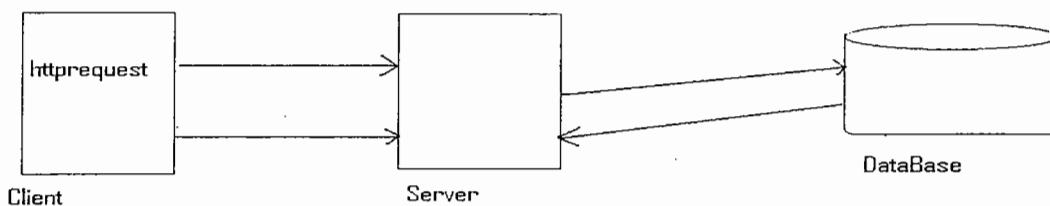
### **Java Server Pages - Introduction**

JSP Stands for JavaServerPages. We can develop a web application dynamic Input screens and dynamic output screens by using JSP. The current Version of JSP is JSP2.2. JSP pages are automatically compiled by the Server, such as Tomcat and Weblogic. Web pages created by using JSP are portable and can be used easily across multiple platforms and WebServers without making any changes. One of the most important benefit of JSP is the separation of business logic from the presentation logic. JSP page can be accessed directly as a simple HTML page.

JSP is actually a powerful scripting language (interpreted language) executed on the server side (like CGI, PHP, ASP, ...) and not on the client side (unlike scripts written in JavaScript or Java applets which run in the browser of the user connected to a site).

JSPs are integrated in a web page in HTML using special tags which will notify the Web server that the code included within these tags are to be interpreted. The result (HTML codes) will be returned to the client browser.

Java Server Pages are part of a **3-tier architecture**: where a server supporting the Java Server Pages (generally referred to as **application server**) will act as a mediator between the client browser and a database (generally referred to as **data server**). JSP provides the necessary elements for the connection to the database management system and allow the manipulation of data through SQL.



### **How Java Server Pages works?**

A page using Java Server Pages is executed during the query, by a JSP engine (generally running with a Web server or an application server). The JSP model is derived from the one used for Java servlets (JSP are indeed a way to write servlets). It is a Java class derived from Servlet class, making use of using `doxxx()` to return an HTTP response.

When a user calls a JSP page, the server calls the JSP engine which creates a Java source code from the JSP script and compile the class to provide a compiled file (with the `.class` extension).

**Note that:** the JSP engine checks if the date of the `.jsp` file corresponds to the `.class` file. The JSP engine will convert and compile the class, only if the JSP script has been updated. Thus, the fact that the compilation only takes place when the JSP script is updated, makes JSP, one of the fastest technologies to create dynamic pages.

**Characteristics of Java Server Pages**

JSPs can be used to create servlets, by including specific tags in the JSP code. In this way, they provide a fast technology to create dynamic pages.

In addition, JSP has all the characteristics of Java:

JSPs are multithreaded.

JSPs are portable.

JSPs are object-oriented.

JSPs are secure.

**Difference between JSP and Servlet**

Servlets	JSP
1. Best suitable for processing logic	1. Best suitable for presentation logic
2. we cannot separate business and presentation logic	2. Separation of presentation and business logic is possible
3. Servlet developer should have strong knowledge in Java	3.JSP author is not required to have strong knowledge in Java
4. For source code change, we have to perform explicitly compilation	4. For source code changes, it is not required to perform explicit compilation
5. Relatively development time is more	5. Relatively development time is less

**JSP Life Cycle**

Every jsp page ends with .jsp extension.

Whenever we give a request to server for a .jsp file then

1<sup>st</sup> .jsp will be converted into .java file

2<sup>nd</sup> .java file is converted into .class file

3<sup>rd</sup> .class is loaded by container and managed by container.

.jsp to .java and .java to .class conversion is performed by JSP engine.

.class file loading and execution is performed by container(servlet container)

JSP page lifecycle contains the following phases:

phase1:Translation phase(.jsp to .java)

phase2:Compilation stage(.java to .class)

phase3:Loading phase

phase4:instantiation phase

phase5:initialization phase

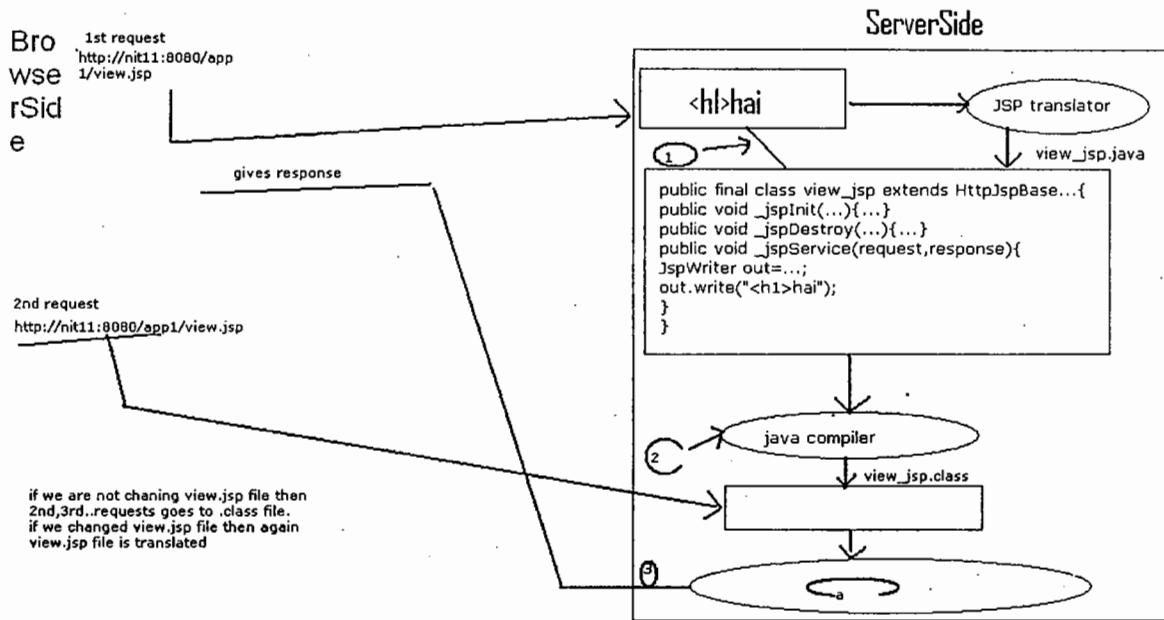
phase6:Request processing and response generation phase

phase7:Destruction phase

**phase1,phase2** are performed by JSP engine.

**phase3 to phase7** are performed by container.





In the above diagram

a---Æ web container loads and execute jsp life cycle methods.

1-ÆJSP translation phase

2-Æcompilation phase

3--Ærequest processing and response generation phase

#### Lifecycle of JSP

The lifecycle of JSP is controlled by three methods which are automatically called, when a JSP is requested and when the JSP terminates normally.

These are:

**jspInit () , \_jspService() , jspDestroy().**

**jspInit()** method is similar to the init() method in a Java Servlet and in applet.

It is called first when the JSP is requested and is used to initialize objects and variables that are used throughout the life of the JSP.

**\_jspService()** method is automatically called and retrieves a connection to HTTP.

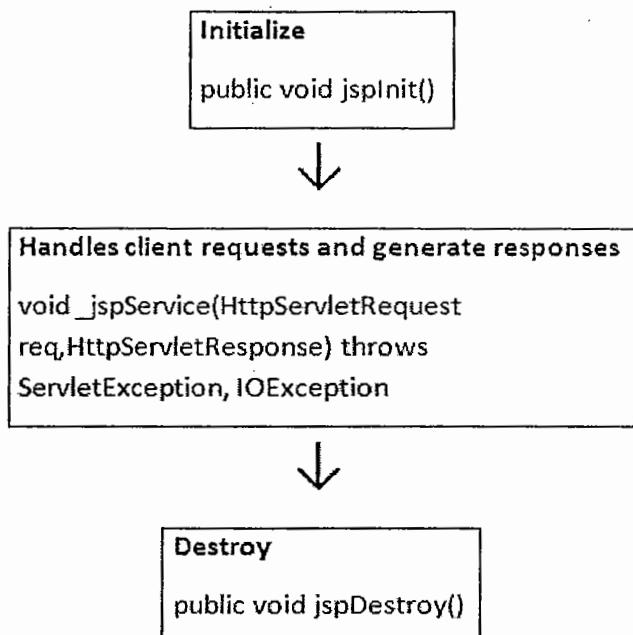
It will call doGet or doPost() method of servlet created.

**jspDestroy()** method is similar to the destroy() method in Servlet.

The destroy() method is automatically called when the JSP terminates normally.

It isn't called by JSP when it terminates abruptly.

It is used for cleanup where resources used during the execution of the JSP are released, such as disconnecting from database.

**LabExercise**

1. JSP embeds in ..... in .....

- a. Servlet, HTML
- b. HTML, Java
- c. HTML, Servlet
- d. Java, HTML

**Answer:D**

2. A JSP is translated into a :

- a. Java applet
- b. Java servlet
- c. Either 1 or 2 above
- d. Neither 1 nor 2 above

**Answer:B**

3. After translation of a JSP source page into its implementation class, The jsp implementation class is \_\_\_\_\_ ?

- a. final
- b. static
- c. abstract
- d. private

**Answer is : A**

**Explanations :** After translation JSP page looks like :

```
public final class test_jsp extends org.apache.jasper.runtime.HttpJspBase  
implements org.apache.jasper.runtime.JspSourceDependent {  
...  
}
```

**4. What is the output of the below test.jsp ?**

```
//test.jsp
<%!
public void _jspService(HttpServletRequest request, HttpServletResponse
response)

throws java.io.IOException, ServletException {
out.println("Hello");
}

%>
```

- a. Hello
- b. Compile Error - \_jspService(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletResponse) is already defined in org.apache.jsp.test\_jsp
- c. Runtime exception
- d. None of the above

**Answer is : D**

**Explanations :** After translation JSP page \_jspService automatically created by JSP compiler. In the JSP page if you define method name \_jspService(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletResponse) then compiler will complain \_jspService(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletResponse) is already defined in org.apache.jsp.test\_jsp.

**5. Which of the following JSP life cycle methods we should not override ?**

- A) jspInit()
- B )\_jspService
- C )jspDestroy
- D )All of the above.

**Answer:B** We cannot override the \_jspService method which is called from the generated servlet's service method.

**6. Which of the following is not a standard method called as part of the JSP life cycle?**

- A.jspInit() B.jspService()
- C.\_jspService() D.jspDestroy()

**Answer (B):** The standard service method for JSP has an \_ in its name

**7. How many copies of a JSP page can be in memory at a time?**

- |                 |                     |
|-----------------|---------------------|
| <u>A.</u> One   | <u>B.</u> Two       |
| <u>C.</u> Three | <u>D.</u> Unlimited |

**Answer:A****8. How does Tomcat execute a JSP?**

- A. As a CGI script
- B. As an independent process
- C. By one of Tomcat's threads
- D. None of the above is correct.

**Answer:A****9: How can we pre compile a JSP ?**

- A ) Invoke JSP by appending query string '?jsp\_precompile'
- B ) Invoke JSP by appending query string '?jsp-precompile=true'
- C ) Invoke JSP after appending query string '?precompile=true'
- D ) We cannot precompile a JSP page. We need to wait till the first request come, to compile JSP.

**Answer:A**

The JSP specification suggests a way to pre compile the JSP by appending the query string '?jsp\_precompile' without sending the actual request. But it is not mandatory to implement this feature. It is vendor dependent. The vendor can decide whether he should compile the JSP when he gets a call with the query string '?jsp\_precompile'

### **Scripting tags in JSP**

JSP scripting elements enable you to insert Java code into the servlet that will be generated from the current JSP page. There are three forms:

- o Expressions of the form <%= expression%> that are evaluated and inserted into output,
- o Scriptlets of the form <% code %> that are inserted into the servlets service method, and
- o Declarations of the form <%! code %> that are inserted into the body of the servlet class, outside of any existing methods.

### **Expression**

The Expression element contains a Java expression that returns a value. This value is then written to the HTML page. The Expression tag can contain any expression that is valid according to the Java Language Specification. This includes variables, method calls then return values or any object that contains a `toString()` method.

### **Syntax**

```
<%= Java expression %>
```

The Java expression is evaluated, converted to a string, and inserted in the page. This evaluation is performed at run-time (when the page is requested), and thus has full access to information about the request. For example, the following shows the client host name and the date/time that the page was requested:

```
<html>
...
<body>
Your hostname : <%=request.getRemoteHost()%><br>
Current time : <%=new java.util.Date()%>
...
</body>
</html>
```

Finally, note that XML authors can use an alternative syntax for JSP expressions:

```
<jsp:expression>
Java Expression
</jsp:expression>
```

Remember that XML elements, unlike HTML ones, are case sensitive. So be sure to use lowercase.

### **Scriptlet**

The scriptlet can contain any number of language statements, variable or method declarations, or expressions that are valid in the page scripting language.

Within a scriptlet, you can do any of the following:

- Declare variables or methods to use later in the JSP page.
- Write expressions valid in the page scripting language.
- Use any of the implicit objects or any object declared with a `<jsp:useBean>` element.
- Write any other statement valid in the scripting language used in the JSP page.

Any text, HTML tags, or JSP elements you write must be outside the scriptlet. For example,

```
<HTML>
<BODY>
<%
// This scriptlet declares and initializes "date"
java.util.Date date = new java.util.Date();
%>
Hello! The time is now
<%
out.println( date );
out.println( "<BR>Your machine's address is " );
out.println( request.getRemoteHost() );
%>
```

```
</BODY>
</HTML>
```

Scriptlets are executed at request time, when the JSP container processes the request. If the scriptlet produces output, the output is stored in the out object.

If you want to use the characters "%>" inside a scriptlet, enter "%\>" instead. Finally, note that the XML equivalent of <% Code %> is

```
<jsp:scriptlet>
Code
</jsp:scriptlet>
```

### Declaration

A declaration can consist of either method declarations or variables, static constants are a good example of what to put in a declaration.

The JSP you write turns into a class definition. All the scriptlets you write are placed inside a single method of this class. You can also add variable and method declarations to this class. You can then use these variables and methods from your scriptlets and expressions.

You can use declarations to declare one or more variables and methods at the class level of the compiled servlet. The fact that they are declared at class level rather than in the body of the page is significant. The class members (variables and methods) can then be used by Java code in the rest of the page.

### Syntax

```
<%! declaration; [ declaration;]+...%>
```

When you write a declaration in a JSP page, remember these rules:

You must end the declaration with a semicolon (the same rule as for a Scriptlet, but the opposite of an Expression).

```
<% int i = 0;%>
```

You can already use variables or methods that are declared in packages imported by the page directive, without declaring them in a declaration element.

You can declare any number of variables or methods within one declaration element, as long as you end each declaration with a semicolon. The declaration must be valid in the Java programming language.

```
<% int i = 0; long l = 5L; %>
```

For example,

```
<%@ page import="java.util.*" %>
<HTML>
<BODY>
<%!
    Date getDate()
    {
        System.out.println( "In getDate() method" );
        return new Date();
    }
%>
```

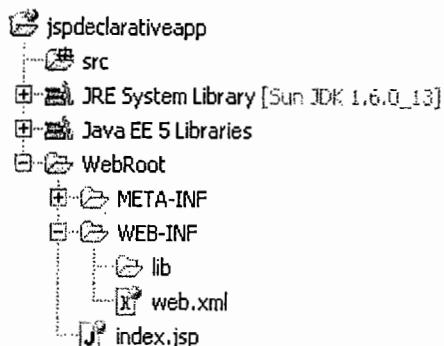
```
Hello! The time is now <%=getDate() %>
```

```
</BODY>
</HTML>
```

A declaration has translation unit scope, so it is valid in the JSP page and any of its static include files. A static include file becomes a part of the source of the JSP page and if any file included with an include directive or a static resource included with a <jsp:include> element. The scope of a declaration does not include dynamic resources included with <jsp:include>.

- As with scriptlets, if you want to use the characters "%>", enter "%\>" instead. Finally, note that the XML equivalent of <%! Code %> is

```
<jsp:declaration>
Code
</jsp:declaration>
```

**Jspdeclaration Example****index.jsp**

```

<%!
//static data member declaration
static int count=0;
//method definition
String greet(String name){
return "Good Evening"+name;
}
%>
<%
count++;
out.println("<h1>" + greet("NITStudent") + "<br>" + "NUMBER OF HITS=" + count);
%>
  
```

**web.xml**

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
<welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
</web-app>
  
```

**LAB Exercise**

If you want to override a JSP file's initialization method, within what type of tags must you declare the method?

- A) <@ @>                      B) <%@ %>  
 C) <% %>                      D) <%! %>

**Answer (D):** Declarations are placed within a set of <%! %> tags.

**What will be the output of the following JSP page?**

```

<html><body>
<% a = 100; %>
<% int a = 200; %>
<%! int a = 300; %>
a = <%= a %>, <%= this.a %>
</body></html>
  
```

- A) a = 200, 100                B) a = 300, 100  
 C) a = 100, 200                D) a = 200, 300  
 E) The code will not compile as written

**Ans:A**

The JSP page will be translated into servlet code similar to the following:

```
public class ...._jsp extends HttpJspBase {
int a = 300;
public void _jspService(HttpServletRequest request, HttpServletResponse response)
throws java.io.IOException, ServletException
{
    out.write("<html>");
    out.write("<body>");

    a = 100;
    int a = 200;

    out.write("a = ");
    out.print( a);
    out.write(", ");
    out.print( this.a);

    out.write("</body>");
    out.write("</html>");
}
}
```

The `<%! int a = 300; %>` declaration will create a class-level instance variable "a", and initialize the variable to 300. The scriplet `<% a = 100; %>` will change the value of the class-level instance variable from 300 to 100.

The second scriplet, `<% int a = 200 %>`, will create a local variable "a" in the `_jspService ()` method and set the initial value to be 200. The first expression `<%= a %>` refers to the local variable "a", which is 200.

The second expression `<%= this.a %>` uses the keyword `this` and refers to the class-level instance variable "a". The class-level instance variable "a" was set to 100 by the scriplet `<% a = 100; %>`. Hence, the correct answer is "a=200, 100"

#### What will be the output of the following JSP code?

```
<html><body>
<% int a = 10; %>
<%! int a = 20; %>
<%! int b = 30; %>
The value of b multiplied by a is <%= b * a %>
</body> </html>
```

- A) The code will not compile
- B) The value of b multiplied by a is 30
- C) The value of b multiplied by a is 300
- D) The value of b multiplied by a is 600
- E) The value of b multiplied by a is 0

**Ans:C**

Although the variable "a" is declared twice, the code should still compile as written. In the first declaration `<% int a = 10; %>`, the variable "a" will be declared as a local variable. In the second declaration `<%! int a = 20; %>`, the variable "a" will be declared as an instance variable, global to the translated servlet class.

Upon translation, the JSP engine will translate the code similar to the following:

```
public class ..._jsp
{
int a = 20;
int b = 30;
public void _jspService (....)
{
int a = 10;
out.write ("The value of b multiplied by a is ");
out.print (b * a);
}
```

Since the local variable "a" has precedence over the global variable "a", the expression `(b * a)` evaluates as `(30 * 10)`, which equals 300.

**Which of the following is a valid JSP declaration ?**

- A) <%= "Hello" %>
- B) <%! int x=10 %>
- C) <%! int x=10; %>
- D) <% int x=10; %>

**Answer:C**

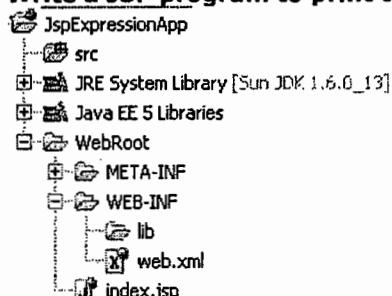
A JSP declaration always starts with a !. Choice A is invalid because it is JSP expression. not a declaration. Choice D is incorrect because it is a valid JSP scriptlet. and B is invalid due to lack of semi colon

**Which of the following is a valid JSP comment?**

- A) <!-- commented part --!>
- B) <comment>commented part</comment>
- C) <--- commented part --->
- D) <%!-- commented part --!>

**Answer :**D is an example for valid JSP comment. Choice A stands for HTML comment.

**Write a JSP program to print the Current time on the browser**



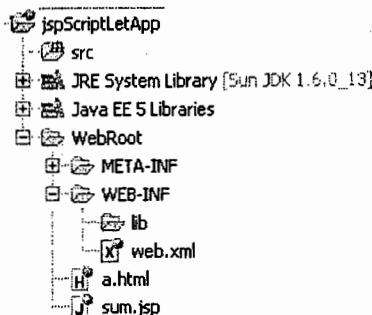
**web.xml**

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
index.jsp
<h1>HAI OUR SERVER TIME IS:</h1>
<%=new java.util.Date()%></h1>

```

**Write a JSP program to print sum of two numbers**



**web.xml**

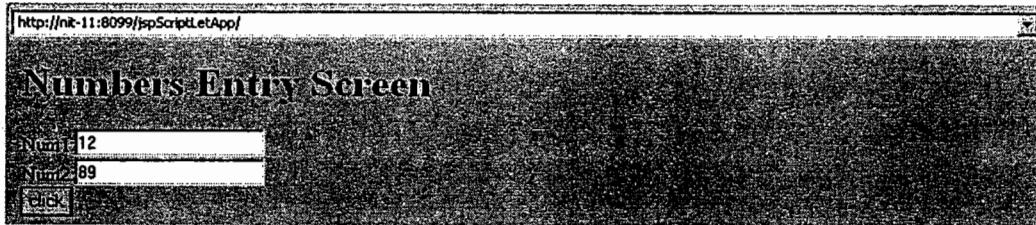
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
<welcome-file-list>
  <welcome-file>a.html</welcome-file>
</welcome-file-list>
</web-app>
```

**a.html**

```
<html>
<h1>Numbers Entry Screen</h1>
<body bgcolor="lightgreen">
<form action="sum.jsp" method="post">
Num1:<input type="text" name="param1"/><br/>
Num2:<input type="text" name="param2"/><br/>
<input type="submit" value="click"/><br/>
</form>
</body>
</html>
```

**sum.jsp**

```
<%String a=request.getParameter("param1");
String b=request.getParameter("param2");
int x=Integer.parseInt(a);
int y=Integer.parseInt(b);
int z=x+y;
out.println("Sum is:"+z);
%>
```

**Implicit Objects in JSP**

Implicit objects are a set of Java objects that the JSP Container makes available to developers in each page. These objects may be accessed as built-in variables via scripting elements and can also be accessed programmatically by JavaBeans and Servlets. You may already meet one of these objects already - the out object which allows you to send information to the output stream.

The implicit objects are created automatically for you within the service method. Furthermore, as summarized below, each object must adhere to a specific Java class or interface definition.

There are nine JSP implicit Objects:

These objects are available to \_jspService so we can use them inside scriptlet and expression tags. But we cannot use them inside declaration tags.

**Nine implicit objects are pointed by the following references:**

The implicit variables are only available within the jspService method and thus are not available within any declarations.

JSP supports nine automatically defined variables, which are also called implicit objects. These variables are:

Objects	Description
request	This is the <b>HttpServletRequest</b> object associated with the request.

response	This is the <b>HttpServletResponse</b> object associated with the response to the client.
out	This is the <b>PrintWriter</b> object used to send output to the client.
session	This is the <b>HttpSession</b> object associated with the request.
application	This is the <b>ServletContext</b> object associated with application context.
config	This is the <b>ServletConfig</b> object associated with the page.
pageContext	This encapsulates use of server-specific features like higher performance <b>JspWriters</b> .
page	This is simply a synonym for <b>this</b> , and is used to call the methods defined by the translated servlet class.
exception	The <b>Exception</b> object allows the exception data to be accessed by designated JSP.

**1) about page reference:-**the page reference points current jsp page object.

```
public final view_jsp extends HttpBase...{  
public void _jspService(request,response)...{  
Object page=this;  
}  
}
```

**2)about out reference:-**

The out reference points **JspWriter** subclass object.

By using the object we can print HTML tags and plain text data on the browser.

**3)about request:-**

The request reference points **HttpServletRequest** implementation object.

By using this reference jsp page can get request parameters and can manage request scope attributes.

**index.html**

```
<form action="add.jsp">  
<h1>num1:<input type="text" name="param1"/>  
<br>num2: <input type="text" name="param2"/>  
<br><input type="submit" value="add"/>  
add.jsp  
<%  
//how to take a,b values  
String s1=request.getParameter("param1");  
String s2=request.getParameter("param2");  
int x=Integer.parseInt(s1);  
int y=Integer.parseInt(s2);  
int z=x+y;  
out.print("<h1>sum="+z);  
%>
```

**4)about response:-**

The reference points **HttpServletResponse** implementation object.

This object is by default available to scriptlets and expressions to get response related information and to set response related information.

**example:**

if a jsp page wants to redirect request from this server to another server then it can use response object and **sendRedirect** method call.

**view.jsp**

```
<%  
response.sendRedirect("http://www.oracle.com");  
%>
```

**5)the session reference:-**

The session reference points **HttpSession** implementation object.

By default this reference is available to scriptlet,expression tags.

These tags can access session related info and they can manage session scope attributes.

By using this reference these tags can delete session object by calling **session.invalidate()** and this reference these tags can set session timeout

**Example:**

```
<%session.setAttribute("a",99);  
session.setMaxInactiveInterval(20);  
%>
```

**6)the config reference**

The config reference points **ServletConfig** implementation object.

By using this config reference a jsp page can:

- 1)get initialization parameters of jsp configuration
- 2)get initialization parameters names

3) get servlet name(jsp file configuration)

4) get servlet context reference.

Example define the following view.jsp file in WEB-INF directory.

#### **view.jsp**

```
<%>
String s=config.getServletName(); //jspfile1
String p1=config.getInitParameter("paramname1"); //NARESHIT
String p2=config.getInitParameter("paramname2"); // studentone
out.print("<h1>ServletName="" + s);
out.print("<h1> param1value=" + p1);
out.print("<h1> param2value = " + p2);
%>
```

#### **web.xml**

```
<web-app>
<servlet>
<servlet-name>jspfile1</servlet-name>
<jsp-file>/WEB-INF/view.jsp</jsp-file>
<init-param>
<param-name>paramname1</param-name>
<param-value>NARESHIT</param-value>
</init-param>
<init-param>
<param-name>paramname2</param-name>
<param-value>studentone</param-value>
</init-param>
</servlet>
<servlet-mapping>
<servlet-name>jspfile1</servlet-name>
<url-pattern>/s1</url-pattern>
</servlet-mapping>
</web-app>
```

#### **7) the application reference:-**

The application reference points ServletContext implementation object.

By using this reference a jsp file can access application related information.

such as:

managing application scope attributes  
getting application init parameters...etc

#### **web.xml**

```
<web-app>
<context-param>
<param-name>paramname</param-name>
<param-value>20</param-value>
</context-param>
</web-app>
```

#### **view.jsp**

```
<%>
String s=application.getInitParameter("paramname");
out.write("<h1>a=" + s); //20
%>
```

#### **8) Exception reference:-**

This reference points any exception object raised in the .jsp page and provides that exception object to error jsp page.

#### **9) pageContext reference:-**

The pageContext reference points PageContext class sub class object.

By using the pageContext reference we can get any other jsp implicit object

```
pageContext.getPage();
pageContext.getOut();
pageContext.getRequest();
pageContext.getResponse();
pageContext.getSession();
pageContext.getServletConfig();
pageContext.getServletContext();
pageContext.getException();
```

By using pageContext object we can manage any scope attributes.

### **Need of PageContext Object**

**Understand the following example:-**

**show.jsp**

```
<%!
public void greet(String name){
out.print("All the Best"+name);
}%
<%
greet("NitStudent");
%>
```

If we send request to show.jsp then we will get 500 status code error because out is not available to declaration tag methods. To make out available

To declaration tag method

**modified show.jsp**

```
<%!
void greet(String name,JspWriter out)throws java.io.IOException{
out.write("<h1>All the Best"+name);
}
%>
<%greet("NitStudent",out);%>
```

### **What we must do if we need all 9 objects to greet method?**

Ans: declare greet() with all nine object parameters

And call greet with all nine object references.

<%!

```
void greet(String name,JspWriter out,Object page,HttpServletRequest request,HttpServletResponse
response,...){
```

}

%>

```
<%greet("NitStudent",out,page,request,response,...);%>
```

This is not flexible.

### **What a is solution to this problem?**

Ans: instead of passing all nine objects refer pass only one reference i.e pageContext ref By using pageContext ref get all other implicit objects.

### **Scopes in JSP**

**1)pageScope**

**2)requestScope**

**3)sessionScope**

**4)applicationScope**

#### **1) pageScope:-**

pageScope is managed by pageContext object.

As pageContext object is created for every jsp page, so, every jsp page will have a specific page scope.

#### **2) requestScope:-**

requestScope is managed by request object.

If same request is shared by more than one jsp pages those sharing the same request object come under same request scope.

requestScope begins whenever request object is created by servlet container and request scope ends whenever request object is deleted by servlet container.

As request object is stored in pageContext object we can manage request scope attributes by using pageContext object.

#### **3) sessionScope:-**

Session scope is managed by session object.

Same session object is available to all jsp pages as long as session is not expired.

Whenever session is expired or session.invalidate() is called then new session is created for next requesting pages.

#### **4) applicationScope:-**

Application scope is maintained by application reference pointing object .

Application scope begins whenever ServletContext implementation object is created.

Application scope ends whenever ServletContext implementation object is deleted.

#### **LabExercise**

```
<%
    request.setAttribute("name","abc");
    session.setAttribute("name","xyz");
    application.setAttribute("name","pqr");
%>
```

**What will be the return value if we are calling <%= pageContext.findAttribute("name") %> on the same page?**

- A ) null
- B ) abc
- C ) xyz
- D ) pqr

**Answer:B**

The findAttribute will first look in the page scope for an attribute. If it finds one, it will return that value. If it is not able to find an attribute in page scope it will start looking at other scopes from most restrictive to least restricted scope. i.e, first request, then session and finally application. If it cannot find the attribute in any of the scope it will return null.

**Which of the following piece of code correctly set an attribute in application scope ?**

- A. We cannot set attributes to application scope using pageContext.
- B. <% pageContext.setAttribute("name", "albin", PageContext.APPLICATION\_SCOPE); %>
- C. <% pageContext.setAttribute("name","albin", PageContext.APPLICATION\_SCOPE); %>
- D. <% pageContext.setAttribute("name","albin",PageContext.CONTEXT\_SCOPE); %>

**Answer:C**

The three arguments version of setAttribute method is used to set an attribute in other scopes using pageContext.

**Which of the following is legal JSP syntax to print the value of i. Select one correct answer**

- A. <%int i = 1;%>  
<%= i; %>
- B. <%int i = 1;  
i; %>
- C. <%int i = 1%>  
<%= i %>
- D. <%int i = 1;%>  
<%= i %>
- E. <%int i = 1%>  
<%= i; %>

**Answer:D**

When using scriptlets (that is code included within <% %>), the included code must have legal Java syntax. So the first statement must end with a semi-colon. The second statement on the other hand is a JSP expression. So it must not end with a semi colon.

**A JSP page called test.jsp is passed a parameter name in the URL using http://localhost/test.jsp?name="Nit". The test.jsp contains the following code.**

```
<%! String myName=request.getParameter();%>
<% String test= "welcome" + myName; %>
<%= test%>
```

- A. The program prints "Welcome Nit"

- B. The program gives a syntax error because of the statement  
`<%! String myName=request.getParameter();%>`
- C. The program gives a syntax error because of the statement  
`<% String test= "welcome" + myName; %>`
- D. The program gives a syntax error because of the statement  
`<%= test%>`

**Answer** B. JSP declarations do not have access to automatically defined variables like request, response etc

Which of the following correctly represents the following JSP statement. Select one correct answer.

`<%=x%>`

- A. `<jsp:expression=x/>`
- B. `<jsp:expression>x</jsp:expression>`
- C. `<jsp:statement>x</jsp:statement>`
- D. `<jsp:declaration>x</jsp:declaration>`
- E. `<jsp:scriptlet>x</jsp:scriptlet>`

**Answer:** B. The XML syntax for JSP expression is `<jsp:expression>Java expression</jsp:expression>`

**Which of the following correctly represents the following JSP statement. Select one correct answer.**

`<%x=1;%>`

- A. `<jsp:expression x=1;/>`
- B. `<jsp:expression>x=1;</jsp:expression>`
- C. `<jsp:statement>x=1;</jsp:statement>`
- D. `<jsp:declaration>x=1;</jsp:declaration>`
- E. `<jsp:scriptlet>x=1;</jsp:scriptlet>`

**Answer:** E. The XML syntax for JSP scriptlets is `<jsp:scriptlet>Java code</jsp:scriptlet>`

**What gets printed when the following JSP code is invoked in a browser. Select one correct answer.**

```
<%= if(Math.random() < 0.5) %>
hello
<%= } else { %>
hi
<%= } %>
```

- A. The browser will print either hello or hi based upon the return value of random.
- B. The string hello will always get printed.
- C. The string hi will always get printed.
- D. The JSP file will not compile.

**Answer:** D. The if statement, else statement and closing parenthesis are JSP scriptlets and not JSP expressions. So these should be included within `<% } %>`

**Which of the following are correct. Select one correct answer.**

- A. JSP scriptlets and declarations result in code that is inserted inside the `_jspService` method.
- B. The JSP statement `<%! int x; %>` is equivalent to the statement `<jsp:scriptlet>int x;</jsp:scriptlet%>`.
- C. The following are some of the predefined variables that maybe used in JSP expression - `httpSession, context`.
- D. To use the character `%>` inside a scriptlet, you may use `%\>` instead.

**Answer:** D. JSP declarations are inserted outside of `_jspService` method. Hence a is incorrect. The JSP statement `<%!int a;%>` is equivalent to `<jsp:declaration>int x;</jsp:declaration>`. Hence b is incorrect. The predefined variables that are available within the JSP expression are session and `pageContext`, and not `httpSession` and `context`. Hence c is incorrect.

**What gets printed when the following is compiled. Select one correct answer.**

```
<% int y = 0; %>
<% int z = 0; %>

<% for(int x=0;x<3;x++) { %>
<% z++;++y;%>
<% }%>

<% if(z<y) {%
<%= z%>
```

```
<% } else {%
<%= z - 1%>
<% }%>
A. 0
B. 1
C. 2
D. 3
E. The program generates compilation error.
```

**Answer:** C. After the for loop z and y are both set to 3. The else statement gets evaluated, and 2 gets printed in the browser.

**Which of the following JSP variables are not available within a JSP expression. Select one correct answer.**

- A. out
- B. session
- C. request
- D. response
- E. httpsession
- F. page

**Answer: E. There is no such variable as httpsession.**

#### The Page Directive in JSP Page

the **pagedirective** of the JSP page which works for the entire JSP page. These directives apply different properties for the page like language support, page information and import etc. by using the different attributes of the directives. There are three types of directives which are as follows:

Page Directive

Include Directive

Taglib Directive

In this section, you will learn about the **page** directive and its attributes and explanation one-by-one. This is the directive of the JSP page which defines the properties for the entire JSP page by using its different attributes and set values of the attributes as per requirements.

Syntax of the declaration of the **page** directive with its attributes is **<%@ page attributeName="values" %>**. The space between the tag **<%@** and **%>** before the **page** (directive name) and after values of the last attribute, is optional, you can leave the space or not.

Following are name of the attributes of the **page** directive used in JSP:

- language**
- extends**
- import**
- session**
- buffer**
- autoFlush**
- isThreadSafe**
- info**
- errorPage**
- contentType**
- isErrorPage**
- pageEncoding**
- isELIgnored**

**language:** This is an attribute of the **page** directive of the JSP which is used for specifying some other scripting languages to be used in your JSP page but at this time, its value becomes **java** that is optional.

**extends:** This is an attribute of the **page** directive of the JSP which is used for specifying some other java classes to be used in your JSP page like **packagename.classname**. The fully qualified name of the superclass of the Java class will be accepted.

**import:** This attribute imports the java packages and its classes more and more. You can import more than one java package and class by separating with comma (,). You can set the name of the class with the package name directly like **packagename.classname** or import all classes of the package by using **packagename.\***.

**session:** This attribute sets a boolean value either **true** or **false**. If the value of session attribute is true then the session object refers to the current or a new session because the client must be in the HTTP session for running the JSP page on the server. If you set the value of session object **false** then you can

not use the session object or <jsp:useBean> element with scope="session" in JSP page. And then if a type of error occurs i.e. called the translation-time error. The default value of session attribute is *true*.

**buffer:** This attribute sets the buffer size in kilobytes i.e. used by the out object to handle output generated by the JSP page on the client web browser. If you specify the buffer size then the output will be buffered with at least 8kb because the default and minimum value of the buffer attribute is 8kb.

**autoFlush:** This attribute of the **page** directive supports for flushing buffer automatically when the buffer is full. The value of the **autoFlush** attribute is either *true* or *false*. If you will specify the *true* value then the buffer will be flushed otherwise it will generate a raised exception if you set the *false* value. You cannot set the *false* value if the buffer size is none.

**isThreadSafe:** This attribute supports the facility of maintaining thread for sending multiple and concurrent requests from the JSP container to the JSP page if you specify the *true* value of the attribute otherwise if you specify the *false* value of the attribute then the JSP container can send only one request at one time. The default value of the attribute is *true*.

**info:** This attribute simply sets the information of the JSP page which is retrieved later by using **Servlet.getServletInfo()** method. The value of the attribute will be a text string.

**errorPage:** This attribute sets a url (relative path starting from the "/" which refers from the root directory of your JSP application). If any exception is generated the the attribute refers to the file which is mentioned in the given url. If you do not specify the url then the attribute refers to the current page of your JSP application if any exception is generated.

**isErrorPage:** This attribute sets the boolean value either *true* or *false*. You can use the exception object in the JSP page if you set the *true* value of the attribute otherwise you cannot use the exception object because the default value of the attribute is *false*.

**contentType:** This attribute specifies the MIME type and the character encoding i.e. used for the JSP response. The default MIME type is "text/html" and the default character set is "ISO-8859-1". You can also specify other.

**pageEncoding:** This attribute specifies the language that the page uses when the page is sent to the browser. This attribute works like the meta tag of the HTML markup language.

**isELIgnored:** This is a boolean attribute that specifies either *true* or *false* value. If you set the attribute value is *true* then any type of the EL expressions will be ignored in the JSP page.

#### Code Description:

In the following program, <% and %> JSP tags. Java codes are written in between the both tag.

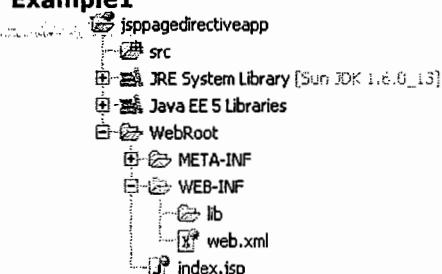
#### Syntax:

<%@ page [attribute="value" attribute="value".....] %>

The table given below describes the possible attributes for the page directive.

Attribute	Description	Syntax (default value is in bold)	Example
<b>language</b>	This tells the server about the language to be used in the JSP file. Presently the only valid value for this attribute is java.	language="java"	<%@page language="java" %>
<b>Import</b>	This attribute defines the list of packages, each separated by comma .	import="package.class"	<%@page import="java.util.* , java.io.*"%>
<b>Extends</b>	Indicates the superclass of servlet when jsp translated in extends="package.class" servlet.	extends="com.Connect"	<%@page extends="com.Connect"%>
<b>Session</b>	true indicates session should be bound to the existing session if one exists, otherwise a new session should be created and bound to it and false indicates that no sessions will be used.	session="true false"	<%@page session="true"%>
<b>Buffer</b>	specifies the buffer size for out and default is 8kb.	buffer="sizekb none"	<%@ page buffer="8kb"%>

<b>isThreadSafe</b>	True indicates multithreading and false indicates that the servlet should implementSingleThreadModel	<code>isThreadSafe="true false"</code>	<code>&lt;%@ page isThreadSafe="true" %&gt;</code>
<b>autoFlush</b>	true indicates that the buffer should be flushed when it is full and false indicates that an exception should be thrown when the buffer overflows.	<code>autoFlush="true false"</code>	<code>&lt;%@ page autoFlush="true" %&gt;</code>
<b>Info</b>	This defines a string that can be retrieved via the getServletInfo method.	<code>info="message"</code>	<code>&lt;%@ page info="This is a simple jsp file created by Java2all team on 22 Feb 2011"%&gt;</code>
<b>pageEncoding</b>	This defines data type of page encoding.	<code>pageEncoding="ISO-8859-1"</code>	<code>&lt;%@ page pageEncoding="ISO-8859-1"%&gt;</code>
<b>contentType</b>	This specifies the MIME type of the output.	<code>contentType="MIME-Type"</code>	<code>&lt;%@ page contentType="text/html; charset=ISO-8859-1"%&gt;</code>
<b>isELIgnored</b>	This defines ENUM data type.	<code>isELIgnored=" true false "</code>	<code>&lt;%@ page isELIgnored="false"%&gt;</code>
<b>isErrorPage</b>	This indicates whether or not the current page can act as the error page.	<code>isErrorPage="true false"</code>	<code>&lt;%@ page isErrorPage="false"%&gt;</code>
<b>errorPage</b>	Define a URL to another JSP that is invoked if an unchecked runtime exception is thrown.	<code>errorPage="url"</code>	<code>&lt;%@ page errorPage="error.jsp"%&gt;</code>

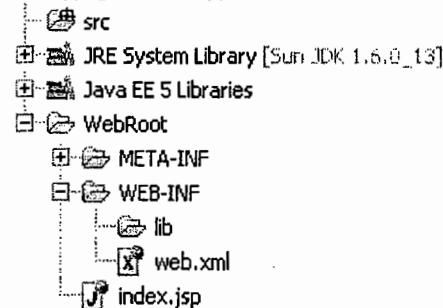
**Example1****index.jsp**

```
<%@ page import="java.util.*" %>
<HTML>
<BODY bgcolor="orange">
<%
    System.out.println( "Evaluating date now" );
    Date date = new Date();
%>
Hai! The time is now <%= date %>
</BODY>
</HTML>
web.xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
<welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
```

```
</welcome-file-list>
</web-app>
```

**Example2**

jsppagedirectiveapp

**web.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

**index.jsp**

```
<%@page language="java" %>
<html>
    <head><title>Hello World JSP Page.</title></head>
    <body>
        <font size="10">
        <%
            String name="Student";
            out.println("Hai " + name + "!");
        %>
        </font>
    </body>
</html>
```

**Include Directive**

The **include** directive is used to include a file during the translation phase. This directive tells the container to merge the content of other external files with the current JSP during the translation phase. You may code *include* directives anywhere in your JSP page.

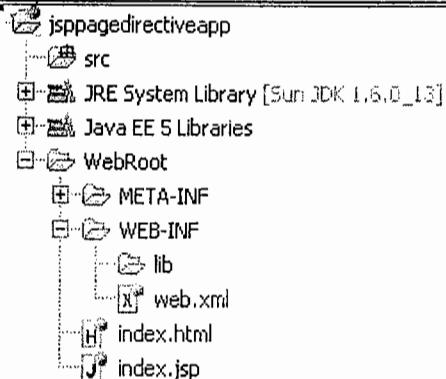
The general usage form of this directive is as follows:

```
<%@ include file="relative url" >
```

The filename in the include directive is actually a relative URL. If you just specify a filename with no associated path, the JSP compiler assumes that the file is in the same directory as your JSP.

You can write XML equivalent of the above syntax as follows:

```
<jsp:directive.include file="relative url"/>
```

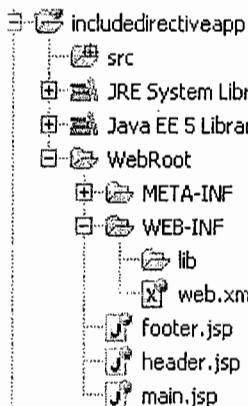
**index.html**

```

<html>
  <head>
    <title>index.html</title>

  </head>

  <body bgcolor="lightgreen">
    <font color="red" size="12"><marquee>Welcome to NareshIT</marquee></font>
  </body>
</html>
index.jsp
<%@include file="index.html" %>
<body>
<font color="orange" size="12"><marquee>Register Here...</marquee></font>
</body>
  
```

**header.jsp**

```

<%!
int pageCount = 0;
void addCount() {
    pageCount++;
}
%>
<% addCount(); %>
<html>
<head>
<title>The include Directive Example</title>
</head>
<body>
<center>
<h2>The include Directive Example</h2>
<p>This site has been visited <%= pageCount %> times.</p>
</center>
<br/><br/>
footer.jsp
  
```

```
<html>
<head></head>
<br/><br/>
<body>
<center>
<p>Copyright © 2010</p>
</center>
</body>
</html>
```

**main.jsp**

```
<%@ include file="header.jsp" %>
<body bgcolor="lightgreen">
<center>
<p>Thanks for visiting my page.</p>
</center>
</body>
<%@ include file="footer.jsp" %>
```

**web.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
<welcome-file-list>
    <welcome-file>main.jsp</welcome-file>
</welcome-file-list>
</web-app>
```

<b>static include</b>	<b>dynamic include</b>
<p>1)we can do static include by Using include directive</p> <p>2)in static include both pages belongs to same page scope and same request scope</p> <p>3)in static include both pages are not translated into .java files</p> <p>4)static include is translation time include(happens at translation phase)</p> <p>5)static include is physically including target jsp into source jsp and works only for jsp to jsp include</p>	<p>1)we can do dynamic include by using request dispatcher object</p> <p>2)in dynamic include both pages belongs to same request scope but not same page scope</p> <p>3)in dynamic include both pages are translated into .java files</p> <p>4)dynamic include is request processing time include(i.e happens at request and response generation phase)</p> <p>5) dynamic include is not physically including target jsp into source jsp and works for request dispatching from jsp to jsp and jsp to servlet also</p>

**Taglib directive****Introduction**

The JavaServer Pages API allows you to define custom JSP tags that look like HTML or XML tags and a tag library is a set of user-defined tags that implement custom behavior.

The taglib directive declares that your JSP page uses a set of custom tags, identifies the location of the library, and provides a means for identifying the custom tags in your JSP page.

**Syntax**

```
<%@ taglib uri="uri" prefix="prefixOfTag" >
```

**Attributes:**

Taglib directive contains two attributes

1. **prefix** : prefix attribute informs a container what bits of markup are custom actions.

2. **uri** : uri represents the location of tag libraries.

**Example:**

```
<%@ taglib uri="http://www.example.com/custlib" prefix="mytag" %>
<html>
<body>
<mytag:hello/>
</body>
</html>
```

**Lab Exercise****How can we create a thread safe JSP ?**

- A ) <%@ page isThreadSafe = "true" %>
- B ) <%@ page implements="SingleThreadModel" %>
- C ) <%@ page isThreadSafe="false" %>
- D ) <%@ page extends="SingleThreadModel" %>

**Answer: C**

The `isThreadSafe` attribute of `page directive` defines whether the generated servlet needs to implement `SingleThreadModel`. The default value for `isThreadSafe` attribute is true and if false, the generated servlet will implement `SingleThreadModel`

**How to set the MIME type of generated servlet to 'image/gif' ?**

- A ) We cannot set JSP's content type to image/gif. The content type of JSP is always text/html only.
- B ) <%@ page contentType="image/gif" %>
- C ) <%@ page content-Type="image/gif" %>
- D ) <%@ page setMimeType="image/gif" %>

**Answer:C**

The `contentType` attribute of `page directive` sets the MIME type for JSP response.

**What is the default value of session Attribute in JSP ?**

- A)<%@ page session="true" %>
- B)<%@ page session="false" %>
- C)<%@ page session="yes" %>,
- D)<%@ page session="no" %>

Correct answer is : A

**Explanations :** <%@ page session="true" %> is the default value.

Questions no--2

**A JSP page does not contain page attribute and trying to access exception implicit variable.**

- A)exception implicit available not available in the JSP page.
- B)exception implicit available is available in the JSP page.
- C)No relation between isErrorPage attribute and exception implicit variable.
- D)None of the above

**Correct answer is : A**

Explanations : exception implicit variable not available in all JSP pages. Need to add page attribute to the JSP to make exception implicit available in the JSP page.

**Which code snippet correctly declares the current JSP page to be an error page?**

- A )<%@ page isErrorPage="true" %>
- B )<%@ page isErrorPage="true" %>
- C )<%@ errorPage="true" %>
- D )All JSP pages are error pages only, we don't need to declare it.

**Answer:B**

The isErrorPage attribute of page directive is used to specify the current JSP page to be an error page. If isErrorPage is false, then the implicit object exception will not be available in this file.

**Which code correctly sets an error page for a JSP?**

- A )<%@ page isErrorPage="true" %>
- B )<%@ page errorPage="mypage.jsp" %>
- C )<%@ page isErrorPage="myerrorpage.jsp" %>
- D )<%@ page isErrorPage="false" %>

**Answer B:**

The errorPage attribute of page directive is used to specify the error page for a JSP page. The page specified in the errorPage attribute must have isErrorPage value true.

```
1 >>>>>>>>>> JSP Tags>>>>>>>>>>>>>>>>>>
2 //JSP
3 Implicit Objects
4 =====
5 request ---> javax.servlet.http.HttpServletRequest(I) ---->request scope
6 response -->javax.servlet.http.HttpServletResponse(I)---->response scope
7 out --> javax.servlet.jsp.JspWriter (AC) --->page scope
8 session --> javax.servlet.http.HttpSession(I)----> session scope
9 page ---> java.lang.Object(C) --> this(current JES class obj reference )---->page scope
10 pageContext --->javax.servlet.jsp.PageContext (AC)--->page scope
11 application --->javax.servlet.ServletContext(I)---->web application scope
12 config --->javax.servlet.ServletConfig(I)---->page scope
13 exception---> java.lang.Throwable(C)--->page scope
14
15
16 Tags
17 ====
18
19 Scripting Tags
20 =====
21
22 1. Scriptlet
23
24 standard syn:
25 -----
26 <%
27     ----- java code
28     ---
29 %>
30
31 xml syn
32 -----
33 <jsp:scriptlet>
34     -----
35         ----- javacode
36     -----
37 </jsp:scriptlet>
38
39
40 ex1:
41     <% int a=10;
42         int b=20;
43         int c=a+b;
44         out.println("sum is"+c); %>
45
46 // This java code of scriptlet goes as it is to _jspService(req,res) of jsp equivalent servlet
47
48 ex2:
49     <jsp:scriptlet>
50         java.util.Date d=new java.util.Date();
51             String s=d.toString();
52             out.println("date is"+s);
53     </jsp:scriptlet>
54
55 note1: if u declare variables in scriptlet they come as local variables of _jspService(req,res)
56
57 ex3:(invalid code)
58 <% public int abc()
59     {
60         -----
```

```
61      -----
62      } %>
63
64 note2:do not place method definitions in scriptlet as java does not support nested methods
65
66 ex4:
67 <jsp:scriptlet>
68   <![CDATA[
69
70     int a=10;
71     int b=20;
72
73     if(a<b)
74       out.println(a+" less than "+b);
75     else
76       out.println(a+" greater than "+b);  ]]>
77
78
79 </jsp:scriptlet>
80
81
82 2. Declaration
83 standard syn
84 -----
85 <%!
86   Decl, of Instance variables,
87   definitions of user defined Methods,
88   definitions of jspInit() & jspDestroy() Convience Life cycle methods
89 %>
90
91 xml syn
92 -----
93 <jsp:declaration>
94 -----
95 -----
96 </jsp:declaration>
97
98
99 ex1:
100   <%! int a=10; %> --> "a" comes as instance variable in jsp equivalent servlet
101
102 ex2:
103   <%! public int sum(int x,int y)
104   {
105     return x+y;
106   } %>
107
108 --->
109 <jsp:declaration>
110   public int sum(int x,int y)
111   {
112     return x+y;
113   }
114 </jsp:declaration>
115
116 ex3:
117   <%! public void jspInit(){
118     // Initialization code
119   } %>
120
```

```
121      <%! public void jspDestory(){
122          //unInitialization code
123      } %>
124 ex4:
125 <jsp:declaration>int xyz=10; </jsp:declaration>
126
127
128 3. Expression
129 standard syn
130 -----
131 <%= <java expression>  %>
132 xml syn
133 -----
134 <jsp:expression>
135     <java expression>
136 </jsp:expression>
137
138
139 ex1:
140     <%=a+b%> -->evaluates a+b expression and writes result to browser
141
142     <%=a+b,c+d %> This is invalid code(becoz two expressions are there)
143     //An expression tag can evaluate only one expression at a time
144 ex2:
145     <%=a %>--->writes "a" variable value to browser
146
147 ex3: <%=sum(10,20) %>-->calls method and writes result to browser
148
149 ex4: <%=new java.util.Date() %>---->creates object writes system date to browser
150
151     <%=Integer.parseInt("30") %>
152 ex5:
153     <jsp:expression>new java.util.Date()</jsp:expression>
154
155
156 JSP Comments
157 =====
158
159 <%--
160
161 --%>
162
163 Directives
164 =====
165
166 <%@directive_name attributes%>
167
168 1. Page Directive
169
170 useful to provide gobal information to jsp page.
171
172 standard syntax
173 =====
174 <%@page attributes%>
175
176 xml syntax
177 =====
178 <jsp:directive.page attributes/>
179
180 attributes
```

```
181 =====
182 language="java" default -->java// java is the default and only one
183           //language that u can pass here as a value
184 import="java.util.*,java.net.*,..." // default :java.lang.*
185
186 extends=" class name " //not recomended to use // no default value
187
188 contentType=" text/html " // resp content Type //default value: text/html
189
190 buffer="18kb" or "none" //default-->8kb
191
192 autoFlush="true" (false) //default -->true
193
194 session="true" (false) //default-->true
195
196 errorPage=" url " //no default
197 isErrorPage="false" (true) //default--> false
198
199 isThreadSafe="true"(false) //default--> true
200 info=" string " //the short desc about jsp page--//no default value
201 isELIgnored="true" (false) //default -->false
202 .
203 .
204 .
205
206 ex1
207 <%@page isThreadSafe="false" %>
208 makes jsp equivalent servlet as Thredsafe servlet by implementing
209           javax.servlet.SingleThreadModel (i).
210
211 ex2: To Specify Buffer size of Jsp Page /prg
212   <%@page buffer="10kb"%>
213
214 ex3:
215 <%@page buffer="none" autoFlush="true"%>
216
217 the above stmt is wrong stmt ,when there is no buffer ,
218 there is no possibility of flushing that buffer.
219
220
221 ex4:
222   <%@page session="false" %>
223 Implicit obj session will not be created
224
225   <%@page session="true" %>
226 implicit obj session will be created
227
228 ex4.
229 <%@page import="java.net.*" import="java.util.*" session="false" session="true" %> (invalid)
230
231 except import attribute , other attributes of page directive should not be repeated having different
values.
232 so the above stmt generates error.
233
234
235
236 ex5.
237 <%@page info="first jsp" message="hello " %>
238 the above code generates page compiler error becoz "message" is an invalid attribute.
239
```

```
240
241 ex6:
242 <%@page session="false" %>
243 implicit obj session will not be created in jsp & jsp equivalent servlet.
244
245 note: attribute names and tag names are case sensitive in jsp.
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275 >>>>>Info about attributes of Page Directive >>>>>>>>>>
276 =====
=====
277 The isThreadSafe attribute controls whether or not the servlet that results
278 from the JSP page will implement the singleThreadModel interface. Use of
279 the isThreadSafe attribute takes one of the following two forms:
280 <%@ page isThreadsafe="true" %> <%!-- Default --%>
281 <%@ page isThreadsafe="false" %>
282 =====
=====
283 The session attribute controls whether or not the page participates in
284 HTTP session. Use of this attribute takes one of the following two forms:
285 <%@ page session="true" %> <%!-- Default --%>
286 <%@ page session="false" %>
287 A value of the true (the default) indicates that the predefined variable session
288 (of type HttpSession) should be bound to the existing Session if one
289 exists; otherwise, a new session should be created and bound to session.
290 A value of false means that no session will be used automatically and
291 attempts to access the variable session will result in errors at the time the
292 JSP page is translated into a servlet.
293 =====
=====
294 The errorPage attribute specifies a JSP page that should process any exceptions
295 .(i.e., something of type Throwable) thrown but not caught in the current page.
296 It is used as follows:
```

```
297 <%@ page errorpage="Relative URL" %>
298 The exception thrown will be automatically available to the designated error page
299 by means of the exception variable.
300 =====
301 =====
301 The isErrorPage attribute indicates whether or not the current page can act
302 as the error page for (another JSP page). Use of isErrorPage takes one of the
303 following two forms:
304 <%@ page isErrorPage="true" %>
305 <%@ page isErrorPage="false" %> <%!-- Default --%>
306 =====
306 =====
307 Use of the contentType attribute takes one of the following two forms:
308 <%@ page contentType="MIME-Type" %>
309 <%@ page contentType="MIME-Type; charset=Character-Set" %>
310 For example, the directive
311 <%@ page contentType="text/plain" %>
312 has the same effect as the scriptlet
313 <% response.setContentType("text/plain"); %>
314 Unlike regular servlets, where the default MIME type is text/plain, the
315 default for (JSP pages is text/html (with a default character set of
316 ISO-8859-1).
317 =====
317 =====
318 The import attribute of the page directive lets you specify the package that
319 should be imported by the servlet into which the JSP page gets translated.
320 If you don't explicitly specify any classes to import, the servlet imports
321 java.lang.*. javax.servlet.*. javax.servlet.jsp.*. javax.servlet.http.*. and
322 possibly some number of server-specific entries. Never
323 write JSP code that relies on any server-specific classes being imported automatically.
324 Use of the import attribute takes one of the following two forms:
325 <%@page import="package.class" %>
326 <%@ page import="package.class1,...,package.classN" %>
327 For example, the following directive signifies that all classes in the
328 java.util package should be available to use without explicit package identifiers.
329 <%@ page import="java.util.*" %>
330
331 The import attribute is the only page attribute that is allowed to appear
332 multiple times within the same document, it is traditional to place import statements
333 either near the top of the document or just before the first place that the referenced
334 package is used.
335 =====
335 =====
336 The buffer Attribute
337 <%@ page buffer="sizekb" %>
338 <%@ page buffer="none" %>
339 Servers can use a larger buffer than you specify, but not a smaller one.
340 For example, <%@ page buffer=32kb" %> means the document
341 content should be buffered and not sent to the client until at least 32
342 kilobytes have been accumulated or the page is completed.
343 =====
343 =====
344 The autoflush /attribute
345 <%@ page autoflush="true" %<%!-- Default --%>
346 <%@ page autoflush="false" %>
347 A value of false is illegal when buffer="none" is also used.
348 =====
348 =====
349 The extends Attribute
350 <%@ page extends="package.class" %>
```

```
351 =====
352 =====
352 The info Attribute
353 <%@ page info="Some Message" %>
354
355
356 2. Include Directive
357
358 standard syn
359 -----
360 <%@include file="dest url "%>
361
362 xml syn
363 -----
364 <jsp:directive.include file="dest url "/>
365
366 --> This tag includes the code of dest prg to the code of source
367     jsp prg's equivalent servlet(JES).
368
369
370
371 3. Taglib Directive
372
373 <%@taglib uri="" prefix=""%>
374
375 decl namespace in
376 <jsp:root xmlns:prefix="uri">
377
378 </jsp:root>
379
380
381 Action Tags
382 =====
383 <jsp:useBean>
384 syn :<jsp:useBean attributes/>
385 //used for creating /locating an object of javabean class.
386 attributes
387 =====
388 id=" "//instance name(Object name)
389
390 class=" " //a fully qualified java bean class name
391
392 scope= "page|session|request|application" (default scope is "page")
393         //specifies the scope of the bean class obj
394
395             page (default)
396             ===
397             bean class obj is available within the current jsp page
398
399             request
400             ====
401             bean class obj is available through the request
402
403             session
404             ====
405             bean class obj is available within a session
406
407             application
408             =====
409             bean class obj is available for all pages within the context(web application)
```

```
410
411
412 D. beanName="logical name of java bean "
413 E. type="reference class type of java bean class obj "
414
415 ex1:
416 <jsp:useBean id="st" class="p1.StudentBean" scope="session" />
417
418 the above stmt creates "st" obj for p1.StudentBean class
419 and keeps that obj in session scope.
420
421 the above stmt creates obj like this:
422     p1.StudentBean st=new p1.StudentBean();
423     "st" object type is p1.StudentBean
424     "st" reference type is p1.StudentBean
425 ex2:
426 ====
427 <jsp:useBean id="st" type="ABC" class="p1.StudentBean" scope="request"/>
428     Bean class obj creation statement
429 -----
430     ABC st=new p1.StudentBean();
431     "st"--> reference type is ABC class.
432     "st"--> object type is p1.StudentBean class
433     --> here p1.StudentBean class extends from ABC class
434
435 note: <jsp:useBean> internally uses pageContext attributes
436      to keep bean class objs in specified scopes or
437      to locate them from specified scopes
438
439 2. <jsp:setProperty>
440 //calls setXxx(-) methods on JavaBean class obj
441     to set given values to bean properties
442
443 syn:
444 <jsp:setProperty attributes/>
445
446 attributes
447 =====
448 property=" "//bean property name (xxx part of setXxx(-))
449 name=" " //((bean class obj name)id attribute value given in the <jsp:useBean >
450 value=" " // value to be set for bean property
451 param=" " //input(request) parameter name
452 Note:
453 value or param --> any one attribute has to be used at a time.
454
455 ex1:
456 <jsp:setProperty name="st" property="sno" value="567"/>
457
458 // This internally calls setSno() method on "st" obj and
459     assigns value "567" to the bean property "sno".
460
461 ex2:
462 <jsp:setProperty name="st" property="sname" param="stname"/>
463
464 //This internally calls setSname(-) method on "st" obj
465 to assign the value of "stname" req param to "sname" bean proeprty
466 -----
467
468 3. <jsp:getProperty>
469 //calls getXxx() methods to read values from bean properties
```

```
470 syn:  
471 <jsp:getProperty attributes/>  
472  
473 attributes  
474 =====  
475 name=" " // (bean obj name) "id" attr value given in <jsp:useBean>  
476 property="bean property name"  
477  
478 ex:  
479 <jsp:getProperty name="st" property="sno" />  
480  
481 reads and displays "sno" bean property value by calling st.getSno()  
482 method .  
483  
484 4. Include  
485 =====  
486 <jsp:include page="relative url" flush="false/true"/>  
487  
488 5. Forward  
489 =====  
490 <jsp:forward page="relative url"/>  
491  
492 6.Param  
493 =====  
494  
495 <jsp:param name=" " value=" " />  
496  
497 //within forward,include and params  
498  
499 7.Params  
500 =====  
501 <jsp:params>  
502   <jsp:param name="" value="" />  
503 </jsp:params>  
504  
505 //used within jsp:plugin  
506  
507 8.Plugin  
508 =====  
509 <jsp:plugin attributes>  
510 <jsp:params></jsp:params>  
511 </jsp:plugin>  
512
```

## JAVASERVER PAGES (JSP) SYNTAX

Element	Description	JSP 1.1 BETA
	<b>Legend</b>	
HTML Comment	Creates a comment that is sent to the client in the viewable page source.	<!-- comment -->
Hidden Comment	Documents the JSP file, but is not sent to the client.	<%-- comment --%>
Declaration	Declares variables or methods valid in the page scripting language.	<%! declaration; %>
Expression	Contains an expression valid in the page scripting language.	<%= expression %>
Scriptlet	Contains a code fragment valid in the page scripting language.	<% code fragment of one or more lines %>
Include Directive	Includes a static file, parsing the file@S JSP elements.	<%@ include file="relativeURL" %>
Page Directive	Defines attributes that apply to an entire JSP page.	<%@ page language="java" [ extends="package.class" ] [ imports="package.class" ] [ package="packageName" ] [ session="true false" ] [ buffer="none 8kb sizeKb" ] [ autoFlush="true false" ] [ isThreadSafe="true false" ] [ info="text" ] [ errorPage="relativeURL" ] [ contentType="mimeType" ] [ charset="characterSet" ] [ isErrorPage="true false" ] %>
Taglib Directive	Defines a tag library and prefix for the custom tags used in the JSP page.	<%@ taglib uri="URItoTagLibrary" prefix="tagPrefix" %> custom tag: <tagPrefix :name attribute="value "+... /> <tagPrefix :name attribute="value "+... > other tags </tagPrefix :name >



## JAVASERVER PAGES (JSP) SYNTAX

Element	Description	Code Examples
<jsp:forward>	Forwards a client request to an HTML file, JSP file, or servlet for processing.	<pre>&lt;jsp:forward page="<i>relativeURL</i>"   &lt;%= expression %&gt;  &gt;   [ &lt;jsp:param name="<i>parameterName</i>" value="<i>parameterValue</i>"   &lt;%= expression %&gt; ]+&gt; &lt;/jsp:forward&gt;</pre> <p>Gets the value of a Bean property so that you can display it in a result page.</p>
<jsp:getProperty>	Includes a static file or sends a request to a dynamic file.	<pre>&lt;jsp:include page="<i>relativeURL</i>"   &lt;%= expression %&gt;" flush="true"&gt;   [ &lt;jsp:param name="<i>parameterName</i>" value="<i>parameterValue</i>"   &lt;%= expression %&gt; ]+&gt; &lt;/jsp:include&gt;</pre>
<jsp:plugin>	Downloads Plug-in software to the Web browser to execute an applet or Bean.	<pre>&lt;jsp:plugin type="bean:applet" code="<i>className</i>" codebase="<i>codebase</i>" classid="&lt;i&gt;classId&lt;/i&gt;" name="&lt;i&gt;instanceName&lt;/i&gt;" archive="&lt;i&gt;URLArchive&lt;/i&gt;" align="&lt;i&gt;align&lt;/i&gt;" bottom="&lt;i&gt;bottom&lt;/i&gt;" left="&lt;i&gt;left&lt;/i&gt;" top="&lt;i&gt;top&lt;/i&gt;" right="&lt;i&gt;right&lt;/i&gt;"&gt;   [ height="&lt;i&gt;displayPixels&lt;/i&gt;"   width="&lt;i&gt;displayPixels&lt;/i&gt;" ] [ vspace="&lt;i&gt;topBottomPixels&lt;/i&gt;"   leftright="&lt;i&gt;leftRightPixels&lt;/i&gt;" ] [ reversionNumber="&lt;i&gt;REVersionNumber&lt;/i&gt;" ] [ pluginurl="&lt;i&gt;URLToPlugin&lt;/i&gt;" URLtoPlugin="&lt;i&gt;URLToPlugin&lt;/i&gt;" ]   [ &lt;jsp:params&gt; [ &lt;jsp:param name="<i>paramName</i>" value="<i>paramValue</i>"   &lt;%= expression %&gt; ]+&gt;   [ &lt;jsp:params&gt; ] [ &lt;jsp:fallback&gt; text message for user ]&gt; &lt;/jsp:plugin&gt;</pre>
<jsp:setProperty>	Sets a property value or values in a Bean.	<pre>&lt;jsp:setProperty name="<i>beanInstanceName</i>" property="<i>propertyName</i>" property="&lt;i&gt;paramName&lt;/i&gt;" paramName="&lt;i&gt;paramName&lt;/i&gt;" value="<i>string</i>"   &lt;%= expression %&gt;  &gt;   [ &lt;jsp:bean id="&lt;i&gt;beanInstanceName&lt;/i&gt;" scope="&lt;i&gt;scope&lt;/i&gt;" &gt;     [ &lt;jsp:property class="&lt;i&gt;package.class&lt;/i&gt;" type="&lt;i&gt;package.class&lt;/i&gt;" beanName="&lt;i&gt;beanName&lt;/i&gt;"   &lt;jsp:putBean&gt; ]+&gt;     [ &lt;jsp:elements&gt; other elements ]&gt;   ]&gt;</pre> <p>Specifies or instantiates a Bean with a specific name and scope.</p>
<jsp:useBean>	Type	<pre>&lt;jsp:useBean type="&lt;i&gt;type&lt;/i&gt;" name="&lt;i&gt;name&lt;/i&gt;" scope="&lt;i&gt;scope&lt;/i&gt;"/&gt;</pre> <p>Some JSP methods (see class of interface for others)</p>
Implicit Objects		
request	Subclass of <i>javax.servlet.ServletRequest</i>	GetAttribute, GetParameter, GetParameterNames, GetParameterValues
response	Subclass of <i>javax.servlet.ServletResponse</i>	Not typically used by JSP page authors
pageContext	<i>javax.servlet.jsp.PageContext</i>	findAttribute, getAttribute, getAttributesScope, getAttributeNamesInScope
session	<i>javax.servlet.http.HttpSession</i>	getid, getValue, getValueNames, putValue
application	<i>java: servlet.ServletContext</i>	getMimeType, getPath
out	<i>java: servlet.jsp.JspWriter</i>	clear, clearBuffer, flush, getBufferSize, getRemaining
config	<i>java: servlet.ServletConfig</i>	getInitParameter, getInitParameterNames
page	<i>java: lang.Object</i>	getMessage, getLocalizedMessage, printStackTrace, toString
exception	<i>java: lang.Throwable</i>	

```
1 =====
2 App1(example using scripting tags)
3 =====
4 -----First.jsp-----
5 <!-- this is HTML comment -->
6 <%-- this is Jsp comments --%>
7 <html>
8   <head>
9     <title>
10    My First JSP
11      </title>
12    </head>
13    <%! int x,y;%>
14    <%! int mul(int a,int b)
15    {
16      int z;
17      z = a*b;
18      return z;
19      //int abc;
20    } %>
21
22
23  <body>
24    <center><h2>WELCOME TO JAVA SERVER PAGES1</h2>
25    Today's date is<%=new java.util.Date()%>
26    <% Class.forName("java.lang.System");           %>
27
28    <br>
29    <br>
30
31    <%x=70;y=69;%>Multiplication of <%=x%>&nbsp;&nbsp;
32    and&nbsp;&nbsp;<%=y%>&nbsp;&nbsp;is&nbsp;<%=mul(x,y)%>
33
34    </center>
35  </body>
36 </html>
37 =====
38 App2(html--->jsp--->db s/w communication)
39 =====
40 -----Third.html-----
41 <!--HTML Page to send input data to Fifth.jsp-->
42 <HTML>
43   <HEAD>
44     <TITLE>
45       Student Registration Form
46     </TITLE>
47   </HEAD>
48   <form action="Third.jsp" method="post">
49
50 <BODY>
51
52   <CENTER><H2><U>Input student details and press SUBMIT button</u></h2>
53   <table align="center" width"100%">
54     <tr>
55       <td align="center">Enter Student Number</td>
56       <td><input type=text name=stno size=10 maxlength=5></td>
57     </tr>
58
59     <tr>
60       <td align="center">Enter Student Name</td>
```

```
61          <td><input type=text name=stname size=20 maxlength=15></td>
62      </tr>
63
64      <tr>
65          <td align="center">Enter Result</td>
66          <td><input type=text name=stresult size=4 maxlength=6></td>
67      </tr>
68
69  </table>
70  <center><input type="submit" value="submit"></center>
71
72 </BODY>
73 </HTML>
74 -----Third.jsp-----
75 <%@page import="java.sql.*;"%>
76
77 <%
78 try
79 {
80     Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
81     Connection con=DriverManager.getConnection("jdbc:odbc:oradsn","scott","tiger");
82     Statement st=con.createStatement();
83 %>
84
85     <% int sno=Integer.parseInt(request.getParameter("stno"));
86     String sname=request.getParameter("stname");
87     String addr=request.getParameter("stresult");
88
89
90 %>
91
92     <%int res=st.executeUpdate
93         ("INSERT INTO STUDENT VALUES("+ sno +",    "+ sname+",""+ addr+")) ;%>
94     <html>
95         <head>
96             <title>
97                 My fifth jsp
98             </title>
99         </head>
100
101         <body>
102             <center><h3>
103
104                 <%
105                 if(res==0)
106                 {
107                     %
108                     Problem in inserting record
109                     <%
110                 }
111                 else
112                 {
113                     %
114                     Sucessfully Inserted
115                     <%
116                 }
117                     %
118             </body>
119         </html>
120             <%con.close();
```

```
121         }
122     catch(ClassNotFoundException cnf)
123     {}
124     catch(Exception e)
125     {}
126
127 %>
128 -----
129 =====
130 App3 (html----->jsp----->Db s/w communication)
131 -----Main.html-----
132
133 <frameset rows="30%,*>
134   <frame src="Search.html" name="f1">
135   <frame name="f2">
136 </frameset>
137
138 -----Search.html-----
139
140 <form action="Search.jsp" method="get" target="f2">
141   <b>sno</b><input type="text" name="tsno"><input type="submit" name="s1"
142   value="Search">
143   <br><br><br>
144   <a href="Search.jsp" target="f2">Get All Student Details </a>
145 -----Search.jsp-----
146 <%@page import="java.sql.*;" %>
147
148 <%! Connection con=null;
149   PreparedStatement ps1=null,ps2=null;
150   ResultSet rs1=null,rs2=null;
151
152   public void jspInit()
153   {
154     try
155     {
156       Class.forName("oracle.jdbc.driver.OracleDriver");
157       con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","s
158       cott","tiger");
159     }
160     catch(Exception e)
161     {
162       e.printStackTrace();
163     }
164   } %>
165
166
167 <%
168
169   String cap=request.getParameter("s1");
170
171   if(cap!=null) // if submit btn is clicked
172   {
173     // read form data
174     String tno=request.getParameter("tsno");
175     int no=Integer.parseInt(tno.trim());
176
177     ps1=con.prepareStatement("select * from student where sno=?");
178     ps1.setInt(1,no);
```

```
179         rs1=ps1.executeQuery(); %>
180         <table>
181             <tr><td>sno</td><td>sname</td><td>sadd</td></tr>
182             <%
183                 while(rs1.next())
184                 { %>
185                     <tr>
186                         <td><%=rs1.getInt(1)%></td>
187                         <td><%=rs1.getString(2)%></td>
188                         <td><%=rs1.getString(3)%></td>
189                     </tr>
190                     <%}// while %
191                 </table>
192             <%//if
193             else // if hyper link is clicked
194             {
195                 ps2=con.prepareStatement("select * from student");
196                 rs2=ps2.executeQuery(); %>
197                 <table>
198                     <tr><td>sno</td><td>sname</td><td>sadd</td></tr>
199                     <%
200                         while(rs2.next())
201                         { %>
202                             <tr>
203                                 <td><%=rs2.getInt(1)%></td>
204                                 <td><%=rs2.getString(2)%></td>
205                                 <td><%=rs2.getString(3)%></td>
206                             </tr>
207                             <%// while %
208                         </table>
209             <% } //else %
210
211         <%! public void jspDestroy()
212         {
213             try
214             {
215                 if(rs1!=null)
216                     rs1.close();
217             }
218             catch(Exception e){ }
219             try
220             {
221                 if(rs2!=null)
222                     rs2.close();
223             }
224             catch(Exception e){ }
225             try
226             {
227                 if(ps1!=null)
228                     ps1.close();
229             }
230             catch(Exception e){ }
231             try
232             {
233                 if(ps2!=null)
234                     ps2.close();
235             }
236             catch(Exception e){ }
237             try
238             {
```

```
239             if(con!=null)
240                 con.close();
241             }
242         catch(Exception e){ }
243     }//jspDestroy()
244     %>
245 -----web.xml-----
246 <!DOCTYPE web-app
247     PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
248     "http://java.sun.com/dtd/web-app_2_3.dtd">
249 <web-app>
250 -----
251 -----
252 App4)Html----->Jsp ----->DB s/w Communication
253 -----Input.html-----
254
255 <form action="dburl" method="get">
256     Number : <input type="text" name="tsno"><br>
257     Name : <input type="text" name="tsname"><br>
258     Address : <input type="text" name="tsadd"><br>
259     <input type="submit" value="register" name="s1"> <br><br>
260 </form>
261     <a href="dburl?s1=link">Get All Student Details </a>
262 -----web.xml-----
263 <web-app>
264     <servlet>
265         <servlet-name>abc</servlet-name>
266         <jsp-file>/DBJsp.jsp</jsp-file>
267             <init-param>
268                 <param-name>driver</param-name>
269                 <param-value>oracle.jdbc.driver.OracleDriver</param-value>
270             </init-param>
271             <init-param>
272                 <param-name>url</param-name>
273                 <param-value>jdbc:oracle:thin:@localhost:1521:orcl</param-value>
274             </init-param>
275             <init-param>
276                 <param-name>dbuser</param-name>
277                 <param-value>scott</param-value>
278             </init-param>
279             <init-param>
280                 <param-name>dbpwd</param-name>
281                 <param-value>tiger</param-value>
282             </init-param>
283             <load-on-startup>2</load-on-startup>
284     </servlet>
285
286     <servlet-mapping>
287         <servlet-name>abc</servlet-name>
288         <url-pattern>/dburl</url-pattern>
289     </servlet-mapping>
290 </web-app>
291 -----DBJsp.jsp-----
292 <%@page import="java.sql.*" %>
293
294 <%!
295     Connection con;
296     PreparedStatement ps1,ps2;
297     public void jspInit()
298     {
```

```
299     System.out.println("DBJsp: jspInit() method");
300     try{
301         //get Access to ServletConfig obj
302         ServletConfig cg=getServletConfig();
303         // read init param values
304         String s1=cg.getInitParameter("driver");
305         String s2=cg.getInitParameter("url");
306         String s3=cg.getInitParameter("dbuser");
307         String s4=cg.getInitParameter("dbpwd");
308         // create jdbc connection ,PreparedStatement objs
309         Class.forName(s1);
310         con=DriverManager.getConnection(s2,s3,s4);
311         ps1=con.prepareStatement("insert into student values(?, ?, ?)");
312         ps2=con.prepareStatement("select * from student");
313         }//try
314         catch(Exception e)
315         { e.printStackTrace(); }
316     }//jspInit()
317 %>
318
319 <%
320     System.out.println("DBJsp: from scriptlet ");
321     //read s1 req param value
322     String pval=request.getParameter("s1");
323     if(pval.equals("register")) // when submit btn is clicked
324     {
325         //read form data
326         int no=Integer.parseInt(request.getParameter("tsno"));
327         String name=request.getParameter("tsname");
328         String addrs=request.getParameter("tsadd");
329         //set form data to the params of insert query
330         ps1.setInt(1,no);
331         ps1.setString(2,name);
332         ps1.setString(3,addrs);
333         //execute the Query
334         int result=ps1.executeUpdate();
335         //process the results
336         if(result==0)
337         { %>
338             <b>Registration Failed</b>
339             <% }
340             else
341             { %>
342                 <b>Registration Success</b>
343             <% } //else
344         } //if
345         else //when hyperlink is clicked
346         {
347             //execute the query
348             ResultSet rs=ps2.executeQuery();
349             // get ResultSet MetaData obj
350             ResultSetMetaData rsmd=rs.getMetaData();
351             int cnt=rsmd.getColumnCount(); %>
352             <table>
353                 <tr>
354                     <% //print col names
355                     for(int i=1;i<=cnt;++i)
356                     { %>
357                         <td> <%=rsmd.getColumnLabel(i) %></td>
358                     <% } //for %>
```



```
359          </tr>
360      <%
361          //print col vlaue
362          while(rs.next())
363          { %>
364              <tr>
365                  <% for(int i=1;i<=cnt;++i)
366                  { %>
367                      <td><%=rs.getString(i) %></td>
368                  <% } //for
369                  //while
370                  rs.close();
371          } //else
372      %>
373  <%! public void jspDestroy()
374  {
375      System.out.println("DBJsp:jspDestroy()");
376      try
377      {
378          //close jdbc objs
379          try{
380              if(ps1!=null)
381                  ps1.close();
382          }catch(Exception e)
383          { e.printStackTrace(); }
384          try{
385              if(ps2!=null)
386                  ps2.close();
387          }catch(Exception e)
388          { e.printStackTrace(); }
389          try{
390              if(con!=null)
391                  con.close();
392          }catch(Exception e)
393          { e.printStackTrace(); }
394          try{
395              if(rs!=null)
396                  rs.close();
397          }catch(Exception e)
398          { e.printStackTrace(); }
399      } //try
400      catch(Exception e)
401      { e.printStackTrace(); }
402  } //jspDestroy()
403  %>
404 =====
405 -----DirectiveEx1.html-----
406 <form action="DirectiveEx1.jsp">
407 <pre>
408     Name : <input type="text" name="name"/>
409
410     age : <input type="text" name="value"/>
411
412     <input type="submit" name="s1" value="Add">
413     <input type="submit" name="s1" value="remove">
414     <input type="submit" name="s1" value="View"/>
415
416 </form>
417 -----DirectiveEx1.jsp-----
418 <%--
```

```
419  Page Directive
420  --%>
421
422  <%@page import="java.util.*"%>
423  <%!
424  Hashtable ht=new Hashtable();
425  %>
426
427  <%
428  String s=request.getParameter("s1");
429
430  if (s.equals("Add"))
431  {
432  ht.put(request.getParameter("name"),request.getParameter("value"));
433  %>
434
435  Value Added
436
437  <%
438  }
439  else if (s.equals("remove"))
440  {
441  ht.remove(request.getParameter("name"));
442  %>
443
444  Value Removed
445
446  <%
447  }
448  else
449  {
450  %>
451
452  <%@include file="ViewJsp.jsp"%>
453
454  <%
455  }
456
457  session.setAttribute("Ht",ht);
458
459  System.out.println(ht.toString());
460  %>
461  -----ViewJsp.jsp-----
462  <%@page import="java.util.*" errorPage="Error.jsp" session="true"%>
463
464  <html>
465  <body>
466  <table>
467  <tr>
468  <th>Name</th>
469  <th>Value</th>
470  </tr>
471
472  <%
473  Hashtable ht=(Hashtable)session.getAttribute("Ht");
474
475  Enumeration names=ht.keys();
476
477  while (names.hasMoreElements())
478  {
```

```
479 String name= (String) names.nextElement();
480
481 String value= (String) ht.get(name);
482 %>
483 <tr>
484   <td><%=name%></td>
485   <td><%=value%></td>
486 </tr>
487 <%
488 }//while
489 %>
490   </table>
491 </body>
492 </html>
493 -----Error.jsp-----
494 <%--
495 Error Page
496 --%>
497
498 <%@page isErrorPage="true"%>
499
500 Output From Error.jsp
501 <br>
502 Error :
503 <%= exception.toString()
504 %>
505 <br>
506 Desc
507 <%= exception.getMessage()
508 %>
511 =====
512 App6(Application on <jsp:plugin>
513 =====
514 -----TestApp.java-----
515 import java.applet.*;
516 import java.awt.*;
517 import java.awt.event.*;
518 public class TestApp extends Applet
519 {
520     public void paint(Graphics g)
521     {
522         setBackground(Color.red);
523         g.drawString("this is from applet to test",50,50);
524     }
526 }
527 -----plug.jsp-----
528 <html>
529 <title> Plugin example </title>
530 <body bgcolor="white">
531 <jsp:plugin type="applet" code="TestApp.class"
532 codebase="/jsp-plug" width="300" height="300" >
533   <jsp:fallback>
534     Plugin tag OBJECT or EMBED not supported by browser.
535   </jsp:fallback>
536 </jsp:plugin>
537 </body>
538 </html>
```

```
539 =====
540 App7(jsp----->java bean)
541 =====
542 -----StudentBean.java-----
543 package com.nt;
544
545
546 public class StudentBean
547 {
548     private int sno;
549     private String sname;
550     private String result;
551     public StudentBean()
552     {
553         System.out.println("inside constructor");
554     }
555     public void setSno(int i)
556     {
557         sno = i;
558     }
559     public int getSno()
560     {
561         return sno;
562     }
563     public void setSname(String s)
564     {
565         sname = s;
566     }
567     public String getSname()
568     {
569         return sname;
570     }
571     public void setResult(String s)
572     {
573         result = s;
574     }
575     public String getResult()
576     {
577         return result;
578     }
579 }
580 -----SetValues.jsp-----
581 <%@ page import="com.nt.StudentBean"%>
582
583 <jsp:useBean id="stud" class="p1.StudentBean" scope="application"></jsp:useBean>
584
585 <jsp:setProperty name="stud" property="sno" value="101"/>
586 <jsp:setProperty name="stud" property="sname" value="raja"/>
587 <jsp:setProperty name="stud" property="result" value="pass"/>
588 -----GetValues.jsp-----
589 <%@ page import="com.nt.StudentBean"%>
590
591 <jsp:useBean id="stud" class="p1.StudentBean" scope="application"></jsp:useBean>
592
593 <html>
594     <head><title>jsp prg</title></head>
595     <body>
596         student number is<jsp:getProperty name="stud" property="sno"/><br>
597         student name is <jsp:getProperty name="stud" property="sname"/><br>
598         student result is <jsp:getProperty name="stud" property="result"/>
```

```
599      </body>
600  </html>
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619 =====
620 App8(Application on Add Rotator)
621 =====
622 -----Rotator.java-----
623 package myapp;
624
625 public class Rotator
626 {
627     private String images[] = { "1.jpg","2.jpg","3.jpg","4.jpg","5.jpg"};
628     private String links[] = {
629         "http://www.yahoo.co.in","http://www.scores.sify.com","http://www.cricinfo.com",
630         "http://www.bazee.com","http://www.google.co.in"};
631     private int counter = 0;
632     public String getImage()
633     {
634         return images[ counter ];
635     }
636     public String getLink()
637     {
638         return links[counter];
639     }
640     public void nextAdvertisement()
641     {
642         // counter = ( counter + 1 ) % images.length;
643         Random rad=new Random();
644         counter=rad.nextInt(5);
645     }
646 }
647 -----adrotator.jsp-----
648 <html>
649 <jsp:useBean id="rotator" scope="session" class="myapp.Rotator" />
650
651 <%
652 response.setIntHeader("Refresh",2);
653 rotator.nextAdvertisement();
654 %>
655 <body bgcolor="red" scroll=no leftmargin=0 topmargin=0>
656 <a href=<jsp:getProperty name="rotator" property="link" />>
657 <img border=0 width=300 height=100
658     src=<jsp:getProperty name="rotator" property="image" /> >
```

```
659 </a>
660 <br>
661 <b> it is rest of the page </b>
662 </body>
663 </html>
664 =====
665 >>>>>>>>>>PageContext attributes>>>>>>>>>>
666 =====
667 Pagecontext allows to create diff types attributes
668 they are a) page scope attributes b) request scope attributes
669 c) session scope attributes d) application scope attributes
670
671 To create pageContext attributes
672 -----
673 pageContext.setAttribute("attr1","val1");
674 --> creates page scope "attr1" attribute
675 pageContext.setAttribute("attr2","val2",PageContext.SESSION_SCOPE);
676 --> creates session scope "attr2" attribute
677 other possible scopes
678 -----
679 PageContext.PAGE_SCOPE
680 PageContext.SESSION_SCOPE
681 PageContext.REQUEST_SCOPE
682 PageContext.APPLICATION_SCOPE
683
684 prefer working with "Declaration tag" member variables instead page scope
685 attributes
686
687 to modify pageContext attribute values
688 -----
689 pageContext.setAttribute("attr1","val1");
690 -> modifies page scope "attr1" attribute value
691 pageContext.setAttribute("attr2","new val2",pageContext.SESSION_SCOPE);
692 -> modifies the session scope "attr2" attribute value.
693
694 to read pageContext attribute values
695 -----
696 String s1=(String)pageContext.getAttribute("attr1");
697 --> reads page scope "attr1" attribute value
698 String s2=(String)pageContext.getAttribute("attr2",PageContext.SESSION_SCOPE);
699 --> reads session scope "attr2" attribute value
700 to find pageContext attribute
701 -----
702 String s1=(String) pageContext.findAttribute("attr1");
703 -> searches and reads attr1 attribute value in multiples scopes
704 like pagescope,request scope,session scope,application scope
705 to remove pageContext attribute
706 -----
707 pageContext.removeAttribute("attr1");
708 --> removes the page scope "attr1" attribute
709 pageContext.removeAttribute("attr2",PageContext.SESSION_SCOPE);
710 --> removes the session scope "attr2" attribute
711 =====
712 App10 (Application on pageContext Attributes )
713 =====
714 -----A.jsp-----
715 <% //create attributes
716 pageContext.setAttribute("attr1","val1",PageContext.REQUEST_SCOPE);
717 pageContext.setAttribute("attr2","val2",PageContext.SESSION_SCOPE);
718 pageContext.setAttribute("attr3","val3",PageContext.APPLICATION_SCOPE); %>
```



```
779 <jsp:forward page="Discount.jsp">
780   <jsp:param name="bill" value="<% =bamt %>" />
781 </jsp:forward>
782 <% }//if
783 else
784 { %
785 from Bill.jsp <br>
786 Name = <% =name %><br>
787 Price = <% =price %><br>
788 Qty = <% =qty %><br>
789 Bill Amt = <% =bamt %><br>
790 <% }//else %>
791 -----Discount.jsp-----
792
793
794 <% //read form data
795 String name= request.getParameter("t1");
796 int price=Integer.parseInt(request.getParameter("t2"));
797 int qty=Integer.parseInt(request.getParameter("t3"));
798 //read additional req param value given by Bill.jsp
799 int bamt=Integer.parseInt(request.getParameter("bill"));
800
801 //Give 30% discount on Bill Amt
802 float discount=bamt*0.3f;
803 float famt=bamt-discount;
804 %>
805
806 <br>
807 From Discount.jsp
808 <br>
809 Item name : <% =name %><br>
810 Price : <% =price %> <br>
811 Qty : <% =qty %><br>
812 Bill Amt : <% =bamt %><br>
813 Discount : <% =discount %><br>
814 Final Bill Amt: <% =famt %>
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
```

839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874

## What is a Tag Library?

In JavaServer Pages technology, *actions* are elements that can create and access programming language objects and affect the output stream. The JSP specification defines 6 standard actions that must be provided by any compliant JSP implementation.

In addition to the standard actions, JSP v1.1 technology supports the development of reusable modules called *custom actions*. A custom action is invoked by using a *custom tag* in a JSP page. A *tag library* is a collection of *custom tags*.

Some examples of tasks that can be performed by custom actions include form processing, accessing databases and other enterprise services such as email and directories, and flow control. Before the availability of custom actions, JavaBeans components in conjunction with scriptlets were the main mechanism for performing such processing. The disadvantage of using this approach is that it makes JSP pages more complex and difficult to maintain.

Custom actions alleviate this problem by bringing the benefits of another level of componentization to JSP pages. Custom actions encapsulate recurring tasks so that they can be reused across more than one application and increase productivity by encouraging division of labor between library developers and library users. JSP tag libraries are created by developers who are proficient at the Java programming language and expert in accessing data and other services. JSP tag libraries are used by Web application designers who can focus on presentation issues rather than being concerned with how to access databases and other enterprise services.

Some features of custom tags are:

They can be customized via attributes passed from the calling page.

They have access to all the objects available to JSP pages.

They can modify the response generated by the calling page.

They can communicate with each other. You can create and initialize a JavaBeans component, create a variable that refers to that bean in one tag, and then use the bean in another tag.

They can be nested within one another, allowing for complex interactions within a JSP page.

## Using Tags

This section describes how a page author specifies that a JSP page is using a tag library and introduces the different types of tags.

### Declaring Tag Libraries

You declare that a JSP page will use tags defined in a tag library by including a taglib directive in the page before any custom tag is used:

```
<%@ taglib uri="/tlt" prefix="tlt" %>
```

The uri attribute refers to a URI that uniquely identifies the tag library. This URI can be relative or absolute. If it is relative it must be mapped to an absolute location in the taglib element of a Web application deployment descriptor, the configuration file associated with Web applications developed according to the Java Servlet and JavaServer Pages specifications. The prefix attribute defines the prefix that distinguishes tags provided by a given tag library from those provided by other tag libraries.

### Types of Tags

JSP custom actions are expressed using XML syntax. They have a start tag and end tag, and possibly a body:

```
<tlt:tag>
```

```
body
```

</tlt:tag>

A tag with no body can be expressed as follows:

<tlt:tag />

#### Simple Tags

The following simple tag invokes an action that creates a greeting:

<tlt:greeting />

#### Tags With Attributes

The start tag of a custom action can contain attributes in the form attr="value". Attributes serve to customize the behavior of a tag just as parameters are used to affect the outcome of executing a method on an object.

Tag attributes can be set from one or more parameters in the request object or from a String constant. The only types of attributes that can be set from request parameter values and String constants are those listed in [Table 1](#); the conversion applied is that shown in the table. When assigning values to indexed attributes the value must be an array; the rules just described apply to the elements.

**Table 1 Valid Tag Attribute Assignments**

Property Type	Conversion on String Value
boolean or Boolean	As indicated in <code>java.lang.Boolean.valueOf(String)</code>
byte or Byte	As indicated in <code>java.lang.Byte.valueOf(String)</code>
char or Character	As indicated in <code>java.lang.Character.valueOf(String)</code>
double or Double	As indicated in <code>java.lang.Double.valueOf(String)</code>
int or Integer	As indicated in <code>java.lang.Integer.valueOf(String)</code>
float or Float	As indicated in <code>java.lang.Float.valueOf(String)</code>
long or Long	As indicated in <code>java.lang.Long.valueOf(String)</code>

An attribute value of the form <%= scriptlet\_expression %> is computed at request time. The value of the expression depends on the type of the attribute's value, which is specified in the object that implements the tag (called a *tag handler*). Request-time expressions can be assigned to attributes of any type; no automatic conversions will be performed.

The following tag has an attribute named date, which accepts a String value obtained by evaluating the variable today:

<tlt:greeting date="<%=\= today %>" />

#### Tags With a Body

A tag can contain custom and core tags, scripting elements, HTML text, and tag-dependent body content between the start and end tag. In the following example, the date information is provided in the body of the tag, instead of as an attribute:

<tlt:greeting>

<%=\= today %>

</tlt:greeting>

#### **Choosing Between Passing Information as Attributes or Body**

As shown in the last two sections, it is possible to pass a given piece of data as an attribute of the tag or to the tag's body. Generally speaking, any data that is a simple string or can be generated by evaluating a simple expression is best passed as an attribute.

**Tags That Define Scripting Variables**

A tag can define a variable that can be used in scripts within a page. The following example illustrates how to define and use a scripting variable that contains an object returned from a JNDI lookup. Examples of such objects include enterprise beans, transactions, databases, environment entries, and so on:

```
<tlt:lookup id="tx" type="UserTransaction"  
name="java:comp/UserTransaction" />  
<% tx.begin(); %>
```

**Cooperating Tags**

Tags cooperate with each other by means of shared objects.

In the following example, tag1 creates a named object called obj1, which is then reused by tag2. The convention encouraged by the JSP specification is that a tag with attribute named id creates and names an object and the object is then referenced by other tags with an attribute named name.

```
<tlt:tag1 id="obj1" attr2="value" />  
<tlt:tag2 name="obj1" />
```

In the next example, an object created by the enclosing tag of a group of nested tags is available to all inner tags. Since the object is not named, the potential for naming conflicts is reduced. The following example illustrates how a set of cooperating nested tags would appear in a JSP page.

```
<tlt:outerTag>  
  <tlt:innerTag />  
</tlt:outerTag>
```

**Defining Tags**

To define a tag, you need to:

Develop a tag handler and helper classes for the tag

Declare the tag in a tag library descriptor

This section describes the properties of tag handlers and tag library descriptors and explains how to develop tag handlers and library descriptor elements for each type of tag introduced in the previous section.

**Tag Handlers**

A **tag handler** is an object invoked by a JSP container to evaluate a custom tag during the execution of the JSP-page that references the tag. Tag handler methods are called by the JSP page implementation class at various points during the evaluation of the tag.

When the start tag of a custom tag is encountered, the JSP page implementation class calls methods to initialize the appropriate handler and then invokes the handler's `doStartTag` method. When the custom end tag is encountered, the handler's `doEndTag` method is invoked. Additional methods are invoked in between when a tag handler needs to interact with the body of the tag. For further information, see ["How Is a Tag Handler Invoked?"](#).

In order to provide a tag handler implementation, you must implement the methods that are invoked at various stages of processing the tag. The methods are summarized in [Table 2](#).

Table 2 Tag Handler Methods

Tag Handler Type	Methods
Simple	doStartTag, doEndTag, release
Attributes	doStartTag, doEndTag, set/getAttribute1...N
Body, No Interaction	doStartTag, doEndTag, release
Body, Interaction	doStartTag, doEndTag, release, doInitBody, doAfterBody

A tag handler has access to an API that allows it to communicate with the JSP page. The entry point to the API is the page context object through which a tag handler can access to all the other implicit objects (request, session, and application) accessible from a JSP page. Implicit objects can have attributes associated with them. Such attributes are accessed using the appropriate [set/get]Attribute method.

If the tag is nested, a tag handler also has access to the handler (called the *parent*) associated with the enclosing tag.

Tag handlers must implement either the Tag or BodyTag interfaces. Interfaces can be used to take an existing Java object and make it a tag handler. For newly created handlers, you can use the TagSupport and BodyTagSupport classes as base classes.

### Tag Library Descriptors

A *tag library descriptor* (TLD) is an XML document that describes a tag library. A TLD contains information about a library as a whole and about each tag contained in the library. TLDs are used by a JSP container to validate the tags and by JSP development tools.

The following TLD elements are used to define a tag library:

```

<taglib>
<tlibversion> - The tag library's version
<jspversion> - The JSP specification version the tag library depends on
<shortname> - A simple default name that could be used by a JSP page authoring tool to create names
with a mnemonic value; for example, shortname may be used as the preferred prefix value in taglib
directives and/or to create prefixes for IDs.
<uri> - A URI that uniquely identifies the tag library
<info> - Descriptive information about the tag library
<tag>
...
</tag>
...
</taglib>
```

The TLD element required for all tags is the one used to specify a tag handler's class:

```

<tag>
<tagclass>classname</tagclass>
...
</tag>
```

### Simple Tags

#### Tag Handlers

The handler for a simple tag must implement the doStartTag and doEndTag methods of the Tag interface. The doStartTag method is invoked when the start tag is encountered. This method returns SKIP\_BODY because a simple tag has no body. The doEndTag method is invoked when the end tag is encountered. The doEndTag method needs to return EVAL\_PAGE if the rest of the page needs to be evaluated; otherwise it should return SKIP\_PAGE.

The following simple tag:

```
<tilt:simple />
```

would be implemented by the following tag handler:

```
public SimpleTag extends TagSupport {
    public int doStartTag() throws JspException {
        try {
            pageContext.getOut().print("Hello.");
        } catch (Exception ex) {
            throw new JspTagException("SimpleTag: " +
                e.getMessage());
        }
        return SKIP_BODY;
    }
    public int doEndTag() {
        return EVAL_PAGE;
    }
}
```

#### TLD *bodycontent* Element

Tags without bodies must declare that their body content is empty:

```
<tag>
...
<bodycontent>empty</bodycontent>
</tag>
```

#### Tags With Attributes

##### Defining Attributes in a Tag Handler

For each tag attribute, you must define a property and JavaBeans style get and set methods in the tag handler. For example, the tag handler for the tag

```
<tlt:twa attr1="value1">
```

where value1 is of type AttributeClass, must contain the following declaration and methods:

```
private AttributeClass attr1;
setAttr1(AttributeClass ac) { ... }
AttributeClass getAttr1() { ... }
```

Note that if your attribute is named id, and your tag handler inherits from the TagSupport class, you do not need to define the property and set and get methods because these are already defined by TagSupport.

A tag attribute whose value is a String can name an attribute of one of the implicit objects available to tag handlers. An implicit object attribute would be accessed by passing the tag attribute value to the [set/get]Attribute method of the implicit object. This is a good way to pass scripting variable names to a tag handler where they are associated with objects stored in the page context.

#### TLD *attribute* Element

For each tag attribute you must specify whether the attribute is required, and whether the value can be determined by an expression:

```
<tag>
...
<attribute>
```

```
<name>attr1</name>
<required>true|false|yes|no</required>
<rteprvalue>true|false|yes|no</rteprvalue>
</attribute>
</tag>
```

If a tag attribute is not required, a tag handler should provide a default value.

```
1 =====
2 Custom Jsp Taglibrary Development
3 =====
4 -----web.xml-----
5 <web-app>
6   <taglib>
7     <taglib-uri>demo</taglib-uri>
8     <taglib-location>/WEB-INF/first.tld</taglib-location>
9   </taglib>
10  </web-app>
11 -----Test.jsp-----
12 <%@ taglib uri="demo" prefix="start"%>
13 <html>
14   <head>
15     <title>Tag Test !!!.....</title>
16   </head>
17   <body>
18     <center><h2>
19       <start:example/>
20       <br>
21       <start:prime />
22       <br>
23       <start:prime n="30" />
24       <br>
25       <start:display font="webdings" >
26       <br>JAVA powered by
27       </start:display>
28       <br>
29       <start:display font="arial" size="56">
30         HCL Technologies
31       </start:display>
32     </h2></center>
33   </body>
34 </html>
35 -----first.tld-----
36 <?xml version="1.0" encoding="UTF-8"?>
37 <!DOCTYPE taglib
38   PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
39   "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">
40
41 <taglib>
42   <tlibversion>1.0</tlibversion>
43   <jspversion>1.2</jspversion>
44   <shortname>start</shortname>
45   <info>
46     An example Tag Library
47   </info>
48   <tag>
49     <name>example</name>
50     <tagclass>tags.ExampleTag</tagclass>
51     <info>Simplest example: inserts one line of output</info>
52     <bodycontent>EMPTY</bodycontent>
53   </tag>
54
55   <tag>
56     <name>prime</name>
57     <tagclass>tags.PrimeTag</tagclass>
58     <info>Prime numbers generation</info>
59     <bodycontent>EMPTY</bodycontent>
60     <attribute>
```

```
61          <name>n</name>
62          <required>false</required>
63      </attribute>
64  </tag>
65
66  <tag>
67      <name>display</name>
68      <tagclass>tags.DisplayTag</tagclass>
69      <info>Displaying text in different fonts</info>
70      <bodycontent>Jsp</bodycontent>
71      <attribute>
72          <name>font</name>
73          <required>true</required>
74      </attribute>
75      <attribute>
76          <name>size</name>
77          <required>false</required>
78      </attribute>
79  </tag>
80
81 </taglib>
82 -----ExampleTag.java-----
83 package tags;
84
85 import javax.servlet.jsp.*;
86 import javax.servlet.jsp.tagext.*;
87 import java.io.*;
88
89 public class ExampleTag extends TagSupport
90 {
91     public int doStartTag()
92     {
93         System.out.println("Inside doStartTag() ExampleTag");
94         try
95         {
96             JspWriter out=pageContext.getOut();
97             out.print("Prime numbers"+<br>);
98         }
99         catch(IOException e)
100        {
101            System.out.println("Error in ExampleTag: "+ e);
102        }
103        return(SKIP_BODY);
104    }//doStartTag()
105    public int doEndTag()
106    {
107        System.out.println("Inside doEndTag() ExampleTag");
108        return EVAL_PAGE;
109    }//doEndTag()
110}//class
111 -----PrimeTag.java-----
112 package tags;
113
114 import javax.servlet.jsp.*;
115 import javax.servlet.jsp.tagext.*;
116 import java.io.*;
117
118 public class PrimeTag extends TagSupport
119 {
120     private int n=10;
```

```
121     public void setN(int n)
122     {
123         this.n=n;
124     }
125     public int getN()
126     {
127         return n;
128     }
129
130
131     private boolean isPrime(int x)
132     {
133         for (int k=2;k<x;k++)
134         {
135             if(x%k==0)
136                 return false;
137         }//for
138         return true;
139     }//isPrime()
140     public int doStartTag()
141     {
142         System.out.println("Inside doStartTag() of PrimeTag");
143         try
144         {
145             JspWriter out=pageContext.getOut();
146             for(int i=1;i<n;i++)
147             {
148                 if(isPrime(i))
149                     out.print(i+" ");
150             }//for
151         }//try
152         catch(IOException ie)
153         {
154             ie.printStackTrace();
155         }
156         return SKIP_BODY;
157     }//doStartTag()
158
159     public int doEndTag()
160     {
161         System.out.println("Inside doEndTag() of PrimeTag");
162         return EVAL_PAGE;
163     }//doEndTag()
164 } //class
165 -----DisplayTag.java-----
166 package tags;
167
168 import javax.servlet.*;
169 import javax.servlet.jsp.*;
170 import javax.servlet.jsp.tagext.*;
171 import java.io.*;
172
173 public class DisplayTag extends TagSupport
174 {
175     private String font;
176     private int size=20;
177
178     public void setFont(String font)
179     {
180         this.font=font;
```



```
241 -----web.xml-----
242 <?xml version="1.0" encoding="ISO-8859-1"?>
243 <!DOCTYPE web-app
244     PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
245     "http://java.sun.com/dtd/web-app_2_3.dtd">
246
247 <web-app>
248     <welcome-file-list>
249         <welcome-file>Main.html</welcome-file>
250     </welcome-file-list>
251
252     <servlet>
253         <servlet-name>insert</servlet-name>
254         <servlet-class>InsertDemo</servlet-class>
255     </servlet>
256     <servlet-mapping>
257         <servlet-name>insert</servlet-name>
258         <url-pattern>/insert</url-pattern>
259     </servlet-mapping>
260
261
262     <servlet>
263         <servlet-name>delete</servlet-name>
264         <servlet-class>DeleteDemo</servlet-class>
265     </servlet>
266     <servlet-mapping>
267         <servlet-name>delete</servlet-name>
268         <url-pattern>/delete</url-pattern>
269     </servlet-mapping>
270
271
272     <servlet>
273         <servlet-name>update</servlet-name>
274         <servlet-class>UpdateDemo</servlet-class>
275     </servlet>
276     <servlet-mapping>
277         <servlet-name>update</servlet-name>
278         <url-pattern>/update</url-pattern>
279     </servlet-mapping>
280
281 </web-app>
282 -----Main.html-----
283 <html>
284 <frameset rows = "15%,*" border = 0>
285 <frame src = "Info.html" name = "info" scrolling = no>
286 <frameset cols = "25%,*">
287 <frame src = "Links.html" name = "links">
288 <frame src = "" name = "display">
289 </frameset>
290 </frameset>
291 </html>
292 -----info.html-----
293 <html>
294 <body>
295 <form name = f>
296 <center><span style = "width=500;height=400;filter:shadow(color=blue,direction=135)">
297 <font color = lavender size = 5>HCL TECHNOLOGIES</font>
298 <hr color = pink width = 60%>
299 </span></center>
300 </form>
```

```
301 </body>
302 </html>
303 -----Links.html-----
304 <html>
305 <body>
306 <form name = f>
307 <div style = "position:absolute;left:80;top:70">
308 <a href = "Insert.jsp" target = "display">INSERT</a>
309 </div>
310 <div style = "position:absolute;left:80;top:140">
311 <a href = "Delete.jsp" target = "display"> DELETE</a>
312 </div>
313 <div style = "position:absolute;left:80;top:210">
314 <a href = "Update.jsp" target = "display">UPDATE</a>
315 </div>
316 </form>
317 </body>
318 </html>
319 -----Insert.jsp-----
320 <html>
321 <script src = "insert.js" language = "javascript">
322 </script>
323 <body>
324 <form name = f action = './insert' method = 'post'
325     onSubmit = "return isValid(this)">
326 <table border = 1 cellpadding = 7 cellspacing = 7 align = center>
327 <caption><i><font size = 4><u>Insert Records</u></font></i></caption>
328 <tr>
329 <th>Student ID<th><input type = "text" name = "stu_id"></tr>
330 <tr>
331 <th>Student NAME<th><input type = "text" name = "stu_name"></tr>
332 <tr>
333 <th>Student Address<th><input type = "text" name = "stu_add"></tr>
334 <tr>
335 <th>click<th><input type = "submit" value = " INSERT "></tr>
336 </table>
337 </form>
338 </body>
339 </html>
340 -----insert.js-----
341 function isValid(frm)
342 {
343     var sid = frm.stu_id;
344     var snm = frm.stu_name;
345     var sadd = frm.stu_add;
346     if(check(sid) && isNumber(sid) && check(snm) && check(sadd))
347     {
348         return true;
349     }
350     else
351     {
352         return false;
353     }
354 }
355 function check(cmd)
356 {
357     if(cmd.value == "")
358     {
359         alert(cmd.name + " " + "Field Missed");
360         cmd.focus();
```

```
361         return false;
362     }
363     return true;
364 }
365 function isNumber(cmd)
366 {
367     if(isNaN(cmd.value))
368     {
369         alert("Stu_ID should be numeric value");
370         cmd.focus();
371         return false;
372     }
373     return true;
374 }
375 -----InsertDemo.java-----
376 import javax.servlet.*;
377 import javax.servlet.http.*;
378 import java.io.*;
379 import java.sql.*;
380
381 public class InsertDemo extends HttpServlet
382 {
383     public void doPost(HttpServletRequest req,HttpServletResponse res)
384     {
385         Connection con = null;
386         PreparedStatement ps = null;
387         int rs = 0;
388         try
389         {
390             PrintWriter pw = res.getWriter();
391             res.setContentType("text/html");
392             pw.println("<html><form target = 'display'>");
393             int sid = Integer.parseInt(req.getParameter("stu_id"));
394             String sname = req.getParameter("stu_name");
395             String sadd = req.getParameter("stu_add");
396             Class.forName("oracle.jdbc.driver.OracleDriver");
397             con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl", "scott", "tiger");
398             System.out.println("connection");
399             ps = con.prepareStatement("insert into student_info values(?, ?, ?)");
400             ps.setInt(1, sid);
401             ps.setString(2, sname);
402             ps.setString(3, sadd);
403             rs = ps.executeUpdate();
404             if(rs!=1)
405                 pw.println("<h2>Record is problem</h2>");
406             else
407                 pw.println("<h2 style = 'position:absolute;left:50;top:50'>One Record Inserted Successfully</h2>");
408
409             ps.close();
410             con.close();
411         }
412         catch(Exception e)
413         {
414             e.printStackTrace();
415         }
416     }
417     public void doGet(HttpServletRequest req,HttpServletResponse res)
418     {
419         try
420         {
```

```
421         doPost(req,res);
422     }
423     catch(Exception e)
424     {
425         e.printStackTrace();
426     }
427 }
428 }
429 -----Delete.jsp-----
430 <html>
431 <script language = 'javascript'>
432 function isValid()
433 {
434     if(f.stu_id.value == "")
435     {
436         alert(f.stu_id.name + " " + "Field Missed");
437         f.stu_id.focus();
438         return false;
439     }
440     if isNaN(f.stu_id.value))
441     {
442         alert("Stu_ID should be numeric value");
443         f.stu_id.focus();
444         return false;
445     }
446     return true;
447 }
448 </script>
449 <body>
450 <form name = f action = './delete' method = 'post' onSubmit ='return isValid()'>
451 <table border = 1 cellpadding = 7 cellspacing = 7 align = center>
452 <caption><i><font size = 4><u>Delete Records</u></font></i></caption>
453 <tr>
454 <th>Student ID<th><input type = "text" name = "stu_id"></tr>
455 <tr>
456 <th>click<th><input type = "submit" value = " DELETE "></tr>
457 </table>
458 </form>
459 </body>
460 </html>
461 -----DeleteDemo.java-----
462 import javax.servlet.*;
463 import javax.servlet.http.*;
464 import java.io.*;
465 import java.sql.*;
466
467 public class DeleteDemo extends HttpServlet
468 {
469     public void doPost(HttpServletRequest req,HttpServletResponse res)
470     {
471         Connection con = null;
472         PreparedStatement ps = null;
473         int rs = 0;
474         try
475         {
476             PrintWriter pw = res.getWriter();
477             res.setContentType("text/html");
478             pw.println("<html><form target = 'display'>");
479             int sid = Integer.parseInt(req.getParameter("stu_id"));
480             Class.forName("oracle.jdbc.driver.OracleDriver");
```

```
481         con = DriverManager.getConnection
482             ("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
483             ps = con.prepareStatement("delete from student_info where stu_id=?");
484             ps.setInt(1,stu_id);
485             rs = ps.executeUpdate();
486             if(rs!=1)
487                 pw.println("<h2>Student id is problem</h2>");
488             else
489                 pw.println("<h2 style = 'position:absolute;left:50;top:50'>
490                         One Record Deleted</h2>");
491
492             ps.close();
493             con.close();
494         }
495     catch(Exception e)
496     {
497         e.printStackTrace();
498     }
499 }
500 public void doGet(HttpServletRequest req,HttpServletResponse res)
501 {
502     try{
503         doPost(req,res);
504     }
505     catch(Exception e)
506     {
507         e.printStackTrace();
508     }
509 }
510 }
511 -----Update.jsp-----
512 <html>
513 <script src = "insert.js" language = "javascript">
514 </script>
515 <body>
516 <form name = f action = './update' method = 'post' onSubmit = "return isValid(this)">
517 <table border = 1 cellpadding = 7 cellspacing = 7 align = center>
518 <caption><i><font size = 4><u>Update Records</u></font></i></caption>
519 <tr>
520 <th>Student ID<th><input type = "text" name = "stu_id"></tr>
521 <tr>
522 <th>Student NAME<th><input type = "text" name = "stu_name"></tr>
523 <tr>
524 <th>Student Address<th><input type = "text" name = "stu_add"></tr>
525 <tr>
526 <th>click<th><input type = "submit" value = " UPDATE "></tr>
527 </table>
528 </form>
529 </body>
530 </html>
531 -----UpdateDemo.java-----
532 import javax.servlet.*;
533 import javax.servlet.http.*;
534 import java.io.*;
535 import java.sql.*;
536
537 public class UpdateDemo extends HttpServlet
538 {
539     public void doPost(HttpServletRequest req,HttpServletResponse res)
540     {
```

```

541     Connection con = null;
542     PreparedStatement ps = null;
543     int rs = 0;
544     try
545     {
546         PrintWriter pw = res.getWriter();
547         res.setContentType("text/html");
548         pw.println("<html><form target = 'display'>");
549         int sid = Integer.parseInt(req.getParameter("stu_id"));
550         String sname = req.getParameter("stu_name");
551         String sadd = req.getParameter("stu_add");
552         Class.forName("oracle.jdbc.driver.OracleDriver");
553         con = DriverManager.getConnection
554             ("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
555         ps = con.prepareStatement
556             ("update student_info set stu_name=? ,stu_add=? where stu_id=?");
557         ps.setString(1,sname);
558         ps.setString(2,sadd);
559         ps.setInt(3,sid);
560         rs = ps.executeUpdate();
561         if(rs!=1)
562             pw.println("<h2>Student ID problem</h2>");
563         else
564             pw.println("<h2 style = 'position:absolute;left:50;top:50'>
565                         One Record Updated</h2>");
566
567         ps.close();
568         con.close();
569     }
570     catch(Exception e)
571     {
572         e.printStackTrace();
573     }
574 }
575 public void doGet(HttpServletRequest req,HttpServletResponse res)
576 {
577     try
578     {
579         doPost(req,res);
580     }
581     catch(Exception e)
582     {
583         e.printStackTrace();
584     }
585 }
586 }
587 =====
588 Main Example (Mini Project2)
589 =====
590 -----Database Details-----
591 SQL>drop table STUDENT_INFO cascade;
592 SQL>create table STUDENT_INFO(STU_ID number,STU_NAME varchar2(30),STU_ADD varchar2(30));
593 -----web.xml-----
594 <?xml version="1.0" encoding="ISO-8859-1"?>
595 <!DOCTYPE web-app
596     PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
597     "http://java.sun.com/dtd/web-app_2_3.dtd">
598
599 <web-app>
600     <servlet>

```

```
601      <servlet-name>First</servlet-name>
602      <servlet-class>StudentDemo</servlet-class>
603    </servlet>
604    <servlet-mapping>
605      <servlet-name>First</servlet-name>
606      <url-pattern>/studentdemo</url-pattern>
607    </servlet-mapping>
608
609    <welcome-file-list>
610      <welcome-file>StudentDetails.jsp</welcome-file>
611    </welcome-file-list>
612  </web-app>
613 -----StudentDetails.jsp-----
614 <%
615   String res =(String)request.getAttribute("msg");
616   String id = (String)request.getAttribute("no");
617   String name = (String)request.getAttribute("name");
618   String addr = (String)request.getAttribute("address");
619 %>
620
621 <html>
622   <script language = 'javascript'>
623
624     function isInsert()
625     {
626       f.setVal.value = "insert";
627       check();
628     }
629     function isDelete()
630     {
631       f.setVal.value = "delete";
632       check1();
633     }
634     function isUpdate()
635     {
636       f.setVal.value = "update";
637       check();
638     }
639     function isDisplay()
640     {
641       f.setVal.value = "display";
642       check1();
643     }
644
645     function check()
646     {
647       if(f.stu_id.value == "")
648       {
649         alert("Please provide Student ID");
650         f.stu_id.focus();
651       }
652       else if(isNaN(f.stu_id.value))
653       {
654         alert("Student ID should be numeric");
655         f.stu_id.value = "";
656         f.stu_id.focus();
657       }
658       else if(f.stu_name.value == "")
659       {
660         alert("Please provide Student Name");
```

```
661         f.stu_name.focus();
662     }
663     else if(f.stu_add.value == "")
664     {
665         alert("Please provide Student Address");
666         f.stu_add.focus();
667     }
668     else
669     {
670         f.submit();
671     }
672     function check1()
673     {
674         if(f.stu_id.value == "")
675         {
676             alert("Please provide Student ID");
677             f.stu_id.focus();
678         }
679         else if(isNaN(f.stu_id.value))
680         {
681             alert("Student ID should be numeric");
682             f.stu_id.value = "";
683             f.stu_id.focus();
684         }
685         else
686         {
687             f.submit();
688         }
689     function disp1_output()
690     {
691         var i = <%=id%>;
692         var n = '<%=name%>';
693         var a = '<%=addr%>';
694         f.stu_id.value = i;
695         f.stu_name.value=n;
696         f.stu_add.value = a;
697     }
698     function isReloading()
699     {
700         <%
701         if(res != null)
702         {
703             if(res.equals("1"))
704                 out.println("alert('Information stored successfully')");
705             else if(res.equals("2"))
706                 out.println("alert('Given Student ID already exists')");
707             else if(res.equals("3"))
708                 out.println("alert('Information deleted successfully')");
709             else if(res.equals("4"))
710                 out.println("alert('Given Student ID is Invalid')");
711             else if(res.equals("5"))
712                 out.println("alert('Information updated successfully')");
713             else if(res.equals("6"))
714             {
715                 out.println("disp1_output()");
716             }
717             else if(res.equals("7"))
718                 out.println("alert('Unknown problem occurred.')");
719         }
720     }
```

```
721      %>
722  }
723
724  </script>
725
726 <body onLoad='isReloading()'%>
727   <form name=f action='./studentdemo' method='post'>
728
729   <center>
730   <span style= "width=500;height=60;filter:shadow(color=blue,direction=135)">
731
732   <font color=lavender size=5>HCL TECHNOLOGIES</font>
733   <hr color=pink width=100%>
734   </span></center>
735
736   <table border=1 cellpadding=7 cellspacing=7 align=center>
737
738   <caption>
739     <font size=6><i><b><u>Student Table</u></b></i></font>
740   </caption>
741
742   <tr>
743     <th>Student ID</th>
744     <th><input type="text" name="stu_id"></th>
745   </tr>
746   <tr>
747     <th>Student Name</th>
748     <th><input type="text" name="stu_name"></th>
749   </tr>
750   <tr>
751     <th>Student Address</th>
752     <th><input type="text" name="stu_add"></th>
753   </tr>
754   <tr>
755     <th colspan=4>
756       <input type='button' value='Insert' onClick='isInsert()'>
757       <input type='button' value='Delete' onClick='isDelete()'>
758       <input type='button' value='Update' onClick='isUpdate()'>
759       <input type='button' value='Display' onClick='isDisplay()'>
760   </tr>
761 </table>
762
763   <input type='text' name='setVal' readonly
764     style='visibility:hidden'>
765   </form>
766 </body>
767 </html>
768 -----StudentDemo.java-----
769 import javax.servlet.*;
770 import javax.servlet.http.*;
771 import java.io.*;
772 import beans.DbConnector;
773
774 public class StudentDemo extends HttpServlet
775 {
776   public void doPost(HttpServletRequest req,HttpServletResponse res)
777   {
778     HttpSession session = null;
779     int flag = 0;
780     int forward = 0;
```

```
781     int output = 0;
782
783     String store = null;
784     String value = null;
785     String dispval1,dispval2,dispval3;
786
787     DbConnector dbc;
788
789     try
790     {
791         session = req.getSession(true);
792         PrintWriter out = res.getWriter();
793         res.setContentType("text/html");
794
795         int s_id = Integer.parseInt(req.getParameter("stu_id"));
796         String s_name = req.getParameter("stu_name");
797         String s_add = req.getParameter("stu_add");
798
799         String checkAction = req.getParameter("setVal");
800
801         dbc = new DbConnector(s_id,s_name,s_add);
802
803         if(checkAction.equals("insert"))
804             output = dbc.insert();
805         else if(checkAction.equals("delete"))
806             output = dbc.delete();
807         else if(checkAction.equals("update"))
808             output = dbc.update();
809         else if(checkAction.equals("display"))
810         {
811             output = dbc.display();
812
813             dispval1 = String.valueOf(dbc.getID());
814             dispval2 = dbc.getName();
815             dispval3 = dbc.getAdd();
816
817             req.setAttribute("no", dispval1);
818             req.setAttribute("name", dispval2);
819             req.setAttribute("address", dispval3);
820         }
821
822         value = String.valueOf(output);
823         req.setAttribute("msg", value);
824         RequestDispatcher rd = req.getRequestDispatcher("StudentDetails.jsp");
825         rd.forward(req,res);
826     } // try
827     catch(Exception e)
828     {
829         e.printStackTrace();
830     }
831     } // doPost()
832     public void doGet(HttpServletRequest req,HttpServletResponse res)
833     {
834         try{
835             doPost(req,res);
836         }
837         catch(Exception e)
838         {
839             e.printStackTrace();
840         }

```

```
841     } // doGet()
842 } // class
843 -----DbConnector.java-----
844 package beans;
845
846 import java.io.*;
847 import java.sql.*;
848
849 public class DbConnector
850 {
851     private Connection con = null;
852
853     private int id;
854     private String name;
855     private String add;
856
857     public DbConnector()
858     {
859     }
860
861     public DbConnector(int id, String name, String add)
862     {
863         this.id = id;
864         this.name = name;
865         this.add = add;
866
867         con = getConnection();
868     }
869
870     public void setID(int id)
871     {
872         this.id = id;
873     }
874
875     public int getID()
876     {
877         return id;
878     }
879
880     public void setName(String name)
881     {
882         this.name = name;
883     }
884
885     public String getName()
886     {
887         return name;
888     }
889
890     public void setAdd(String add)
891     {
892         this.add = add;
893     }
894
895     public String getAdd()
896     {
897         return add;
898     }
899
900     public Connection getConnection()
```

```
901     {
902         try
903         {
904             Class.forName("oracle.jdbc.driver.OracleDriver");
905             con=DriverManager.getConnection
906                 ("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
907             }catch(Exception e)
908             {
909                 e.printStackTrace();
910             }
911             return con;
912     }
913
914     public boolean exists(int id)
915     {
916         PreparedStatement ps = null;
917         try
918         {
919             ps = con.prepareStatement("SELECT COUNT(*) FROM STUDENT_INFO " +
920                             " WHERE STU_ID = ?");
921             ps.setInt(1, id);
922
923             ResultSet rs = ps.executeQuery();
924             rs.next();
925
926             if(rs.getInt(1) == 0)
927                 return false;
928             else
929                 return true;
930         }
931         catch(Exception e)
932         {
933             e.printStackTrace();
934             return false;
935         }
936         finally
937         {
938             if(ps != null)
939             {
940                 try
941                 {
942                     ps.close();
943                 }
944                 catch(Exception e)
945                 {
946                     e.printStackTrace();
947                 }
948             }
949         }
950     } // exists()
951
952     public int insert()
953     {
954         PreparedStatement ps = null;
955         try
956         {
957             boolean alreadyPresent = exists(this.getID());
958
959             if(! alreadyPresent)
960             {
```

```
961         ps = con.prepareStatement("INSERT INTO STUDENT_INFO VALUES(?, ?, ?)");
962         ps.setInt(1, id);
963         ps.setString(2, name);
964         ps.setString(3, add);
965         ps.executeUpdate();
966         return 1;
967     }
968     else
969         return 2;
970 }
971 catch(Exception e)
972 {
973     e.printStackTrace();
974     return 7;
975 }
976 finally
977 {
978     if(ps != null)
979     {
980         try
981         {
982             ps.close();
983         }
984         catch(Exception e)
985         {
986             e.printStackTrace();
987         }
988     }
989 }
990 } // insert()
991
992 public int delete()
993 {
994     PreparedStatement ps = null;
995     try
996     {
997         boolean alreadyPresent = exists(this.getID());
998
999         if(alreadyPresent)
1000         {
1001             ps = con.prepareStatement("DELETE FROM STUDENT_INFO WHERE STU_ID=?");
1002             ps.setInt(1,id);
1003             ps.executeUpdate();
1004             return 3;
1005         }
1006         else
1007             return 4;
1008     }
1009     catch(Exception e)
1010     {
1011         e.printStackTrace();
1012         return 7;
1013     }
1014     finally
1015     {
1016         if(ps != null)
1017         {
1018             try
1019             {
1020                 ps.close();
```

```
1021             }
1022         catch(Exception e)
1023         {
1024             e.printStackTrace();
1025         }
1026     }
1027 }
1028 } // delete()
1029
1030 public int update()
1031 {
1032     PreparedStatement ps = null;
1033     try
1034     {
1035         boolean alreadyPresent = exists(this.getID());
1036
1037         if(alreadyPresent)
1038         {
1039             ps = con.prepareStatement("UPDATE STUDENT_INFO SET
1040                 STU_NAME=? ,STU_ADD=? WHERE STU_ID = ?");
1041             ps.setString(1,name);
1042             ps.setString(2,add);
1043             ps.setInt(3,id);
1044             ps.executeUpdate();
1045             return 5;
1046         }
1047         else
1048             return 4;
1049     }
1050     catch(Exception e)
1051     {
1052         e.printStackTrace();
1053         return 7;
1054     }
1055     finally
1056     {
1057         if(ps != null)
1058         {
1059             try
1060             {
1061                 ps.close();
1062             }
1063             catch(Exception e)
1064             {
1065                 e.printStackTrace();
1066             }
1067         }
1068     }
1069 } // update()
1070
1071 public int display()
1072 {
1073     PreparedStatement ps = null;
1074
1075     boolean alreadyPresent = exists(this.getID());
1076
1077     try
1078     {
1079         if(alreadyPresent)
1080         {
```

```
1081         ps = con.prepareStatement("SELECT STU_NAME,STU_ADD FROM
1082                               STUDENT_INFO WHERE STU_ID=?");
1083         ps.setInt(1,id);
1084         ResultSet rs = ps.executeQuery();
1085
1086         if(rs.next())
1087         {
1088             String nm = rs.getString(1);
1089             this.setName(nm);
1090
1091             String ad = rs.getString(2);
1092             this.setAdd(ad);
1093         }
1094
1095         rs.close();
1096
1097         return 6;
1098     }
1099     else
1100     {
1101     }
1102     catch(Exception e)
1103     {
1104         e.printStackTrace();
1105         return 7;
1106     }
1107     finally
1108     {
1109         if(ps != null)
1110         {
1111             try
1112             {
1113                 ps.close();
1114             }
1115             catch(Exception e)
1116             {
1117                 e.printStackTrace();
1118             }
1119         }
1120     }
1121 } // display()
1122 } // class
1123 =====
1124 PROJECT On BookSearch using mvc2 architecture (MinProject3)
1125 =====
1126 -----SQL.txt-----
1127 CREATE TABLE SELECT_BOOKS (BOOKID VARCHAR2(10),
1128                           BOOKNAME VARCHAR2(30),
1129                           AUTHORNAME VARCHAR2(30),
1130                           STATUS VARCHAR2(10),
1131                           CATEGORY VARCHAR2(20)
1132 );
1133 -----web.xml-----
1134 <?xml version="1.0" encoding="ISO-8859-1"?>
1135 <!DOCTYPE web-app
1136   PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
1137   "http://java.sun.com/dtd/web-app_2_3.dtd">
1138
1139 <web-app>
1140     <welcome-file-list>
```

```
1141      <welcome-file>Search.jsp</welcome-file>
1142  </welcome-file-list>
1143
1144  <servlet>
1145    <servlet-name>search</servlet-name>
1146    <servlet-class>com.nt.controller.MainSrv</servlet-class>
1147  </servlet>
1148
1149  <servlet-mapping>
1150    <servlet-name>search</servlet-name>
1151    <url-pattern>/BookSearchServlet</url-pattern>
1152  </servlet-mapping>
1153 </web-app>
1154 -----Search.jsp-----
1155 <html>
1156
1157  <script language = "javascript">
1158
1159  function isHtml()
1160  {
1161    f.source.value = "Html";
1162    validate();
1163  }
1164
1165  function isExcel()
1166  {
1167    f.source.value = "Excel";
1168    validate();
1169  }
1170
1171  function validate()
1172  {
1173    if(f.category.selectedIndex == '0')
1174    {
1175      alert("You should select Category !!!");
1176      f.category.focus();
1177      return false;
1178    }
1179    else
1180    {
1181      f.submit();
1182      return true;
1183    }
1184  }
1185
1186 </script>
1187
1188 <body>
1189   <form name=f action=".//BookSearchServlet" method="post">
1190     <center>
1191       <span style= "width=500;height=60;filter:shadow(color=pink,direction=135)">
1192         <font color=red size=5>Search for Books</font>
1193         <hr color=orange width=50%>
1194       </span></center>
1195
1196   <table border=1 cellpadding=4 cellspacing=4 align=center bgcolor='lavender'>
1197     <tr>
1198       <th>Select Category</th>
1199       <th>
1200         <select name='category'>
```

```
1201             <option selected value="">Select a value</option>
1202             <option value='java'>JAVA</option>
1203             <option value=''.net'>.NET</option>
1204             <option value='jscript'>JavaScript</option>
1205         </select>
1206     </th>
1207   </tr>
1208   <tr>
1209     <th><input type='button' value='Html Output' onClick='isHtml()'></th>
1210     <th><input type='button' value='Excel Output' onClick='isExcel()'></th>
1211   </tr>
1212 </table>
1213
1214 <input type='text' name='source' readonly style='visibility:hidden'>
1215 </form>
1216 </body>
1217 </html>
1218 -----MainSrv.java-----
1219 package com.nt.controller;
1220 import javax.servlet.*;
1221 import javax.servlet.http.*;
1222 import java.io.*;
1223 import java.util.*;
1224 import com.nt.service.DbConnector;
1225
1226 public class MainSrv extends HttpServlet
1227 {
1228     public void doGet(HttpServletRequest req, HttpServletResponse res)
1229     {
1230         try
1231         {
1232             String cat = req.getParameter("category");
1233             String checkAction=req.getParameter("source");
1234
1235             DbConnector dbc = new DbConnector();
1236
1237             ArrayList al = dbc.search(cat);
1238             req.setAttribute("list", al);
1239             req.setAttribute("category", cat);
1240
1241             String target;
1242             if(checkAction.equalsIgnoreCase("Html"))
1243                 target = "HtmlPrint.jsp";
1244             else
1245                 target = "ExcelScreen.jsp";
1246
1247             RequestDispatcher rd = null;
1248             rd = req.getRequestDispatcher(target);
1249             if(rd != null)
1250                 rd.forward(req,res);
1251         } // try
1252         catch(Exception e)
1253         {
1254             e.printStackTrace();
1255         }
1256     } // doPost()
1257     public void doPost(HttpServletRequest req, HttpServletResponse res)
1258     {
1259         doGet(requeste,response);
1260     }
```

```
1261
1262 } // class
1263 -----DbConnector.java-----
1264 package com.nt.service;
1265
1266 import java.io.*;
1267 import java.sql.*;
1268 import java.util.ArrayList;
1269 import com.nt.model.BookBean;
1270
1271 public class DbConnector
1272 {
1273     private int found=0;
1274
1275     public Connection getConnection()
1276     {
1277         Connection con = null;
1278         try
1279         {
1280             Class.forName("oracle.jdbc.driver.OracleDriver");
1281             con = DriverManager.getConnection
1282                 ("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
1283         }
1284         catch(Exception e)
1285         {
1286             e.printStackTrace();
1287         }
1288         return con;
1289     }
1290
1291     public ArrayList search(String category)
1292     {
1293         Connection con = getConnection();
1294         PreparedStatement ps = null;
1295         ResultSet rs = null;
1296         ArrayList al = new ArrayList();
1297
1298         try
1299         {
1300             String searchQuery;
1301             searchQuery = "SELECT BOOKID, BOOKNAME, AUTHORNAME, STATUS " +
1302                         " FROM SELECT_BOOKS WHERE CATEGORY = ? ";
1303             ps = con.prepareStatement(searchQuery);
1304
1305             ps.setString(1, category);
1306             rs = ps.executeQuery();
1307
1308             while(rs.next())
1309             {
1310                 BookBean b = new BookBean();
1311                 b.setBookId(rs.getString(1));
1312                 b.setBookName(rs.getString(2));
1313                 b.setAuthorName(rs.getString(3));
1314                 b.setStatus(rs.getString(4));
1315                 al.add(b);
1316             }
1317             rs.close();
1318         }
1319         catch(Exception e)
1320         {
```

```
1321         e.printStackTrace();
1322     }
1323     finally
1324     {
1325         if(ps != null)
1326         {
1327             try
1328             {
1329                 ps.close();
1330             }
1331             catch(Exception e)
1332             {
1333                 e.printStackTrace();
1334             }
1335         }
1336         if(con != null)
1337         {
1338             try
1339             {
1340                 con.close();
1341             }
1342             catch(Exception e)
1343             {
1344                 e.printStackTrace();
1345             }
1346         }
1347     } // finally
1348     return al;
1349 } // search()
1350 } // class
1351 -----BookBean.java-----
1352 package com.nt.model;
1353
1354 public class BookBean implements java.io.Serializable
1355 {
1356     private String bookid;
1357     private String bookname;
1358     private String authorname;
1359     private String status;
1360
1361     public void setBookId(String bookid)
1362     {
1363         this.bookid = bookid;
1364     }
1365     public String getBookId()
1366     {
1367         return bookid;
1368     }
1369     public void setBookName(String bookname)
1370     {
1371         this.bookname = bookname;
1372     }
1373     public String getBookName()
1374     {
1375         return bookname;
1376     }
1377     public void setAuthorName(String authorname)
1378     {
1379         this.authorname = authorname;
1380     }
```

```
1381     public String getAuthorName()
1382     {
1383         return authorname;
1384     }
1385     public void setStatus(String status)
1386     {
1387         this.status = status;
1388     }
1389     public String getStatus()
1390     {
1391         return status;
1392     }
1393 } // BookBean class.
1394 -----HtmlPrint.jsp-----
1395 <%@page import="java.util.ArrayList, com.nt.model.BookBean" %>
1396 <%
1397     ArrayList al = (ArrayList)request.getAttribute("list");
1398     String cat = (String)request.getAttribute("category");
1399 %>
1400
1401 <html>
1402
1403     <script language='javascript'>
1404         function showprint()
1405         {
1406             frames.focus();
1407             frames.print();
1408         }
1409     </script>
1410
1411 <body>
1412     <form name='f'>
1413     <center><h2><u>
1414 Books belonging to category <%= cat.toUpperCase() %>
1415 </u></h2></center>
1416 <br>
1417
1418 <table border="1" width="100%">
1419     <tr>
1420         <th>Sno</th>
1421         <th>BookId</th>
1422         <th>BookName</th>
1423         <th>AuthorName</th>
1424         <th>Status</th>
1425     </tr>
1426     <%
1427         for(int i = 0; i < al.size(); i++)
1428         {
1429             BookBean sb=(BookBean)al.get(i);
1430         %}
1431     <tr>
1432         <td><%= (i+1) %></td>
1433         <td><%= sb.getBookId() %></td>
1434         <td><%= sb.getBookName() %></td>
1435         <td><%= sb.getAuthorName() %></td>
1436         <td><%= sb.getStatus() %></td>
1437     </tr>
1438     <%
1439         }
1440     %>
```

```
1441      </table>
1442      <center>
1443      <a href="javascript:showprint()">Print</a>
1444      </center>
1445      </form>
1446  </body>
1447  </html>
1448 -----ExcelScreen.jsp-----
1449 <%@page import="java.util.ArrayList,com.nt.model.BookBean"%>
1450 <%
1451     response.setHeader("Content-Disposition","attachment;filename=Title1.xls");
1452     response.setContentType("application/ms-excel");
1453
1454     ArrayList al = (ArrayList)request.getAttribute("list");
1455     String cat = (String)request.getAttribute("category");
1456 %>
1457 <center><h2><u>
1458     Books belonging to category <%= cat.toUpperCase() %>
1459 <u></h2></center>
1460 <br>
1461 <table border="1" width="100%">
1462     <tr>
1463         <th>Sno</th>
1464         <th>BookId</th>
1465         <th>BookName</th>
1466         <th>AuthorName</th>
1467         <th>Status</th>
1468     </tr>
1469     <%
1470         for(int i = 0; i < al.size(); i++)
1471     {
1472         BookBean sb=(BookBean)al.get(i);
1473     %>
1474         <tr>
1475             <td><%= (i+1) %></td>
1476             <td><%= sb.getBookId() %></td>
1477             <td><%= sb.getBookName() %></td>
1478             <td><%= sb.getAuthorName() %></td>
1479             <td><%= sb.getStatus() %></td>
1480         </tr>
1481     <%
1482     }
1483 %>
1484 </table>
1485 =====
1486 >>>>>>>> MiniProject on file uploading and Downloading operations>>>>>
1487 =====
1488 -----DB script.txt-----
1489 SQL> desc employereg;
1490 Name          Null?    Type
1491 -----
1492 EMP_ID           NOT NULL NUMBER(4)
1493 EMP_NAME        VARCHAR2(10)
1494 EMP_ADD         VARCHAR2(10)
1495 EMP_RESUME      VARCHAR2(50)
1496 EMP_PIC         VARCHAR2(50)
1497
1498 create table employereg (EMP_ID    NUMBER(4) primary key,
1499                           EMP_NAME   VARCHAR2(10),
1500                           EMP_ADD    VARCHAR2(10),
```

```
1501           EMP_RESUME VARCHAR2(50),  
1502           EMP_PIC   VARCHAR2(50))  
1503 -----Home.html-----  
1504 <center>  
1505   <a href="Register.jsp">Register new Employee</a>  
1506 <br><br><br>  
1507   <a href="View.jsp">view and Download Emp Details</a>  
1508 -----Register.jsp-----  
1509 body bgcolor=wheat>  
1510 <center><h1>Employee Registration Page</h1>  
1511 <form action="reg" method="post" enctype="multipart/form-data">  
1512  
1513 <table>  
1514   <tr>  
1515     <td>Employee ID</td><td><input type=text name=tid></td>  
1516   </tr>  
1517   <tr>  
1518     <td>Employee Name</td><td><input type=text name=tname></td>  
1519   </tr>  
1520   <tr>  
1521     <td>Employee Add</td><td><input type=text name=tadd></td>  
1522   </tr>  
1523   <tr>  
1524     <td>Employee photo</td><td><input type=file name=tphoto></td>  
1525   </tr>  
1526   <tr>  
1527     <td>Employee resume</td><td><input type=file name=tresume></td>  
1528   </tr>  
1529   <tr>  
1530     <td><input type=submit value=register></td>  
1531   </tr>  
1532 </form>  
1533 </center>  
1534 </body>  
1535 -----web.xml-----  
1536 <web-app>  
1537  
1538   <servlet>  
1539     <servlet-name>reg</servlet-name>  
1540     <servlet-class>com.nt.RegisterServlet</servlet-class>  
1541   </servlet>  
1542  
1543   <servlet-mapping>  
1544     <servlet-name>reg</servlet-name>  
1545     <url-pattern>/reg</url-pattern>  
1546   </servlet-mapping>  
1547  
1548   <welcome-file-list>  
1549     <welcome-file>Home.html</welcome-file>  
1550   </welcome-file-list>  
1551  
1552 </web-app>  
1553 -----RegisterServlet.java-----  
1554 //RegisterServlet.java  
1555 package com.nt;  
1556 import java.io.IOException;  
1557 import java.io.PrintWriter;  
1558 import java.sql.Connection;  
1559 import java.sql.DriverManager;  
1560 import java.sql.PreparedStatement;
```

```
1561 import java.util.ArrayList;
1562 import java.util.Vector;
1563
1564 import javax.servlet.ServletException;
1565 import javax.servlet.http.HttpServlet;
1566 import javax.servlet.http.HttpServletRequest;
1567 import javax.servlet.http.HttpServletResponse;
1568
1569 import javazoom.upload.MultipartFormDataRequest;
1570 import javazoom.upload.UploadBean;
1571 import javazoom.upload.UploadParameters;
1572
1573 public class RegisterServlet extends HttpServlet
1574 {
1575     public void doPost (HttpServletRequest req, HttpServletResponse res)
1576             throws ServletException, IOException
1577     {
1578
1579         String resumePath="c:/store/resumes";
1580         String photoPath="c:/store/photoes";
1581         PrintWriter out = res.getWriter();
1582         try
1583         {
1584             // get special request obj given by Java zoom api
1585             MultipartFormDataRequest nreq = new MultipartFormDataRequest(req);
1586             int eId=Integer.parseInt(nreq.getParameter("tid"));
1587             String eName=nreq.getParameter("tname");
1588             String eAdd=nreq.getParameter("tadd");
1589             /*String ePhoto=nreq.getParameter("tphoto"); shows Null
1590             String eResume=nreq.getParameter("tresume"); shows Null */
1591
1592             // settings related to Resume uploading
1593             UploadBean upb = new UploadBean();
1594             upb.setFolderstore(resumePath);
1595             upb.setOverwrite(false);
1596             upb.store(nreq,"tresume");// complets files uploading
1597
1598             // setting related photo uploading
1599             upb.setFolderstore(photoPath);
1600             upb.setOverwrite(false);
1601             upb.store(nreq,"tphoto"); complets files uploading
1602
1603             // get the file names of the uploaded files
1604             Vector history = upb.getHistory();
1605
1606             ArrayList <String> filesName=new ArrayList<String>();
1607             for (int i=0;i<history.size();i++)
1608             {
1609                 UploadParameters up = (UploadParameters) history.elementAt(i);
1610                 filesName.add(up.getFilename());
1611             }
1612             System.out.println("resume"+filesName.get(0));
1613             System.out.println("photo"+filesName.get(1));
1614
1615             // store the paths of the uploaded files to c:\store folder
1616             Class.forName("oracle.jdbc.driver.OracleDriver");
1617             Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
1618             PreparedStatement ps=con.prepareStatement("insert into EmployeeReg values(?, ?, ?, ?, ?)");
1619             ps.setInt(1,eId);
1620             ps.setString(2, eName);
```

```
1621     ps.setString(3, eAdd);
1622     ps.setString(4, resumePath + "/" + fileName.get(0));
1623     ps.setString(5, photoPath + "/" + fileName.get(1));
1624
1625     int i=ps.executeUpdate();
1626     if(i==1)
1627         out.println("Successfully uploaded and Stored in DataBase");
1628     else
1629         out.println("Failed in uploading");
1630
1631 } //try
1632 catch(Exception e)
1633 {
1634     out.println(e);
1635 } //catch
1636 } //doPost()
1637 } //class
1638
1639 -----View.jsp-----
1640 <%@page import="java.io.File,java.util.* ,java.sql.*"%>
1641
1642 <!-- Retrive records from DB table -->
1643 <H1>List of All files under C:\store</H1>
1644 <%
1645 Class.forName("oracle.jdbc.driver.OracleDriver");
1646 Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
1647 PreparedStatement ps=con.prepareStatement("select * from EmployeeReg");
1648 ResultSet rs=ps.executeQuery();
1649 %>
1650
1651 <!-- display employee details as html table content-->
1652 <body bgcolor=wheat>
1653 <table border=1>
1654 <tr><td>Employee Name</td><td>Employee Address</td><td>Employee Resume</td><td>Employee Photo</td></tr>
1655
1656 <%
1657 while(rs.next())
1658 {
1659 %>
1660     <tr>
1661         <td><%=rs.getString(2)%></td>
1662         <td><%=rs.getString(3)%></td>
1663         <td><a href='Download.jsp?resumeId=<%=rs.getString(1)%>'>Download Here</a></td>
1664         <td><a href='Download.jsp?photoId=<%=rs.getString(1)%>'>Download Here</a></td>
1665     </tr>
1666 <%}>
1667
1668 </table>
1669 </body>
1670 -----Download.jsp-----
1671 <%@page import="java.io.* ,javax.servlet.* ,javax.servlet.http.* ,java.sql.* "%>
1672
1673 <!-- get the path of the file to downloaded -->
1674 <%
1675 String fileName="";
1676 String queryText="";
1677
1678 Class.forName("oracle.jdbc.driver.OracleDriver");
1679 Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
1680
```

```
1681 if(request.getParameter("resumeId")!=null)
1682 queryText="select emp_resume from employeereg where emp_id="+request.getParameter("resumeId");
1683 else
1684     queryText="select emp_pic from employeereg where emp_id="+request.getParameter("photoId");
1685
1686 PreparedStatement ps=con.prepareStatement(queryText);
1687
1688 ResultSet rs=ps.executeQuery();
1689 while(rs.next()){
1690     fileName=rs.getString(1);
1691 }
1692 // prepare settings to download the files
1693 File f= new File(fileName);
1694 int length = 0;
1695 ServletOutputStream op = response.getOutputStream();
1696
1697 String mimetype = application.getMimeType(fileName);
1698
1699 response.setContentType( (mimetype != null) ? mimetype : "application/octet-stream" );
1700
1701 response.setContentLength( (int)f.length() );
1702
1703 response.setHeader( "Content-Disposition", "attachment;filename="+fileName );
1704
1705 // comple file downloading using buffering
1706 byte[] buf = new byte[1024];
1707 DataInputStream in = new DataInputStream(new FileInputStream(f));
1708
1709 while ((in != null) && ((length = in.read(buf)) != -1))
1710 {
1711     op.write(buf,0,length);
1712 }
1713 //close streams
1714 in.close();
1715 op.flush();
1716 op.close();
1717
1718 %>
1719
```

## **JavaServer Pages Standard Tag Library**

The JavaServer Pages Standard Tag Library (JSTL) encapsulates core functionality common to many JSP applications. Instead of mixing tags from numerous vendors in your JSP applications, JSTL allows you to employ a single, standard set of tags. This standardization allows you to deploy your applications on any JSP container supporting JSTL and makes it more likely that the implementation of the tags is optimized.

JSTL has tags such as iterators and conditionals for handling flow control, tags for manipulating XML documents, internationalization tags, tags for accessing databases using SQL, and commonly used functions.

### **Using JSTL**

JSTL includes a wide variety of tags that fit into discrete functional areas. To reflect this, as well as to give each area its own namespace, JSTL is exposed as multiple tag libraries. The URIs for the libraries are as follows:

*Core:* <http://java.sun.com/jsp/jstl/core>  
*XML:* <http://java.sun.com/jsp/jstl/xml>  
*Internationalization:* <http://java.sun.com/jsp/jstl/fmt>  
*SQL:* <http://java.sun.com/jsp/jstl/sql>  
*Functions:* <http://java.sun.com/jsp/jstl/functions>

Table 14-2 summarizes these functional areas along with the prefixes used in this tutorial.

**Table 14-2 JSTL Tags**

Area	Subfunction	Prefix
Core	Variable support	c
	Flow control	
	URL management	
	Miscellaneous	
XML	Core	x
	Flow control	
	Transformation	
I18n	Locale	fmt
	Message formatting	
	Number and date formatting	
Database	SQL	sql
Functions	Collection length	fn
	String manipulation	

**JSTL Core:****Standard Syntax:**

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

<b><u>catch</u></b>	Catches any Throwable that occurs in its body and optionally exposes it.
<b><u>choose</u></b>	Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by <when> and <otherwise>
<b><u>if</u></b>	Simple conditional tag, which evaluates its body if the supplied condition is true and optionally exposes a Boolean scripting variable representing the evaluation of this condition
<b><u>import</u></b>	Retrieves an absolute or relative URL and exposes its contents to either the page, a String in 'var', or a Reader in 'varReader'.
<b><u>forEach</u></b>	The basic iteration tag, accepting many different collection types and supporting subsetting and other functionality
<b><u>forTokens</u></b>	Iterates over tokens, separated by the supplied delimiters
<b><u>out</u></b>	Like <%= ... >, but for expressions.
<b><u>otherwise</u></b>	Subtag of <choose> that follows <when> tags and runs only if all of the prior conditions evaluated to 'false'
<b><u>param</u></b>	Adds a parameter to a containing 'import' tag's URL.
<b><u>redirect</u></b>	Redirects to a new URL.
<b><u>remove</u></b>	Removes a scoped variable (from a particular scope, if specified).
<b><u>set</u></b>	Sets the result of an expression evaluation in a 'scope'
<b><u>url</u></b>	Creates a URL with optional query parameters.
<b><u>when</u></b>	Subtag of <choose> that includes its body if its condition evaluates to 'true'

**JSTL fmt:****Standard Syntax:**

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
```

<b><u>requestEncoding</u></b>	Sets the request character encoding
<b><u>setLocale</u></b>	Stores the given locale in the locale configuration variable
<b><u>timeZone</u></b>	Specifies the time zone for any time formatting or parsing actions nested in its body
<b><u>setTimeZone</u></b>	Stores the given time zone in the time zone configuration variable
<b><u>bundle</u></b>	Loads a resource bundle to be used by its tag body
<b><u>setBundle</u></b>	Loads a resource bundle and stores it in the named scoped variable or the bundle configuration variable
<b><u>message</u></b>	Maps key to localized message and performs parametric replacement
<b><u>param</u></b>	Supplies an argument for parametric replacement to a containing <message> tag
<b><u>formatNumber</u></b>	Formats a numeric value as a number, currency, or percentage
<b><u>parseNumber</u></b>	Parses the string representation of a number, currency, or percentage
<b><u>formatDate</u></b>	Formats a date and/or time using the supplied styles and pattern

**parseDate** Parses the string representation of a date and/or time

**JSTL Sql:****Standard Syntax:**

```
<% taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
```

<b><u>transaction</u></b>	Provides nested database action elements with a shared Connection, set up to execute all statements as one transaction.
<b><u>query</u></b>	Executes the SQL query defined in its body or through the sql attribute.
<b><u>update</u></b>	Executes the SQL update defined in its body or through the sql attribute.
<b><u>param</u></b>	Sets a parameter in an SQL statement to the specified value.
<b><u>dateParam</u></b>	Sets a parameter in an SQL statement to the specified java.util.Date value.
<b><u>setDataSource</u></b>	Creates a simple DataSource suitable only for prototyping.

**JSTL XML:****Standard Syntax:**

```
<% taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
```

<b><u>choose</u></b>	Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by <when> and <otherwise>
<b><u>out</u></b>	Like <%= ... >, but for XPath expressions.
<b><u>if</u></b>	XML conditional tag, which evaluates its body if the supplied XPath expression evaluates to 'true' as a boolean
<b><u>forEach</u></b>	XML iteration tag.
<b><u>otherwise</u></b>	Subtag of <choose> that follows <when> tags and runs only if all of the prior conditions evaluated to 'false'
<b><u>param</u></b>	Adds a parameter to a containing 'transform' tag's Transformer
<b><u>parse</u></b>	Parses XML content from 'source' attribute or 'body'
<b><u>set</u></b>	Saves the result of an XPath expression evaluation in a 'scope'
<b><u>transform</u></b>	Conducts a transformation given a source XML document and an XSLT stylesheet
<b><u>when</u></b>	Subtag of <choose> that includes its body if its expression evaluates to 'true'

**JSTL functions:****Standard Syntax:**

```
<% taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
```

boolean **contains**( java.lang.String, java.lang.String)

Tests if an input string contains the specified substring.

boolean **containsIgnoreCase**( java.lang.String, java.lang.String)

Tests if an input string contains the specified substring in a case insensitive way.

boolean **endsWith**( java.lang.String, java.lang.String)

Tests if an input string ends with the specified

		<b>suffix.</b>
java.lang.String	<b><u>escapeXml(</u></b> java.lang.String)	Escapes characters that could be interpreted as XML markup.
int	<b><u>indexOf(</u></b> java.lang.String, java.lang.String)	Returns the index within a string of the first occurrence of a specified substring.
java.lang.String	<b><u>join(</u></b> java.lang.String[], java.lang.String)	Joins all elements of an array into a string.
int	<b><u>length(</u></b> java.lang.Object)	Returns the number of items in a collection, or the number of characters in a string.
java.lang.String	<b><u>replace(</u></b> java.lang.String, java.lang.String, java.lang.String)	Returns a string resulting from replacing in an input string all occurrences of a "before" string into an "after" substring.
java.lang.String[]	<b><u>split(</u></b> java.lang.String, java.lang.String).	Splits a string into an array of substrings.
boolean	<b><u>startsWith(</u></b> java.lang.String, java.lang.String)	Tests if an input string starts with the specified prefix.
java.lang.String	<b><u>substring(</u></b> java.lang.String, int, int)	Returns a subset of a string.
java.lang.String	<b><u>substringAfter(</u></b> java.lang.String, java.lang.String)	Returns a subset of a string following a specific substring.
java.lang.String	<b><u>substringBefore(</u></b> java.lang.String, java.lang.String)	Returns a subset of a string before a specific substring.
java.lang.String	<b><u>toLowerCase(</u></b> java.lang.String)	Converts all of the characters of a string to lower case.
java.lang.String	<b><u>toUpperCase(</u></b> java.lang.String)	Converts all of the characters of a string to upper case.
java.lang.String	<b><u>trim(</u></b> java.lang.String)	Removes white spaces from both ends of a string.

```
1 =====
2 Custom Jsp Taglibrary Development
3 =====
4 -----web.xml-----
5 <web-app>
6   <taglib>
7     <taglib-uri>demo</taglib-uri>
8     <taglib-location>/WEB-INF/first.tld</taglib-location>
9   </taglib>
10  </web-app>
11 -----Test.jsp-----
12 <%@ taglib uri="demo" prefix="start"%>
13 <html>
14   <head>
15     <title>Tag Test !!!.....</title>
16   </head>
17   <body>
18     <center><h2>
19       <start:example/>
20       <br>
21       <start:prime />
22         <br>
23       <start:prime n="30" />
24       <br>
25       <start:display font="webdings" >
26       <br>JAVA powered by
27       </start:display>
28       <br>
29       <start:display font="arial" size="56">
30         HCL Technologies
31       </start:display>
32     </h2></center>
33   </body>
34 </html>
35 -----first.tld-----
36 <?xml version="1.0" encoding="UTF-8"?>
37 <!DOCTYPE taglib
38 PUBLIC "-//Sun Microsystems, Inc./DTD JSP Tag Library 1.2//EN"
39   "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">
40
41 <taglib>
42   <tlibversion>1.0</tlibversion>
43   <jspversion>1.2</jspversion>
44   <shortname>start</shortname>
45   <info>
46     An example Tag Library
47   </info>
48   <tag>
49     <name>example</name>
50     <tagclass>tags.ExampleTag</tagclass>
51     <info>Simplest example: inserts one line of output</info>
52     <bodycontent>EMPTY</bodycontent>
53   </tag>
54
55   <tag>
56     <name>prime</name>
57     <tagclass>tags.PrimeTag</tagclass>
58     <info>Prime numbers generation</info>
59     <bodycontent>EMPTY</bodycontent>
60     <attribute>
```

```
61          <name>n</name>
62          <required>false</required>
63      </attribute>
64  </tag>
65
66  <tag>
67      <name>display</name>
68      <tagclass>tags.DisplayTag</tagclass>
69      <info>Displaying text in different fonts</info>
70      <bodycontent>Jsp</bodycontent>
71      <attribute>
72          <name>font</name>
73          <required>true</required>
74      </attribute>
75      <attribute>
76          <name>size</name>
77          <required>false</required>
78      </attribute>
79  </tag>
80
81 </taglib>
82 -----ExampleTag.java-----
83 package tags;
84
85 import javax.servlet.jsp.*;
86 import javax.servlet.jsp.tagext.*;
87 import java.io.*;
88
89 public class ExampleTag extends TagSupport
90 {
91     public int doStartTag()
92     {
93         System.out.println("Inside doStartTag() ExampleTag");
94         try
95         {
96             JspWriter out=pageContext.getOut();
97             out.print("Prime numbers"+<br>);
98         }
99         catch(IOException e)
100        {
101            System.out.println("Error in ExampleTag: "+ e);
102        }
103        return(SKIP_BODY);
104    }//doStartTag()
105    public int doEndTag()
106    {
107        System.out.println("Inside doEndTag() ExampleTag");
108        return EVAL_PAGE;
109    }//doEndTag()
110 } //class
111 -----PrimeTag.java-----
112 package tags;
113
114 import javax.servlet.jsp.*;
115 import javax.servlet.jsp.tagext.*;
116 import java.io.*;
117
118 public class PrimeTag extends TagSupport
119 {
120     private int n=10;
```

```
121
122     public void setN(int n)
123     {
124         this.n=n;
125     }
126     public int getN()
127     {
128         return n;
129     }
130
131     private boolean isPrime(int x)
132     {
133         for (int k=2;k<x;k++)
134         {
135             if(x%k==0)
136                 return false;
137         }//for
138         return true;
139     }//isPrime()
140     public int doStartTag()
141     {
142         System.out.println("Inside doStartTag() of PrimeTag");
143         try
144         {
145             JspWriter out=pageContext.getOut();
146             for(int i=1;i<n;i++)
147             {
148                 if(isPrime(i))
149                     out.print(i+" ");
150             }//for
151         }//try
152         catch(IOException ie)
153         {
154             ie.printStackTrace();
155         }
156         return SKIP_BODY;
157     }//doStartTag()
158
159     public int doEndTag()
160     {
161         System.out.println("Inside doEndTag() of PrimeTag");
162         return EVAL_PAGE;
163     }//doEndTag()
164 } //class
165 -----DisplayTag.java-----
166 package tags;
167
168 import javax.servlet.*;
169 import javax.servlet.jsp.*;
170 import javax.servlet.jsp.tagext.*;
171 import java.io.*;
172
173 public class DisplayTag extends TagSupport
174 {
175     private String font;
176     private int size=20;
177
178     public void setFont(String font)
179     {
180         this.font=font;
```



```
241 -----web.xml-----
242 <?xml version="1.0" encoding="ISO-8859-1"?>
243 <!DOCTYPE web-app
244     PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
245     "http://java.sun.com/dtd/web-app_2_3.dtd">
246
247 <web-app>
248     <welcome-file-list>
249         <welcome-file>Main.html</welcome-file>
250     </welcome-file-list>
251
252     <servlet>
253         *
254         <servlet-name>insert</servlet-name>
255         <servlet-class>InsertDemo</servlet-class>
256     </servlet>
257     <servlet-mapping>
258         <servlet-name>insert</servlet-name>
259         <url-pattern>/insert</url-pattern>
260     </servlet-mapping>
261
262     <servlet>
263         <servlet-name>delete</servlet-name>
264         <servlet-class>DeleteDemo</servlet-class>
265     </servlet>
266     <servlet-mapping>
267         <servlet-name>delete</servlet-name>
268         <url-pattern>/delete</url-pattern>
269     </servlet-mapping>
270
271     <servlet>
272         <servlet-name>update</servlet-name>
273         <servlet-class>UpdateDemo</servlet-class>
274     </servlet>
275     <servlet-mapping>
276         <servlet-name>update</servlet-name>
277         <url-pattern>/update</url-pattern>
278     </servlet-mapping>
279
280 </web-app>
281 -----Main.html-----
282 <html>
283 <frameset rows = "15%,*" border = 0>
284 <frame src = "Info.html" name = "info" scrolling = no>
285 <frameset cols = "25%,*>
286 <frame src = "Links.html" name = "links">
287 <frame src = "" name = "display">
288 </frameset>
289 </frameset>
290 </html>
291
292 -----info.html-----
293 <html>
294 <body>
295 <form name = f>
296 <center><span style = "width=500;height=400;filter:shadow(color=blue,direction=135)">
297 <font color = lavender size = 5>HCL TECHNOLOGIES</font>
298 <hr color = pink width = 60%>
299 </span></center>
300 </form>
```

```
301 </body>
302 </html>
303 -----Links.html-----
304 <html>
305 <body>
306 <form name = f>
307 <div style = "position:absolute;left:80;top:70">
308 <a href = "Insert.jsp" target = "display">INSERT</a>
309 </div>
310 <div style = "position:absolute;left:80;top:140">
311 <a href = "Delete.jsp" target = "display"> DELETE</a>
312 </div>
313 <div style = "position:absolute;left:80;top:210">
314 <a href = "Update.jsp" target = "display">UPDATE</a>
315 </div>
316 </form>
317 </body>
318 </html>
319 -----Insert.jsp-----
320 <html>
321 <script src = "insert.js" language = "javascript">
322 </script>
323 <body>
324 <form name = f action = './insert' method = 'post'
325     onSubmit = "return isValid(this)">
326 <table border = 1 cellpadding = 7 cellspacing = 7 align = center>
327 <caption><i><font size = 4><u>Insert Records</u></font></i></caption>
328 <tr>
329 <th>Student ID<th><input type = "text" name = "stu_id"></tr>
330 <tr>
331 <th>Student NAME<th><input type = "text" name = "stu_name"></tr>
332 <tr>
333 <th>Student Address<th><input type = "text" name = "stu_add"></tr>
334 <tr>
335 <th>click<th><input type = "submit" value = " INSERT "></tr>
336 </table>
337 </form>
338 </body>
339 </html>
340 -----insert.js-----
341 function isValid(frm)
342 {
343     var sid = frm.stu_id;
344     var snm = frm.stu_name;
345     var sadd = frm.stu_add;
346     if(check(sid) && isNumber(sid) && check(snm) && check(sadd))
347     {
348         return true;
349     }
350     else
351     {
352         return false;
353     }
354 }
355 function check(cmd)
356 {
357     if(cmd.value == "")
358     {
359         alert(cmd.name + " " + "Field Missed");
360         cmd.focus();
```

```
361         return false;
362     }
363     return true;
364 }
365 function isNumber(cmd)
366 {
367     if(isNaN(cmd.value))
368     {
369         alert("Stu_ID should be numeric value");
370         cmd.focus();
371         return false;
372     }
373     return true;
374 }
375 -----InsertDemo.java-----
376 import javax.servlet.*;
377 import javax.servlet.http.*;
378 import java.io.*;
379 import java.sql.*;
380
381 public class InsertDemo extends HttpServlet
382 {
383     public void doPost(HttpServletRequest req,HttpServletResponse res)
384     {
385         Connection con = null;
386         PreparedStatement ps = null;
387         int rs = 0;
388         try
389         {
390             PrintWriter pw = res.getWriter();
391             res.setContentType("text/html");
392             pw.println("<html><form target = 'display'>");
393             int sid = Integer.parseInt(req.getParameter("stu_id"));
394             String sname = req.getParameter("stu_name");
395             String sadd = req.getParameter("stu_add");
396             Class.forName("oracle.jdbc.driver.OracleDriver");
397             con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
398             System.out.println("connection");
399             ps = con.prepareStatement("insert into student_info values(?, ?, ?)");
400             ps.setInt(1,sid);
401             ps.setString(2,sname);
402             ps.setString(3,sadd);
403             rs = ps.executeUpdate();
404             if(rs!=1)
405                 pw.println("<h2>Record is problem</h2>");
406             else
407                 pw.println("<h2 style = 'position:absolute;left:50;top:50'>One Record Inserted Successfully</h2>");
408
409             ps.close();
410             con.close();
411         }
412         catch(Exception e)
413         {
414             e.printStackTrace();
415         }
416     }
417     public void doGet(HttpServletRequest req,HttpServletResponse res)
418     {
419         try
420         {
```

```
421         doPost(req,res);
422     }
423     catch(Exception e)
424     {
425         e.printStackTrace();
426     }
427 }
428 }  
429 -----Delete.jsp-----  
430 <html>
431 <script language = 'javascript'>
432 function isValid()
433 {
434     if(f.stu_id.value == "")
435     {
436         alert(f.stu_id.name + " " + "Field Missed");
437         f.stu_id.focus();
438         return false;
439     }
440     if(isNaN(f.stu_id.value))
441     {
442         alert("Stu_ID should be numeric value");
443         f.stu_id.focus();
444         return false;
445     }
446     return true;
447 }
448 </script>
449 <body>
450 <form name = f action = './delete' method = 'post' onSubmit ='return isValid()'>
451 <table border = 1 cellpadding = 7 cellspacing = 7 align = center>
452 <caption><i><font size = 4><u>Delete Records</u></font></i></caption>
453 <tr>
454 <th>Student ID<th><input type = "text" name = "stu_id"></tr>
455 <tr>
456 <th>click<th><input type = "submit" value = " DELETE "></tr>
457 </table>
458 </form>
459 </body>
460 </html>  
461 -----DeleteDemo.java-----  
462 import javax.servlet.*;
463 import javax.servlet.http.*;
464 import java.io.*;
465 import java.sql.*;
466
467 public class DeleteDemo extends HttpServlet
468 {
469     public void doPost(HttpServletRequest req,HttpServletResponse res)
470     {
471         Connection con = null;
472         PreparedStatement ps = null;
473         int rs = 0;
474         try
475         {
476             PrintWriter pw = res.getWriter();
477             res.setContentType("text/html");
478             pw.println("<html><form target = 'display'>");
479             int sid = Integer.parseInt(req.getParameter("stu_id"));
480             Class.forName("oracle.jdbc.driver.OracleDriver");
```

```
481         con = DriverManager.getConnection
482             ("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
483             ps = con.prepareStatement("delete from student_info where stu_id=?");
484             ps.setInt(1,sid);
485             rs = ps.executeUpdate();
486             if(rs!=1)
487                 pw.println("<h2>Student id is problem</h2>");
488             else
489                 pw.println("<h2 style = 'position:absolute;left:50;top:50'>
490                             One Record Deleted</h2>");
491
492             ps.close();
493             con.close();
494         }
495         catch(Exception e)
496         {
497             e.printStackTrace();
498         }
499     }
500     public void doGet(HttpServletRequest req,HttpServletResponse res)
501     {
502         try{
503             doPost(req,res);
504         }
505         catch(Exception e)
506         {
507             e.printStackTrace();
508         }
509     }
510 }
511 -----Update.jsp-----
512 <html>
513 <script src = "insert.js" language = "javascript">
514 </script>
515 <body>
516 <form name = f action = './update' method = 'post' onSubmit = "return isValid(this)">
517 <table border = 1 cellpadding = 7 cellspacing = 7 align = center>
518 <caption><i><font size = 4><u>Update Records</u></font></i></caption>
519 <tr>
520 <th>Student ID<th><input type = "text" name = "stu_id"></tr>
521 <tr>
522 <th>Student NAME<th><input type = "text" name = "stu_name"></tr>
523 <tr>
524 <th>Student Address<th><input type = "text" name = "stu_add"></tr>
525 <tr>
526 <th>click<th><input type = "submit" value = " UPDATE "></tr>
527 </table>
528 </form>
529 </body>
530 </html>
531 -----UpdateDemo.java-----
532 import javax.servlet.*;
533 import javax.servlet.http.*;
534 import java.io.*;
535 import java.sql.*;
536
537 public class UpdateDemo extends HttpServlet
538 {
539     public void doPost(HttpServletRequest req,HttpServletResponse res)
540     {
```

```

541     Connection con = null;
542     PreparedStatement ps = null;
543     int rs = 0;
544     try
545     {
546         PrintWriter pw = res.getWriter();
547         res.setContentType("text/html");
548         pw.println("<html><form target = 'display'>");
549         int sid = Integer.parseInt(req.getParameter("stu_id"));
550         String sname = req.getParameter("stu_name");
551         String sadd = req.getParameter("stu_add");
552         Class.forName("oracle.jdbc.driver.OracleDriver");
553         con = DriverManager.getConnection
554             ("jdbc:oracle:thin:@localhost:1521:orcl", "scott", "tiger");
555         ps = con.prepareStatement
556             ("update student_info set stu_name=? ,stu_add=? where stu_id=?");
557         ps.setString(1,sname);
558         ps.setString(2,sadd);
559         ps.setInt(3,sid);
560         rs = ps.executeUpdate();
561         if(rs!=1)
562             pw.println("<h2>Student ID problem</h2>");
563         else
564             pw.println("<h2 style = 'position:absolute;left:50;top:50'>
565                         One Record Updated</h2>");
566
567         ps.close();
568         con.close();
569     }
570     catch(Exception e)
571     {
572         e.printStackTrace();
573     }
574 }
575 public void doGet(HttpServletRequest req,HttpServletResponse res)
576 {
577     try
578     {
579         doPost(req,res);
580     }
581     catch(Exception e)
582     {
583         e.printStackTrace();
584     }
585 }
586 }
587 =====
588 Main Example (Mini Project2)
589 =====
590 -----Database Details-----
591 SQL>drop table STUDENT_INFO cascade;
592 SQL>create table STUDENT_INFO(STU_ID number,STU_NAME varchar2(30),STU_ADD varchar2(30));
593 -----web.xml-----
594 <?xml version="1.0" encoding="ISO-8859-1"?>
595 <!DOCTYPE web-app
596 PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
597 "http://java.sun.com/dtd/web-app_2_3.dtd">
598
599 <web-app>
600     <servlet>

```

```
601      <servlet-name>First</servlet-name>
602      <servlet-class>StudentDemo</servlet-class>
603    </servlet>
604    <servlet-mapping>
605      <servlet-name>First</servlet-name>
606      <url-pattern>/studentdemo</url-pattern>
607    </servlet-mapping>
608
609    <welcome-file-list>
610      <welcome-file>StudentDetails.jsp</welcome-file>
611    </welcome-file-list>
612  </web-app>
613 -----StudentDetails.jsp-----
614 <%
615   String res =(String)request.getAttribute("msg");
616   String id = (String)request.getAttribute("no");
617   String name = (String)request.getAttribute("name");
618   String addr = (String)request.getAttribute("address");
619 %>
620
621 <html>
622   <script language = 'javascript'>
623
624     function isInsert()
625     {
626       f.setVal.value = "insert";
627       check();
628     }
629     function isDelete()
630     {
631       f.setVal.value = "delete";
632       check1();
633     }
634     function isUpdate()
635     {
636       f.setVal.value = "update";
637       check();
638     }
639     function isDisplay()
640     {
641       f.setVal.value = "display";
642       check1();
643     }
644
645     function check()
646     {
647       if(f.stu_id.value == "")
648       {
649         alert("Please provide Student ID");
650         f.stu_id.focus();
651       }
652       else if(isNaN(f.stu_id.value))
653       {
654         alert("Student ID should be numeric");
655         f.stu_id.value = "";
656         f.stu_id.focus();
657       }
658       else if(f.stu_name.value == "")
659       {
660         alert("Please provide Student Name");
```

```
661         f.stu_name.focus();
662     }
663     else if(f.stu_add.value == "")
664     {
665         alert("Please provide Student Address");
666         f.stu_add.focus();
667     }
668     else
669     {
670         f.submit();
671     }
672     function check1()
673     {
674         if(f.stu_id.value == "")
675         {
676             alert("Please provide Student ID");
677             f.stu_id.focus();
678         }
679         else if(isNaN(f.stu_id.value))
680         {
681             alert("Student ID should be numeric");
682             f.stu_id.value = "";
683             f.stu_id.focus();
684         }
685         else
686         {
687             f.submit();
688         }
689     function disp1_output()
690     {
691         var i = <%=id%>;
692         var n ='<%=name%>';
693         var a ='<%=addr%>';
694         f.stu_id.value = i;
695         f.stu_name.value=n;
696         f.stu_add.value = a;
697     }
698     function isReloading()
699     {
700         <%
701         if(res != null)
702         {
703             if(res.equals("1"))
704                 out.println("alert('Information stored successfully')");
705             else if(res.equals("2"))
706                 out.println("alert('Given Student ID already exists')");
707             else if(res.equals("3"))
708                 out.println("alert('Information deleted successfully')");
709             else if(res.equals("4"))
710                 out.println("alert('Given Student ID is Invalid')");
711             else if(res.equals("5"))
712                 out.println("alert('Information updated successfully')");
713             else if(res.equals("6"))
714             {
715                 out.println("disp1_output()");
716             }
717             else if(res.equals("7"))
718                 out.println("alert('Unknown problem occurred.')");
719         }
720     }
```

```
721      %>
722  }
723
724  </script>
725
726 <body onLoad='isReloading()'>
727   <form name=f action='./studentdemo' method='post'>
728
729   <center>
730     <span style= "width=500;height=60;filter:shadow(color=blue,direction=135)">
731       <font color=lavender size=5>HCL TECHNOLOGIES</font>
732       <hr color=pink width=100%>
733     </span></center>
734
735   <table border=1 cellpadding=7 cellspacing=7 align=center>
736
737   <caption>
738     <font size=6><i><b><u>Student Table</u></b></i></font>
739   </caption>
740
741   <tr>
742     <th>Student ID</th>
743     <th><input type="text" name="stu_id"></th>
744   </tr>
745   <tr>
746     <th>Student Name</th>
747     <th><input type="text" name="stu_name"></th>
748   </tr>
749   <tr>
750     <th>Student Address</th>
751     <th><input type="text" name="stu_add"></th>
752   </tr>
753   <tr>
754     <th colspan=4>
755       <input type='button' value='Insert' onClick='isInsert()'>
756       <input type='button' value='Delete' onClick='isDelete()'>
757       <input type='button' value='Update' onClick='isUpdate()'>
758       <input type='button' value='Display' onClick='isDisplay()'>
759     </th>
760   </tr>
761 </table>
762
763 <input type='text' name='setVal' readonly
764   style='visibility:hidden'>
765 </form>
766 </body>
767 </html>
768 -----StudentDemo.java-----
769 import javax.servlet.*;
770 import javax.servlet.http.*;
771 import java.io.*;
772 import beans.DbConnector;
773
774 public class StudentDemo extends HttpServlet
775 {
776   public void doPost(HttpServletRequest req,HttpServletResponse res)
777   {
778     HttpSession session = null;
779     int flag = 0;
780     int forward = 0;
```

```
781     int output = 0;
782
783     String store = null;
784     String value = null;
785     String dispval1,dispval2,dispval3;
786
787     DbConnector dbc;
788
789     try
790     {
791         session = req.getSession(true);
792         PrintWriter out = res.getWriter();
793         res.setContentType("text/html");
794
795         int s_id = Integer.parseInt(req.getParameter("stu_id"));
796         String s_name = req.getParameter("stu_name");
797         String s_add = req.getParameter("stu_add");
798
799         String checkAction = req.getParameter("setVal");
800
801         dbc = new DbConnector(s_id,s_name,s_add);
802
803         if(checkAction.equals("insert"))
804             output = dbc.insert();
805         else if(checkAction.equals("delete"))
806             output = dbc.delete();
807         else if(checkAction.equals("update"))
808             output = dbc.update();
809         else if(checkAction.equals("display"))
810         {
811             output = dbc.display();
812
813             dispval1 = String.valueOf(dbc.getID());
814             dispval2 = dbc.getName();
815             dispval3 = dbc.getAdd();
816
817             req.setAttribute("no", dispval1);
818             req.setAttribute("name", dispval2);
819             req.setAttribute("address", dispval3);
820         }
821
822         value = String.valueOf(output);
823         req.setAttribute("msg", value);
824         RequestDispatcher rd = req.getRequestDispatcher("StudentDetails.jsp");
825         rd.forward(req,res);
826     } // try
827     catch(Exception e)
828     {
829         e.printStackTrace();
830     }
831 } // doPost()
832 public void doGet(HttpServletRequest req,HttpServletResponse res)
833 {
834     try{
835         doPost(req,res);
836     }
837     catch(Exception e)
838     {
839         e.printStackTrace();
840     }
}
```

```
841     } // doGet()
842 } // class
843 -----DbConnector.java-----
844 package beans;
845
846 import java.io.*;
847 import java.sql.*;
848
849 public class DbConnector
850 {
851     private Connection con = null;
852
853     private int id;
854     private String name;
855     private String add;
856
857     public DbConnector()
858     {
859     }
860
861     public DbConnector(int id, String name, String add)
862     {
863         this.id = id;
864         this.name = name;
865         this.add = add;
866
867         con = getConnection();
868     }
869
870     public void setID(int id)
871     {
872         this.id = id;
873     }
874
875     public int getID()
876     {
877         return id;
878     }
879
880     public void setName(String name)
881     {
882         this.name = name;
883     }
884
885     public String getName()
886     {
887         return name;
888     }
889
890     public void setAdd(String add)
891     {
892         this.add = add;
893     }
894
895     public String getAdd()
896     {
897         return add;
898     }
899
900     public Connection getConnection()
```

```
901     {
902         try
903         {
904             Class.forName("oracle.jdbc.driver.OracleDriver");
905             con=DriverManager.getConnection
906                 ("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
907             }catch(Exception e)
908             {
909                 e.printStackTrace();
910             }
911             return con;
912     }
913
914     public boolean exists(int id)
915     {
916         PreparedStatement ps = null;
917         try
918         {
919             ps = con.prepareStatement("SELECT COUNT(*) FROM STUDENT_INFO " +
920                             " WHERE STU_ID = ?");
921             ps.setInt(1, id);
922
923             ResultSet rs = ps.executeQuery();
924             rs.next();
925
926             if(rs.getInt(1) == 0)
927                 return false;
928             else
929                 return true;
930         }
931         catch(Exception e)
932         {
933             e.printStackTrace();
934             return false;
935         }
936         finally
937         {
938             if(ps != null)
939             {
940                 try
941                 {
942                     ps.close();
943                 }
944                 catch(Exception e)
945                 {
946                     e.printStackTrace();
947                 }
948             }
949         }
950     } // exists()
951
952     public int insert()
953     {
954         PreparedStatement ps = null;
955         try
956         {
957             boolean alreadyPresent = exists(this.getID());
958
959             if(! alreadyPresent)
960             {
```

```
961         ps = con.prepareStatement("INSERT INTO STUDENT_INFO VALUES(?,?,?)");
962         ps.setInt(1, id);
963         ps.setString(2, name);
964         ps.setString(3, add);
965         ps.executeUpdate();
966         return 1;
967     }
968     else
969         return 2;
970 }
971 catch(Exception e)
972 {
973     e.printStackTrace();
974     return 7;
975 }
976 finally
977 {
978     if(ps != null)
979     {
980         try
981         {
982             ps.close();
983         }
984         catch(Exception e)
985         {
986             e.printStackTrace();
987         }
988     }
989 }
990 } // insert()
991
992 public int delete()
993 {
994     PreparedStatement ps = null;
995     try
996     {
997         boolean alreadyPresent = exists(this.getID());
998
999         if(alreadyPresent)
1000         {
1001             ps = con.prepareStatement("DELETE FROM STUDENT_INFO WHERE STU_ID=?");
1002             ps.setInt(1,id);
1003             ps.executeUpdate();
1004             return 3;
1005         }
1006         else
1007             return 4;
1008     }
1009     catch(Exception e)
1010     {
1011         e.printStackTrace();
1012         return 7;
1013     }
1014     finally
1015     {
1016         if(ps != null)
1017         {
1018             try
1019             {
1020                 ps.close();
```

```
1021         }
1022     catch(Exception e)
1023     {
1024         e.printStackTrace();
1025     }
1026 }
1027 } // delete()
1028
1029 public int update()
1030 {
1031     PreparedStatement ps = null;
1032     try
1033     {
1034         boolean alreadyPresent = exists(this.getID());
1035
1036         if(alreadyPresent)
1037         {
1038             ps = con.prepareStatement("UPDATE STUDENT_INFO SET
1039                         STU_NAME=? ,STU_ADD=? WHERE STU_ID = ?");
1040             ps.setString(1,name);
1041             ps.setString(2,add);
1042             ps.setInt(3,id);
1043             ps.executeUpdate();
1044             return 5;
1045         }
1046         else
1047             return 4;
1048     }
1049     catch(Exception e)
1050     {
1051         e.printStackTrace();
1052         return 7;
1053     }
1054     finally
1055     {
1056         if(ps != null)
1057         {
1058             try
1059             {
1060                 ps.close();
1061             }
1062             catch(Exception e)
1063             {
1064                 e.printStackTrace();
1065             }
1066         }
1067     }
1068 }
1069 } // update()
1070
1071 public int display()
1072 {
1073     PreparedStatement ps = null;
1074
1075     boolean alreadyPresent = exists(this.getID());
1076
1077     try
1078     {
1079         if(alreadyPresent)
1080         {
```

```
1081         ps = con.prepareStatement("SELECT STU_NAME,STU_ADD FROM
1082                               STUDENT_INFO WHERE STU_ID=?");
1083         ps.setInt(1,id);
1084         ResultSet rs = ps.executeQuery();
1085
1086         if(rs.next())
1087         {
1088             String nm = rs.getString(1);
1089             this.setName(nm);
1090
1091             String ad = rs.getString(2);
1092             this.setAdd(ad);
1093         }
1094
1095         rs.close();
1096
1097         return 6;
1098     }
1099     else
1100     {
1101         return 4;
1102     }
1103     catch(Exception e)
1104     {
1105         e.printStackTrace();
1106         return 7;
1107     }
1108     finally
1109     {
1110         if(ps != null)
1111         {
1112             try
1113             {
1114                 ps.close();
1115             }
1116             catch(Exception e)
1117             {
1118                 e.printStackTrace();
1119             }
1120         }
1121     } // display()
1122 } // class
1123 =====
1124 PROJECT On BookSearch using mvc2 architecture (MinProject3)
1125 =====
1126 -----SQL.txt-----
1127 CREATE TABLE SELECT_BOOKS (BOOKID VARCHAR2(10),
1128                           BOOKNAME VARCHAR2(30),
1129                           AUTHORNAME VARCHAR2(30),
1130                           STATUS VARCHAR2(10),
1131                           CATEGORY VARCHAR2(20)
1132 );
1133 -----web.xml-----
1134 <?xml version="1.0" encoding="ISO-8859-1"?>
1135 <!DOCTYPE web-app
1136   PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
1137   "http://java.sun.com/dtd/web-app_2_3.dtd">
1138
1139 <web-app>
1140   <welcome-file-list>
```

```
1141      <welcome-file>Search.jsp</welcome-file>
1142  </welcome-file-list>
1143
1144  <servlet>
1145    <servlet-name>search</servlet-name>
1146    <servlet-class>com.nt.controller.MainSrv</servlet-class>
1147  </servlet>
1148
1149  <servlet-mapping>
1150    <servlet-name>search</servlet-name>
1151    <url-pattern>/BookSearchServlet</url-pattern>
1152  </servlet-mapping>
1153 </web-app>
1154 -----Search.jsp-----
1155 <html>
1156
1157   <script language = "javascript">
1158
1159     function isHtml()
1160     {
1161       f.source.value = "Html";
1162       validate();
1163     }
1164
1165     function isExcel()
1166     {
1167       f.source.value = "Excel";
1168       validate();
1169     }
1170
1171     function validate()
1172     {
1173       if(f.category.selectedIndex == '0')
1174       {
1175         alert("You should select Category !!!");
1176         f.category.focus();
1177         return false;
1178       }
1179       else
1180       {
1181         f.submit();
1182         return true;
1183       }
1184     }
1185
1186 </script>
1187
1188 <body>
1189   <form name=f action=".//BookSearchServlet" method="post">
1190     <center>
1191       <span style= "width=500;height=60;filter:shadow(color=pink,direction=135)">
1192         <font color=red size=5>Search for Books</font>
1193         <hr color=orange width=50%>
1194       </span></center>
1195
1196       <table border=1 cellpadding=4 cellspacing=4 align=center      bgcolor='lavender'>
1197         <tr>
1198           <th>Select Category</th>
1199           <th>
1200             <select name='category'>
```

```
1201             <option selected value="">Select a value</option>
1202             <option value='java'>JAVA</option>
1203             <option value=''.net'>.NET</option>
1204             <option value='jscript'>JavaScript</option>
1205         </select>
1206     </th>
1207 </tr>
1208 <tr>
1209     <th><input type='button' value='Html Output' onClick='isHtml()'></th>
1210     <th><input type='button' value='Excel Output' onClick='isExcel()'></th>
1211 </tr>
1212 </table>
1213
1214     <input type='text' name='source' readonly style='visibility:hidden'>
1215 </form>
1216 </body>
1217 </html>
1218 -----MainSrv.java-----
1219 package com.nt.controller;
1220 import javax.servlet.*;
1221 import javax.servlet.http.*;
1222 import java.io.*;
1223 import java.util.*;
1224 import com.nt.service.DbConnector;
1225
1226 public class MainSrv extends HttpServlet
1227 {
1228     public void doGet(HttpServletRequest req, HttpServletResponse res)
1229     {
1230         try
1231         {
1232             String cat = req.getParameter("category");
1233             String checkAction=req.getParameter("source");
1234
1235             DbConnector dbc = new DbConnector();
1236
1237             ArrayList al = dbc.search(cat);
1238             req.setAttribute("list", al);
1239             req.setAttribute("category", cat);
1240
1241             String target;
1242             if(checkAction.equalsIgnoreCase("Html"))
1243                 target = "HtmlPrint.jsp";
1244             else
1245                 target = "ExcelScreen.jsp";
1246
1247             RequestDispatcher rd = null;
1248             rd = req.getRequestDispatcher(target);
1249             if(rd != null)
1250                 rd.forward(req,res);
1251         } // try
1252         catch(Exception e)
1253         {
1254             e.printStackTrace();
1255         }
1256     } // doPost()
1257     public void doPost(HttpServletRequest req, HttpServletResponse res)
1258     {
1259         doGet(requeste,response);
1260     }
```

```
1261
1262 } // class
1263 -----DbConnector.java-----
1264 package com.nt.service;
1265
1266 import java.io.*;
1267 import java.sql.*;
1268 import java.util.ArrayList;
1269 import com.nt.model.BookBean;
1270
1271 public class DbConnector
1272 {
1273     private int found=0;
1274
1275     public Connection getConnection()
1276     {
1277         Connection con = null;
1278         try
1279         {
1280             Class.forName("oracle.jdbc.driver.OracleDriver");
1281             con = DriverManager.getConnection
1282                 ("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
1283         }
1284         catch(Exception e)
1285         {
1286             e.printStackTrace();
1287         }
1288         return con;
1289     }
1290
1291     public ArrayList search(String category)
1292     {
1293         Connection con = getConnection();
1294         PreparedStatement ps = null;
1295         ResultSet rs = null;
1296         ArrayList al = new ArrayList();
1297
1298         try
1299         {
1300             String searchQuery;
1301             searchQuery = "SELECT BOOKID, BOOKNAME, AUTHORNAME, STATUS " +
1302                         " FROM SELECT_BOOKS WHERE CATEGORY = ? ";
1303             ps = con.prepareStatement(searchQuery);
1304
1305             ps.setString(1, category);
1306             rs = ps.executeQuery();
1307
1308             while(rs.next())
1309             {
1310                 BookBean b = new BookBean();
1311                 b.setBookId(rs.getString(1));
1312                 b.setBookName(rs.getString(2));
1313                 b.setAuthorName(rs.getString(3));
1314                 b.setStatus(rs.getString(4));
1315                 al.add(b);
1316             }
1317             rs.close();
1318         }
1319         catch(Exception e)
1320         {
```

```
1321         e.printStackTrace();
1322     }
1323     finally
1324     {
1325         if(ps != null)
1326         {
1327             try
1328             {
1329                 ps.close();
1330             }
1331             catch(Exception e)
1332             {
1333                 e.printStackTrace();
1334             }
1335         }
1336         if(con != null)
1337         {
1338             try
1339             {
1340                 con.close();
1341             }
1342             catch(Exception e)
1343             {
1344                 e.printStackTrace();
1345             }
1346         }
1347     } // finally
1348     return al;
1349 } // search()
1350 } // class
1351 -----BookBean.java-----
1352 package com.nt.model;
1353
1354 public class BookBean implements java.io.Serializable
1355 {
1356     private String bookid;
1357     private String bookname;
1358     private String authorname;
1359     private String status;
1360
1361     public void setBookId(String bookid)
1362     {
1363         this.bookid = bookid;
1364     }
1365     public String getBookId()
1366     {
1367         return bookid;
1368     }
1369     public void setBookName(String bookname)
1370     {
1371         this.bookname = bookname;
1372     }
1373     public String getBookName()
1374     {
1375         return bookname;
1376     }
1377     public void setAuthorName(String authorname)
1378     {
1379         this.authorname = authorname;
1380     }
```

```
1381     public String getAuthorName()
1382     {
1383         return authorname;
1384     }
1385     public void setStatus(String status)
1386     {
1387         this.status = status;
1388     }
1389     public String getStatus()
1390     {
1391         return status;
1392     }
1393 } // BookBean class.
1394 -----
1395 <%@page import="java.util.ArrayList, com.nt.model.BookBean" %>
1396 <%
1397     ArrayList al = (ArrayList)request.getAttribute("list");
1398     String cat = (String)request.getAttribute("category");
1399 %>
1400
1401 <html>
1402
1403     <script language='javascript'>
1404         function showprint()
1405         {
1406             frames.focus();
1407             frames.print();
1408         }
1409     </script>
1410
1411 <body>
1412     <form name='f'>
1413     <center><h2><u>
1414         Books belonging to category <%= cat.toUpperCase() %>
1415     <u></h2></center>
1416     <br>
1417
1418     <table border="1" width="100%">
1419         <tr>
1420             <th>Sno</th>
1421             <th>BookId</th>
1422             <th>BookName</th>
1423             <th>AuthorName</th>
1424             <th>Status</th>
1425         </tr>
1426         <%
1427             for(int i = 0; i < al.size(); i++)
1428             {
1429                 BookBean sb=(BookBean)al.get(i);
1430             %>
1431             <tr>
1432                 <td><%= (i+1) %></td>
1433                 <td><%= sb.getBookId() %></td>
1434                 <td><%= sb.getBookName() %></td>
1435                 <td><%= sb.getAuthorName() %></td>
1436                 <td><%= sb.getStatus() %></td>
1437             </tr>
1438             <%
1439             }
1440         %>
```

```
1441      </table>
1442      <center>
1443          <a href="javascript:showprint()">Print</a>
1444      </center>
1445      </form>
1446  </body>
1447  </html>
1448  -----ExcelScreen.jsp-----
1449  <%@page import="java.util.ArrayList,com.nt.model.BookBean"%>
1450  <%
1451      response.setHeader("Content-Disposition","attachment;filename=Title1.xls");
1452      response.setContentType("application/ms-excel");
1453
1454      ArrayList al = (ArrayList)request.getAttribute("list");
1455      String cat = (String)request.getAttribute("category");
1456  %>
1457  <center><h2><u>
1458      Books belonging to category <%= cat.toUpperCase() %>
1459  <u></h2></center>
1460  <br>
1461  <table border="1" width="100%">
1462      <tr>
1463          <th>Sno</th>
1464          <th>BookId</th>
1465          <th>BookName</th>
1466          <th>AuthorName</th>;
1467          <th>Status</th>
1468      </tr>
1469      <%
1470      for(int i = 0; i < al.size(); i++)
1471      {
1472          BookBean sb=(BookBean)al.get(i);
1473      %>
1474      <tr>
1475          <td><%= (i+1) %></td>
1476          <td><%= sb.getBookId() %></td>
1477          <td><%= sb.getBookName() %></td>
1478          <td><%= sb.getAuthorName() %></td>
1479          <td><%= sb.getStatus() %></td>
1480      </tr>
1481      <%
1482      }
1483      %>
1484  </table>
1485  =====
1486  =====
1487  >>>>>>>>>MiniProject on file uploading and Downloading operations>>>>>>>>>>>>>>>>>>>
1488  =====
1489  -----DB script.txt-----
1490 SQL> desc employereg;
1491 Name           Null?    Type
1492 -----
1493 EMP_ID          NOT NULL NUMBER(4)
1494 EMP_NAME        VARCHAR2(10)
1495 EMP_ADD         VARCHAR2(10)
1496 EMP_RESUME      VARCHAR2(50)
1497 EMP_PIC          VARCHAR2(50)
1498
1499 create table employereg (EMP_ID      NUMBER(4) primary key,
1500                           EMP_NAME    VARCHAR2(10),
```

```
1501           EMP_ADD      VARCHAR2(10),  
1502           EMP_RESUME   VARCHAR2(50),  
1503           EMP_PIC      VARCHAR2(50))  
1504 -----Home.html-----  
1505 <center>  
1506   <a href="Register.jsp">Register new Employee</a>  
1507   <br><br><br>  
1508   <a href="View.jsp">view and Download Emp Details</a>  
1509 -----Register.jsp-----  
1510 body bgcolor=wheat>  
1511 <center><h1>Employee Registration Page</h1>  
1512 <form action="reg" method="post" enctype="multipart/form-data">  
1513  
1514 <table>  
1515   <tr>  
1516     <td>Employee ID</td><td><input type=text name=tid></td>  
1517   </tr>  
1518   <tr>  
1519     <td>Employee Name</td><td><input type=text name=tname></td>  
1520   </tr>  
1521   <tr>  
1522     <td>Employee Add</td><td><input type=text name=tadd></td>  
1523   </tr>  
1524   <tr>  
1525     <td>Employee photo</td><td><input type=file name=tphoto></td>  
1526   </tr>  
1527   <tr>  
1528     <td>Employee resume</td><td><input type=file name=tresume></td>  
1529   </tr>  
1530   <tr>  
1531     <td><input type=submit value=register></td>  
1532   </tr>  
1533 </form>  
1534 </center>  
1535 </body>  
1536 -----web.xml-----  
1537 <web-app>  
1538  
1539   <servlet>  
1540     <servlet-name>reg</servlet-name>  
1541     <servlet-class>com.nt.RegisterServlet</servlet-class>  
1542   </servlet>  
1543  
1544   <servlet-mapping>  
1545     <servlet-name>reg</servlet-name>  
1546     <url-pattern>/reg</url-pattern>  
1547   </servlet-mapping>  
1548  
1549   <welcome-file-list>  
1550     <welcome-file>Home.html</welcome-file>  
1551   </welcome-file-list>  
1552  
1553 </web-app>  
1554 -----RegisterServlet.java-----  
1555 //RegisterServlet.java  
1556 package com.nt;  
1557 import java.io.IOException;  
1558 import java.io.PrintWriter;  
1559 import java.sql.Connection;  
1560 import java.sql.DriverManager;
```

```
1561 import java.sql.PreparedStatement;
1562 import java.util.ArrayList;
1563 import java.util.Vector;
1564
1565 import javax.servlet.ServletException;
1566 import javax.servlet.http.HttpServlet;
1567 import javax.servlet.http.HttpServletRequest;
1568 import javax.servlet.http.HttpServletResponse;
1569
1570 import javazoom.upload.MultipartFormDataRequest;
1571 import javazoom.upload.UploadBean;
1572 import javazoom.upload.UploadParameters;
1573
1574 public class RegisterServlet extends HttpServlet
1575 {
1576     public void doPost (HttpServletRequest req, HttpServletResponse res)
1577             throws ServletException, IOException
1578     {
1579
1580         String resumePath="c:/store/resumes";
1581         String photoPath="c:/store/photoes";
1582         PrintWriter out = res.getWriter();
1583         try
1584         {
1585             // get special request obj given by Java zoom api
1586             MultipartFormDataRequest nreq = new MultipartFormDataRequest(req);
1587             int eId=Integer.parseInt(nreq.getParameter("tid"));
1588             String eName=nreq.getParameter("tname");
1589             String eAdd=nreq.getParameter("tadd");
1590             /*String ePhoto=nreq.getParameter("tphoto"); shows Null
1591             String eResume=nreq.getParameter("tresume"); shows Null */
1592
1593             // settings related to Resume uploading
1594             UploadBean upb = new UploadBean();
1595             upb.setFolderstore(resumePath);
1596             upb.setOverwrite(false);
1597             upb.store(nreq,"tresume");// complets files uploading
1598
1599             // setting related photo uploading
1600             upb.setFolderstore(photoPath);
1601             upb.setOverwrite(false);
1602             upb.store(nreq,"tphoto"); complets files uploading
1603
1604             // get the file names of the uploaded files
1605             Vector history = upb.getHistory();
1606
1607             ArrayList <String> filesName=new ArrayList<String>();
1608             for (int i=0;i<history.size();i++)
1609             {
1610                 UploadParameters up = (UploadParameters) History.elementAt(i);
1611                 filesName.add(up.getFilename());
1612             }
1613             System.out.println("resume"+filesName.get(0));
1614             System.out.println("photo"+filesName.get(1));
1615
1616             // store the paths of the uploaded files to c:\store folder
1617             Class.forName("oracle.jdbc.driver.OracleDriver");
1618             Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
1619             PreparedStatement ps=con.prepareStatement("insert into EmployeeReg values(?,?,?,?,?)");
1620             ps.setInt(1,eId);
```

```
1621     ps.setString(2, eName);
1622     ps.setString(3, eAdd);
1623     ps.setString(4, resumePath+"/"+filesName.get(0));
1624     ps.setString(5, photoPath+"/"+filesName.get(1));
1625
1626     int i=ps.executeUpdate();
1627     if(i==1)
1628         out.println("Successfully uploaded and Stored in DataBase");
1629     else
1630         out.println("Failed in uploading");
1631
1632     }//try
1633     catch(Exception e)
1634     {
1635         out.println(e);
1636     }//catch
1637     }//doPost ()
1638 } //class
1639
1640 -----View.jsp-----
1641 <%@page import="java.io.File,java.util.* ,java.sql.*"%>
1642
1643 <!-- Retrive records from DB table -->
1644 <H1>List of All files under C:\store</H1>
1645 <%
1646     Class.forName("oracle.jdbc.driver.OracleDriver");
1647     Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
1648     PreparedStatement ps=con.prepareStatement("select * from EmployeeReg");
1649     ResultSet rs=ps.executeQuery();
1650 %>
1651
1652 <!-- display employee details as html table content-->
1653 <body bgcolor=wheat>
1654 <table border=1>
1655 <tr><td>Employee Name</td><td>Employee Address</td><td>Employee Resume</td><td>Employee Photo</td></tr>
1656
1657 <%
1658 while(rs.next())
1659 {
1660 %>
1661     <tr>
1662         <td><%=rs.getString(2)%></td>
1663         <td><%=rs.getString(3)%></td>
1664         <td><a href='Download.jsp?resumeId=<%=rs.getString(1)%>'>Download Here</a></td>
1665         <td><a href='Download.jsp?photoid=<%=rs.getString(1)%>'>Download Here</a></td>
1666     </tr>
1667 <%} %>
1668
1669 </table>
1670 </body>
1671 -----Download.jsp-----
1672 <%@page import="java.io.* ,javax.servlet.* ,javax.servlet.http.* ,java.sql.*" %>
1673
1674 <!-- get the path of the file to downloaded -->
1675 <%
1676 String fileName="";
1677 String queryText="";
1678
1679 Class.forName("oracle.jdbc.driver.OracleDriver");
1680 Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
```

```
1681
1682 if(request.getParameter("resumeId")!=null)
1683     queryText="select emp_resume from employeereg where emp_id="+request.getParameter("resumeId");
1684 else
1685     queryText="select emp_pic from employeereg where emp_id="+request.getParameter("photoId");
1686
1687 PreparedStatement ps=con.prepareStatement(queryText);
1688
1689 ResultSet rs=ps.executeQuery();
1690 while(rs.next()){
1691     fileName=rs.getString(1);
1692 }
1693 // prepare settings to download the files
1694 File f= new File(fileName);
1695 int length = 0;
1696 ServletOutputStream op = response.getOutputStream();
1697
1698 String mimetype = application.getMimeType(fileName);
1699
1700 response.setContentType( (mimetype != null) ? mimetype : "application/octet-stream" );
1701
1702 response.setContentLength( (int)f.length() );
1703
1704 response.setHeader( "Content-Disposition", "attachment;filename="+fileName );
1705
1706 // comple file downloading using buffering
1707 byte[] buf = new byte[1024];
1708 DataInputStream in = new DataInputStream(new FileInputStream(f));
1709
1710 while ((in != null) && ((length = in.read(buf)) != -1))
1711 {
1712     op.write(buf,0,length);
1713 }
1714 //close streams
1715 in.close();
1716 op.flush();
1717 op.close();
1718
1719 %>
1720
```

```
1 =====
2 JSTL Applications
3 =====
4 -----ex1.jsp-----
5 <%@ taglib uri="core" prefix="c" %>
6 <HTML>
7 <HEAD>
8     <TITLE>Example on JSTL Coretags</TITLE>
9 </HEAD>
10    <BODY BGCOLOR="#FFFFFF">
11        <c:set var="var1" value="Welcome" />
12
13        <c:if test="${!empty param.uname}">
14            <c:out value="${var1}" />
15            <c:out value="${param.uname}" />
16        </c:if>
17    </BODY>
18 </HTML>
19
20 -----web.xml-----
21 <!DOCTYPE web-app
22 PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
23 "http://java.sun.com/dtd/web-app_2_3.dtd">
24
25 <web-app>
26
27     <taglib>
28         <taglib-uri>core</taglib-uri>
29         <taglib-location>/WEB-INF/c.tld</taglib-location>
30     </taglib>
31
32     <taglib>
33         <taglib-uri>sql</taglib-uri>
34         <taglib-location>/WEB-INF/sql.tld</taglib-location>
35     </taglib>
36 </web-app>
37 -----ex2.jsp-----
38 <%@ taglib uri="core" prefix="c" %>
39 <HTML>
40 <HEAD>
41 <TITLE>Example on JSTL Coretags</TITLE>
42 </HEAD>
43 <BODY BGCOLOR="#FFFFFF">
44     <c:choose>
45         <c:when test="${param.p>0}">
46             p is +ve
47         </c:when>
48         <c:when test="${param.p<0}">
49             p is -ve
50         </c:when>
51         <c:otherwise>
52             p is zero
53         </c:otherwise>
54     </c:choose>
55
56 </BODY>
57 </HTML>
58
59 -----ex3.jsp-----
60 <%@ taglib uri="core" prefix="c" %>
```

```
61 <HTML>
62 <HEAD>
63 <TITLE>Example on JSTL Coretags</TITLE>
64 </HEAD>
65 <BODY BGCOLOR="#FFFFFF">
66 </BODY>
67 List of parameters <br><br>
68 <c:forEach var="p" items="${paramValues}">
69     parameter name = <c:out value="${p.key}" /><br>
70     values =
71     <c:forEach var="pv" items="${p.value}">
72         <c:out value="${pv}" /><br>
73     </c:forEach>
74 </c:forEach>
75 </HTML>
76
77 -----ex4.jsp-----
78 <%@ taglib uri="core" prefix="c" %>
79 <HTML>
80 <HEAD>
81     <TITLE>Example on JSTL Coretags</TITLE>
82 </HEAD>
83 <BODY BGCOLOR="#FFFFFF">
84
85 List of students<br><br>
86     <c:set var="str" value="raja,ravi,jani,rakesh" scope="page" />
87
88     <c:forTokens var="sname" items="${str}" delims="," >
89         <c:out value="${sname}" /> <br>
90     </c:forTokens>
91
92 </BODY>
93 </HTML>
94
95 -----ex5.jsp-----
96 <%@ taglib uri="core" prefix="c" %>
97 <HTML>
98 <HEAD>
99     <TITLE>Example on JSTL Coretags</TITLE>
100 </HEAD>
101 <BODY BGCOLOR="#FFFFFF">
102 <table>
103     <c:forEach var="x" begin="1" end="10" step="1">
104         <tr>
105             <td><c:out value="2 * ${x} = "/></td>
106             <td><c:out value="${2 * x}" /></td>
107         </tr>
108     </c:forEach>
109 </table>
110
111 </BODY>
112 </HTML>
113 -----ex6.jsp-----
114 <%@ taglib uri="sql" prefix="sql" %>
115 <%@ taglib uri="core" prefix="c" %>
116
117 <sql:setDataSource var="ds" driver="oracle.jdbc.driver.OracleDriver"
118     url="jdbc:oracle:thin:@localhost:1521:orcl" user="scott" password="tiger" />
119 <sql:query var="res_set" dataSource="${ds}" sql="select * FROM emp" />
120
```

```
121 <HTML>
122 <HEAD>
123 <TITLE>Example on JSTL tags</TITLE>
124 </HEAD>
125 <BODY BGCOLOR="#FFFFFF">
126
127 <c:forEach var="row" items="${res_set.rows}">
128   <c:out value="${row.ename}" />
129   <c:out value="${row.job}" />
130   <br><br>
131 </c:forEach>
132
133 </BODY>
134 </HTML>
135 -----ex7.jsp-----
136 <%@ taglib uri="http://java.sun.com/jstl/sql" prefix="sql" %>
137 <%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
138
139 <sql:setDataSource var="ds" driver="oracle.jdbc.driver.OracleDriver"
140   url="jdbc:oracle:thin:@localhost:1521:orcl" user="scott" password="tiger" />
141
142 <HTML>
143 <HEAD>
144 <TITLE>Example on JSTL tags</TITLE>
145 </HEAD>
146 <BODY BGCOLOR="#FFFFFF">
147
148 <sql:transaction dataSource="${ds}">
149   <sql:update>
150     update emp SET comm =comm + ? WHERE empno = ?
151     <sql:param value="1000"/>
152     <sql:param value="7499"/>
153   </sql:update>
154 </sql:transaction>
155   Record updated.
156 </BODY>
157 </HTML>
158 -----ex8.jsp-----
159 <%@ taglib uri="sql" prefix="sql" %>
160 <%@ taglib uri="core" prefix="c" %>
161
162 <sql:setDataSource var="ds" driver="oracle.jdbc.driver.OracleDriver"
163   url="jdbc:oracle:thin:@localhost:1521:orcl" user="scott" password="tiger" />
164
165 <HTML>
166 <HEAD>
167 <TITLE>Example on JSTL tags</TITLE>
168 </HEAD>
169 <BODY BGCOLOR="#FFFFFF">
170 <sql:transaction dataSource="${ds}">
171
172   <sql:update>
173     delete from emp WHERE empno = ?
174     <sql:param value="7499"/>
175   </sql:update>
176 </sql:transaction>
177
178 Record Deleted
179
180 </BODY>
181 </HTML>
182 ======
```

## **Expression Language (EL)**

It has introduced in Jsp 2.0 version.

The main objective of EL is to eliminate java code from the jsp.

In General we can use EL with JSTL and custom tags for complete elimination of java code.

### **What can we do with EL**

- 1.we can get request parameters
- 2.we can get any scope attributes
- 3.we can get values of bean objects stored in a scope.
- 4.we can get request header values.
- 5.we can get context parameter values.
- 6.we can get cookie values
- 7.we can get jsp implicit objects
- 8.we can do arithmetic, relational, logical and conditional operations.
- 9.we can call static methods of a class by using EL functions.

### **What cannot we do with EL**

- 1.we cannot set attributes in a scope.
- 2.we cannot remove attributes from a scope.
- 3.we cannot set values to array elements stored in a scope
- 4.we cannot set values to collection to collection object stored in scope.
- 5.we cannot execute if else and switch case conditions.
- 6.we cannot execute looping operations.
- 7.we cannot catch exceptions
- 8.we cannot request dispatching and redirecting etc...

### **To print the value of request parameter uname.**

```
<%=request.getParameter("uname")%>  
(or)  
${param.uname}
```

### **To print the value of scoped attribute x.**

```
<%=pageContext.findAttribute("x")%>  
  
 ${x}
```

**Note:** To use any variable x in EL directly compulsorily it should be an attribute in some scope.

It prints the value of x attribute and if there is no such attribute then we will get blank space but not null.but EL supreses null and doesnot print any thing.

- 1)EL implicit objects
- 2)EL operators
- 3)EL Functions

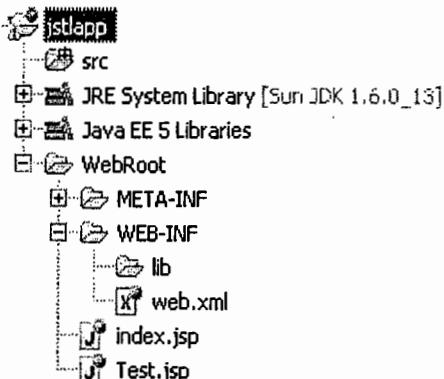
### **1)EL implicit Objects:**

EL contains 11 implicit objects.

The power of EL is just because of These Objects only.

The Following is the list of all EL implicit Objects.

- 1.page scope
- 2.request scope
- 3.session scope
- 4.application scope
- 5.param
- 6.param values
- 7.header
- 8.header values
- 9.cookie
- 10.init param
- 11.page context



page scope, request scope, session scope, application scope By using these implicit objects we can retrieve the attributes of a particular scope.

- 1.page scope → pageContext.getAttribute();
- 2.request scope → request.getAttribute();
- 3.session scope → session.getAttribute();
- 4.application scope → application.getAttribute();

**Ex1:** To access the value of session scoped attribute 'x'  
    \${sessionScope.x}

**Ex2:** It prints the value of request scoped attribute 'x'.  
If there is no such type of attribute we will get blank space.  
    \${requestScope.x}

**Ex3:** \${x}

Jsp container first checks in page Scope for the attribute x.  
If it is available it prints its value .If it is not available it will check in requestScope followed by Session and Application scope.It simply acts as pageContext.findAttribute.

```
<%@page isELIgnored="false"%>
<%
pageContext.setAttribute("x",100,2);
pageContext.setAttribute("y",1000,3);
pageContext.setAttribute("x",10);
%>
<h1>${x} <br>
${requestScope.x} <br>
${sessionScope.x} <br>
${y}
output
10
100
space
1000
5)param
6)param values
```

we can use these implicit objects to retrieve request parameter values.  
param → request.getParameter();  
**paramValues** -> request.getParameterValues();

<b>String(param name)</b>	<b>String[](param value)</b>
name	{NIT}
course	{java,sql}
mail	{hr@nareshit.com,info@nit.com}

Ex: \${param.x}

It prints the value associates with request parameter x.  
If the parameters is associated with multiple values.then it prints the first value.  
If the specified parameter is not available then we will get blank space.  
\${paramValues.x} → internally toString[] will be called as String[]  
    \${paramValues.x[0]} → It prints first value.  
    \${paramValues.x[1]} → It prints second value.

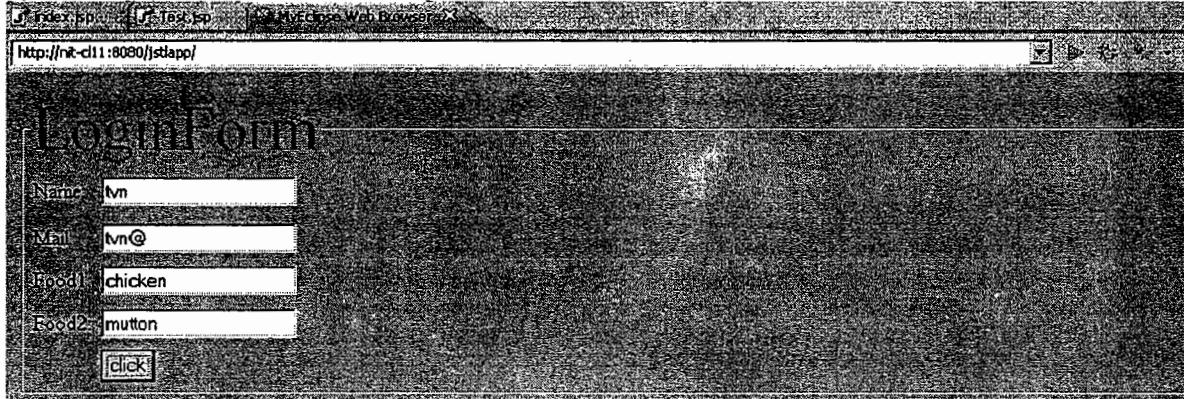
```
index.jsp
<html>
<body bgcolor="ORANGE">
<form action="Test.jsp" method="post">
<fieldset>
<legend>
<font size="9" color="Green">LoginForm</font></legend>
<table border="2" cellpadding="2" bordercolor="orange">
<tr><td>Name:</td>
<td><input type="text" name="name"/></td></tr>
<tr>
<td>Mail:</td>
<td><input type="text" name="mail"/></td></tr>
<tr><td>Food1:</td>
<td><input type="text" name="food"/></td></tr>
<tr><td>Food2:</td>
<td><input type="text" name="food"/></td></tr>

<tr><td></td>
<td><input type="submit" value="click"/></td></tr>
</table>
</fieldset>
</form>
</body>
</html>
```

**Test.jsp**

```
<%@page isELIgnored="false"%>
<h1>
${param.name}<br/>
${param.age}<br/>
${param.food}<br/>
${paramValues.name}<br/>
${paramValues.name[0]}<br/>
${paramValues.food[0]}<br/>
${paramValues.food[100]}<br/>
${paramValues.food[1]}<br/>
```

&lt;/h1&gt;

**output**

tvn

```
chicken
[Ljava.lang.String;@162ba99
tvn
chicken
```

mutton

**Note:** EL handles very nicely null and ArrayIndexOut of Bounds Exceptions. Suppresses them and prints blank space.

**header, header value:-** These are exactly same as param&param value except that these for retrieving request headers.

```
header ${request.getHeader()}
headerValues ${request.getHeaders()}
${header.accept}
${headerValues.accept[0]}
${headerValues.accept} ${toString()} method will be executed.
```

#### **Cookie:-**

We can use this implicit object to retrieve cookies associated with the request.

```
cookie ${request.getCookies()}
```

```
 ${cookie.JSESSIONID}
```

```
 ${cookie.JSESSIONID.name}
```

```
 ${cookie.JSESSIONID.value}
```

Output

```
javax.servlet.http.Cookie@21d23b  
JSESSIONID  
1A46E26C2BD6F96BD1B5012789B4234B
```

cookieapp

- src
- JRE System Library [Sun JDK 1.6.0\_18]
- Java EE 5 Libraries
- WebRoot
  - META-INF
  - WEB-INF
    - lib
    - web.xml
  - index.jsp

#### **web.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

#### **index.jsp**

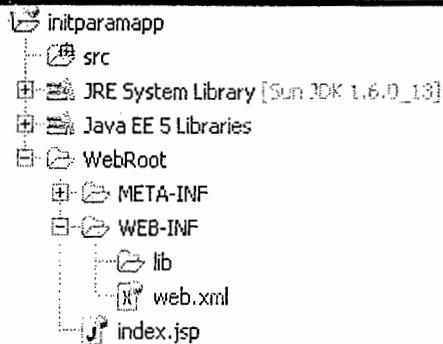
```
 ${cookie.JSESSIONID}<br/>
 ${cookie.JSESSIONID.name}<br/>
 ${cookie.JSESSIONID.value}<br/>
```

#### **init param:-**

by using this implicit object we can access context initialization parameters  
 initparameters - - - \${application.getInitParameter();}

#### **example**

```
 ${initparam.user} ---- ${INITAdmin}
 ${initparam.pwd} ---- ${Blankspace}
```

**web.xml**

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
    <context-param>
        <param-name>user</param-name>
        <param-value>NITAdmin</param-value>
    </context-param>
</web-app>

```

**index.jsp**

```

${initParam.user}<br/>
${initParam.pwd}<br/>

```

The JSP expression language supports the following implicit objects:

<b>Implicit object</b>	<b>Description</b>
pageScope	Scoped variables from page scope
requestScope	Scoped variables from request scope
sessionScope	Scoped variables from session scope
applicationScope	Scoped variables from application scope
param	Request parameters as strings
paramValues	Request parameters as collections of strings
header	HTTP request headers as strings
headerValues	HTTP request headers as collections of strings
initParam	Context-initialization parameters
cookie	Cookie values
pageContext	The JSP PageContext object for the current page

**ELOperators**

EL(Expression Language) contains its own set of operators the following is the list of all available operators in EL.

- 1)property access operator(.)
- 2)collection access operator([])
- 3)Arithmetic operators
- 4)Relational operators
- 5)Logical operators

**1) property access operator:-**

**Syntax:** \${leftvariable.rightvariable}  
\${param.uname},\${pageContext.session}

**2)collection Access operator:-**

**Syntax:** \${left variable[right variable]}

**Example:**

```

${initParam.mail}
${initParam['mail']}
${initParam["mail"]}

```

**Example For Arrays:**

```

<%
String[] s={"CoreJava","Spring","Hibernate","WebServices"}
pageContext.setAttribute("s",s);
%>
${s[0]}--&Core Java
${s[1]}--&Spring
${s[2]}--&Hibernate
${s[3]}--&WebServices
${s[100]}--&blankspace

```

**Example for List:-**

```

<%@page isELIgnored="false"%>
<%
java.util.ArrayList al=new java.util.ArrayList();
al.add("nitstudent1");
al.add("nitstudent2");
al.add("nitstudent3");
pageContext.setAttribute("list",l,2);
%>
<h1>
${list[0]}-& nitstudent1
${list["2"]}-& nitstudent2
${list[100]}-& blank space

```

**Arithmetic Operator:-**

EL does not support operator overloading hence '+' operator always meant for arithmetic addition operator.

**1) '+' operator**

- 1) \${2+3} &5
- 2) \${"2"+"3"} &5
- 3) \${"2"+'3'} &5
- 4) \${abc+'3'} &3
- 5) \${null+"3"} &3
- 6) \${"abc"+"3"} &NumberFormatException
- 7) {" "+"3"} &NumberFormatException

**2) '-' operator:**

```

${a-b}
All rules is exactly same as '+' operator
${10-3} &7
${"abc"-3}&NumberFormatException
${"abc"-3}& -3

```

**3) '\*' operator:-**

```

${a*b}
All rules are exactly similar to "+" operator
${19*2}&38

```

**4) '/' operator:-**

```

${a/b} or ${a div b}
All rules are exactly same as "+" operator
${10/2}&5
${10/0}&infinity
division operator always follow floating point arithmetic but not integral arithmetic
System.out.println(10/0): java.lang.ArithmaticException: / by zero
System.out.println(10.0/0): Infinity
System.out.println(0/0): java.lang.ArithmaticException: / by zero
System.out.println(0/0.0):NaN(Not any Number)

```

**5)Module operator:** % or mod

All the rules are exactly same as '+' operator and this operator can provide support for both integral and floating point arithmetic.

```
 ${10%3}-->1
 ${10%0}-->ArithmetricException
 ${10.0/0}=>NaN (Not any Number)
```

**Relational Operators:-**

> or gt  
\${2>1}-->true

< or lt

```
 ${3<4}-->true
 == or eq
 ${abcd==abcd}&gt;true

 ${"abcd"=="abc"}=>false
 != or ne
 >= or ge
 < = or le
```

**Logical Operators:-**

and | &&  
\${true and false}-->false
 or | ||  
\${true or false}&gt;true
 not | !  
\${not true} &gt;false

**Conditional Operator:-**

```
 ${(3<4) ? "yes" : "no"} &gt; yes
 ${(true==false) ? "Right" : "wrong"} &gt; wrong
```

**empty Operator:-**

\${empty object}  
Returns true  
(a) if object does not exists  
(b) if object is an empty array  
(c) object is an empty string  
(d) object is an collection  
In all other cases it returns false.  
\${empty abcd} | true  
\${empty "abcd"} | false  
\${empty null} | true

JSP Expression Language (EL) supports most of the arithmetic and logical operators supported by Java.  
Below is the list of most frequently used operators:

Operator	Description
.	Access a bean property or Map entry
[]	Access an array or List element
()	Group a subexpression to change the evaluation order
+	Addition
-	Subtraction or negation of a value
*	Multiplication
/ or div	Division

% or mod	Modulo (remainder)
== or eq	Test for equality
!= or ne	Test for inequality
< or lt	Test for less than
> or gt	Test for greater than
<= or le	Test for less than or equal
>= or gt	Test for greater than or equal
&& or and	Test for logical AND
or or	Test for logical OR
! or not	Unary Boolean complement
empty	

**Objective Questions on JSTL and EL**

1. What gets printed when the following expression is evaluated? Select one correct answer.  
 $\${(1==2) ? 4 : 5}$

- A. 1
- B. 2
- C. 4
- D. 5

**Answer:D**

2. What gets printed when the following expression is evaluated? Select one correct answer.

$\${4 \text{ div } 5}$

- A. 0
- B. 0.8
- C. 1
- D. -1

**Answer :B div operator is used for dividing in EL.**

3. What gets printed when the following expression is evaluated? Select one correct answer.

$\${12 \% 4}$

- A. 0
- B. 3
- C. 8
- D. 16

**Answer: A % operator gives the remainder after performing division.**

4. What is the effect of executing the following JSP statement, assuming a class with name Employee exists in classes package.

```
<%@ page import = "classes.Employee" %> <jsp:useBean id="employee" class="classes.Employee"
scope="session"/> <jsp:setProperty name="employee" property="*"/>
```

- A. The code does not compile as there is no property attribute of setProperty tag.
- B. The code does not compile as property attribute cannot take \* as a value.
- C. The code sets value of all properties of employee bean to \*.
- D. The code sets the values of all properties of employee bean to matching parameters in request object.

**Answer: D. This is a valid syntax for setProperty. All properties of the bean are set from the corresponding parameter names in the request object**

5. What is the effect of evaluation of following expression? Select the one correct answer.

$\${(5*5) ne 25}$

- A. true
- B. false
- C. 25

D. The expression does not compile as ne is not a valid operator.

**Answer B: The code prints false. ne is a valid operator. Since both left hand side and right hand side are equal to 25, false gets printed.**

6. What is the effect of evaluation of following expression? Select the one correct answer.

$\${'cat' gt 'cap'}$

- A. true
- B. false
- C. catcap
- D. The expression does not compile as gt operator cannot be applied on strings.

**Answer:** A EL considers <cat> to be greater than <cap>, as the letter t comes after the letter p.

7. How many numbers are printed, when the following JSTL code fragment is executed? Select the one correct answer.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:forEach var="item" begin="0" end="10" step="2">
${item}
</c:forEach>
```

A. 1                    B. 5  
C. 6                    D. 11

**Answer:** C The following numbers get printed - 0, 2, 4, 6, 8, 10.

8. What gets printed when the following JSTL code fragment is executed? Select the one correct answer.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:set var="item" value="2"/>
<c:if test="${var==1}" var="result" scope="session">
<c:out value="${result}"/>
</c:if>
```

- A. The JSTL code does not compile as attribute for if tag are not correct.  
B. true                    C. false  
D. Nothing gets printed.

**Answer:** D if evaluates to false, hence the c.out statement does not get executed.

9. What gets printed when the following JSTL code fragment is executed? Select the one correct answer.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:set var="item" value="2"/>
<c:forEach var="item" begin="0" end="0" step="2">
<c:out value="${item}" default="abc"/>
</c:forEach>
```

- A. The JSTL code does not compile as an attribute for forEach tag is not correct.  
B. 0                    C. 2                    D. ABC  
E. Nothing gets printed as c.out statement does not get executed.

**Answer B:** The forEach tag gets executed once, and prints zero.

10. How many numbers gets printed when the following JSTL code fragment is executed? Select the one correct answer.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:set var="item" value="2"/>
<c:choose>
<c:when test="${item>0}">
<c:out value="1"/>
</c:when>
<c:when test="${item==2}">
<c:out value="2"/>
</c:when>
<c:when test="${item<2}">
<c:out value="3"/>
</c:when>
<c:otherwise>
<c:out value="4"/>
</c:otherwise>
</c:choose>
```

- A. No number gets printed.                    B. One number gets printed.  
C. Two numbers gets printed.                    D. Three numbers gets printed.  
E. Four numbers gets printed.

**Answer B:** Only one number gets printed - the number 1.

11. Which numbers gets printed when the following JSTL code fragment is executed? Select the two correct answers.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:set var="j" value="4,3,2,1"/>
<c:forEach items="${j}" var="item" begin="1" end="2">
```

```
<c:out value="${item}" default="abc"/>
</c:forEach>
A. 1      B. 2
C. 3      D. 4
E. abc
```

The program does not compile.

**Answer B, C: In this case the forEach tag iterates through two elements of the array named j.**

12. Which numbers gets printed when the following JSTL code fragment is executed? Select the two correct answers.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:set var="j" value="4,3,2,1"/>
<c:forEach items="${j}" var="item" begin="1" end="2" varStatus="status">
<c:out value="${status.count}" default="abc"/>
</c:forEach>
```

- A. 1      B. 2
- C. 3      D. 4
- E. abc

The program does not compile.

**Answer: B, C. varStatus is set to a class of type LoopTagStatus. This class has a property named count which is being printed. count is the loop index, beginning with 1. So for two iterations 1 and 2 get printed. In this case the forEach tag iterates through two elements of the array named j.**

13. Which number gets printed when the following JSTL code fragment is executed? Select the one correct answers.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:set var="j" value="4,3,2,1"/>
<c:forEach items="${j}" var="item" varStatus="status">
<c:if test="${status.first}">
<c:out value="${status.index}" default="abc"/>
</c:if>
</c:forEach>
```

- a. 1      b. 2      c. 3      d. 4
- E. abc

The program does not compile.

**Answer: A status.first is true for the first iteration. The index is set to 0 in the first iteration.**

14. Which of these represent the correct path for the core JSTL library in JSTL version 1.1? Select the one correct answer.

- a. <http://java.sun.com/jsp/jstl/core>
- b. <http://java.sun.com/jsp/core>
- c. <http://java.sun.com/core>
- d. <http://java.sun.com/jstl1.1/core>

**Answer: A The path of core tag library in JSTL 1.1 is <http://java.sun.com/jsp/jstl/core>**

---

## JSP interview questions and answers

**LabExercise****1. JSP embeds in ..... in .....**

- a. Servlet, HTML      b. HTML, Java      c. HTML, Servlet      d. Java, HTML

**Answer:D****2. A JSP is translated into a :**

- a. Java applet      b. Java servlet    c. Either 1 or 2 above    d. Neither 1 nor 2 above

**Answer:B****3. After translation of a JSP source page into its implementation class, The jsp implementation class is \_\_\_\_\_ ?**

- a. final      b. static      c. abstract      d. private

**Answer is : A****Explanations :** After translation JSP page looks like :

```
public final class test_jsp extends org.apache.jasper.runtime.HttpJspBase
implements org.apache.jasper.runtime.JspSourceDependent {

    ...
}
```

**4. What is the output of the below test.jsp ?**

```
//test.jsp
<%!
public void _jspService(HttpServletRequest request, HttpServletResponse
response)

throws java.io.IOException, ServletException {
    out.println("Hello");
}
%>
a. Hello
b. Compile Error -_jspService(javax.servlet.http.HttpServletRequest,javax.servlet.http.
HttpServletResponse) is already defined in org.apache.jsp.test_jsp

c. Runtime exception
d. None of the above
```

**Answer is : D****Explanations :** After translation JSP page \_jspService automatically created by JSP compiler. In the JSP page if you define method name \_jspService(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletResponse) then compiler will complain \_jspService(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletResponse) is already defined in org.apache.jsp.test\_jsp.**5. Which of the following JSP life cycle methods we should not override ?**

- A) jspInit()      B )\_jspService  
C )jspDestroy      D )All of the above.

**Answer:B** We cannot override the \_jspService method which is called from the generated servlet's service method.**6. Which of the following is not a standard method called as part of the JSP life cycle?**

- A. jspInit()      B.jspService()  
C.\_jspService()      D.jspDestroy()

**Answer (B):** The standard service method for JSP has an \_ in its name

**7. How many copies of a JSP page can be in memory at a time?**

- |                 |                     |
|-----------------|---------------------|
| <u>A.</u> One   | <u>B.</u> Two       |
| <u>C.</u> Three | <u>D.</u> Unlimited |

**Answer:A****8. How does Tomcat execute a JSP?**

- A. As a CGI script
- B. As an independent process
- C. By one of Tomcat's threads
- D. None of the above is correct.

**Answer:A****9. How can we pre compile a JSP ?**

- A ) Invoke JSP by appending query string '?jsp\_precompile'
- B ) Invoke JSP by appending query string '?jsp-precompile=true'
- C ) Invoke JSP after appending query string '?precompile=true'
- D ) We cannot precompile a JSP page. We need to wait till the first request come, to compile JSP.

**Answer:A**

The JSP specification suggests a way to pre compile the JSP by appending the query string '?jsp\_precompile' without sending the actual request. But it is not mandatory to implement this feature. It is vendor dependent. The vendor can decide whether he should compile the JSP when he gets a call with the query string '?jsp\_precompile'

```
<%
    request.setAttribute("name","abc");
    session.setAttribute("name","xyz");
    application.setAttribute("name","pqr");
%>
```

**10. What will be the return value if we are calling <%= pageContext.getAttribute("name") %> on the same page?**

- A ) null
- B ) abc
- C ) xyz
- D ) pqr

**Answer:B**

The findAttribute will first look in the page scope for an attribute. If it finds one, it will return that value. If it is not able to find an attribute in page scope it will start looking at other scopes from most restrictive to least restricted scope. i.e, first request, then session and finally application. If it cannot find the attribute in any of the scope it will return null.

**11. Which of the following piece of code correctly set an attribute in application scope ?**

- A. We cannot set attributes to application scope using pageContext.
- B. <% pageContext.setAttribute("name", "albin", PageContext\_APPLICATION\_SCOPE); %>
- C. <% pageContext.setAttribute("name", "albin", PageContext.APPLICATION\_SCOPE); %>
- D. <% pageContext.setAttribute("name", "albin", PageContext.CONTEXT\_SCOPE); %>

**Answer:C**

The three arguments version of setAttribute method is used to set an attribute in other scopes using pageContext.

**12. Which of the following is legal JSP syntax to print the value of i. Select one correct answer**

- A. <%int i = 1;%>  
<%= i; %>
- B. <%int i = 1;  
i; %>
- C. <%int i = 1%>  
<%= i %>

- D. <%int i = 1;%>  
     <%= i %>  
 E. <%int i = 1%>  
     <%= i; %>

**Answer:D**

When using scriptlets (that is code included within <% %>), the included code must have legal Java syntax. So the first statement must end with a semi-colon. The second statement on the other hand is a JSP expression. So it must not end with a semi colon.

**13. A JSP page called test.jsp is passed a parameter name in the URL using [http://localhost/test.jsp?name="Nit"](http://localhost/test.jsp?name='Nit'). The test.jsp contains the following code.**

```
<%! String myName=request.getParameter();%>
<% String test= "welcome" + myName; %>
<%= test%>
```

- A. The program prints "Welcome Nit"
- B. The program gives a syntax error because of the statement  
`<%! String myName=request.getParameter();%>`
- C. The program gives a syntax error because of the statement  
`<% String test= "welcome" + myName; %>`
- D. The program gives a syntax error because of the statement  
`<%= test%>`

**Answer B.** JSP declarations do not have access to automatically defined variables like request, response etc

**14. Which of the following correctly represents the following JSP statement. Select one correct answer.**

```
<%=x%>
A. <jsp:expression=x/>
B. <jsp:expression>x</jsp:expression>
C. <jsp:statement>x</jsp:statement>
D. <jsp:declaration>x</jsp:declaration>
E. <jsp:scriptlet>x</jsp:scriptlet>
```

**Answer: B.** The XML syntax for JSP expression is <jsp:expression>Java expression</jsp:expression>

**15. Which of the following correctly represents the following JSP statement. Select one correct answer.**

```
<%x=1;%>
A. <jsp:expression x=1;/>
B. <jsp:expression>x=1;</jsp:expression>
C. <jsp:statement>x=1;</jsp:statement>
D. <jsp:declaration>x=1;</jsp:declaration>
E. <jsp:scriptlet>x=1;</jsp:scriptlet>
```

**Answer: E.** The XML syntax for JSP scriptlets is <jsp:scriptlet>Java code</jsp:scriptlet>

**Objective Questions on tag library**

1. When implementing a tag, if the tag just includes the body verbatim, or if it does not include the body, then the tag handler class must extend the BodyTagSupport class. Is this statement true or false.
2. Fill in the blanks. A tag handler class must implement the javax.servlet.jsp.tagext.Tag interface. This is accomplished by extending the class TagSupport or another class named in one of the options below. Select one correct answer.
  - A. IterationTag
  - B. TagClass
  - C. BodyTag
  - D. BodyTagSupport
3. Is this statement true or false. The deployment descriptor of a web application must have the name web.xml . In the same way the tag library descriptor file must be called taglib.xml .
4. A JSP file that uses a tag library must declare the tag library first. The tag library is defined using the taglib directive - <%= taglib uri="..." prefix="..."%>
 Which of the following specifies the correct purpose of prefix attribute. Select the one correct answer.

- A. The prefix defines the name of the tag that may be used for a tag library.
  - B. The prefix attribute defines the location of the tag library descriptor file.
  - C. The prefix attribute should refer to the short name attribute of the tag library file that is defined by the uri attribute of taglib directive.
  - D. The prefix attribute is used in front of a tagname of a tag defined within the tag library.
5. A JSP file uses a tag as <myTaglib:myTag>. The myTag element here should be defined in the tag library descriptor file in the tag element using which element. Select one correct answer.
- A. tagname
  - B. name
  - C. tag
  - D. prefix
6. Which of the elements defined within the taglib element of taglib descriptor file are required. Select two correct answers.
- A. tlib-version
  - B. short-name
  - C. uri
  - D. display-name
7. Which of the elements defined within the taglib element of taglib descriptor files are required. Select two correct answers.
- A. name
  - B. description
  - C. validator
  - D. tag-class
  - E. display-name
8. Name the element within the tag element that defines the name of the class that implements the functionality of tag. Select one correct answer.
- A. class-name
  - B. tag
  - C. class
  - D. tag-class
  - E. tei-class
9. Which of these are legal return types of the doStartTag method defined in a class that extends TagSupport class. Select two correct answers.
- A. EVAL\_PAGE
  - B. EVAL\_BODY
  - C. EVAL\_PAGE\_INCLUDE
  - D. EVAL\_BODY\_INCLUDE
  - E. SKIP\_PAGE
  - F. SKIP\_BODY
  - G. SKIP\_PAGE\_INCLUDE
  - H. SKIP\_BODY\_INCLUDE
10. Which of these are legal return types of the doAfterBody method defined in a class that extends TagSupport class. Select two correct answers.
- A. EVAL\_PAGE
  - B. EVAL\_BODY
  - C. EVAL\_PAGE AGAIN
  - D. EVAL\_BODY AGAIN
  - E. SKIP\_PAGE
  - F. SKIP\_BODY
  - G. SKIP\_PAGE AGAIN
  - H. SKIP\_BODY AGAIN
11. Which of these are legal return types of the doEndTag method defined in a class that extends TagSupport class. Select two correct answers.
- A. EVAL\_PAGE
  - B. EVAL\_BODY
  - C. EVAL\_PAGE\_INCLUDE
  - D. EVAL\_BODY\_INCLUDE
  - E. SKIP\_PAGE
  - F. SKIP\_BODY
  - G. SKIP\_PAGE\_INCLUDE
  - H. SKIP\_BODY\_INCLUDE

**Answers to questions on tag library**

1. false. Such a class should extend the TagSupport class.

2. D. BodyTagSupport extends Tag Support and implements interfaces Tag and IterationTag.
3. false. The tag library descriptor file can have any name. It need not have the name taglib.xml .
4. d. If the taglib directive directive defines a prefix of test, and a tag is called myTag, then the tag is used as <test:myTag>.
5. b. The name element inside the tag element defines the tag name to which the prefix of the taglib is attached. For example <name> myTag </name>
6. a,b. tlib-version and short-name are required elements within the taglib element of the tag library descriptor file.
7. a,d. name and tag-class are required elements within the tag element of tag library descriptor file.
8. d. tag-class element defines the fully qualified class name of the tag in the TLD file.
9. d,f. EVAL\_BODY\_INCLUDE, and SKIP\_BODY are legal return types of doStartTag.
10. d,f. EVAL\_BODY\_AGAIN, and SKIP\_BODY are legal return types of doAfterBody.
11. a,e. EVAL\_PAGE and SKIP\_PAGE are legal return types of doEndTag

**What gets printed when the following JSP code is invoked in a browser. Select one correct answer.**

```
<%= if(Math.random() < 0.5) %>
  hello
<%= } else { %>
  hi
<%= } %>
```

- A. The browser will print either hello or hi based upon the return value of random.
- B. The string hello will always get printed.
- C. The string hi will always get printed.
- D. The JSP file will not compile.

Answer: D. The if statement, else statement and closing parenthesis are JSP scriptlets and not JSP expressions. So these should be included within <% } %>

**Which of the following are correct. Select one correct answer.**

- A. JSP scriptlets and declarations result in code that is inserted inside the \_jspService method.
- B. The JSP statement <%! int x; %> is equivalent to the statement <jsp:scriptlet>int x;</jsp:scriptlet%>.
- C. The following are some of the predefined variables that maybe used in JSP expression - httpSession, context.
- D. To use the character %> inside a scriptlet, you may use %\> instead.

Answer:D. JSP declarations are inserted outside of \_jspService method. Hence a is incorrect. The JSP statement <%!int a;%> is equivalent to <jsp:declaration>int x;</jsp:declaration>. Hence b is incorrect. The predefined variables that are available within the JSP expression are session and pageContext, and not httpSession and context. Hence c is incorrect.

**What gets printed when the following is compiled. Select one correct answer.**

```
<% int y = 0; %>
<% int z = 0; %>

<% for(int x=0;x<3;x++) { %>
<% z++;++y;%>
<% }%>
```

```
<% if(z<y) {%
<%= z%>
<% } else {%
<%= z - 1%>
<% }%>
```

A. 0  
 B. 1  
 C. 2  
 D. 3  
 E. The program generates compilation error.

**Answer:**C. After the for loop z and y are both set to 3. The else statement gets evaluated, and 2 gets printed in the browser.

**Which of the following JSP variables are not available within a JSP expression. Select one correct answer.**

- A. out
- B. session
- C. request
- D. response
- E. httpsession
- F. page

**Answer: E. There is no such variable as httpsession.**

**How can we create a thread safe JSP ?**

- A ) <%@ page isThreadSafe = "true" %>
- B ) <%@ page implements="SingleThreadModel" %>
- C ) <%@ page isThreadSafe="false" %>
- D ) <%@ page extends="SingleThreadModel" %>

**Answer: C**

The is ThreadSafe attribute of page directive defines whether the generated servlet needs to implement SingleThreadModel. The default value for isThreadSafe attribute is true and if false, the generated servlet will implement SingleThreadModel

**How to set the MIME type of generated servlet to 'image/gif' ?**

- A ) We cannot set JSP's content type to image/gif. The content type of JSP is always text/html only.
- B ) <%@ page contentType="image/gif" %>
- C ) <%@ page content-Type="image/gif" %>
- D ) <%@ page setMimeType="image/gif" %>

**Answer:C**

The contentType attribute of page directive sets the MIME type for JSP response.

**What is the default value of session Attribute in JSP ?**

- A)<%@ page session="true" %>
- B)<%@ page session="false" %>
- C)<%@ page session="yes" %>
- D)<%@ page session="no" %>

Correct answer is : A

**Explanations :** <%@ page session="true" %> is the default value.

Questions no -2

**A JSP page does not contain page attribute and trying to access exception implicit variable.**

- A)exception implicit available not available in the JSP page.
- B)exception implicit available is available in the JSP page.
- C)No relation between isErrorPage attribute and exception implicit variable.
- D)None of the above

**Correct answer is : A**

**Explanations :** exception implicit variable not available in all JSP pages. Need to add page attribute to the JSP to available exception implicit in the JSP page.

**Which code snippet correctly declares the current JSP page to be an error page?**

- A )<%@ page isErrorPage="true" %>
- B )<%@ page isErrorPage="true" %>

C )<%@ errorPage="true" %>

D )All JSP pages are error pages only, we don't need to declare it.

**Answer:B**

The isErrorPage attribute of page directive is used to specify the current JSP page to be an error page. If isErrorPage is false, then the implicit object exception will not be available in this file.

**Which code correctly sets an error page for a JSP?**

A )<%@ page isErrorPage="true" %>

B )<%@ page errorPage="mypage.jsp" %>

C )<%@ page isErrorPage="myerrorpage.jsp" %>

D )<%@ page isErrorPage="false" %>

**Answer B:**

The errorPage attribute of page directive is used to specify the error page for a JSP page. The page specified in the errorPage attribute must have isErrorPage value true.

#### Objective Questions on JSTL and EL

1. What gets printed when the following expression is evaluated? Select one correct answer.

$\${1==2 ? 4 : 5}$

- A. 1
- B. 2
- C. 4
- D. 5

**Answer:D**

2. What gets printed when the following expression is evaluated? Select one correct answer.

$\${4 \text{ div } 5}$

- A. 0
- B. 0.8
- C. 1
- D. -1

**Answer :B div operator is used for dividing in EL.**

3. What gets printed when the following expression is evaluated? Select one correct answer.

$\${12 \% 4}$

- A. 0
- B. 3
- C. 8
- D. 16

**Answer: A % operator gives the remainder after performing division.**

4. What is the effect of executing the following JSP statement, assuming a class with name Employee exists in classes package.

<%@ page import = "classes.Employee" %> <jsp:useBean id="employee" class="classes.Employee" scope="session"/> <jsp:setProperty name="employee" property="\*"/>

- A. The code does not compile as there is no property attribute of setProperty tag.
- B. The code does not compile as property attribute cannot take \* as a value.
- C. The code sets value of all properties of employee bean to \*.
- D. The code sets the values of all properties of employee bean to matching parameters in request object.

**Answer: D. This is a valid syntax for setProperty. All properties of the bean are set from the corresponding parameter names in the request object**

5. What is the effect of evaluation of following expression? Select the one correct answer.

$\${(5*5) ne 25}$

- A. true
- B. false
- C. 25
- D. The expression does not compile as ne is not a valid operator.

**Answer B: The code prints false. ne is a valid operator. Since both left hand side and right hand side are equal to 25, false gets printed.**

6. What is the effect of evaluation of following expression? Select the one correct answer.

$\${'cat' gt 'cap'}$

- A. true
- B. false

- C. catcap              D. The expression does not compile as gt operator cannot be applied on strings.

**Answer: A EL considers <cat> to be greater than <cap>, as the letter t comes after the letter p.**

7. How many numbers are printed, when the following JSTL code fragment is executed? Select the one correct answer.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:forEach var="item" begin="0" end="10" step="2">
```

- ```
${item}
```
- ```
</c:forEach>
```
- A. 1              B. 5  
C. 6              D. 11

**Answer: C The following numbers get printed - 0, 2, 4, 6, 8, 10.**

8. What gets printed when the following JSTL code fragment is executed? Select the one correct answer.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:set var="item" value="2"/>
<c:if test="${var==1}" var="result" scope="session">
<c:out value="${result}"/>
</c:if>
```

- A. The JSTL code does not compile as attribute for if tag are not correct.  
B. true              C. false  
D. Nothing gets printed.

**Answer: D if evaluates to false, hence the c.out statement does not get executed.**

9. What gets printed when the following JSTL code fragment is executed? Select the one correct answer.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:set var="item" value="2"/>
<c:forEach var="item" begin="0" end="0" step="2">
<c:out value="${item}" default="abc"/>
</c:forEach>
```

- A. The JSTL code does not compile as an attribute for forEach tag is not correct.  
B. 0  
C. 2  
D. ABC  
E. Nothing gets printed as c.out statement does not get executed.

**Answer B: The forEach tag gets executed once, and prints zero.**

10. How many numbers gets printed when the following JSTL code fragment is executed?

Select the one correct answer.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:set var="item" value="2"/>
<c:choose>
<c:when test="${item>0}">
<c:out value="1"/>
</c:when>
<c:when test="${item==2}">
<c:out value="2"/>
</c:when>
<c:when test="${item<2}">
<c:out value="3"/>
</c:when>
<c:otherwise>
<c:out value="4"/>
</c:otherwise>
</c:choose>
```

- A. No number gets printed.              B. One number gets printed.  
C. Two numbers gets printed.              D. Three numbers gets printed.  
E. Four numbers gets printed.

**Answer B: Only one number gets printed - the number 1.**

11. Which numbers gets printed when the following JSTL code fragment is executed? Select the two correct answers.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:set var="j" value="4,3,2,1"/>
<c:forEach items="${j}" var="item" begin="1" end="2">
<c:out value="${item}" default="abc"/>
</c:forEach>
```

- A. 1      B. 2  
 C. 3      D. 4  
 E. abc

The program does not compile.

**Answer B, C: In this case the forEach tag iterates through two elements of the array named j.**

**12. Which numbers gets printed when the following JSTL code fragment is executed? Select the two correct answers.**

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:set var="j" value="4,3,2,1"/>
<c:forEach items="${j}" var="item" begin="1" end="2" varStatus="status">
<c:out value="${status.count}" default="abc"/>
</c:forEach>
```

- A. 1      B. 2  
 C. 3      D. 4  
 E. abc

The program does not compile.

**Answer: B, C. varStatus is set to a class of type LoopTagStatus. This class has a property named count which is being printed. count is the loop index, beginning with 1. So for two iterations 1 and 2 get printed. In this case the forEach tag iterates through two elements of the array named j.**

**13. Which number gets printed when the following JSTL code fragment is executed? Select the one correct answers.**

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:set var="j" value="4,3,2,1"/>
<c:forEach items="${j}" var="item" varStatus="status">
<c:if test="${status.first}">
<c:out value="${status.index}" default="abc"/>
</c:if>
</c:forEach>
```

- a. 1      b. 2      c. 3      d. 4  
 E. abc

The program does not compile.

**Answer:A status.first is true for the first iteration. The index is set to 0 in the first iteration.**

**14.Which of these represent the correct path for the core JSTL library in JSTL version 1.1? Select the one correct answer.**

- a. <http://java.sun.com/jsp/jstl/core>    b. <http://java.sun.com/jsp/core>  
 c. <http://java.sun.com/core>    d. <http://java.sun.com/jsp/jstl1.1/core>

**Answer:A The path of core tag library in JSTL 1.1 is <http://java.sun.com/jsp/jstl/core>**

#### **Q: What is a output comment?**

A comment that is sent to the client in the viewable page source. The JSP engine handles an output comment as uninterpreted HTML text, returning the comment in the HTML output sent to the client. You can see the comment by viewing the page source from your Web browser.

#### **JSP Syntax**

<!-- comment [ <%= expression %> ] -->

#### **Example 1**

```
<!-- This is a comment sent to client on
<%= (new java.util.Date()).toLocaleString() %>
-->
```

#### **Q. Displays in the page source:**

<!-- This is a comment sent to client on January 24, 2004 -->

#### **Q. What are stored procedures? How is it useful?**

A stored procedure is a set of statements/commands which reside in the database. The stored procedure is pre-compiled and saves the database the effort of parsing and compiling SQL statements everytime a query is run. Each database has its own stored procedure language, usually a variant of C with a SQL

preprocessor. Newer versions of db's support writing stored procedures in Java and Perl too. Before the advent of 3-tier/n-tier architecture it was pretty common for stored procs to implement the business logic( A lot of systems still do it). The biggest advantage is of course speed. Also certain kind of data manipulations are not achieved in SQL. Stored procs provide a mechanism to do these manipulations. Stored procs are also useful when you want to do Batch updates/exports/houseKeeping kind of stuff on the db. The overhead of a JDBC Connection may be significant in these cases.

#### **Q.What is a Hidden Comment?**

A comments that documents the JSP page but is not sent to the client. The JSP engine ignores a hidden comment, and does not process any code within hidden comment tags. A hidden comment is not sent to the client, either in the displayed JSP page or the HTML page source. The hidden comment is useful when you want to hide or "comment out" part of your JSP page.

You can use any characters in the body of the comment except the closing `-%>` combination. If you need to use `-%>` in your comment, you can escape it by typing `-%\>`.

#### JSP Syntax

```
<%-- comment --%>
```

#### Examples

```
<%@ page %>
<html>
<head><title>A Hidden Comment </title></head>
<body>
<%-- This comment will not be visible to the client in the page source --%>
</body>
</html>
```

#### **Q.How do I include static files within a JSP page?**

Static resources should always be included using the JSP include directive. This way, the inclusion is performed just once during the translation phase. note that you should always supply a relative URL for the file attribute. Although you can also include static resources using the action, this is not advisable as the inclusion is then performed for each and every request.

#### **Q.What are the two kinds of comments in JSP and what's the difference between them.**

```
<%-- JSP Comment --%>
```

```
<!-- HTML Comment -->
```

#### **Q.What is JSP technology?**

Java Server Page is a standard Java extension that is defined on top of the servlet Extensions. The goal of JSP is the simplified creation and management of dynamic Web pages. JSPs are secure, platform-independent, and best of all, make use of Java as a server-side scripting language.

#### **Q. What is JSP page?**

A JSP page is a text-based document that contains two types of text: static template data, which can be expressed in any text-based format such as HTML, SVG, WML, and XML, and JSP elements, which construct dynamic content.

#### **Q. What is a Expression?**

An expression tag contains a scripting language expression that is evaluated, converted to a String, and inserted where the expression appears in the JSP file. Because the value of an expression is converted to a String, you can use an expression within text in a JSP file. Like

```
<%= someexpression %>
```

```
<%= (new java.util.Date()).toLocaleString() %>
```

You cannot use a semicolon to end an expression

#### **Q.What is a Declaration?**

A declaration declares one or more variables or methods for use later in the JSP source file.

A declaration must contain at least one complete declarative statement. You can declare any number of variables or methods within one declaration tag, as long as they are separated by semicolons. The declaration must be valid in the scripting language used in the JSP file.

```
<%! somedeclarations %>
```

```
<%! int i = 0; %>
```

```
<%! int a, b, c; %>
```

#### **Q.What is a Scriptlet?**

A scriptlet can contain any number of language statements, variable or method declarations, or expressions that are valid in the page scripting language. Within scriptlet tags, you can

- 1.Declare variables or methods to use later in the file (see also Declaration).

- 2.Write expressions valid in the page scripting language (see also Expression).

- 3.Use any of the JSP implicit objects or any object declared with a `<jsp:useBean>` tag.

You must write plain text, HTML-encoded text, or other JSP tags outside the scriptlet.

Scriptlets are executed at request time, when the JSP engine processes the client request. If the scriptlet produces output, the output is stored in the `out` object, from which you can display it.

**Q.What are the implicit objects?**

Implicit objects are objects that are created by the web container and contain information related to a particular request, page, or application. They are:

- request
- response
- pageContext
- session
- application
- out
- config
- page
- exception

**Q. How many JSP scripting elements and what are they?**

There are three scripting language elements:

- declarations
- scriptlets
- expressions

**Q. Why are JSP pages the preferred API for creating a web-based client program?**

Because no plug-ins or security policy files are needed on the client systems(applet does). Also, JSP pages enable cleaner and more module application design because they provide a way to separate applications programming from web page design. This means personnel involved in web page design do not need to understand Java programming language syntax to do their job.

**Q.Is JSP technology extensible?**

YES. JSP technology is extensible through the development of custom actions, or tags, which are encapsulated in tag libraries.

**Q. Can we use the constructor, instead of init(), to initialize servlet?**

Yes, of course you can use the constructor instead of init(). There's nothing to stop you. But you shouldn't. The original reason for init() was that ancient versions of Java couldn't dynamically invoke constructors with arguments, so there is no way to give the constructur a ServletConfig. That no longer applies, but servlet containers still will only call your no-arg constructor. So you won't have access to a ServletConfig or ServletContext.

**Q. How can a servlet refresh automatically if some new data has entered the database?**

You can use a client-side Refresh or Server Push.

**Q: Difference between forward and sendRedirect?**

When you invoke a forward request, the request is sent to another resource on the server, without the client being informed that a different resource is going to process the request. This process occurs completely with in the web container. When a sendRedirect method is invoked, it causes the web container to return to the browser indicating that a new URL should be requested. Because the browser issues a completely new request any object that are stored as request attributes before the redirect occurs will be lost. This extra round trip a redirect is slower than forward.

**Q: What are the different scope values for the <jsp:useBean>?**

The different scope values for <jsp:useBean> are

1. page
2. request
- 3.session
- 4.application

**Q: Explain the life-cycle mehtods in JSP?**

A: The generated servlet class for a JSP page implements the HttpJspPage interface of the javax.servlet.jsp package. The HttpJspPage interface extends the JspPage interface which inturn extends the Servlet interface of the javax.servlet package. The generated servlet class thus implements all the methods of the these three interfaces. The JspPage interface declares only two mehtods - jspInit() and jspDestroy() that must be implemented by all JSP pages regardless of the client-server protocol. However the JSP specification has provided the HttpJspPage interfaec specifically for the JSp pages serving HTTP requests. This interface declares one method \_jspService().

The jspInit()- The container calls the jspInit() to initialize te servlet instance. It is called before any other method, and is called only once for a servlet instance.

The \_jspService()- The container calls the \_jspService() for each request, passing it the request and the response objects.

The jspDestroy()- The container calls this when it decides take the instance out of service. It is the last method called n the servlet instance.

**Q: How do I prevent the output of my JSP or Servlet pages from being cached by the browser?**

A: You will need to set the appropriate HTTP header attributes to prevent the dynamic content output by the JSP page from being cached by the browser. Just execute the following scriptlet at the beginning of your JSP pages to prevent them from being cached at the browser. You need both the statements to take care of some of the older browser versions.

```
<%
response.setHeader("Cache-Control","no-store"); //HTTP 1.1
response.setHeader("Pragma\","no-cache"); //HTTP 1.0
response.setDateHeader ("Expires", 0); //prevents caching at the proxy server
%>
```

**Q: What is the difference b/w variable declared inside a declaration part and variable declared in scriptlet part?**

A: Variable declared inside declaration part is treated as a global variable, that means after conversion jsp file into servlet that variable will be in outside of service method or it will be declared as instance variable. And the scope is available to complete jsp and to complete in the converted servlet class, where if u declare a variable inside a scriptlet that variable will be declared inside a service method and the scope is with in the service method.

**Q: Is there a way to execute a JSP from the commandline or from my own application?**

A: There is a little tool called JSPExecutor that allows you to do just that. The developers (Hendrik Schreiber <hs@webapp.de> & Peter Rossbach <pr@webapp.de>) aim was not to write a full blown servlet engine, but to provide means to use JSP for generating source code or reports. Therefore most HTTP-specific features (headers, sessions, etc) are not implemented, i.e. no response or header is generated. Nevertheless you can use it to precompile JSP for your website.

**Q: How does JSP handle run-time exceptions?**

A: You can use the errorPage attribute of the page directive to have uncaught run-time exceptions automatically forwarded to an error processing page. For example:

```
<%@ page errorPage="error.jsp" %> redirects the browser to the JSP page error.jsp if an uncaught exception is encountered during request processing. Within error.jsp, if you indicate that it is an error-processing page, via the directive: <%@ page isErrorPage="true" %> Throwable object describing the exception may be accessed within the error page via the exception implicit object. Note: You must always use a relative URL as the value for the errorPage attribute.
```

**Q: How can I implement a thread-safe JSP page? What are the advantages and Disadvantages of using it?**

A: You can make your JSPs thread-safe by having them implement the SingleThreadModel interface. This is done by adding the directive <%@ page isThreadSafe="false" %> within your JSP page. With this, instead of a single instance of the servlet generated for your JSP page loaded in memory, you will have N instances of the servlet loaded and initialized, with the service method of each instance effectively synchronized. You can typically control the number of instances (N) that are instantiated for all servlets implementing SingleThreadModel through the admin screen for your JSP engine. More importantly, avoid using the tag for variables. If you do use this tag, then you should set isThreadSafe to true, as mentioned above. Otherwise, all requests to that page will access those variables, causing a nasty race condition. SingleThreadModel is not recommended for normal use. There are many pitfalls, including the example above of not being able to use <%! %>. You should try really hard to make them thread-safe the old fashioned way: by making them thread-safe .

**Q: How do I use a scriptlet to initialize a newly instantiated bean?**

A: A jsp:useBean action may optionally have a body. If the body is specified, its contents will be automatically invoked when the specified bean is instantiated. Typically, the body will contain scriptlets or jsp:setProperty tags to initialize the newly instantiated bean, although you are not restricted to using those alone.

The following example shows the "today" property of the Foo bean initialized to the current date when it is instantiated. Note that here, we make use of a JSP expression within the jsp:setProperty action.

```
<jsp:useBean id="foo" class="com.Bar.Foo" >
<jsp:setProperty name="foo" property="today"
value="<%java.text.DateFormat.getDateInstance().format(new java.util.Date())%>" />
<%-- scriptlets calling bean setter methods go here --%>
</jsp:useBean >
```

**Q: How can I prevent the word "null" from appearing in my HTML input text fields when I populate them with a resultset that has null values?**

A: You could make a simple wrapper function, like

```
<%
String blanknull(String s) {
return (s == null) ? "\\" : s;
}
%>
```

then use it inside your JSP form, like

```
<input type="text" name="lastName" value="<%blanknull(lastName)%>" >
```

**Q: What's a better approach for enabling thread-safe servlets and JSPs? SingleThreadModel Interface or Synchronization?**

A: Although the SingleThreadModel technique is easy to use, and works well for low volume sites, it does not scale well. If you anticipate your users to increase in the future, you may be better off implementing explicit synchronization for your shared data. The key however, is to effectively minimize the amount of code that is synchronized so that you take maximum advantage of multithreading.

Also, note that SingleThreadModel is pretty resource intensive from the server's perspective. The most serious issue however is when the number of concurrent requests exhaust the servlet instance pool. In that case, all the unserviced requests are queued until something becomes free – which results in poor performance. Since the usage is non-deterministic, it may not help much even if you did add more memory and increase the size of the instance pool.

#### **Q: How can I enable session tracking for JSP pages if the browser has disabled cookies?**

A: We know that session tracking uses cookies by default to associate a session identifier with a unique user. If the browser does not support cookies, or if cookies are disabled, you can still enable session tracking using URL rewriting. URL rewriting essentially includes the session ID within the link itself as a name/value pair. However, for this to be effective, you need to append the session ID for each and every link that is part of your servlet response. Adding the session ID to a link is greatly simplified by means of a couple of methods: response.encodeURL() associates a session ID with a given URL, and if you are using redirection, response.encodeRedirectURL() can be used by giving the redirected URL as input. Both encodeURL() and encodeRedirectedURL() first determine whether cookies are supported by the browser; if so, the input URL is returned unchanged since the session ID will be persisted as a cookie.

Consider the following example, in which two JSP files, say hello1.jsp and hello2.jsp, interact with each other. Basically, we create a new session within hello1.jsp and place an object within this session. The user can then traverse to hello2.jsp by clicking on the link present within the page. Within hello2.jsp, we simply extract the object that was earlier placed in the session and display its contents. Notice that we invoke the encodeURL() within hello1.jsp on the link used to invoke hello2.jsp; if cookies are disabled, the session ID is automatically appended to the URL, allowing hello2.jsp to still retrieve the session object. Try this example first with cookies enabled. Then disable cookie support, restart the browser, and try again. Each time you should see the maintenance of the session across pages. Do note that to get this example to work with cookies disabled at the browser, your JSP engine has to support URL rewriting.

#### **hello1.jsp**

```
<%@ page session="true" %>
<%
Integer num = new Integer(100);
session.putValue("num",num);
String url =response.encodeURL("hello2.jsp");
%>
<a href='<%=url%>'>hello2.jsp</a>
```

#### **hello2.jsp**

```
<%@ page session="true" %>
<%
Integer i= (Integer )session.getValue("num");
out.println("Num value in session is " + i.intValue());
%>
```

#### **What is a JSP and what is it used for?**

Java Server Pages (JSP) is a platform independent presentation layer technology that comes with SUN's J2EE platform. JSPs are normal HTML pages with Java code pieces embedded in them. JSP pages are saved to \*.jsp files. A JSP compiler is used in the background to generate a Servlet from the JSP page.

#### **What is difference between custom JSP tags and beans?**

Custom JSP tag is a tag you defined. You define how a tag, its attributes and its body are interpreted, and then group your tags into collections called tag libraries that can be used in any number of JSP files. To use custom JSP tags, you need to define three separate components:

1. the tag handler class that defines the tag's behavior
2. the tag library descriptor file that maps the XML element names to the tag implementations
3. the JSP file that uses the tag library

When the first two components are done, you can use the tag by using taglib directive:

```
<%@ taglib uri="xxx.tld" prefix="..." %>
```

Then you are ready to use the tags you defined. Let's say the tag prefix is test:

MyJSPTag or

JavaBeans are Java utility classes you defined. Beans have a standard format for Java classes. You use tags to declare a bean and use to set value of the bean class and use to get value of the bean class.

```
<%=identifier.getClassField() %>
```

Custom tags and beans accomplish the same goals -- encapsulating complex behavior into simple and accessible forms. There are several differences:

Custom tags can manipulate JSP content; beans cannot.

Complex operations can be reduced to a significantly simpler form with custom tags than with beans. Custom tags require quite a bit more work to set up than do beans.

Custom tags usually define relatively self-contained behavior, whereas beans are often defined in one servlet and used in a different servlet or JSP page.

Custom tags are available only in JSP 1.1 and later, but beans can be used in all JSP 1.x versions.

**What are the two kinds of comments in JSP and what's the difference between them ?**

<%-- JSP Comment --%>

<!-- HTML Comment -->

**What is JSP technology?**

Java Server Page is a standard Java extension that is defined on top of the servlet Extensions. The goal of JSP is the simplified creation and management of dynamic Web pages. JSPs are secure, platform-independent, and best of all, make use of Java as a server-side scripting language.

**What is JSP page?**

A JSP page is a text-based document that contains two types of text: static template data, which can be expressed in any text-based format such as HTML, SVG, WML, and XML, and JSP elements, which construct dynamic content.

**What are the implicit objects?**

Implicit objects are objects that are created by the web container and contain information related to a particular request, page, or application. They are:

--request  
--response  
--pageContext  
--session  
--application  
--out  
--config  
--page  
--exception

**How many JSP scripting elements and what are they?**

There are three scripting language elements:

--declarations  
--scriptlets  
--expressions

**Why are JSP pages the preferred API for creating a web-based client program?**

Because no plug-ins or security policy files are needed on the client systems(applet does). Also, JSP pages enable cleaner and more modular application design because they provide a way to separate applications programming from web page design. This means personnel involved in web page design do not need to understand Java programming language syntax to do their job.

**Is JSP technology extensible?**

YES. JSP technology is extensible through the development of custom actions, or tags, which are encapsulated in tag libraries.

**Can we use the constructor, instead of init(), to initialize servlet?**

Yes , of course you can use the constructor instead of init(). There's nothing to stop you. But you shouldn't. The original reason for init() was that ancient versions of Java couldn't dynamically invoke constructors with arguments, so there was no way to give the constructor a ServletConfig. That no longer applies, but servlet containers still will only call your no-arg constructor. So you won't have access to a ServletConfig or ServletContext.

**How can a servlet refresh automatically if some new data has entered the database?**

You can use a client-side Refresh or Server Push.

**he code in a finally clause will never fail to execute, right?**

Using System.exit(1); in try block will not allow finally code to execute.

**How many messaging models do JMS provide for and what are they?**

JMS provide for two messaging models, publish-and-subscribe and point-to-point queuing.

**What information is needed to create a TCP Socket?**

The Local Systems IP Address and Port Number. And the Remote System's IPAddress and Port Number.

**What Class.forName will do while loading drivers?**

It is used to create an instance of a driver and register it with the DriverManager. When you have loaded a driver, it is available for making a connection with a DBMS.

**How to Retrieve Warnings?**

SQLWarning objects are a subclass of SQLException that deal with database access warnings. Warnings do not stop the execution of an application, as exceptions do; they simply alert the user that something did

not happen as planned. A warning can be reported on a Connection object, a Statement object (including PreparedStatement and CallableStatement objects), or a ResultSet object. Each of these classes has a getWarnings method, which you must invoke in order to see the first warning reported on the calling object

```
SQLWarning warning = stmt.getWarnings();
if (warning != null)
{
while (warning != null)
{
System.out.println("Message: " + warning.getMessage());
System.out.println("SQLState: " + warning.getSQLState());
System.out.print("Vendor error code: ");
System.out.println(warning.getErrorCode());
warning = warning.getNextWarning();
}
}
```

#### **How many JSP scripting elements are there and what are they?**

There are three scripting language elements: declarations, scriptlets, expressions.

#### **In the Servlet 2.4 specification SingleThreadModel has been deprecated, why?**

Because it is not practical to have such model. Whether you set isThreadSafe to true or false, you should take care of concurrent client requests to the JSP page by synchronizing access to any shared objects defined at the page level.

#### **What are stored procedures? How is it useful?**

A stored procedure is a set of statements/commands which reside in the database. The stored procedure is pre-compiled and saves the database the effort of parsing and compiling sql statements every time a query is run. Each database has its own stored procedure language, usually a variant of C with a SQL preprocessor. Newer versions of db's support writing stored procedures in Java and Perl too. Before the advent of 3-tier/n-tier architecture it was pretty common for stored procs to implement the business logic(A lot of systems still do it). The biggest advantage is of course speed. Also certain kind of data manipulations are not achieved in SQL. Stored procs provide a mechanism to do these manipulations. Stored procs are also useful when you want to do Batch updates/exports/houseKeeping kind of stuff on the db. The overhead of a JDBC Connection may be significant in these cases.

#### **How do I include static files within a JSP page?**

Static resources should always be included using the JSP include directive. This way, the inclusion is performed just once during the translation phase. Do note that you should always supply a relative URL for the file attribute. Although you can also include static resources using the action, this is not advisable as the inclusion is then performed for each and every request.

#### **Why does JComponent have add() and remove() methods but Component does not?**

because JComponent is a subclass of Container, and can contain other components and jcomponents. How can I implement a thread-safe JSP page? - You can make your JSPs thread-safe by having them implement the SingleThreadModel interface. This is done by adding the directive <%@ page isThreadSafe="false" %> within your JSP page.

#### **How do I prevent the output of my JSP or Servlet pages from being cached by the browser?**

You will need to set the appropriate HTTP header attributes to prevent the dynamic content output by the JSP page from being cached by the browser. Just execute the following scriptlet at the beginning of your JSP pages to prevent them from being cached at the browser. You need both the statements to take care of some of the older browser versions.

#### **How do you restrict page errors display in the JSP page?**

You first set "Errorpage" attribute of PAGE directory to the name of the error page (ie Errorpage="error.jsp") in your jsp page .Then in the error jsp page set "isErrorpage=TRUE". When an error occur in your jsp page it will automatically call the error page.

#### **How can I enable session tracking for JSP pages if the browser has disabled cookies?**

We know that session tracking uses cookies by default to associate a session identifier with a unique user. If the browser does not support cookies, or if cookies are disabled, you can still enable session tracking using URL rewriting. URL rewriting essentially includes the session ID within the link itself as a name/value pair.

However, for this to be effective, you need to append the session ID for each and every link that is part of your servlet response. Adding the session ID to a link is greatly simplified by means of a couple of methods: response.encodeURL() associates a session ID with a given URL, and if you are using redirection, response.encodeRedirectURL() can be used by giving the redirected URL as input. Both encodeURL() and encodeRedirectedURL() first determine whether cookies are supported by the browser; if so, the input URL is returned unchanged since the session ID will be persisted as a cookie.

Consider the following example, in which two JSP files, say hello1.jsp and hello2.jsp, interact with each other.

Basically, we create a new session within hello1.jsp and place an object within this session. The user can then traverse to hello2.jsp by clicking on the link present within the page. Within hello2.jsp, we simply extract the object that was earlier placed in the session and display its contents. Notice that we invoke the encodeURL() within hello1.jsp on the link used to invoke hello2.jsp; if cookies are disabled, the session ID is automatically appended to the URL, allowing hello2.jsp to still retrieve the session object. Try this example first with cookies enabled. Then disable cookie support, restart the browser, and try again. Each time you should see the maintenance of the session across pages.

Do note that to get this example to work with cookies disabled at the browser, your JSP engine has to support URL rewriting. hello1.jsp

```
hello2.jsp
hello2.jsp
<%
Integer i= (Integer )session.getValue("num");
out.println("Num value in session is "+i.intValue());
```

#### **What JSP lifecycle methods can I override?**

You cannot override the \_jspService() method within a JSP page. You can however, override the jspInit() and jspDestroy() methods within a JSP page. jspInit() can be useful for allocating resources like database connections, network connections, and so forth for the JSP page. It is good programming practice to free any allocated resources within jspDestroy().

The jspInit() and jspDestroy() methods are each executed just once during the lifecycle of a JSP page and are typically declared as JSP declarations:

#### **How do I perform browser redirection from a JSP page?**

You can use the response implicit object to redirect the browser to a different resource, as:

```
response.sendRedirect("http://www.exforsys.com/path/error.html");
```

You can also physically alter the Location HTTP header attribute, as shown below:

You can also use the: Also note that you can only use this before any output has been sent to the client. I believe this is the case with the response.sendRedirect() method as well. If you want to pass any parameters then you can pass using >

#### **How does JSP handle run-time exceptions?**

You can use the errorPage attribute of the page directive to have uncaught runtime exceptions automatically forwarded to an error processing page.

For example:

redirects the browser to the JSP page error.jsp if an uncaught exception is encountered during request processing. Within error.jsp, if you indicate that it is an error-processing page, via the directive: the Throwable object describing the exception may be accessed within the error page via the exception implicit object.

Note: You must always use a relative URL as the value for the errorPage attribute.

#### **How do I use comments within a JSP page?**

You can use "JSP-style" comments to selectively block out code while debugging or simply to comment your scriptlets. JSP comments are not visible at the client.

For example:

```
--%>
```

You can also use HTML-style comments anywhere within your JSP page. These comments are visible at the client. For example:

Of course, you can also use comments supported by your JSP scripting language within your scriptlets.

#### **Is it possible to share an HttpSession between a JSP and EJB? What happens when I change a value in the HttpSession from inside an EJB?**

You can pass the HttpSession as parameter to an EJB method, only if all objects in session are serializable. This has to be considered as "passed-by-value", that means that it's read-only in the EJB. If anything is altered from inside the EJB, it won't be reflected back to the HttpSession of the Servlet Container. The "pass-byreference" can be used between EJBs Remote Interfaces, as they are remote references.

While it IS possible to pass an HttpSession as a parameter to an EJB object, it is considered to be "bad practice" in terms of object oriented design. This is because you are creating an unnecessary coupling between back-end objects (ejbs) and front-end objects (HttpSession). Create a higher-level of abstraction for your ejb's api. Rather than passing the whole, fat, HttpSession (which carries with it a bunch of http

semantics), create a class that acts as a value object (or structure) that holds all the data you need to pass back and forth between front-end/back-end.

Consider the case where your ejb needs to support a non-http-based client. This higher level of abstraction will be flexible enough to support it.

#### **How can I implement a thread-safe JSP page?**

You can make your JSPs thread-safe by having them implement the SingleThreadModel interface. This is done by adding the directive `<%@ page isThreadSafe="false" %>` within your JSP page.

#### **How can I declare methods within my JSP page?**

You can declare methods for use within your JSP page as declarations. The methods can then be invoked within any other methods you declare, or within JSP scriptlets and expressions. Do note that you do not have direct access to any of the JSP implicit objects like request, response, session and so forth from within JSP methods. However, you should be able to pass any of the implicit JSP variables as parameters to the methods you declare.

#### **For example:**

Another Example:

file1.jsp:

file2.jsp

<%@test(out);%>

#### **Can I stop JSP execution while in the midst of processing a request?**

Yes. Preemptive termination of request processing on an error condition is a good way to maximize the throughput of a high-volume JSP engine. The trick (assuming Java is your scripting language) is to use the return statement when you want to terminate further processing.

#### **Can a JSP page process HTML FORM data?**

Yes. However, unlike Servlet, you are not required to implement HTTP-protocol specific methods like doGet() or doPost() within your JSP page. You can obtain the data for the FORM input elements via the request implicit object within a scriptlet or expression as.

#### **Is there a way to reference the "this" variable within a JSP page?**

Yes, there is. Under JSP 1.0, the page implicit object is equivalent to "this", and returns a reference to the Servlet generated by the JSP page.

#### **How do you pass control from one JSP page to another?**

Use the following ways to pass control of a request from one servlet to another or one jsp to another.

The RequestDispatcher object's forward method to pass the control.

The response.sendRedirect method

#### **Is there a way I can set the inactivity lease period on a per-session basis?**

Typically, a default inactivity lease period for all sessions is set within your JSPengine admin screen or associated properties file. However, if your JSP engine supports the Servlet 2.1 API, you can manage the inactivity lease period on a per-session basis. This is done by invoking the HttpSession.setMaxInactiveInterval() method, right after the session has been created.

#### **How does a servlet communicate with a JSP page?**

The following code snippet shows how a servlet instantiates a bean and initializes it with FORM data posted by a browser. The bean is then placed into the request, and the call is then forwarded to the JSP page, Bean1.jsp, by means of a request dispatcher for downstream processing.

```
public void doPost (HttpServletRequest request, HttpServletResponse response)
```

```
{
```

```
try {
```

```
govi.FormBean f = new govi.FormBean();
String id = request.getParameter("id");
f.setName(request.getParameter("name"));
f.setAddr(request.getParameter("addr"));
f.setAge(request.getParameter("age"));
```

```
//use the id to compute
```

```
//additional bean properties like info
```

```
//maybe perform a db query, etc.
```

```
// ...
```

```
f.setPersonalizationInfo(info);
request.setAttribute("fBean", f);
getServletConfig().getServletContext().getRequestDispatcher
("/jsp/Bean1.jsp").forward(request, response);
} catch (Exception ex) {
```

```
...  
}}
```

The JSP page Bean1.jsp can then process fBean, after first extracting it from the default request scope via the useBean action.

```
jsp:useBean id="fBean" class="govi.FormBean" scope="request"/ jsp:getProperty
name="fBean" property="name" / jsp:getProperty name="fBean" property="addr"
/jsp:getProperty name="fBean" property="age" / jsp:getProperty name="fBean"
property="personalizationInfo" /
```

#### **Can you make use of a ServletOutputStream object from within a JSP page?**

No. You are supposed to make use of only a JSPWriter object (given to you in the form of the implicit object out) for replying to clients.

A JSPWriter can be viewed as a buffered version of the stream object returned by response.getWriter(), although from an implementational perspective, it is not.

A page author can always disable the default buffering for any page using a page directive as:

#### **How do I include static files within a JSP page?**

Static resources should always be included using the JSP include directive. This way, the inclusion is performed just once during the translation phase.

The following example shows the syntax:

```
<% @ include file="copyright.html" %>
```

Do note that you should always supply a relative URL for the file attribute. Although you can also include static resources using the action, this is not advisable as the inclusion is then performed for each and every request.

How do I have the JSP-generated servlet subclass my own custom servlet class, instead of the default? One should be very careful when having JSP pages extend custom servlet classes as opposed to the default one generated by the JSP engine. In doing so, you may lose out on any advanced optimization that may be provided by the JSPengine.

In any case, your new super class has to fulfill the contract with the JSP engine by: Implementing the HttpJspPage interface, if the protocol used is HTTP, or implementing JspPage otherwise Ensuring that all the methods in the Servlet interface are declared final.

Additionally, your servlet super class also needs to do the following:

The service() method has to invoke the \_jspService() method

The init() method has to invoke the jspInit() method

The destroy() method has to invoke jspDestroy()

If any of the above conditions are not satisfied, the JSP engine may throw a translation error. Once the super class has been developed, you can have your JSP extend it as follows:

#### **Can a JSP page instantiate a serialized bean?**

No problem! The use Bean action specifies the beanName attribute, which can be used for indicating a serialized bean.

For example:

A couple of important points to note. Although you would have to name your serialized file "filename.ser", you only indicate "filename" as the value for the beanName attribute. Also, you will have to place your serialized file within the WEB-INF\jspbeans directory for it to be located by the JSP engine.

#### **How do you prevent the Creation of a Session in a JSP Page and why? What is the difference between include directive & jsp:include action?**

By default, a JSP page will automatically create a session for the request if one does not exist. However, sessions consume resources and if it is not necessary to maintain a session, one should not be created. For example, a marketing campaign may suggest the reader visit a web page for more information. If it is anticipated that a lot of traffic will hit that page, you may want to optimize the load on the machine by not creating useless sessions.

#### **How do I use a scriptlet to initialize a newly instantiated bean?**

A jsp:useBean action may optionally have a body. If the body is specified, its contents will be automatically invoked when the specified bean is instantiated. Typically, the body will contain scriptlets or jsp:setProperty tags to initialize the newly instantiated bean, although you are not restricted to using those alone.

The following example shows the "today" property of the Foo bean initialized to the current date when it is instantiated. Note that here, we make use of a JSP expression within the jsp:setProperty action. value="" />

#### **How can I set a cookie and delete a cookie from within a JSP page?**

A cookie, mycookie, can be deleted using the following scriptlet:

**How do you connect to the database from JSP?**

A Connection to a database can be established from a jsp page by writing the code to establish a connection using a jsp scriptlets.

Further then you can use the resultset object "res" to read data in the following way.

**What is the page directive is used to prevent a JSP page from automatically creating a session?**

<%@ page session="false">

**Can we implement an interface in a JSP?**

No

**What is the difference between ServletContext and PageContext?**

ServletContext: Gives the information about the container

PageContext: Gives the information about the Request

**What is the difference in using request.getRequestDispatcher() and context.getRequestDispatcher()?**

request.getRequestDispatcher(path): In order to create it we need to give the relative path of the resource  
context.getRequestDispatcher(path): In order to create it we need to give the absolute path of the resource.

**How to pass information from JSP to included JSP?**

Using <%jsp:param> tag.

**How is JSP include directive different from JSP include action. ?**

When a JSP include directive is used, the included file's code is added into the added JSP page at page translation time, this happens before the JSP page is translated into a servlet. While if any page is included using action tag, the page's output is returned back to the added page. This happens at runtime.

**Can we override the `jspInit()`, `_jspService()` and `jspDestroy()` methods?**

We can override `jspInit()` and `jspDestroy()` methods but not `_jspService()`.

**Why is `_jspService()` method starting with an '`_`' while other life cycle methods do not?**

`_jspService()` method will be written by the container hence any methods which are not to be overridden by the end user are typically written starting with an '`_`'. This is the reason why we don't override `_jspService()` method in any JSP page.

**What happens when a page is statically included in another JSP page?**

An include directive tells the JSP engine to include the contents of another file (HTML, JSP, etc.) in the current page. This process of including a file is also called as static include.

**A JSP page, `include.jsp`, has a instance variable "int a", now this page is statically included in another JSP page, `index.jsp`, which has a instance variable "int a" declared. What happens when the `index.jsp` page is requested by the client?**

Compilation error, as two variables with same name can't be declared. This happens because, when a page is included statically, entire code of included page becomes part of the new page. At this time there are two declarations of variable 'a'. Hence compilation error.

**Can you override `jspInit()` method? If yes, In which cases?**

ye, we can. We do it usually when we need to initialize any members which are to be available for a servlet/JSP throughout its lifetime.

**What is the difference between directive include and jsp include?**

<%@ include> : Used to include static resources during translation time. : Used to include dynamic content or static content during runtime.

**What is the difference between RequestDispatcher and sendRedirect?**

RequestDispatcher: server-side redirect with request and response objects. sendRedirect : Client-side redirect with new request and response objects.

**How does JSP handle runtime exceptions?**

Using `errorPage` attribute of page directive and also we need to specify `isErrorPage=true` if the current page is intended to URL redirecting of a JSP.

**How can my application get to know when a HttpSession is removed?**

Define a Class `HttpSessionNotifier` which implements `HttpSessionBindingListener` and implement the functionality what you need in `valueUnbound()` method.

Create an instance of that class and put that instance in `HttpSession`.

**What Class.forName will do while loading drivers?**

It is used to create an instance of a driver and register it with the `DriverManager`. When you have loaded a driver, it is available for making a connection with a DBMS.

**How many JSP scripting elements are there and what are they?**

There are three scripting language elements: declarations, scriptlets, expressions.

**In the Servlet 2.4 specification SingleThreadModel has been deprecated, why?**

Because it is not practical to have such model. Whether you set `isThreadSafe` to true or false, you should take care of concurrent client requests to the JSP page by synchronizing access to any shared objects defined at the page level.

**Is JSP technology extensible?**

YES. JSP technology is extensible through the development of custom actions, or tags, which are encapsulated in tag libraries.

**Can we use the constructor, instead of init(), to initialize servlet?**

Yes , of course you can use the constructor instead of init(). There's nothing to stop you. But you shouldn't. The original reason for init() was that ancient versions of Java couldn't dynamically invoke constructors with arguments, so there was no way to give the constructur a ServletConfig. That no longer applies, but servlet containers still will only call your no-arg constructor. So you won't have access to a ServletConfig or ServletContext.

**How can a servlet refresh automatically if some new data has entered the database?**  
You can use a client-side Refresh or Server Push.

**How many messaging models do JMS provide for and what are they?**

JMS provide for two messaging models, publish-and-subscribe and point-to-point queuing.

```
1 >>>>>>>>>>>>>>>>>>>> Additional Example Apps on JDBC>>>>>>>>>>>>>>
2 App1
3 -----OracleJoinRowSet.java-----
4 package com.nt;
5 import oracle.jdbc.rowset.OracleCachedRowSet;
6 import oracle.jdbc.rowset.OracleJoinRowSet;
7
8
9 class OracleJoinRowSetDemo {
10 public static void main(String args[]) {
11     try {
12         OracleJoinRowSet joinRS = new OracleJoinRowSet();
13
14         OracleCachedRowSet crs1 = new OracleCachedRowSet();
15         crs1.setUrl("jdbc:oracle:thin:@localhost:1521:orcl");
16         crs1.setUsername("scott");
17         crs1.setPassword("tiger");
18         crs1.setCommand("select empno,ename,deptno from emp");
19         crs1.execute();
20         crs1.setMatchColumn(3);
21
22         joinRS.addRowSet(crs1);
23
24
25         OracleCachedRowSet crs2 = new OracleCachedRowSet();
26         crs2.setUrl("jdbc:oracle:thin:@localhost:1521:orcl");
27         crs2.setUsername("scott");
28         crs2.setPassword("tiger");
29         crs2.setCommand("select * from dept");
30         crs2.execute();
31         crs2.setMatchColumn(1);
32
33         joinRS.addRowSet(crs2);
34
35
36     while (joinRS.next()) {
37         System.out.println(joinRS.getString(1)+" "+joinRS.getString(2)+" "+joinRS.getString(3)+" "+joinRS.getString(5));
38     }
39
40 } catch(Exception e) {
41     e.printStackTrace();
42 }
43 }
44 }
45 -----
46 App2
47 -----FilteredRowSet.java-----
48 package com.nt;
49
50 import java.sql.SQLException;
51
52 import javax.sql.RowSet;
53 import javax.sql.rowset.CachedRowSet;
54 import javax.sql.rowset.Predicate;
55
56 import oracle.jdbc.rowset.OracleFilteredRowSet;
57
58
59 class Filter1 implements Predicate {
60     private String colName;
```

```
61
62     public Filter1(String colName) {
63         this.colName = colName;
64     }
65
66     public boolean evaluate(ResultSet rs) {
67         try {
68             CachedRowSet crs = (CachedRowSet) rs;
69             String object = crs.getString(colName);
70             if (object != null && (object.charAt(0) == 'A' || object.charAt(0) == 'a')) {
71                 return true;
72             } else {
73                 return false;
74             }
75         } catch (Exception e) {
76             return false;
77         }
78     }
79
80     public boolean evaluate(Object arg0, int arg1) throws SQLException {
81         return false;
82     }
83
84
85     public boolean evaluate(Object arg0, String arg1) throws SQLException {
86         return false;
87     }
88 }
89
90
91     public class FilteredRowSet {
92         public static void main(String[] args) throws Exception {
93             OracleFilteredRowSet frs = new OracleFilteredRowSet();
94             frs.setUsername("scott");
95             frs.setPassword("tiger");
96             frs.setUrl("jdbc:oracle:thin:@localhost:1521:orcl");
97             frs.setCommand("select * from emp");
98             frs.setFilter(new Filter1("ename"));
99             frs.execute();
100            while (frs.next()) {
101                System.out.println(frs.getInt(1)+" "+frs.getString(2)+" "+frs.getString(3));
102            }
103        }
104    }
105
106 -----
107 App3 >>> Example App Oracle Array Type
108 -----InsertPassportDetails.java-----
109 package com.nt;
110 import java.sql.Connection;
111 import java.sql.Driver;
112 import java.sql.DriverManager;
113 import java.sql.PreparedStatement;
114 import java.util.Properties;
115
116 import oracle.sql.ARRAY;
117 import oracle.sql.ArrayDescriptor;
118 public class InsertPassportDetails {
119     public static void main(String[] args) throws Exception {
120         Class.forName("oracle.jdbc.driver.OracleDriver");
```

```
121 Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
122
123 PreparedStatement ps=con.prepareStatement("INSERT INTO EMPPASSPORTDETAILS VALUES(?, ?, ?)");
124 ps.setInt(1,99);
125
126 ps.setString(2,"9989A555");
127
128 String s1[]={ "v1", "v2", "v3", "v4", "v5"};
129 ArrayDescriptor ad=ArrayDescriptor.createDescriptor("VISA_NOS",con);
130 ARRAY a=new ARRAY(ad, con, s1);
131 ps.setArray(3, a);
132
133 int i=ps.executeUpdate();
134 System.out.println("Rows Inserted,Count:"+i);
135
136 con.close();
137 }
138 }
139
140 /* SQL> CREATE TYPE VISA_NOS AS VARRAY(5) OF VARCHAR2(19)
141 2 /
142 Type created.
143 SQL> CREATE TABLE EMPPASSPORTDETAILS(EMPNO NUMBER,PASSPORTNO VARCHAR2(19),VISA_TAKEN VISA_NOS);
144 */
145
146 -----GetPassportDetails.java-----
147 package com.nt;
148 import java.sql.Array;
149 import java.sql.Connection;
150 import java.sql.Driver;
151 import java.sql.PreparedStatement;
152 import java.sql.ResultSet;
153 import java.util.Properties;
154 public class GetPassportDetails {
155 public static void main(String[] args) throws Exception {
156
157     Driver d=(Driver)Class.forName("oracle.jdbc.driver.OracleDriver").newInstance();
158
159     Properties p=new Properties();
160     p.put("user","scott");
161     p.put("password","tiger");
162     Connection con=d.connect("jdbc:oracle:thin:@localhost:1521:orcl",p);
163
164
165 PreparedStatement ps=con.prepareStatement("SELECT PASSPORTNO,VISA_TAKEN FROM EMPPASSPORTDETAILS WHERE EMPNO=99");
166     ResultSet rs=ps.executeQuery();
167     if(rs.next()){
168         System.out.println("\n Employee Found,His Passport Details are:\n");
169         System.out.println("PassportNo:" +rs.getString(1)+"\n");
170         System.out.print("Visa's Taken are :\n\t");
171         Array a=rs.getArray(2);
172         ResultSet rs1=a.getResultSet();
173         while(rs1.next()){
174             System.out.println(rs1.getString(2)+",");
175         }
176     }
177     con.close();
178 }
179 }
180 }
```

```
181 -----  
182 >>>> Example on Array Struct Data type  
183 -----StudentDetails.java-----  
184 package com.nt;  
185 import java.sql.SQLData;  
186 import java.sql.SQLException;  
187 import java.sql.SQLInput;  
188 import java.sql.SQLOutput;  
189 public class StudentAddress implements SQLData{  
190     private String street,city,state,typename;  
191     private int fno,pin;  
192     public String getStreet() {  
193         return street;  
194     }  
195     public void setStreet(String street) {  
196         this.street = street;  
197     }  
198     public String getCity() {  
199         return city;  
200     }  
201     public void setCity(String city) {  
202         this.city = city;  
203     }  
204     public String getState() {  
205         return state;  
206     }  
207     public void setState(String state) {  
208         this.state = state;  
209     }  
210     public String getTypename() {  
211         return typename;  
212     }  
213     public void setTypename(String typename) {  
214         this.typename = typename;  
215     }  
216     public int getFno() {  
217         return fno;  
218     }  
219     public void setFno(int fno) {  
220         this.fno = fno;  
221     }  
222     public int getPin() {  
223         return pin;  
224     }  
225     public void setPin(int pin) {  
226         this.pin = pin;  
227     }  
228     @Override  
229     public String getSQLTypeName() throws SQLException {  
230         return typename;  
231     }  
232     @Override  
233     public void readSQL(SQLInput stream, String name) throws SQLException {  
234         fno=stream.readInt();  
235     }  
236 }
```

```
241     street=stream.readString();
242     city=stream.readString();
243     state=stream.readString();
244     pin=stream.readInt();
245     typename=name;
246 }
247
248     @Override
249     public void writeSQL(SQLOutput stream) throws SQLException {
250         stream.writeInt(fno);
251         stream.writeString(city);
252         stream.writeString(state);
253         stream.writeString(street);
254         stream.writeInt(pin);
255     }
256 }
257
258
259
260 -----InsertStudentDetails.java-----
261 package com.nt;
262 import java.io.File;
263 import java.io.FileInputStream;
264 import java.sql.Connection;
265 import java.sql.Driver;
266 import java.sql.DriverManager;
267 import java.sql.PreparedStatement;
268 import java.util.Properties;
269 public class InsertStudentDetails {
270     public static void main(String[] args) throws Exception{
271         Class.forName("oracle.jdbc.driver.OracleDriver");
272         Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
273
274         PreparedStatement ps=con.prepareStatement("INSERT INTO NARESHIT_STUDENTDETAILS(SNO,PHOTO,PERMENTADDRESS) VALUES(?, ?, ?)");
275
276         ps.setInt(1, 99);
277
278         File f=new File("C:\\\\Users\\\\Public\\\\Pictures\\\\Sample Pictures\\\\koala.jpg");
279         FileInputStream fis=new FileInputStream(f);
280         ps.setBinaryStream(2,fis,(int)f.length());
281
282         StudentAddress addr=new StudentAddress();
283         addr.setFno(19);
284         addr.setCity("hyd");
285         addr.setStreet("Ameerpet");
286         addr.setPin(500099);
287         addr.setState("TG");
288         addr.setTypename("NIT_STUDENTADDR2");
289
290         ps.setObject(3,addr);
291
292         int i=ps.executeUpdate();
293
294         System.out.println("Personal Details of Student 99 inserted successfully");
295
296         con.close();
297     }
298 }
299
300 /*SQL> CREATE TYPE NIT_STUDENTADDR2 AS OBJECT (FLATNO NUMBER,STREET VARCHAR2(19),CITY VARCHAR2(19),STATE VARCHAR2(19),PINCODE NUMBER)
```

```
301 2 /
302 Type created.
303 SQL> CREATE TABLE NARESHIT_STUDENTDETAILS(SNO NUMBER,PHOTO BLOB,PERMENTADDRESS NIT_STUDENTADDR2); */
304
305
306
307 -----GestStudentAddress.java-----
308 //GetStudentAddress.java
309 package com.nt;
310 import java.sql.Connection;
311 import java.sql.Driver;
312 import java.sql.DriverManager;
313 import java.sql.ResultSet;
314 import java.sql.Statement;
315 import java.util.HashMap;
316
317
318 public class GetStudentAddress {
319 public static void main(String[] args) throws Exception {
320
321     Class.forName("oracle.jdbc.driver.OracleDriver");
322 Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
323
324
325     Statement st=con.createStatement();
326 ResultSet rs=st.executeQuery("SELECT PERMENTADDRESS FROM NARESHIT_STUDENTDETAILS WHERE SNO=99");
327 if(rs.next()){
328     HashMap map=new HashMap();
329     map.put("NIT_STUDENTADDR2",StudentAddress.class);
330
331 StudentAddress addr=(StudentAddress)rs.getObject(1,map);
332
333 System.out.println("Student Address Found");
334 System.out.println("Flatno:"+addr.getFno());
335 System.out.println("Street:"+addr.getStreet());
336 System.out.println("Pin:"+addr.getPin());
337 System.out.println("State"+addr.getState());
338 }
339 con.close();
340 }
341 }
342
343
344 -----GetStudentUsingStruct.java-----
345 package com.nt;
346 import java.sql.Connection;
347 import java.sql.Driver;
348 import java.sql.DriverManager;
349 import java.sql.ResultSet;
350 import java.sql.Statement;
351 import java.sql.Struct;
352 import java.util.Properties;
353
354
355 public class GetStudentUsingStruct {
356 public static void main(String[] args) throws Exception {
357
358     Class.forName("oracle.jdbc.driver.OracleDriver");
359 Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
360 }
```

```
361
362     Statement st=con.createStatement();
363 ResultSet rs=st.executeQuery("SELECT PERMENTADDRESS FROM NARESHIT_STUDENTDETAILS WHERE SNO=99");
364 if(rs.next()){
365 System.out.println("Student Address Found");
366 Struct struct=(Struct)rs.getObject(1);
367 Object addr[]=struct.getAttributes();
368 System.out.println("Flatno:"+addr[0]);
369 System.out.println("Street:"+addr[1]);
370 System.out.println("Pin:"+addr[4]);
371 System.out.println();
372 }
373 con.close();
374 }
375 }
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
```



```
61  @Override
62  protected void doGet(HttpServletRequest request, HttpServletResponse response)
63      throws ServletException, IOException {
64      processRequest(request, response);
65  }
66
67  @Override
68  protected void doPost(HttpServletRequest request, HttpServletResponse response)
69      throws ServletException, IOException {
70      processRequest(request, response);
71  }
72
73
74 }//class
75 -----
76 App2 (Working with multiple hyperlinks)
77 -----page.html-----
78
79
80 <a href="linkurl?s1=link1">All Countries </a>
81 <br>
82 <br>
83 <a href="linkurl?s1=link2">All Languages </a>
84 <br>
85 <br>
86 <a href="linkurl?s1=link3">Sys Date </a>
87 -----LinksSrv.java-----
88 package com.nt;
89
90 import java.io.IOException;
91 import java.io.PrintWriter;
92 import java.util.Arrays;
93 import java.util.Date;
94 import java.util.Locale;
95 import javax.servlet.ServletException;
96 import javax.servlet.http.HttpServlet;
97 import javax.servlet.http.HttpServletRequest;
98 import javax.servlet.http.HttpServletResponse;
99
100 public class LinksSrv extends HttpServlet {
101     @Override
102     protected void doGet(HttpServletRequest req, HttpServletResponse res)
103         throws ServletException, IOException {
104         // General settings
105         PrintWriter pw=res.getWriter();
106         res.setContentType("text/html");
107         //read additional req param (s1) value
108         String pval=req.getParameter("s1");
109
110         if(pval.equals("link1")){
111             Locale locale[]=Locale.getAvailableLocales(); // USe all locale of this world
112             for(Locale loc:locale){
113                 pw.println(loc.getDisplayCountry()+" ");
114             }
115         }
116         else if(pval.equals("link2")){
117             Locale locale[]=Locale.getAvailableLocales();
118             for(Locale loc:locale){
119                 pw.println(loc.getDisplayLanguage()+" ");
120             }
121         }
122     }
123 }
```

```

121      }
122      else
123      {
124          pw.println("Date and Time"+new Date());
125      }
126  }//doGet(-,-)
127
128  @Override
129  protected void doPost(HttpServletRequest req, HttpServletResponse res)
130      throws ServletException, IOException {
131      doGet(req,res);
132  }
133
134 }
135 -----
136 App3 >>>> (Handling multiple submit Buttons and hyperlinks )
137 -----Form.html-----
138
139 <form action="turl">
140     Value 1 : <input type="text" name="t1"><br></br>
141     Value 2 : <input type="text" name="t2"><br></br>
142     <input type="submit" name="s1" value="add">
143     <input type="submit" name="s1" value="sub">
144     <input type="submit" name="s1" value="mul">
145     <br><br>
146     <a href="turl?s1=link1"> System properties </a> &nbsp; &nbsp;
147     <a href="turl?s1=link2"> Date and Time </a> &nbsp; &nbsp;
148 </form>
149 -----TestSrv3.java-----
150 package com.nt;
151 import java.io.IOException;
152 import java.io.PrintWriter;
153 import java.util.Date;
154 import javax.servlet.ServletException;
155 import javax.servlet.http.HttpServlet;
156 import javax.servlet.http.HttpServletRequest;
157 import javax.servlet.http.HttpServletResponse;
158
159 public class TestSrv3 extends HttpServlet {
160
161     protected void processRequest(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
162     {
163         //General settings
164         PrintWriter pw=res.getWriter();
165         res.setContentType("text/html");
166
167         //read s1 req param value
168         String pval=req.getParameter("s1");
169         if(pval.equals("link1")){
170             pw.println("Sys Properties===== "+System.getProperties());
171         }
172         else if(pval.equals("link2")){
173             pw.println("Date and Time :"+new Date());
174         }
175         else if(pval.equals("add")){
176             int v1=Integer.parseInt(req.getParameter("t1"));
177             int v2=Integer.parseInt(req.getParameter("t2"));
178             pw.println("Sum:"+ (v1+v2));
179         }
180         else if(pval.equals("sub")){

```

```
181     int v1=Integer.parseInt(req.getParameter("t1"));
182     int v2=Integer.parseInt(req.getParameter("t2"));
183     pw.println("Sub:"+ (v1-v2));
184 }
185 else{
186     int v1=Integer.parseInt(req.getParameter("t1"));
187     int v2=Integer.parseInt(req.getParameter("t2"));
188     pw.println("Mul:"+(v1*v2));
189 }
190 //close stream
191 pw.close();
192 }//processRequest(-,-)

193
194 protected void doGet(HttpServletRequest request, HttpServletResponse response)
195     throws ServletException, IOException {
196     processRequest(request, response);
197 }
198
199 protected void doPost(HttpServletRequest request, HttpServletResponse response)
200     throws ServletException, IOException {
201     processRequest(request, response);
202 }
203 }//class
204
205
206
207
208
209
210
211
212
213
```

