



ADVANCED JAVA

COURSE MATERIAL



TM

NARESH

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

technologies

SOFTWARE TRAINING & DEVELOPMENT

Branches : CHENNAI PH:044-6555 5557/58 VIJAYAWADA PH:0866-6576789 NELLORE PH:81068 22902

info@nareshit.com

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.

JDBC 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 Pacakage 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 SQL type.
4. SQL exception handling enhancements.
5. DataSet implementation of SQL using Annotations.
6. SQL XML support

JDBC 3.0 & JDBC 4.0

JDBC 3.0	JDBC 4.0
<p>Features:</p> <ul style="list-style-type: none"> - Reusability of prepared statements by connection pools. - 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. - 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. - Retrieval of parameter metadata. - It has added a means of retrieving values from columns containing automatically generated values. - Added a new data type i.e. java.sql.BOOLEAN. - Passing parameters to CallableStatement. - The data in the Blob and Clob can be altered: JDBC 3.0 introduces a standard mechanism for updating BLOB and CLOB data. - DatabaseMetaData API has been added. - It allows stored procedure parameters to be called by name. 	<p>Features:</p> <ul style="list-style-type: none"> - Auto-loading of JDBC driver class: In JDBC 4 invoking the getConnection() on DriverManager will automatically load a driver. - 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. - 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. - 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. - SQL XML support. - 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.

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 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. `DataSource` 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.

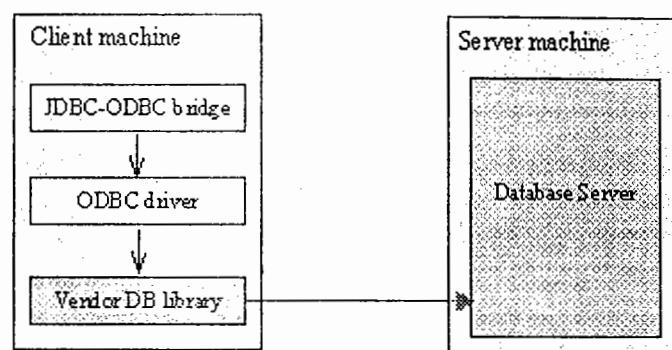


Figure 1. Type 1: JDBC-ODBC Bridge

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.

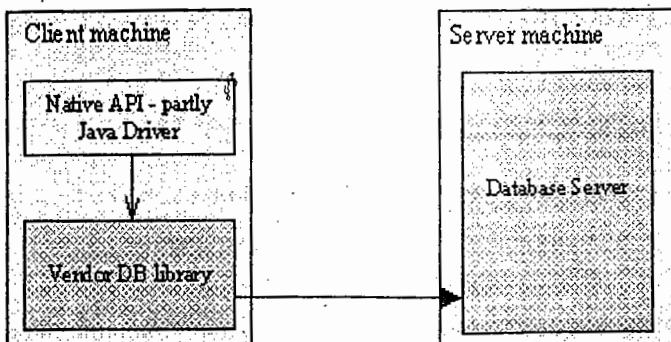


Figure 2. Type 2: Native API/partly Java driver

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.

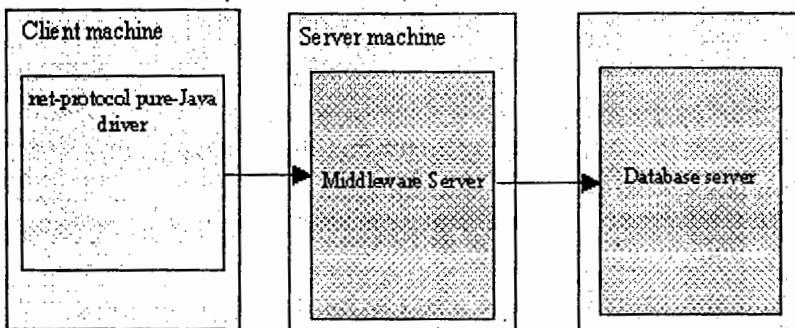


Figure 3. Type 3: Net-protocol/all-Java driver

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.

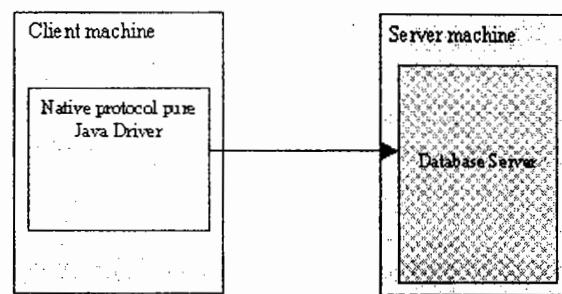


Figure 4. Type 4: Native-protocol/all-Java driver

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.

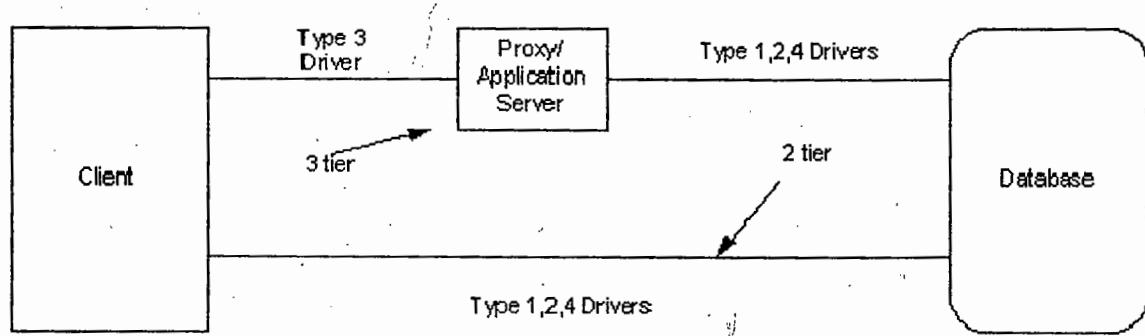
Driver is the key player in a JDBC application, it acts as a mediator between Java application and database. It implements JDBC API interfaces for a database, for example Oracle driver for oracle database, Sybase driver for Sybase database. It maps Java language to database specific language including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

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

Interface Summary

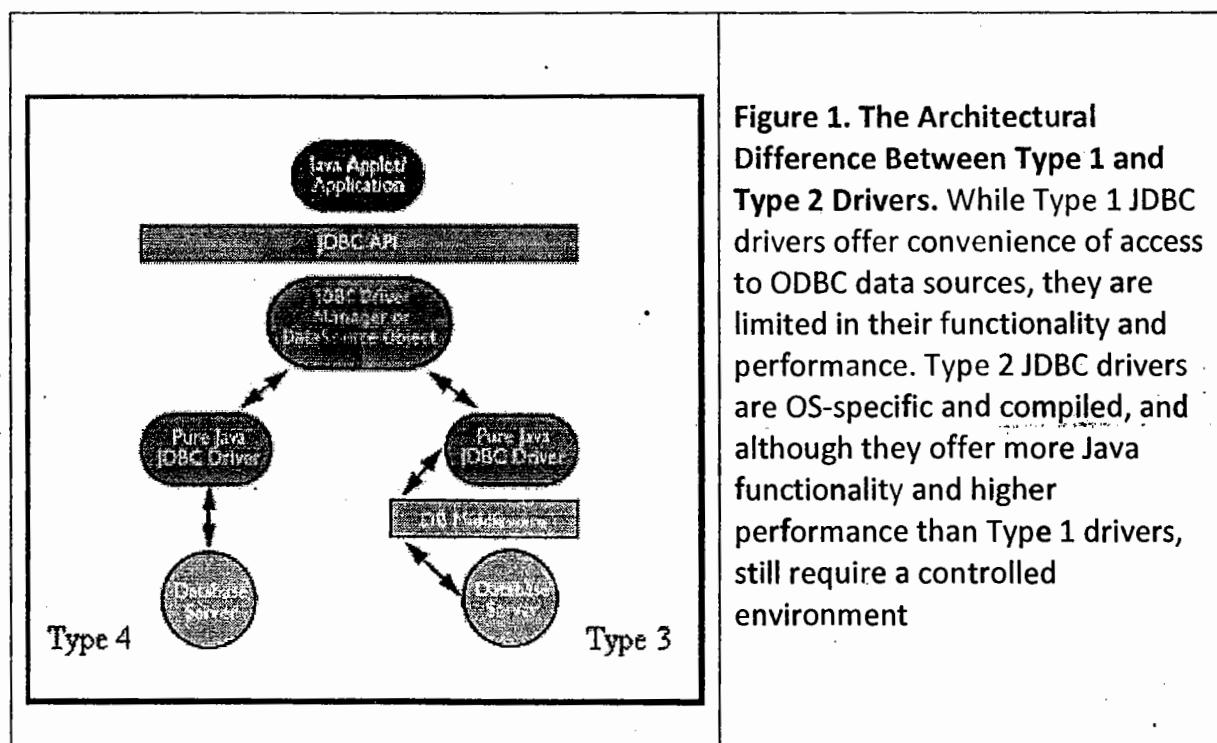
<u>Array</u>	The mapping in the Java programming language for the SQL type ARRAY.
<u>Blob</u>	The representation (mapping) in the Java™ programming language of an SQL BLOB value.
<u>CallableStatement</u>	The interface used to execute SQL stored procedures.
<u>Clob</u>	The mapping in the Java™ programming language for the SQL CLOB type.
<u>Connection</u>	A connection (session) with a specific database.
<u>DatabaseMetaData</u>	Comprehensive information about the database as a whole.
<u>Driver</u>	The interface that every driver class must implement.

<u>PreparedStatement</u>	An object that represents a precompiled SQL statement.
<u>Ref</u>	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.
<u>ResultSet</u>	A table of data representing a database result set, which is usually generated by executing a statement that queries the database.
<u>ResultSetMetaData</u>	An object that can be used to get information about the types and properties of the columns in a ResultSet object.
<u>SQLData</u>	The interface used for the custom mapping of SQL user-defined types.
<u>SQLInput</u>	An input stream that contains a stream of values representing an instance of an SQL structured or distinct type.
<u>SQLOutput</u>	The output stream for writing the attributes of a user-defined type back to the database.
<u>Statement</u>	The object used for executing a static SQL statement and obtaining the results produced by it.
<u>Struct</u>	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 :



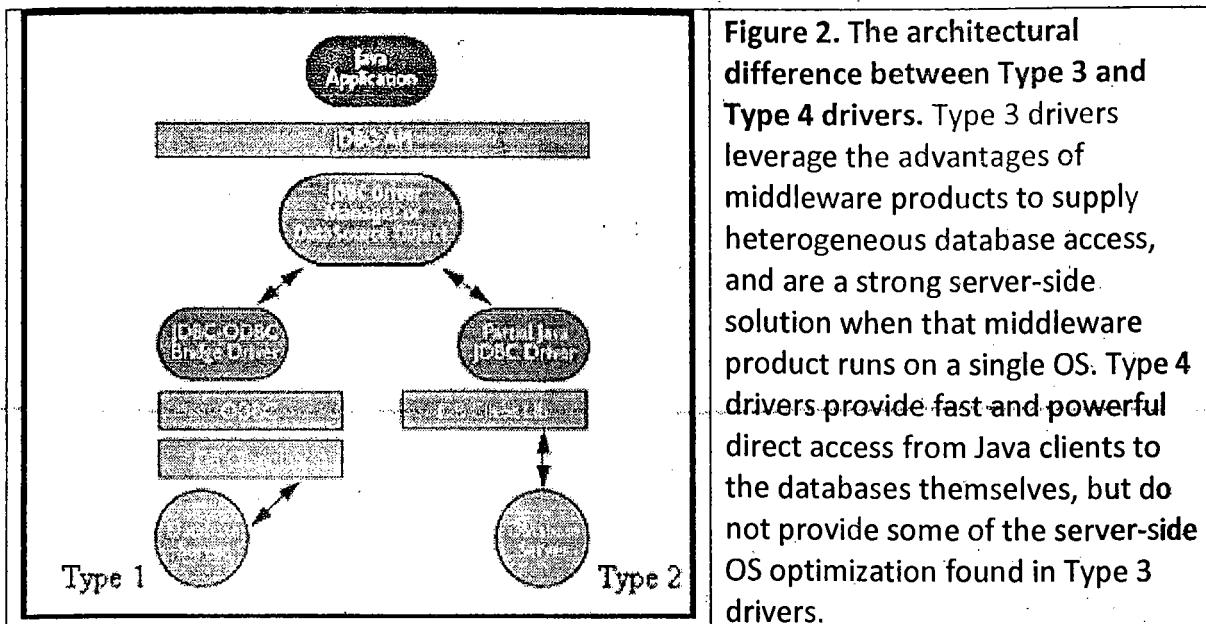


Figure 2. The architectural difference between Type 3 and Type 4 drivers. Type 3 drivers leverage the advantages of middleware products to supply heterogeneous database access, and are a strong server-side solution when that middleware product runs on a single OS. Type 4 drivers provide fast and powerful 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 Type 4 Limitations: Taking the Next Step

There is a movement afoot to address these limitations of Type 4 JDBC drivers by proposing the next step in 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.

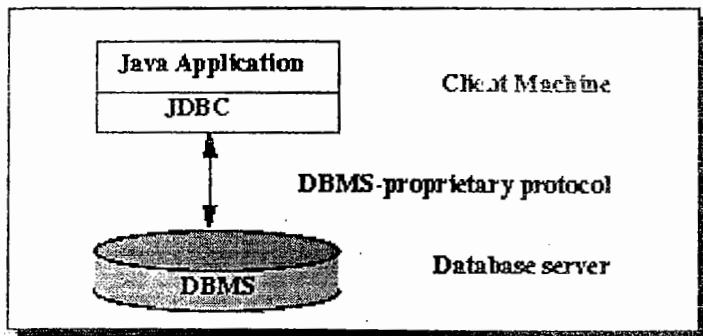
JDBC drivers have not traditionally been viewed as a strategic investment for many businesses. Type 5 JDBC has the potential to turn that trend on its head by offering businesses an easy way take advantage of years of innovation in database features, data access models, and virtualization technologies without requiring changes to the application. Whether the benefits sought are a simplified data connectivity infrastructure, accelerated front-line productivity, or dramatically reduced IT operational costs, businesses of all shapes and sizes should make it a priority to evaluate Type 5 JDBC drivers and continue to push for innovation over stagnation.

JDBC Architecture

Two-tier and Three-tier Processing Models

The JDBC API supports both two-tier and three-tier processing models for database access.

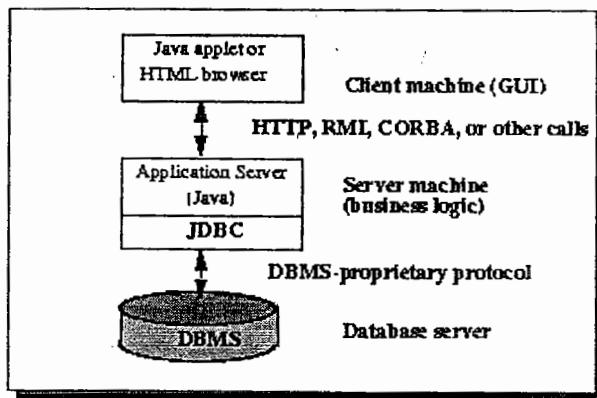
Figure 1: Two-tier Architecture for Data Access.



In the two-tier model, a Java applet or application talks directly to the data source. This requires a JDBC driver that can communicate with the particular data source being accessed. A user's commands are delivered to the database or other data source, and the results of those statements are sent back to the user. The data source may be located on another machine to which the user is connected via a network. This is referred to as a client/server configuration, with the user's machine as the client, and the machine housing the data source as the server. The network can be an intranet, which, for example, connects employees within a corporation, or it can be the Internet.

In the three-tier model, commands are sent to a "middle tier" of services, which then sends the commands to the data source, the data source processes the commands and sends the results back to the middle tier, which then sends them to the user. MIS directors find the three-tier model very attractive because the middle tier makes it possible to maintain control over access and the kinds of updates that can be made to corporate data. Another advantage is that it simplifies the deployment of applications. Finally, in many cases, the three-tier architecture can provide performance advantages.

Figure 2: Three-tier Architecture for Data Access.



Until recently, the middle tier has often been written in languages such as C or C++, which offer fast performance. However, with the introduction of optimizing compilers that translate Java bytecode into efficient machine-specific code and technologies such as Enterprise JavaBeans™, the Java platform is fast becoming the standard platform for middle-tier development. This is a big plus point, making it possible to take advantage of Java's robustness, multithreading, and security features.

With enterprises increased using the Java programming language for writing server code, the JDBC API is being used more and more in the middle tier of a three-tier architecture. Some of the features that make JDBC a server technology are its support for connection pooling, distributed

transactions, and disconnected rowsets. The JDBC API is also what allows access to a data source from a Java middle tier.

HardCoding Verses GeneralizedCoding

HardCoding

Directly writing values in the program is called as hardcoding.

Simple example is find sum of two numbers:-

SumDemo.java(HardCoding)

```
public class SumDemo{  
    public static void main(String[] args){  
        int a=5;//hard coded value  
        int b=6;//hard coded value  
        int c=a+b;  
        System.out.println(c);  
    }  
}
```

GeneralizedCoding

Taking (reading)values as input from external resources such as keyboard or properties file or xml file...

SumDemo.java (GeneralizedCoding)

```
import java.util.Scanner;  
public class SumDemo{  
    public static void main(String[] args){  
        Scanner sc=new Scanner(System.in);  
        System.out.printf("Enter 1st Number:");  
        int a=sc.nextInt();  
        System.out.printf("Enter 2nd Number:");  
        int b= sc.nextInt();  
        int c=a+b;  
        System.out.printf(c);  
    }  
}
```

In Realtime industry there are 3 types of environments for normal projects and 4 types for environments for sensitive projects. They are:

- 1) Development Environment
- 2) Testing Environment
- 3) Pre Production Environment
- 4) Post Production Environment

The project is developed at development environment, after development, the same project must be moved from development environment to testing environment for testing, after tested ok the same project must be moved from test environment to preproduction, preproduction for producing it to enduser(client). During testing, if the project is having some issue then, that project from test environment must be taken back to development environment. In every environment separate database server is maintained, but all database server systems should have the same data base management system software.

For example : Oracle11g

But one thing we which must be rembered is, if DataBase Server System ip address changes, database SID, portnumber, username and password may change.

If the above values(SID, portnumber, username, password) are hardcoded in JDBC project.

Whenever the project is moving from one environment to another environment then to execute that JDBC project we must modify the project source code.The project source code created in development environment souldnot be modified in any other environment. If modification is required, it should be done at development environment. To solve hard coding problem we must use generalized coding.Three types of generalized coding solutions are:

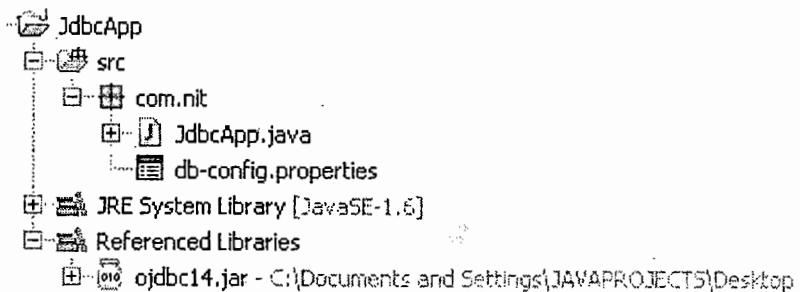
Solution one:-

Read driver class name, database url, database username and database password from keyboard.
(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 -----SelectTest1.java-----
5 //SelectTest.java
6 import java.sql.*;
7 public class SelectTest1
8 {
9     public static void main(String args[])throws Exception
10    {      //register jdbc driver
11        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
12        Connection con=DriverManager.getConnection("jdbc:odbc:oradsn","scott","tiger");
13        // create Staement obj
14        Statement st=con.createStatement();
15        // execute the query
16        ResultSet rs=st.executeQuery("select * from student");
17        //process the ResultSet
18        while(rs.next())
19        {
20            System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3))
21        }
22        //close all jdbc stream objects
23        rs.close();
24        st.close();
25        con.close();
26    }//main
27 } //class
28 //javac SelectTest1.java
29 //java SelectTest1
30 =====
31 App2) Select Operation
32 =====
33 -----SelectTest2.java-----
34 //SelectTest2.java
35 import java.sql.*; //jdbc api
36 import java.util.*; // Scanner class
37 public class SelectTest2
38 {
39     public static void main(String args[])throws Exception
40     {
41         // read inputs
42         Scanner sc=new Scanner(System.in);
43         System.out.println("Enter city");
44         String city=sc.next(); //gives hyd
45         city="""+city+"""; //gives 'hyd'
46
47         //register type1 jdbc driver
48         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
49         //Establish the connection with Db s/w.
50         Connection con=DriverManager.getConnection("jdbc:odbc:oradsn","scott","tiger");
51         //create Jdbc Statement obj
52         Statement st=con.createStatement();
53
54         //send and execute SQL Query in DB s/w
55         //select * from student where sadd='hyd'
56         ResultSet rs=st.executeQuery("select * from student where sadd='"+city);
57
58         //process the ResultSet
59         boolean flag=false;
60         while(rs.next()!=false){
61             System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3))
62             flag=true;
63         }//while
64         if(flag==false)
65             System.out.println("No records found");
66 }
```

```
67         //close jdbc objs
68         rs.close();
69         st.close();
70         con.close();
71     }//main
72 }//class
73 //>javac SelectTest2.java
74 //>java SelectTest2
75 =====
76 App3 )Select Operation
77 =====
78 -----SelectTest3.java-----
79 //SelectTest3.java
80 import java.sql.*;
81 import java.util.*;
82
83 public class SelectTest3
84 {
85     public static void main(String args[])throws Exception
86     {
87         //read inputs
88         Scanner sc=new Scanner(System.in);
89         System.out.println("Enter start range no:");
90         int stno=sc.nextInt(); //gives 100
91         System.out.println("Enter end range no:");
92         int endno=sc.nextInt(); //gives 200
93
94         //register type1 jdbc driver
95         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
96         //Establish the connection with Db s/w.
97         Connection con=DriverManager.getConnection("jdbc:odbc:oradsn","scott","tiger");
98         //create Jdbc Statement obj
99         Statement st=con.createStatement();
100
101        //prepare SQL Query
102        // select * from student where sno>=100 and sno<=300
103        String qry="select * from student where sno>="+stno+" and sno<="+endno;
104        System.out.println(qry);
105        //send and execute SQL Query in DB s/w
106        ResultSet rs=st.executeQuery(qry);
107        //Process the ResultSet
108        boolean flag=false;
109        while(rs.next()){
110            flag=true;
111            System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
112        }
113        if(flag==false)
114            System.out.println("Records not found");
115        //close jdbc objs
116        rs.close();
117        st.close();
118        con.close();
119    }//main
120 }//class
121 =====
122 App4) Select Operation
123 =====
124 -----SelectTest4.java-----
125 //SelectTest4.java
126 import java.sql.*;
127 import java.util.*;
128
129 public class SelectTest4
130 {
131     public static void main(String args[])throws Exception
132     {
```

```

133          //read inputs
134          Scanner sc=new Scanner(System.in);
135          System.out.println("Enter initchars of Emp name");
136          String initchars=sc.next(); //gives AD
137          // convert input value as required for the SQL Query
138          initchars="""+initchars+"%"; //gives 'AD%'
139
140          //register type1 jdbc driver
141          Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
142          //Establish the connection with Db s/w.
143          Connection con=DriverManager.getConnection("jdbc:odbc:oradsn","scott","tiger");
144          //create Jdbc Statement obj
145          Statement st=con.createStatement();
146
147          //pearce SQL Query
148          //select empno,ename,job,sal from emp where ename like 'AD%'
149          String qry="select empno,ename,job,sal from emp where ename like "+initchars;
150          System.out.println(qry);
151          //send and execute SQL Query in DB s/w
152          ResultSet rs=st.executeQuery(qry);
153          //Process the ResultSet
154          boolean flag=false;
155          while(rs.next()){
156              flag=true;
157              System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3)+" "+rs.get
158          }//while
159          if(flag==false)
160              System.out.println("records not found");
161
162          //close jdbc objs
163          rs.close();
164          st.close();
165          con.close();
166      }//main
167  }//class
168 =====
169 App5) Select Operation
170 =====
171 -----SelectTest5.java-----
172 //SelectTest5.java (with java coding standards)
173 /* App to get emp details based on the given designations
174 * Version : 1.0
175 * author: Team-Nt
176 * Date : 2015-7-30 */
177 package com.nt.jdbc;
178 import java.sql.DriverManager;
179 import java.sql.Connection;
180 import java.sql.Statement;
181 import java.sql.ResultSet;
182 import java.sql.SQLException;
183 import java.util.Scanner;
184
185 public class SelectTest5
186 {
187     public static void main(String args[]){
188         Connection con=null;
189         Statement st=null;
190         ResultSet rs=null;
191         Scanner sc=null;
192         try{
193             //read inputs
194             sc=new Scanner(System.in);
195             String desg1=null,desg2=null,desg3=null;
196             if(sc!=null){
197                 System.out.println("Enter Desg1");
198                 desg1=sc.nextLine().toUpperCase(); //gives CLERK

```

```
199     System.out.println("Enter Desg2");
200     desg2=sc.next().toUpperCase(); //gives MANAGER
201     System.out.println("Enter Desg3");
202     desg3=sc.next().toUpperCase(); //gives SALESMAN
203 } //if
204 //prepare Condition as required for SQL Query
205 // ('CLERK','MANAGER','SALESMAN')
206 String cond="('"+desg1+"','"+desg2+"','"+desg3+"')";
207
208 //register jdbc driver
209 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver1");
210 //Establish the connection
211 con=DriverManager.getConnection("jdbc:odbc:oradsn","scott","tiger");
212 //create Statement obj
213 if(con!=null)
214     st=con.createStatement();
215 //prepare SQL Query
216 //select empno,ename,job,sal from emp where job in('CLERK','MANAGER','
217 String qry="select empno,ename,job,sal from emp where job in"+cond+" orde
218 System.out.println(qry);
219 //send and execute SQL Query in DB s/w
220 if(st!=null)
221     rs=st.executeQuery(qry);
222 //process the ResultSet
223 boolean flag=false;
224 if(rs!=null){
225     while(rs.next()){
226         System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getStr
227         flag=true;
228     }//while
229 } //if
230
231 if(flag==false)
232     System.out.println("Records not found");
233
234 } //try
235 catch(ClassNotFoundException cnf){ //for known exception
236     System.out.println(cnf.toString());
237 }
238 catch(SQLException se){ //for known exception
239     se.printStackTrace();
240 }
241 catch(Exception e){ //for unknown exception
242     e.printStackTrace();
243 }
244 finally{
245     //close jdbc objs
246     try{
247         if(rs!=null){
248             rs.close();
249         }
250     }
251     catch(SQLException se){
252         se.printStackTrace();
253     }
254
255     try{
256         if(st!=null){
257             st.close();
258         }
259     }
260     catch(SQLException se){
261         se.printStackTrace();
262     }
263
264     try{
```

```

265         if(con!=null){
266             con.close();
267         }
268     } catch(SQLException se){
269         se.printStackTrace();
270     }
271     try{
272         if(sc!=null){
273             sc.close();
274         }
275     } catch(Exception e){
276         e.printStackTrace();
277     }
278 } //finally
279
280 } //main
281 } //class
282 //javac -d . SelectTest4.java
283 //java com.nt.jdbc.SelectTest4
284 =====
285 App6 ) Select Operation
286 =====
287 -----SelectTest6.java-----
288 //SelectTest6.java
289 package com.nt;
290
291 import java.sql.Connection;
292 import java.sql.DriverManager;
293 import java.sql.Statement;
294 import java.sql.ResultSet;
295 import java.sql.SQLException;
296 import java.util.Date;
297 /* Application that gives employee details who is having highest salary
298 * Version: 1.0
299 * Author :Team -s */
300
301
302
303
304 public class SelectTest6
305 {
306     public static void main(String args[]){
307         Connection con=null;
308         Statement st=null;
309         ResultSet rs=null;
310         try{
311             //register jdbc driver with DriverManager Service
312             Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
313             //Establish the connection
314             con=DriverManager.getConnection("jdbc:odbc:oradsn","scott","tiger");
315             // create Statement obj
316             if(con!=null)
317                 st=con.createStatement();
318             //send and execute SQL Query in DB s/w
319             if(st!=null)
320                 rs=st.executeQuery("select empno,ename,sal,job from emp where sal=(select m.");
321             //process the ResultSet obj
322             int cnt=0;
323             if(rs!=null){
324                 while(rs.next()){
325                     System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+
326                     cnt=1;
327                 }//while
328             }//if
329             if(cnt==0)
330

```

```
331             System.out.println("No Records Found");
332
333         } //try
334         catch(ClassNotFoundException cnf){ // To handle known Exception
335             cnf.printStackTrace();
336         }
337         catch(SQLException se) { // To handle known Exception
338             se.printStackTrace();
339         }
340         catch(Exception e){ // To handle unknown Exceptions
341             e.printStackTrace();
342         }
343     finally{
344         try{
345             if(rs!=null)
346                 rs.close();
347         }
348         catch(Exception e){
349             e.printStackTrace();
350         }
351
352         try{
353             if(st!=null)
354                 st.close();
355         }
356         catch(Exception e){
357             e.printStackTrace();
358         }
359
360         try{
361             if(con!=null)
362                 con.close();
363         }
364         catch(Exception e){
365             e.printStackTrace();
366         }
367     } //finally
368 } //main
369 } //class
370 //>javac -d . SelectTest5.java          (Compilation)
371 //>java com.nt.SelectTest5              (Execution)
372 =====
373 App7) Select Operation
374 =====
375 -----SelectTest7.java-----
376 //SelectTest7.java (Application that gives Emp Details based on given deptno)
377 package com.nt;
378
379 import java.sql.Connection;
380 import java.sql.DriverManager;
381 import java.sql.ResultSet;
382 import java.sql.SQLException;
383 import java.sql.Statement;
384 import java.util.Scanner;
385
386 public class SelectTest7{
387     public static void main(String[] args){
388         Scanner sc=null;
389         Statement st=null;
390         Connection con=null;
391         ResultSet rs=null;
392         try{
393             //read inputs
394             sc=new Scanner(System.in);
395             System.out.println("Enter dept no");
396             int deptno=0;
```

```
397         if(sc!=null){
398             deptno=sc.nextInt();
399         }
400         // register jdbc driver with DriverManager service
401         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
402         //Establish the connection
403         con=DriverManager.getConnection("jdbc:odbc:oradsn","scott","tiger");
404         //Create Jdbc Statement obj
405         if(con!=null)
406             st=con.createStatement();
407         //send and execute SQL Query in DB s/w
408         if(st!=null)
409             rs=st.executeQuery("select empno,ename,job,sal from emp where deptno");
410
411         if(rs!=null){
412             boolean flag=false;
413             while(rs.next()){
414                 flag=true;
415                 System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3)+"");
416             }
417
418             if(flag==false)
419                 System.out.println("Records not found");
420
421         } //if
422     } //try
423     catch(ClassNotFoundException cnf){ //To handle known exception
424         cnf.printStackTrace();
425     }
426     catch(SQLException se){ // To handle known exception
427         se.printStackTrace();
428     }
429     catch(Exception e){ // To handle unknown exception
430         e.printStackTrace();
431     }
432     finally{
433         //Close jdbc objs
434         try{
435             if(rs!=null)
436                 rs.close();
437         } //try
438         catch(Exception e){
439             e.printStackTrace();
440         }
441
442         try{
443             if(st!=null)
444                 st.close();
445         } //try
446         catch(Exception e){
447             e.printStackTrace();
448         }
449
450         try{
451             if(con!=null)
452                 con.close();
453         }
454         catch(Exception e){
455             e.printStackTrace();
456         }
457     } //finally
458 } //main
459 //javac -d . SelectTest7.java
460 //java com.nt.SelectTest7
461
462
```

```

463 =====
464 App8(Application to call SQL Aggragate Function)
465 =====
466 -----+-----AggregateTest.java-----+
467 //AggregateTest.java
468 import java.sql.*;
469 import java.util.*;
470 import java.io.*;
471 public class AggragateTest
472 {
473     public static void main(String args[])throws Exception
474     {
475         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
476         Connection con=DriverManager.getConnection("jdbc:odbc:oradsn","scott","tiger");
477         // create Staement obj
478         Statement st=con.createStatement();
479         // execute the query
480         ResultSet rs=st.executeQuery("select count(*) from emp");
481         if(rs.next())
482         {
483             System.out.println("The Number of Records are :" +rs.getInt(1));
484         }
485         //close all jdbc stream objects
486         rs.close();
487         st.close();
488         con.close();
489     }//main
490 } //class
491 =====
492 App9(Simple Delete operation taking sno value as criteria)
493 =====
494 -----DeleteTest.java-----+
495 //DeleteTest.java
496 import java.sql.*;
497 import java.util.*;
498 public class DeleteTest
499 {
500     public static void main(String args[])throws Exception
501     {
502         // read input values from key board
503         Scanner sc=new Scanner(System.in);
504         System.out.println("Enter student no to delete record");
505         int no=sc.nextInt();
506
507         // create jdbc connection
508         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
509         Connection con=DriverManager.getConnection("jdbc:odbc:oradsn","scott","tiger");
510
511         //disable autocommit mode on DB s/w
512         con.setAutoCommit(false);
513
514         //create jdbc Statement obj
515         Statement st=con.createStatement();
516
517         //prepare query
518         String qry="delete from student where sno="+no;
519         System.out.println(qry);
520
521         // execute the query.....
522         int result=st.executeUpdate(qry);
523
524         //commit or rollback
525         con.commit(); // (or)    con.rollback();
526
527         // process the result
528         if(result==0)

```

```
529         System.out.println("record not found to delete");
530     else
531         System.out.println(result+" no.of Record(s) found and deleted");
532
533     //close jdbc objects
534     st.close();
535     con.close();
536 } //main
537 } //class
538 =====
539 App10(Simple Insert operation)
540 =====
541 -----InsertTest.java-----
542 package com.nt;
543 //InsertTest.java (having Coding standards)
544 /* Application to insert record into DB table
545 * version :1.0
546 * author: Team-s */
547 import java.util.Scanner;
548 import java.sql.DriverManager;
549 import java.sql.Connection;
550 import java.sql.Statement;
551 import java.sql.SQLException;
552
553 public class InsertTest
554 {
555     public static void main(String args[])
556     {
557         Connection con=null;
558         Statement st=null;
559         Scanner sc=null;
560         try
561         {
562             // read record related input values from keyboard
563             int no=0;
564             String name=null,addr=null;
565             sc=new Scanner(System.in);
566             if(sc!=null)
567             {
568                 System.out.println("Enter student no");
569                 no=sc.nextInt();
570                 System.out.println("Enter student name");
571                 name=sc.next();
572                 System.out.println("Enter student address");
573                 addr=sc.next();
574             }
575             //prepare variable values as required for the SQL(insert) query
576             name+"'"+name+"'";
577             addr+"'"+addr+"'";
578
579             // create jdbc con with DB s/w
580             Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
581             con=DriverManager.getConnection("jdbc:odbc:oradsn","scott","tiger");
582
583             // create jdbc statement obj
584             if(con!=null)
585                 st=con.createStatement();
586
587             //prepare example query with direct values
588             //String qry="insert into student values(101,'raja','hyd')";
589
590             //frame the insert query with variables
591             String qry="insert into student values("+no+","+name+","+addr+ ")";
592             System.out.println(qry);
593
594             // send and execute the query in DB s/w
```

```
595     int res=0;
596     if(st!=null)
597         res=st.executeUpdate(qry);
598
599         //process the result
600         if(res==0)
601             System.out.println("Record insertion Failed");
602         else
603             System.out.println("Record inserted");
604     }//try
605     catch(ClassNotFoundException cnf) // to handle known exception
606     {
607         cnf.printStackTrace();
608     }
609     catch(SQLException se) // to handle known exception
610     {
611         se.printStackTrace();
612     }
613     catch(Exception e) // to unknown exceptions
614     {
615         e.printStackTrace();
616     }
617     finally
618     {
619         //close jdbc objects.....
620         try
621         {
622             if(st!=null)
623                 st.close();
624         }
625         catch(SQLException se)
626         {
627             se.printStackTrace();
628         }
629
630         try
631         {
632             if(con!=null)
633                 con.close();
634         }
635         catch(SQLException se1)
636         {
637             se1.printStackTrace();
638         }
639     }//finally
640     }//main
641 }//class
642
643 //>javac -d . InsertTest.java
644 //>java com.nt.InsertTest
645
646 =====
647 App11(Program to update the record based on the student number)
648 =====
649 -----UpdateTest.java-----
650 package com.nt;
651 //UpdateTest.java (with coding standards)
652 /* App to update student details
653 * version : 1.0
654 * author : Team-s*/
655
656 import java.util.Scanner;
657 import java.sql.DriverManager;
658 import java.sql.Connection;
659 import java.sql.Statement;
660 import java.sql.SQLException;
```

```
661 public class UpdateTest
662 {
663     public static void main(String args[])
664     {
665         Connection con=null;
666         Statement st=null;
667         Scanner sc=null;
668         try
669         {
670             // read input values from keyboard
671             sc=new Scanner(System.in);
672             int no=0;
673             String name=null;
674             String addrs=null;
675             if (sc!=null)
676             {
677                 System.out.println("Enter existing student number");
678                 no=sc.nextInt();
679                 System.out.println("Enter new name:");
680                 name=sc.next();
681                 System.out.println("Enter new address:");
682                 addrs=sc.next();
683             }
684         }//if
685
686         // register jdbc driver and establish the connection
687         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
688         con=DriverManager.getConnection("jdbc:odbc:oradsn","scott","tiger");
689         //create Statement obj
690         if(con!=null)
691             st=con.createStatement();
692
693         // frame example qry with direct values
694         //String qry="update student set sname='maharaja',sadd='vizag' where sno=101
695         String qry="update student set sname='"+name+"',sadd='"+addrs+"' where sno="+
696         System.out.println(qry);
697
698         //send and execute the qry in db s/w
699         int res=0;
700         if(st!=null)
701             res=st.executeUpdate(qry);
702
703         // process results
704         if (res==0)
705         {
706             System.out.println("record not found");
707         }
708         else
709         {
710             System.out.println(res+"no.of records are updated");
711         }
712     }//try
713     catch(ClassNotFoundException cnf)//handles known exception
714     {
715         cnf.printStackTrace();
716     }
717     catch(SQLException se) //handles known exception
718     {
719         se.printStackTrace();
720     }
721     catch(Exception e) //handles unknown exceptions
722     {
723         e.printStackTrace();
724     }
725     finally
726     {
```

```
727 //close jdbc objs
728     try
729     {
730         if(st!=null)
731             st.close();
732     }
733     catch(SQLException se)
734     {
735         se.printStackTrace();
736     }
737
738     try
739     {
740         if(con!=null)
741             con.close();
742     }
743     catch(SQLException se)
744     {
745         se.printStackTrace();
746     }
747 } //finally
748 } //main
749 } //class
750 //>javac -d . UpdateTest.java
751 //>java com.nt.UpdateTest
752 =====
753 App12 (App to execute both select and non-select SQL Queries)
754 =====
755 -----SelectNonSelectTest.java-----
756 //SelectNonSelectTest.java (Example on execute(-) method)
757 import java.sql.*;
758 import java.util.*;
759 public class SelectNonSelectTest
760 {
761     public static void main(String args[])throws Exception
762     {
763         // read sql query (any) from keyboard
764         Scanner sc=new Scanner(System.in);
765         System.out.println("Enter the qry");
766         String qry=sc.nextLine();
767
768         // create jdbc con
769         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
770         Connection con=DriverManager.getConnection("jdbc:odbc:oradsn","scott","tiger");
771         // create jdbc statement obj
772         Statement st=con.createStatement();
773
774         //send and execute qry in db s/w
775         boolean flag=st.execute(qry);
776         // gather and process the result
777         if(flag==true) // when select sql qry is executed
778         {
779             ResultSet rs=st.getResultSet();
780             while(rs.next())
781             {
782                 System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getStri
783             }
784             rs.close();
785         }//if
786         else // when non-select qry is executed
787         {
788             int rowcount=st.getUpdateCount();
789             System.out.println("the no.of records that effected"+rowcount);
790         }
791
792     //close jdbc objs
```

```
793             st.close();
794             con.close();
795         }//main
796     }//class
797 =====
798 App13 ) CreateTable
799 =====
800 -----CreateTableTest.java-----
801 package com.nt.jdbc;
802
803 import java.sql.Connection;
804 import java.sql.DriverManager;
805 import java.sql.Statement;
806
807 public class CreateTableTest {
808
809     public static void main(String[] args) throws Exception {
810
811         //register jdbc driver
812         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
813         //Establish the connection
814         Connection con=DriverManager.getConnection("jdbc:odbc:oradsn","scott","tiger");
815         //create Statement obj
816         Statement st=con.createStatement();
817         //send and execute SQL Query in DB s/w
818         int result=st.executeUpdate("create table temp(col1 number(5))");
819         //process the result
820         if(result==0)
821             System.out.println("Table not created");
822         else
823             System.out.println("Table created");
824         //close jdbc objs
825         st.close();
826         con.close();
827     }//main
828 }//class
829 =====
830 App14) Drop table Test
831 =====
832 -----DropTableTest.java-----
833 package com.nt.jdbc;
834
835 import java.sql.Connection;
836 import java.sql.DriverManager;
837 import java.sql.Statement;
838 import java.util.Scanner;
839
840 public class DropTableTest {
841
842     public static void main(String[] args) throws Exception {
843         //read inputs
844         Scanner sc=new Scanner(System.in);
845         System.out.println("Enter db talbe");
846         String tab=sc.next();
847
848         //register jdbc driver
849         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
850         //Establish the connection
851         Connection con=DriverManager.getConnection("jdbc:odbc:oradsn","scott","tiger");
852         //create Statement obj
853         Statement st=con.createStatement();
854         //send execute SQL Query in DB s/w
855         int result=st.executeUpdate("drop table "+tab);
856         //process the result
857         if(result==0)
858             System.out.println("Table not found");
```

```
859         else
860             System.out.println("Table dropped");
861
862             //close jdbc objs
863             st.close();
864             con.close();
865     }//main
866 } //class
867
868 =====
869 App15(Program to insert multiple records using PreparedStatement)
870 =====
871 -----PstInsertTest.java-----
872 //PstInsertTest.java
873 import java.sql.DriverManager;
874 import java.sql.Connection;
875 import java.sql.PreparedStatement;
876 import java.sql.SQLException;
877 import java.util.Scanner;
878
879 public class PstInsertTest
880 {
881     public static void main(String args[])
882     {
883         Connection con=null;
884         Scanner sc=null;
885         PreparedStatement ps=null;
886         try
887         {
888             //read input value from key board
889             sc=new Scanner(System.in);
890             int n=0;
891             if(sc!=null)
892             {
893                 System.out.println("Enter no.of students");
894                 n=sc.nextInt();
895             }
896             //create jdbc con object
897             Class.forName("oracle.jdbc.driver.OracleDriver");
898             con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","scott
899
900             // create PreparedStatement obj
901             if(con!=null)
902                 ps=con.prepareStatement("insert into student values(?, ?, ?)");
903
904             // read each student details from keyboard, set them to query
905             // and execute the query
906             if(ps!=null && sc!=null)
907             {
908                 for(int i=1;i<=n;++i)
909                 {
910                     System.out.println("Enter "+i+" student details");
911                     System.out.println("Enter student no");
912                     int no=sc.nextInt();
913
914                     System.out.println("Enter student name");
915                     String name=sc.next();
916
917                     System.out.println("Enter student address");
918                     String addrs=sc.next();
919                     //set these values query place holders
920                     ps.setInt(1,no);
921                     ps.setString(2,name);
922                     ps.setString(3,addrs);
923                     //execute the query
924                     int res=ps.executeUpdate();
```

```

925                     if(res==0)
926                         System.out.println(i+"student details are not inserted");
927                     else
928                         System.out.println(i+"student details are inserted");
929                 }//for
930             }//if
931         }//try
932         catch(ClassNotFoundException cnf) // to handle known exceptions
933         {
934             cnf.printStackTrace();
935         }
936         catch(SQLException se) // to handle known exceptions
937         {
938             se.printStackTrace();
939         }
940         catch(Exception e) // to hanlde unknown exceptions
941         {
942             e.printStackTrace();
943         }
944     finally
945     {
946         //close jdbc stream objects
947         try
948         {
949             if(ps!=null)
950                 ps.close();
951         }
952         catch(SQLException se)
953         {
954             se.printStackTrace();
955         }
956     }
957     try
958     {
959         if(con!=null)
960             con.close();
961     }
962     catch(SQLException se1)
963     {
964         se1.printStackTrace();
965     }
966 } //finally
967 } //main
968 } //class
969 =====
970 App16) (Showing SQL Injection Problem)
971 =====
972 -----LoginApp.java-----
973 /* DB table in oracle
974     userlist
975
976 uname vc2          pwd vc2
977 -----
978     raja           hyd
979     ramesh         hyd
980 */
981
982 import java.sql.*;
983 import java.util.*;
984
985 public class LoginApp
986 {
987     public static void main(String args[])throws Exception
988     {
989         //read inputs
990         Scanner sc=new Scanner(System.in);

```

```

991         System.out.println("Enter username:");
992         String user=sc.nextLine(); //gives raja
993         System.out.println("Enter Password:");
994         String pass=sc.nextLine(); //gives rao
995     // there is no need of converting input values..
996
997     // register jdbc driver and establish the connection
998     Class.forName("oracle.jdbc.driver.OracleDriver");
999     Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","sc
1000
1001    //prepare SQL Query
1002    String qry="select count(*) from userlist where uname='"+user+"' and pwd='"+pass+"'
1003    //create Jdbc Statement obj
1004    Statement st=con.createStatement();
1005    //send and execute the Query in DB s/w
1006    ResultSet rs=st.executeQuery(qry);
1007    //process the ResultSet
1008    int cnt=0;
1009    if(rs.next())
1010    {
1011        cnt=rs.getInt(1);
1012    }
1013
1014    if(cnt==0)
1015        System.out.println("InValid Credentials");
1016    else
1017        System.out.println("Valid Credentials");
1018
1019    //close jdbc objs
1020    rs.close();
1021    st.close();
1022    con.close();
1023 } //main
1024 } //class
1025 //>javac LoginApp.java
1026 //>java LoginApp
1027 enter user name:raja' --
1028 enter password : hyd1 (wrong password)
1029 output: Valid Credentials ( SQL Injection problem)
1030 =====
1031 App17 (Solving SQL Injection Problem)
1032 =====
1033 -----LoginApp1.java-----
1034 /* DB table in oracle
1035     userlist
1036
1037     uname vc2          pwd vc2
1038
1039     raja              hyd
1040     ramesh            hyd
1041 */
1042 import java.sql.*;
1043 import java.util.*;
1044
1045 public class LoginApp1
1046 {
1047     public static void main(String args[])throws Exception
1048     {
1049         //read inputs
1050         Scanner sc=new Scanner(System.in);
1051         System.out.println("Enter username:");
1052         String user=sc.nextLine(); //gives raja
1053         System.out.println("Enter Password:");
1054         String pass=sc.nextLine(); //gives rao
1055     // there is no need of converting input values..
1056

```

```
1057     // register jdbc driver and establish the connection
1058     Class.forName("oracle.jdbc.driver.OracleDriver");
1059     Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","sc
1060
1061     //prepare SQL Query
1062     String qry="select count(*) from userlist where uname=? and pwd=?";
1063     //create Jdbc PreparedStatement obj
1064     PreparedStatement ps=con.prepareStatement(qry);
1065     // set values to Query params
1066     ps.setString(1,user);
1067     ps.setString(2,pass);
1068     //send and execute the Query in DB s/w
1069     ResultSet rs=ps.executeQuery();
1070
1071     //process the ResultSet
1072     int cnt=0;
1073     if(rs.next())
1074     {
1075         cnt=rs.getInt(1);
1076     }
1077
1078     if(cnt==0)
1079         System.out.println("InValid Credentials");
1080     else
1081         System.out.println("Valid Credentials");
1082
1083     //close jdbc objs
1084     rs.close();
1085     ps.close();
1086     con.close();
1087 } //main
1088 } //class
1089 //javac LoginApp1.java
1090 //java LoginApp1
1091 enter user name:raja' --
1092 enter password : hyd1 (wrong password)
1093 output: InValid Credentials ( No SQL Injection problem)
1094 =====
1095 App18 (prg on Inserting Dates and Retreving Dates)
1096 =====
1097 -----DateInsert.java-----
1098 //DateInsert.java
1099 import java.util.*;
1100 import java.sql.*;
1101
1102 public class DateInsert
1103 {
1104     public static void main(String args[])throws Exception
1105     {
1106         // read input values from keyboard
1107         Scanner sc=new Scanner(System.in);
1108         System.out.println("Enter person no");
1109         int no=sc.nextInt();
1110
1111         System.out.println("Enter person name:");
1112         String name=sc.next();
1113
1114         System.out.println("Enter DOB(dd-MM-yy)");
1115         String sdob=sc.next();
1116
1117         System.out.println("Enter DOJ(yyyy-MM-dd)");
1118         String sdoj=sc.next();
1119
1120         //convert String Date values to java.sql.Date class objs
1121         // for DOB
1122         SimpleDateFormat sdf1=new SimpleDateFormat("dd-MM-yy");
```

```
1123         java.util.Date udob=sdf1.parse(sdob);
1124
1125         java.sql.Date sqdob=new java.sql.Date(udob.getTime());
1126
1127         //for DOJ (yyyy-MM-dd)
1128         java.sql.Date sqdoj=java.sql.Date.valueOf(sdoj);
1129
1130         //create jdbc con object
1131         Class.forName("oracle.jdbc.driver.OracleDriver");
1132         Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","");
1133
1134         //create PreparedStatement obj pointing to insert query
1135         PreparedStatement ps=con.prepareStatement("insert into person_tab values(?, ?, ?, ?)");
1136
1137         //set values to the parameters of query
1138         ps.setInt(1,no);
1139         ps.setString(2,name);
1140         ps.setDate(3,sqdob);
1141         ps.setDate(4,sqdoj);
1142
1143         //execute the query
1144         int res=ps.executeUpdate();
1145
1146         //process the result
1147         if(res==0)
1148             System.out.println("record not inserted");
1149         else
1150             System.out.println("record inserted");
1151
1152         //close jdbc objs
1153         ps.close();
1154         con.close();
1155     }//main
1156 }//class
1157 //>javac DateInsert.java
1158 //>java DateInsert
1159 -----DateRetrieve.java-----
1160 //DateRetrieve.java
1161 import java.sql.*;
1162 import java.util.*;
1163 import java.text.*;
1164
1165 public class DateRetrieve
1166 {
1167     public static void main(String args[])throws Exception
1168     {
1169         // create jdbc con obj
1170         Class.forName("oracle.jdbc.driver.OracleDriver");
1171         Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","");
1172
1173         //create jdbc Statement object
1174         Statement st=con.createStatement();
1175
1176         // execute the query
1177         ResultSet rs=st.executeQuery("select * from person_tab");
1178
1179         //process the ResultSet
1180         while(rs.next())
1181         {
1182             int no=rs.getInt(1);
1183             String name=rs.getString(2);
1184             java.sql.Date sqdob=rs.getDate(3);
1185             java.sql.Date sqdoj=rs.getDate(4);
1186
1187             //convert java.sql.Date class objs to java.util.Date class objs
1188             java.util.Date udob=(java.util.Date)sqdob;
```

```

1189         java.util.Date udoj=(java.util.Date)sqdoj;
1190         //convert java.util.Date class objs to String date values
1191         SimpleDateFormat sdf1=new SimpleDateFormat("MMM-yy-dd");
1192         String sdob=sdf1.format(udob);
1193
1194         SimpleDateFormat sdf2=new SimpleDateFormat("yyyy-dd-MMM");
1195         String sdoj=sdf2.format(udoj);
1196
1197         System.out.println(no+" "+name+" "+sdob+" "+sdoj);
1198     }//while
1199
1200     //close jdbc objs
1201     rs.close();
1202     st.close();
1203     con.close();
1204
1205 } //main
1206 } //class
1207 =====
1208 App19 (working with Large objs (BLOB))
1209 =====
1210 -----PhotoInsert.java-----
1211 /* create table empall(eno number(4),ename varchar2(20),esalary number(7,2), ephoto blob);*/
1212 import java.sql.*;
1213 import java.io.*;
1214 class PhotoInsert
1215 {
1216     public static void main(String args[])throws Exception
1217     {
1218         //register jdbc driver and establish the connection
1219         Class.forName("oracle.jdbc.driver.OracleDriver");
1220         Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","scott","tig");
1221         //create PreparedStatement obj
1222         PreparedStatement ps=con.prepareStatement("insert into empall values(?, ?, ?, ?)");
1223         ps.setInt(1,234);
1224         ps.setString(2,"raja");
1225         ps.setFloat(3,5000);
1226
1227         File f=new File("tips.gif");
1228         FileInputStream fis=new FileInputStream(f);
1229         ps.setBinaryStream(4,fis,(int)f.length());
1230
1231         //execute the query
1232         ps.executeUpdate();
1233         System.out.println("Photo Inserted");
1234         //close the jdbc obj
1235         fis.close();
1236         ps.close();
1237         con.close();
1238     }//main
1239 } //class
1240 -----PhotoRetrieve.java-----
1241 import java.sql.*;
1242 import java.io.*;
1243 public class PhotoRetrieve
1244 {
1245     public static void main(String args[])throws Exception
1246     {
1247         //register jdbc driver and estblish the connection
1248         Class.forName("oracle.jdbc.driver.OracleDriver");
1249         Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","scott",
1250
1251         //create JDBC Statement obj
1252         Statement st=con.createStatement();
1253         // execute the SQL Query
1254         ResultSet rs=st.executeQuery("select * from empall");

```

```

1255      //process the ResultSet obj
1256      if(rs.next())
1257      {
1258          //get InputStream
1259          InputStream in=rs.getBinaryStream("ephoto");
1260          //create Destination file
1261          FileOutputStream fos=new FileOutputStream("newpict.gif");
1262
1263          //Buffering based logic to complete photo retriving
1264          int bytesRead=0;
1265          byte [] buffer=new byte[4096];
1266
1267          while((bytesRead=in.read(buffer))!=-1)
1268          {
1269              fos.write(buffer,0,bytesRead);
1270          }
1271
1272
1273          System.out.println("photo is stored in newpict.gif");
1274          //close jdbc objs
1275          fos.close();
1276          in.close();
1277          rs.close();
1278          st.close();
1279          con.close();
1280      }//if
1281  }//main
1282 }//class
1283 =====
1284 App20) Working with Large objs (CLOB)
1285 =====
1286 -----CLOBInsert.java-----
1287 package com.nt.jdbc;
1288
1289 import java.io.File;
1290 import java.io.FileReader;
1291 import java.io.Reader;
1292 import java.sql.Connection;
1293 import java.sql.DriverManager;
1294 import java.sql.PreparedStatement;
1295 import java.util.Scanner;
1296
1297 public class CLOBInsert {
1298     public static void main(String[] args) throws Exception {
1299         //read inputs
1300         Scanner sc=new Scanner(System.in);
1301         System.out.println("Enter sno");
1302         int no=sc.nextInt();
1303         System.out.println("Enter sname");
1304         String sname=sc.next();
1305         System.out.println("Enter address");
1306         String sadd=sc.next();
1307         System.out.println("Enter resume path");
1308         String resumePath=sc.next();
1309
1310         //register jdbc driver
1311         Class.forName("oracle.jdbc.driver.OracleDriver");
1312         //Establish the connection
1313         Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "scott"
1314         //create PreparedStatement obj
1315         PreparedStatement ps=con.prepareStatement("insert into studentAll values(?, ?, ?, ?)");
1316         // Create java.io.File obj
1317         File file=new File(resumePath);
1318         long length=file.length();
1319         //Create Reader obj pointing resume
1320         Reader reader=new FileReader(file);

```

```
1321 //set param values(?) to Query
1322 ps.setInt(1,no);
1323 ps.setString(2,sname);
1324 ps.setString(3,sadd);
1325 ps.setCharacterStream(4,reader,(int)length);
1326 //ps.setClob(4,reader,length);
1327 //execute the Query
1328 int result=ps.executeUpdate();
1329 // process the Result
1330 if(result==0)
1331     System.out.println("Record not inserted");
1332 else
1333     System.out.println("Record inserted");
1334
1335 //close jdbc objs
1336 reader.close();
1337 ps.close();
1338 con.close();
1339 }
1340
1341 }
-----CLOBRetrieve.java-----
1343 package com.nt.jdbc;
1344
1345 import java.io.FileWriter;
1346 import java.io.Reader;
1347 import java.io.Writer;
1348 import java.sql.Connection;
1349 import java.sql.DriverManager;
1350 import java.sql.PreparedStatement;
1351 import java.sql.ResultSet;
1352 import java.util.Scanner;
1353
1354 public class CLOBRetrieve {
1355     public static void main(String[] args) throws Exception {
1356         //read inputs
1357         Scanner sc=new Scanner(System.in);
1358         System.out.println("Enter student no");
1359         int no=sc.nextInt();
1360         //Establish the connection
1361         Class.forName("oracle.jdbc.driver.OracleDriver");
1362         Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "s
1363         //create PreparedStatement obj
1364         PreparedStatement ps=con.prepareStatement("select * from StudentAll where sno=?");
1365         //set Query param values
1366         ps.setInt(1,no);
1367         //execute the Query
1368         ResultSet rs=ps.executeQuery();
1369         //process the ResultSet
1370         Reader reader=null;
1371         if(rs.next()){
1372             reader=rs.getCharacterStream(4);
1373         }
1374         //create Writer
1375         Writer writer=new FileWriter("myResume.txt");
1376         //use Buffer based logic to write CLOB value to Dest file
1377         char buffer[]=new char[2048];
1378         int charsRead=0;
1379         while((charsRead=reader.read(buffer))!=-1){
1380             writer.write(buffer,0,charsRead);
1381         }
1382
1383         System.out.println("File has been Retrieved");
1384
1385         //close jdbc objs
1386         reader.close();
```

```
1387         writer.close();
1388         rs.close();
1389         ps.close();
1390         con.close();
1391     } //main
1392 } //class
1393
1394 -----CLOBRetrieve1.java-----
1395 package com.nt.jdbc;
1396
1397 import java.io.FileOutputStream;
1398 import java.io.InputStream;
1399 import java.io.OutputStream;
1400 import java.sql.Clob;
1401 import java.sql.Connection;
1402 import java.sql.DriverManager;
1403 import java.sql.PreparedStatement;
1404 import java.sql.ResultSet;
1405 import java.util.Scanner;
1406
1407 public class CLOBRetrieve1 {
1408     public static void main(String[] args) throws Exception {
1409         //read inputs
1410         Scanner sc=new Scanner(System.in);
1411         System.out.println("Enter student no");
1412         int no=sc.nextInt();
1413         //Establish the connection
1414         Class.forName("oracle.jdbc.driver.OracleDriver");
1415         Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "s
1416 //create PrpearedStaement obj
1417         PreparedStatement ps=con.prepareStatement("select * from StudentAll where sno=?");
1418         //set Query param values
1419         ps.setInt(1,no);
1420         //execute the Query
1421         ResultSet rs=ps.executeQuery();
1422         //process the ResultSet
1423         Clob clb=null;
1424         if(rs.next()){
1425             clb=rs.getBlob(4);
1426         }
1427         //get InputStream
1428         InputStream is=clb.getAsciiStream();
1429         //create Writer
1430         OutputStream os=new FileOutputStream("newResume.txt");
1431         //use Buffer logic
1432         byte buffer[]=new byte[4096];
1433         int bytesRead=0;
1434         while((bytesRead=(is.read(buffer)))!=-1){
1435             os.write(buffer,0,bytesRead);
1436         }
1437         System.out.println("file has been retrived");
1438         //close jdbc objs
1439         os.close();
1440         is.close();
1441         rs.close();
1442         ps.close();
1443         con.close();
1444     } //main
1445 } //class
1446
1447 =====
1448 App21(Program to call pl/sql procedure using CallableStatement)
1449 =====Procedure in oracle=====
1450 create or replace procedure first_pro1(x in number,y out number) as
1451 begin
1452     y:=x*x;
```

```

1453 end;
1454 -----
1455 import java.sql.*;
1456 class UsingProcedure
1457 {
1458     public static void main( String args[ ] ) throws Exception
1459     {
1460         //register jdbc driver
1461         Class.forName("oracle.jdbc.driver.OracleDriver");
1462         //establish the connection
1463         Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","scott",
1464             // create CallableStatement obj
1465             CallableStatement cst = con.prepareCall( "{CALL first_pro1(?,?)}" );
1466             //register out param with jdbc types
1467             cst.registerOutParameter( 2, Types.INTEGER );
1468             //set value to IN param
1469             cst.setInt( 1, 20 );
1470             // call pl/sql procedure
1471             cst.execute();
1472             // gather results from out params
1473             int res = cst.getInt(2);
1474
1475             System.out.println( " Result is: " + res );
1476             //close jdbc objs
1477             cs.close();
1478             con.close();
1479     }
1480 }
1481 =====
1482 App22 (App on CallableStatement obj)
1483 =====
1484 -----CsTest1.java-----
1485 /* create or replace procedure get_EmpDetails(no in number,name out varchar2, salary out number)as
1486 begin
1487     select ename,sal into name,salary from emp where empno=no;
1488 end;
1489 */
1490 //CsTest1.java
1491 import java.sql.*;
1492 import java.util.*;
1493 public class CsTest1
1494 {
1495     public static void main(String[] args) throws Exception
1496     {
1497         // read employee no from keyboard
1498         Scanner sc=new Scanner(System.in);
1499         System.out.println("Enter employee no:");
1500         int no=sc.nextInt();
1501
1502         // create jdbc con object
1503         Class.forName("oracle.jdbc.driver.OracleDriver");
1504         Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","scott");
1505         // prepare qry calling pl/sql procedure
1506         String qry="{ call get_EmpDetails(?,?,?)}";
1507         //create CallableStatement obj
1508         CallableStatement cs=con.prepareCall(qry);
1509         // register outparameters with jdbc types
1510         cs.registerOutParameter(2,Types.VARCHAR);
1511         cs.registerOutParameter(3,Types.INTEGER);
1512         // set input value to IN parameter
1513         cs.setInt(1,no);
1514         //execute pl/sql procedure
1515         cs.execute();
1516         //gather results from out parameters
1517         System.out.println(no+" employee salary is"+cs.getInt(3));
1518         System.out.println(no+" employee name is"+cs.getString(2));

```

```

1519                     //close jdbc objects
1520                     cs.close();
1521                     con.close();
1522
1523             }//main
1524
1525 }//class
1526
1527 //>javac CsTest1.java
1528 //>java CsTest1
1529
1530 =====
1531 App23 (Application on CallableStatement obj Using Curosor)
1532 =====
1533 -----CursorsTest.java-----
1534
1535 /*create or replace procedure fetch_AllEmpDetails(initchars in varchar,details out sys_refcursor)as
1536 begin
1537 open details for
1538     select * from emp where ename like initchars;
1539 end;
1540 */
1541 //CursorCsTest.java
1542
1543 import java.sql.*;
1544 import java.util.*;
1545 import oracle.jdbc.*; // for OracleTypes class ( this pkg is available in ojdbc14.jar file)
1546
1547 public class CursorCsTest
1548 {
1549     public static void main( String args[ ] ) throws Exception
1550     {
1551
1552         // read cond character from key board
1553         Scanner sc=new Scanner(System.in);
1554         System.out.println("Enter characters (first letters of emp name)");
1555         String cond=sc.next();
1556         cond=cond+"%";
1557
1558         //create jdbc con obj
1559         Class.forName("oracle.jdbc.driver.OracleDriver");
1560         Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","scott",
1561
1562             CallableStatement cs = con.prepareCall("{call fetch_AllEmpDetails(?,?)}");
1563
1564             // register out parameter with jdbc types (Oracle corp supplied jdbc types)
1565             cs.registerOutParameter(2,OracleTypes.CURSOR);
1566
1567             // set value to IN parameter
1568             cs.setString(1,cond);
1569
1570             // execute pl/sql procudre
1571             cs.execute();
1572
1573             // gather result from out parameter
1574             ResultSet rs=(ResultSet)cs.getObject(2);
1575
1576             //display result
1577             while(rs.next())
1578             {
1579                 System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
1580             }
1581
1582             // close jdbc objects
1583             rs.close();
1584             cs.close();

```

```
1585                     con.close();
1586 =====
1587 App 24) App calling pl/sql function
1588 =====
1589 -----CsFxTest1.java-----
1590 /*create or replace function Fx_Get_EmpDetails(no in number,name out varchar,salary out number)return
1591 as
1592     desg varchar2(20);
1593 begin
1594     select ename,sal,job into name,salary,desg from emp where empno=no;
1595
1596     return desg;
1597 end;
1598 */
1599 package com.nt.jdbc;
1600
1601 import java.sql.CallableStatement;
1602 import java.sql.Connection;
1603 import java.sql.DriverManager;
1604 import java.sql.Types;
1605 import java.util.Scanner;
1606
1607 public class CsFxTest1 {
1608     public static void main(String[] args)throws Exception {
1609         //read input value
1610         Scanner sc=new Scanner(System.in);
1611         System.out.println("Enter no:");
1612         int no=sc.nextInt();
1613         //register driver and establish the connection
1614         Class.forName("oracle.jdbc.driver.OracleDriver");
1615         Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","sc");
1616         //prepare Query
1617         String qry="{?= call Fx_Get_EmpDetails(?, ?, ?)}";
1618         //create CallableStatement obj
1619         CallableStatement cs=con.prepareCall(qry);
1620         //register return,OUT params with JDBC types
1621         cs.registerOutParameter(1,Types.VARCHAR);//return param
1622         cs.registerOutParameter(3,Types.VARCHAR);//out param
1623         cs.registerOutParameter(4,Types.INTEGER);// out param
1624         //set value to IN param
1625         cs.setInt(2,no);
1626         //call pl/sql function
1627         cs.execute();
1628         //Gather results from OUT,return parameters
1629         int salary=cs.getInt(4); //from OUT
1630         String desg=cs.getString(1); //from return
1631         String name=cs.getString(3); //from OUT
1632         System.out.println("Name:"+name+"desg="+desg+"Salary="+salary);
1633         //close jdbc objs
1634         cs.close();
1635         con.close();
1636     }//main
1637 } //class
1638 =====
1639 App25 (calls pl/sql function that contains cursor based select query and non-select query)
1640 =====
1641 -----CsFxTest1.java-----
1642 //CsFxTest2.java
1643 /* create or replace function Fx_Get_Student_FOR_DELETION (no in number, cnt out number)return sys_
1644 as
1645     details sys_refcursor;
1646 begin
1647     open details for
1648         select * from student where sno=no;
1649
1650
```

```
1651      delete from student where sno=no;
1652      cnt:=SQL%ROWCOUNT;
1653      return details;
1654  end;
1655  */
1656 import java.sql.*;
1657 import java.util.*;
1658 import oracle.jdbc.*;
1659
1660 public class CsFxTest1
1661 {
1662     public static void main(String args[])throws Exception
1663     {
1664         // read input values from keyboard
1665         Scanner sc=new Scanner(System.in);
1666         System.out.println("Enter student no");
1667         int no=sc.nextInt();
1668
1669         // create jdbc con object
1670         Class.forName("oracle.jdbc.driver.OracleDriver");
1671         Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","sc
1672
1673         //create jdbc CallableStatement obj
1674         CallableStatement cs=con.prepareCall("{?= call FX_GET_STUDENT_FOR_DELETION(?,?)}");
1675
1676         // register out,return params with jdbc types
1677         cs.registerOutParameter(1,OracleTypes.CURSOR); //return param
1678         cs.registerOutParameter(3,Types.INTEGER);//out param
1679
1680         //set values to IN parameters
1681         cs.setInt(2,no);
1682
1683         // call and execute pl/sql function
1684         cs.execute();
1685
1686
1687         // gather results from return,out params
1688         ResultSet rs=(ResultSet)cs.getObject(1); // from return param(cursor)
1689         //process return value result
1690         while(rs.next())
1691         {
1692             System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
1693         }
1694
1695         int cnt=cs.getInt(3); //out param result
1696         if(cnt==0)
1697             System.out.println("Record not deleted");
1698         else
1699             System.out.println("Record deleted");
1700         //close jdbc objs
1701         rs.close();
1702         cs.close();
1703         con.close();
1704     }//main
1705 } //class
1706 //javac CsFxTest1.java
1707 //java CsFxTest1
1708 =====
1709 =====
1710 App26(Using Swings+JDBC)(program using all three Statement Objects)
1711 =====Procedure in oracle=====
1712 create or replace procedure FIND_PASS_FAIL(m1 in number,m2 in number,m3 in number,res out varchar) a
1713 begin
1714     if(m1<35 or m2<35 or m3<35)then
1715         res:='fail';
1716     else
```

```
1717     res:='pass';
1718     end if;
1719   end;
1720 -----AllStmtsTest.java-----
1721 package com.nt.jdbc;
1722
1723 import java.awt.Color;
1724 import java.awt.FlowLayout;
1725 import java.awt.event.ActionEvent;
1726 import java.awt.event.ActionListener;
1727 import java.sql.CallableStatement;
1728 import java.sql.Connection;
1729 import java.sql.DriverManager;
1730 import java.sql.PreparedStatement;
1731 import java.sql.ResultSet;
1732 import java.sql.Statement;
1733 import java.sql.Types;
1734
1735 import javax.swing.JButton;
1736 import javax.swing.JComboBox;
1737 import javax.swing.JFrame;
1738 import javax.swing.JLabel;
1739 import javax.swing.JTextField;
1740
1741 public class AllStmtsTest extends JFrame implements ActionListener{
1742     private JLabel lno,lname,lm1,lm2,lm3,lres;
1743     private JTextField tname,tm1,tm2,tm3,tres;
1744     private JButton bdetails,bresult;
1745     private JComboBox tno;
1746     private Connection con;
1747     private Statement st;
1748     private PreparedStatement ps;
1749     private CallableStatement cs;
1750     private ResultSet rs1,rs2;
1751
1752     //constructor
1753     public AllStmtsTest(){
1754         System.out.println("Constructor");
1755         setSize(400,400);
1756         setBackground(Color.cyan);
1757         setLayout(new FlowLayout());
1758         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
1759         //prpeare and Add comps
1760         lno=new JLabel("sno");
1761         add(lno);
1762         tno=new JComboBox();
1763         add(tno);
1764
1765         bdetails=new JButton("details");
1766         bdetails.addActionListener(this);
1767         add(bdetails);
1768
1769         lname=new JLabel("sname");
1770         add(lname);
1771         tname=new JTextField(10);
1772         add(tname);
1773
1774         lm1=new JLabel("Marks1");
1775         add(lm1);
1776         tm1=new JTextField(10);
1777         add(tm1);
1778
1779         lm2=new JLabel("Marks2");
1780         add(lm2);
1781         tm2=new JTextField(10);
1782         add(tm2);
```

```
1783     lm3=new JLabel("Marks3");
1784     add(lm3);
1785     tm3=new JTextField(10);
1786     add(tm3);
1787
1788     bresult=new JButton("Result");
1789     bresult.addActionListener(this);
1790     add(bresult);
1791
1792     lres=new JLabel("Result");
1793     add(lres);
1794     tres=new JTextField(10);
1795     add(tres);
1796
1797     setVisible(true);
1798
1799     myInit();
1800
1801 } //constructor
1802
1803 private void myInit(){
1804     System.out.println("myInit()");
1805     try{
1806         //register driver
1807         Class.forName("oracle.jdbc.driver.OracleDriver");
1808         //Establish the connection
1809         con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","scott","tiger");
1810
1811         //create Statement obj
1812         st=con.createStatement();
1813         //exeute logic to get sno values into combo box
1814         rs1=st.executeQuery("select sno from All_student");
1815         while(rs1.next()){
1816             tno.addItem(rs1.getString(1));
1817         }
1818         rs1.close();
1819         st.close();
1820         //create PrparedStatement obj
1821         ps=con.prepareStatement("select * from All_Student where sno=?");
1822         //create CallableStatement obj
1823         cs=con.prepareCall("{call FIND_PASS_FAIL(?,?,?,?,?)}");
1824         cs.registerOutParameter(4,Types.VARCHAR);
1825         //try
1826         catch(Exception e){
1827             e.printStackTrace();
1828         }
1829     } //myInit()
1830
1831
1832 @Override
1833 public void actionPerformed(ActionEvent ae) {
1834     System.out.println("action Performed");
1835     if(ae.getSource()==bdetails){
1836         System.out.println("Details btn is clicked");
1837         //get selected item from combo box
1838         String tsno=(String)tno.getSelectedItem();
1839         int no=Integer.parseInt(tsno);
1840         try{
1841             //set value to PrepareStatement obj Query
1842             ps.setInt(1,no);
1843             //execute the Query
1844             rs2=ps.executeQuery();
1845             //set values to text boxes
1846             if(rs2.next()){
1847                 tname.setText(rs2.getString(2));
1848                 tm1.setText(rs2.getString(3));
1849             }
1850         }
1851     }
1852 }
```

```
1849             tm2.setText(rs2.getString(4));
1850             tm3.setText(rs2.getString(5));
1851         }//if
1852     }//try
1853     catch(Exception e){
1854         e.printStackTrace();
1855     }
1856 } //if
1857 else{
1858     try{
1859         System.out.println("Result Btn is clicked");
1860         //read values from marks text boxes and set them IN param values
1861         cs.setInt(1,Integer.parseInt(tm1.getText()));
1862         cs.setInt(2,Integer.parseInt(tm2.getText()));
1863         cs.setInt(3,Integer.parseInt(tm3.getText()));
1864         //call pl/sql procedure
1865         cs.execute();
1866         //gather result from OUT param and set to text box
1867         tres.setText(cs.getString(4));
1868     }
1869     catch(Exception e){
1870         e.printStackTrace();
1871     }
1872 } //else
1873 }//actionPerformed(-)

1874
1875
1876
1877     public static void main(String[] args) {
1878         System.out.println("main(-)");
1879         AllStmtsTest stmts=new AllStmtsTest();
1880
1881     }
1882
1883 }
1884
1885 =====
1886 App27( Scrollable ResultSet obj)
1887 =====
1888 -----ScrollTest.java-----
1889 //ScrollableTest.java
1890 import java.sql.*;
1891
1892 public class ScrollableTest
1893 {
1894
1895     public static void main(String args[])throws Exception
1896     {
1897         //load jdbc driver class,create jdbc con obj
1898         Class.forName("oracle.jdbc.driver.OracleDriver");
1899         Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","sc";
1900         // create Jdbc Statement obj with type,mode values
1901         Statement st=con.createStatement	ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPD;
1902         // create Scrollable ResultSet obj
1903         ResultSet rs=st.executeQuery("select * from student");
1904         // display records (top-bottom)
1905         System.out.println("Top-to bottom");
1906         while(rs.next())
1907         {
1908             System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
1909         }//while
1910
1911         // display records (bottom-top)
1912         System.out.println("Bottom-Top");
1913         rs.afterLast();
1914         while(rs.previous())
```

```

1915     {
1916         System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
1917     }//while
1918
1919     //display records randomly
1920     rs.first(); //gives First record
1921     System.out.println(rs.getRow()+"--->"+rs.getInt(1)+" "+rs.getString(2)+" "+rs.getS
1922
1923     rs.last(); //gives Last record
1924     System.out.println(rs.getRow()+"--->"+rs.getInt(1)+" "+rs.getString(2)+" "+rs.getS
1925     rs.relative(-2); //gives Last but 3
1926     System.out.println(rs.getRow()+"--->"+rs.getInt(1)+" "+rs.getString(2)+" "+rs.getS
1927     rs.absolute(4); //gives 4th record
1928     System.out.println(rs.getRow()+"--->"+rs.getInt(1)+" "+rs.getString(2)+" "+rs.getS
1929     rs.absolute(-2); //gives last but 1 record
1930     System.out.println(rs.getRow()+"--->"+rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3)
1931         rs.relative(1); //gives last record
1932     System.out.println(rs.getRow()+"--->"+rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3)
1933     //close jdbc objs
1934     rs.close();
1935     ps.close();
1936     con.close();
1937 }
1938 }
=====

1940 App28(Using Scrollable ResultSet Object)
1941 =====
1942 -----ScrollFrame.java-----
1943 /*perform first,last,previous and next navigation buttons using scrollable ResultSet*/
1944
1945 import java.sql.*;
1946 import java.awt.*;
1947 import java.awt.event.*;
1948
1949 public class ScrollFrame extends JFrame implements ActionListener
1950 {
1951     private JTextField tsno, tsna, tsadd;
1952     private JLabel lno, lna, ladd;
1953     private JButton bfirst, blast, bnext, bprevious;
1954     private Connection con=null;
1955     private Statement st=null;
1956     private ResultSet rs=null;
1957     ScrollFrame()
1958     {
1959         System.out.println("constructor");
1960         setLayout(new FlowLayout());
1961         setBackground(Color.green);
1962         setSize(400,200);
1963
1964         //add comps to Frame window
1965         lno=new JLabel("SNO");
1966         add(lno);
1967
1968         tsno = new JTextField(20);
1969         add(tsno);
1970
1971         lna=new JLabel("NAME");
1972         add(lna);
1973
1974         tsna = new JTextField(20);
1975         add(tsna);
1976         ladd=new JLabel("ADDRESS");
1977         add(ladd);
1978
1979         tsadd = new JTextField(20);
1980         add(tsadd);

```

```
1981         bfirst = new JButton("first");
1982         add(bfirst);
1983         bfirst.addActionListener(this);
1984
1985
1986         blast =new JButton("last");
1987         add(blast);
1988         blast.addActionListener(this);
1989
1990
1991         bprevious = new JButton("previous");
1992         add(bprevious);
1993         bprevious.addActionListener(this);
1994
1995         bnext = new JButton("next");
1996         add(bnext);
1997         bnext.addActionListener(this);
1998
1999         setVisible(true);
2000
2001     }
2002     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
2003     makeConnection();
2004
2005     private void makeConnection()
2006     {
2007         try
2008         {
2009             System.out.println("makeConnection()");
2010             //register jdbc driver
2011             Class.forName("oracle.jdbc.driver.OracleDriver");
2012             //estblish the connection
2013             con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","scott","tiger")
2014             //create Statement obj
2015             st = con.createStatement	ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_RE_
2016             // create Scrollable ResultSet
2017             rs = st.executeQuery("select * from student");
2018             System.out.println("scrollble resultset object is ready");
2019         }
2020         catch(Exception ce)
2021         {
2022             ce.printStackTrace();
2023         }
2024     }//make connection
2025
2026     public void actionPerformed(ActionEvent ae)
2027     {
2028         System.out.println("actionPerformed()");
2029         boolean found=false;
2030         try
2031         {
2032             System.out.println(ae.getActionCommand());
2033             if(ae.getSource()==bfirst) //when first btn is clicked
2034             {
2035                 rs.first();
2036                 found=true;
2037             }
2038             else if(ae.getSource()==blast) //when last btn is clicked
2039             {
2040                 rs.last();
2041                 found=true;
2042             }
2043             else if(ae.getSource()==bnext) //when last btn is clicked
2044             {
2045                 if(!rs.isLast())
2046                 {
```

```

2047                               rs.next();
2048                               found=true;
2049                           }
2050           }
2051           else if(ae.getSource()==bprevious) //when previous btn is clicked
2052           {
2053               if(!rs.isFirst())
2054               {
2055                   rs.previous();
2056                   found=true;
2057               }
2058           }
2059
2060
2061           if(found)
2062           {
2063               tsno.setText(rs.getString(1));
2064               tsna.setText(rs.getString(2));
2065               tsadd.setText(rs.getString(3));
2066           }
2067       }catch(Exception e)
2068       {
2069           e.printStackTrace();
2070       }
2071   }//actionPerformed
2072
2073   public static void main(String args[])
2074   {
2075       System.out.println("main method");
2076       new ScrollFrame();
2077   }
2078 }
2079 =====
2080 App29(Program on SensitiveResultSet Object)
2081 =====
2082 -----SensitiveTest.java-----
2083 // prg to read data from DB table
2084 import java.sql.*;
2085
2086 public class SensitiveTest
2087 {
2088     public static void main(String args[])throws Exception
2089     {
2090         //register jdbc driver and establish the connection
2091         Class.forName("oracle.jdbc.driver.OracleDriver");
2092         Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","s
2093
2094         //create Scrollable and Sensitive ResultSet obj
2095         Statement st=con.createStatement	ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_U
2096         ResultSet rs=st.executeQuery("select sno,sname,sadd from student");
2097
2098         //logic to demonstrate sensitive behaviour
2099         int cnt=1;
2100         while(rs.next())
2101         {
2102             rs.refreshRow();
2103             System.out.println(rs.getString(1)+" "+rs.getString(2)+" "+rs.getString(3));
2104
2105             if(cnt==2)
2106                 Thread.sleep(40000); //modify records from SQL prompt
2107
2108             cnt++;
2109         }//while
2110
2111         //close jdbc objs
2112         rs.close();
2113         st.close();

```

```
2113                                     con.close();
2114     } //main
2115 } //class
2116 =====
2117 App30( prg on Updatable ResultSet)
2118 =====
2119 ----- UpdatableTest.java -----
2120 import java.sql.*;
2121 public class UpdatableTest
2122 {
2123     public static void main(String args[])throws Exception
2124     {
2125         //register jdbc driver and establish the connection
2126         Class.forName("oracle.jdbc.driver.OracleDriver");
2127         Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","sc
2128
2129 // create jdbc Statement obj and create UpdatableResultSet obj
2130     Statement st=con.createStatement	ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPD
2131     ResultSet rs=st.executeQuery("select sno,sname,sadd from student");
2132
2133     /*System.out.println("updates the 4th record");
2134         rs.absolute(4);
2135         rs.updateString(2,"babu");
2136         rs.updateRow();*/
2137
2138
2139     System.out.println("Inserting new row");
2140     rs.moveToInsertRow(); //creates an empty record in ResultSet obj
2141     rs.updateInt(1,131);
2142     rs.updateString(2,"keyboard");
2143     rs.updateInt(3,57);
2144     rs.insertRow();
2145
2146
2147 /* System.out.println("deleting 2 th row");
2148     rs.absolute(2);
2149     rs.deleteRow(); */
2150
2151     rs.close();
2152     st.close();
2153     con.close();
2154
2155 } //main
2156 } //class
2157 =====
2158 App31(To interact with Mysql DB s/w)
2159 =====
2160 ----- MySqlSelectTest.java -----
2161 import java.sql.*;
2162 import java.util.*;
2163 import java.io.*;
2164 public class MySqlSelectTest
2165 {
2166     public static void main(String args[])throws Exception
2167     {
2168         //register jdbc driver and establish the connection
2169         //Class.forName("org.gjt.mm.mysql.Driver"); (or)
2170         Class.forName("com.mysql.jdbc.Driver");
2171         //Establish the connection
2172         Connection con=DriverManager.getConnection("jdbc:mysql://mydb1","root","root");
2173         //Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb1","r
2174         // create Staement obj
2175         Statement st=con.createStatement();
2176         // execute the query
2177         ResultSet rs=st.executeQuery("select * from product");
2178         while(rs.next())
```

```
2179     {
2180         System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3))
2181     }
2182     //close all jdbc stream objects
2183     rs.close();
2184     st.close();
2185     con.close();
2186 } //main
2187 } //class
2188 =====
2189 App32 (Interacting with Excel as DB s/w)
2190 =====
2191 -----ExcelTest.java-----
2192 ./ExcelTest.java
2193 package com.nt.jdbc;
2194 import java.sql.*;
2195 public class ExcelTest {
2196     public static void main(String[] args) throws Exception {
2197         //register jdbd driver
2198         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
2199         //Establish the connection
2200         Connection con=DriverManager.getConnection("jdbc:odbc:xlsdsn");
2201         /* //create Staement obj
2202             Statement st=con.createStatement();
2203             // send and execute SQL Query in Db s/w
2204             ResultSet rs=st.executeQuery("select * from [Sheet1$]");
2205             // Process the ResultSet
2206             while(rs.next()){
2207                 System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
2208             }*/
2209         PreparedStatement ps=con.prepareStatement("insert into [sheet1$] values(?, ?, ?)");
2210         ps.setInt(1, 300);
2211         ps.setString(2, "ramesh");
2212         ps.setString(3, "indore");
2213         int result=ps.executeUpdate();
2214         System.out.println(result+" no.of records are inserted");
2215
2216         /* //close jdbc objs
2217             rs.close();
2218             st.close();
2219             con.close(); */
2220     }
2221 }
2222 =====
2223 App33) Interacting with CSV File
2224 =====
2225 -----TxtTest.java-----
2226 package com.nt.jdbc;
2227
2228 import java.sql.Connection;
2229 import java.sql.DriverManager;
2230 import java.sql.ResultSet;
2231 import java.sql.Statement;
2232
2233 public class TextTest {
2234     public static void main(String[] args) throws Exception {
2235
2236         //register jdbc driver
2237         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
2238         //Establish the connection
2239         Connection con=DriverManager.getConnection("jdbc:odbc:txtdsn");
2240
2241         //create Statement obj
2242         Statement st=con.createStatement();
2243
2244 }
```

```
2245 //send and execute SQL Query in Db s/w
2246 ResultSet rs=st.executeQuery("select * from file1.csv");
2247
2248 //process the ResultSet
2249 while(rs.next()){
2250     System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
2251 }
2252
2253 //close jdbc objs
2254 rs.close();
2255 st.close();
2256 con.close();
2257 } //main
2258 } //class
2259
2260 =====
2261 App34) prg to read data from Excel Sheet and to insert into Oracle DB table)
2262 =====
2263 ----- Excel To Oracle.java -----
2264 import java.sql.*;
2265 public class ExcelToOracle
2266 {
2267     public static void main(String args[])throws Exception
2268     {
2269         Connection excelcon=null,oracon=null;
2270         Statement st=null;
2271         PreparedStatement ps=null;
2272         ResultSet rs=null;
2273
2274         //register jdbc drivers
2275         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
2276         Class.forName("oracle.jdbc.driver.OracleDriver");
2277         System.out.println("Drivers Loaded");
2278
2279         //establish the connections
2280         excelcon=DriverManager.getConnection("jdbc:odbc:xlsdsn");
2281         oracon=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","scott","t");
2282         System.out.println("connections established");
2283
2284         //create Statement objs
2285         st=excelcon.createStatement();
2286         ps=oracon.prepareStatement("insert into student values(?, ?, ?)");
2287         System.out.println("statement prepared");
2288
2289         //get records from excel sheet
2290         rs=st.executeQuery("select * from [student$]");
2291
2292         while(rs.next())
2293         {
2294             //read each record from excel sheet
2295             int no=rs.getInt(1);
2296             String na=rs.getString(2);
2297             String add=rs.getString(3);
2298
2299             //insert each record into oracle db table
2300             ps.setInt(1,no);
2301             ps.setString(2,na);
2302             ps.setString(3,add);
2303
2304             ps.executeUpdate();
2305         } //while
2306
2307         System.out.println("Data has been copied");
2308         //close jdbc objs
2309         rs.close();
2310         st.close();
```

```

2311                     ps.close();
2312                     oracon.close();
2313                     excelcon.close();
2314
2315             }//main
2316 } //class
2317
2318 =====
2319 App35) Program to read MetaData of the Database)
2320 =====
2321 -----DBcap.java-----
2322 //DatabaseMetaData program
2323 import java.io.*;
2324 import java.sql.*;
2325
2326 public class DBcap
2327 {
2328     public static void main(String args[])throws Exception
2329     {
2330         //register jdbc driver and establish the connection
2331         Class.forName("oracle.jdbc.driver.OracleDriver");
2332         Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","scott","ti
2333
2334 //get MetaData
2335 DatabaseMetaData dbmd =con.getMetaData();
2336
2337         System.out.println("class that implements DatabaseMetaData(i) is"+dbmd.getCl
2338
2339         System.out.println("database name "+dbmd.getDatabaseProductName());
2340         System.out.println("database version "+dbmd.getDatabaseProductVersion());
2341         System.out.println("jdbc driver version "+dbmd.getDriverVersion());
2342         System.out.println("sql key words = "+dbmd.getSQLKeywords());
2343         System.out.println("numeric functions "+dbmd.getNumericFunctions());
2344         System.out.println("String Functions "+dbmd.getStringFunctions());
2345         System.out.println("System Functions"+dbmd.getSystemFunctions());
2346         System.out.println("Search String Escape"+dbmd.getSearchStringEscape());
2347         System.out.println(" supportsStoredProcedures = "+dbmd.supportsStoredProcedures());
2348         System.out.println(" getMaxRowSize = "+dbmd.getMaxRowSize());
2349         System.out.println(" getMaxStatementLength = "+dbmd.getMaxStatementLength());
2350         System.out.println("get Max tables in a select query="+dbmd.getMaxTablesInSelect());
2351         System.out.println("get Max Length of Table Name="+dbmd.getMaxTableNameLength());
2352         System.out.println("jdbc api version is"+dbmd.getJDBCMajorVersion()+" . "+dbmd.getJDBCMinorV
2353     }
2354 }
2355 =====
2356 App36 (Application on ResultSetMetaData to print table data along with col names)
2357 =====
2358 -----RsmSelectTest.java-----
2359 import java.sql.*;
2360 import java.util.*;
2361 import java.io.*;
2362 public class RsmSelectTest
2363 {
2364     public static void main(String args[])throws Exception
2365     {
2366         Class.forName("oracle.jdbc.driver.OracleDriver");
2367         Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","scott","t
2368         // create Statement obj
2369         Statement st=con.createStatement();
2370         // execute the query
2371         ResultSet rs=st.executeQuery("select * from student");
2372         //get ResultSetMetaData obj
2373         ResultSetMetaData rsmd=rs.getMetaData();
2374         // get col count
2375         int cnt=rsmd.getColumnCount();
2376         //print col names

```

```

2377         for(int i=0;i<=cnt;++i)
2378         {
2379             System.out.print(rsmd.getColumnLabel(i)+"\t\t\t");
2380         }//for
2381         System.out.println();
2382         //print col vlaues
2383         while(rs.next())
2384         {
2385             for(int i=0;i<cnt;++i)
2386             {
2387                 System.out.print(rs.getString(i)+"\t\t\t");
2388             }//for
2389             System.out.println();
2390         }//while
2391
2392         //close all jdbc stream objects
2393         rs.close();
2394         st.close();
2395         con.close();
2396     }//main
2397 } //class
2398 =====
2400 App37(Program using ParameterMetaData)
2401 =====
2402 -----PMDTest.java-----
2403 import java.sql.*;
2404
2405 public class PMDTest {
2406
2407     public static void main(java.lang.String[] args) throws Exception
2408     {
2409         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
2410         Connection con=DriverManager.getConnection("jdbc:odbc:mysqldsn","root","root");
2411
2412         PreparedStatement ps = con.prepareStatement("insert into student values(?, ?, ?)");
2413
2414         ParameterMetaData pmd=ps.getParameterMetaData();
2415
2416         System.out.println("impl class name of ParameterMetaData(i)"+pmd.getClass().getName());
2417
2418         int cnt=pmd.getParameterCount();
2419
2420         for (int i = 1; i<=cnt; i++) {
2421             System.out.println("Parameter number " + i);
2422
2423             System.out.println(" Mode is " + pmd.getParameterMode(i));
2424             System.out.println(" Type is " + pmd.getParameterType(i));
2425             System.out.println(" Type name is " + pmd.getParameterTypeName(i));
2426             System.out.println(" Precision is " + pmd.getPrecision(i));
2427             System.out.println(" Scale is " + pmd.getScale(i));
2428             System.out.println(" Nullable? is " + pmd.isNullable(i));
2429             System.out.println(" Signed? is " + pmd.isSigned(i));
2430         }
2431     }
2432 }
2433 }
2434 =====
2436 App38(program on jdbc rowset )
2437 =====
2438 -----Row1.java-----
2439 package com.nt.jdbc;
2440 import java.sql.*;
2441 import javax.sql.*;
2442 import oracle.jdbc.rowset.*;
```

```
2443 public class Row1
2444 {
2445     public static void main (String []args)
2446     {
2447         try
2448         {
2449             //create RowsSet obj
2450             RowSet jrowset = new OracleJDBCRowSet();
2451             //set jdbc properties
2452             jrowset.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
2453             jrowset.setUsername("scott");
2454             jrowset.setPassword("tiger");
2455             jrowset.setCommand("select empno from emp");
2456             jrowset.execute();
2457             System.out.println("command executed");
2458             //process the Rowset
2459             while (jrowset.next())
2460             {
2461                 System.out.println (jrowset.getInt(1));
2462             }
2463             jrowset.close();
2464         }
2465         catch(Exception ee)
2466         {
2467             System.out.println(ee.toString());
2468         }
2469     }
2470 }
2471 =====
2472 App39) CachedRowset
2473 =====
2474 -----Row2.java-----
2475 package com.nt.jdbc;
2476 import java.sql.*;
2477 import javax.sql.*;
2478 public class Row2
2479 {
2480     public static void main (String []args)throws Exception
2481     {
2482         //create RowSet obj
2483         oracle.jdbc.rowset.OracleCachedRowSet crowset= new oracle.jdbc.rowset.OracleCachedRowSet();
2484         //set jdbc properties
2485         crowset.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
2486         crowset.setUsername("scott");
2487         crowset.setPassword("tiger");
2488         crowset.setCommand("select * from student");
2489         crowset.execute(); //executes Query
2490         System.out.println("query executed");
2491         //process the Rowset
2492         while (crowset.next())
2493         {
2494             System.out.println (crowset.getInt(1)+" "+crowset.getString(2)+" "+crowset.getString(3) )
2495         }
2496
2497         crowset.setReadOnly(false);
2498         crowset.absoluter(2);
2499         crowset.updateString(3,"new delhi");
2500         crowset.updateRow();
2501         crowset.acceptChanges();
2502         //close Rowset
2503         crowset.close();
2504     } //main
2505 } //class
2506 =====
2507 App40) program on webrowset
2508 =====
```

```
2509 -----Row3.java-----
2510 package com.nt.jdbc;
2511 import java.io.File;
2512 import java.io.FileWriter;
2513 import java.io.StringWriter;
2514 import javax.sql.rowset.WebRowSet;
2515 import oracle.jdbc.rowset.OracleWebRowSet;
2516
2517 public class Row3 {
2518     public static void main(String[] args)  {
2519         try
2520     {
2521             //create WebRowSet obj
2522             OracleWebRowSet webRS= new OracleWebRowSet();
2523             //set jdbc properties
2524             webRS.setUsername("scott");
2525             webRS.setPassword("tiger");
2526             webRS.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
2527             webRS.setCommand("SELECT * from employee");
2528             //execute the query
2529             webRS.execute();
2530             //create or locate file
2531             File myFile=new File("c:/employee1.xml");
2532             FileWriter fw = new FileWriter(myFile);
2533
2534             System.out.println("Writing db data to file " + myFile.getAbsolutePath());
2535             webRS.writeXml(fw); // writes the records of Rowset to file
2536
2537             // convert xml to a String object for display purpose
2538             StringWriter sw = new StringWriter();
2539             webRS.writeXml(sw);
2540             System.out.println(sw.toString());
2541
2542             fw.flush();
2543             fw.close();
2544 }catch(Exception e)
2545 {
2546     e.printStackTrace();
2547 }
2548
2549 }
2550 }
2551 =====
2552 App40) JoinRowSet
2553 =====
2554 -----Row4.java-----
2555 package com.nt.jdbc;
2556 import oracle.jdbc.rowset.OracleCachedRowSet;
2557 import oracle.jdbc.rowset.OracleJoinRowSet;
2558
2559
2560 public class Row4{
2561     public static void main(String args[]) {
2562         try {
2563
2564             //Cached RowSet1
2565             OracleCachedRowSet crs1 = new OracleCachedRowSet();
2566             crs1.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
2567             crs1.setUsername("scott");
2568             crs1.setPassword("tiger");
2569             crs1.setCommand("select empno,ename,deptno from emp");
2570             crs1.execute();
2571             crs1.setMatchColumn(3);
2572
2573             //Cached RowSet2
2574             OracleCachedRowSet crs2 = new OracleCachedRowSet();
```

```
2575     crs2.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
2576     crs2.setUsername("scott");
2577     crs2.setPassword("tiger");
2578     crs2.setCommand("select * from dept");
2579     crs2.execute();
2580     crs2.setMatchColumn(1);
2581
2582     //Add multiple RowSets to JoinRowSet
2583     OracleJoinRowSet joinRS = new OracleJoinRowSet();
2584     joinRS.addRowSet(crs1);
2585     joinRS.addRowSet(crs2);
2586
2587     //Process JoinRowSet
2588     while (joinRS.next()) {
2589         System.out.println(joinRS.getString(1)+" "+joinRS.getString(2)+" "+joinRS.getString(3)+"");
2590     }
2591
2592     } catch(Exception e) {
2593         e.printStackTrace();
2594     }
2595 }
2596 }
2597 =====
2598 App 41) FilteredRowset
2599 =====
2600 -----Row5.java-----
2601 package com.nt.jdbc;
2602 import java.sql.SQLException;
2603 import javax.sql.RowSet;
2604 import javax.sql.rowset.CachedRowSet;
2605 import javax.sql.rowset.Predicate;
2606
2607 import oracle.jdbc.rowset.OracleFilteredRowSet;
2608
2609
2610 class Filter1 implements Predicate {
2611     private String colName;
2612
2613     public Filter1(String colName) {
2614         this.colName = colName;
2615     }
2616
2617     public boolean evaluate(RowSet rs) {
2618         try {
2619             CachedRowSet crs = (CachedRowSet) rs; //get Each record of ResultSet as CachedRowSet
2620             String object = crs.getString(colName);
2621             if (object != null && (object.charAt(0) == 'A' || object.charAt(0) == 'a')) {
2622                 return true;
2623             } else {
2624                 return false;
2625             }
2626         } catch (Exception e) {
2627             return false;
2628         }
2629     }
2630
2631     public boolean evaluate(Object arg0, int arg1) throws SQLException {
2632
2633         return false;
2634     }
2635
2636     public boolean evaluate(Object arg0, String arg1) throws SQLException {
2637
2638         return false;
2639     }
2640 }
```

```
2641 public class Row5 {
2642     public static void main(String[] args) throws Exception {
2643         OracleFilteredRowSet frs = new OracleFilteredRowSet();
2644         frs.setUsername("scott");
2645         frs.setPassword("tiger");
2646         frs.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
2647         frs.setCommand("select * from emp");
2648         //Apply Filter
2649         frs.setFilter(new Filter1("ename"));
2650         frs.execute();
2651         while (frs.next()) {
2652             System.out.println(frs.getInt(1)+" "+frs.getString(2)+" "+frs.getString(3));
2653         }
2654     }
2655 }
2656 =====
2657 App42) BatchProcessing/Updataion)
2658 =====
2659 -----BatchProcess.java-----
2660 package com.nt.jdbc;
2661 import java.sql.*;
2662 public class BatchProcess
2663 {
2664     public static void main(String args[])throws Exception
2665     {
2666         //load jdbc driver and establish the connection
2667         Class.forName("oracle.jdbc.driver.OracleDriver");
2668         Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "sc
2669         //disable the autocommit mode
2670         con.setAutoCommit(false);
2671         //create Statement obj
2672         Statement st=con.createStatement();
2673         // add queries to batch
2674         st.addBatch("update student set sname='chinna1' where sno=10070");
2675         st.addBatch("insert into student values(169,'ccc','muak;')");
2676         st.addBatch("delete from student where sno=34");
2677         // execute the batch
2678         int result[]=st.executeBatch();
2679         //process the results
2680         int sum=0;
2681         for(int i=0;i<result.length;++i)
2682         {
2683             sum=sum+result[i];
2684         }
2685         System.out.println("no.of records that are effected"+sum);
2686
2687         st.close();
2688         con.close();
2689
2690     }//main
2691 } //class
2692 =====
2693 App43( Transcation Management)
2694 =====
2695 -----TxMgmtTest.java-----
2696 //TxMgmtTest.java
2697 package com.nt.jdbc;
2698 import java.sql.*;
2699 import java.util.*;
2700
2701 public class TxMgmtTest
2702 {
2703     public static void main(String args[])
2704     {
2705         Connection con=null;
```

```
2707     Statement st=null;
2708     try
2709     {
2710         //read input value from keyboard
2711         Scanner sc=new Scanner(System.in);
2712         System.out.println("Enter src a/c no:");
2713         int srcacno=sc.nextInt();
2714         System.out.println("Enter Dest a/c no:");
2715         int destacno=sc.nextInt();
2716         System.out.println("Enter Amt to transfer:");
2717         int amt=sc.nextInt();
2718
2719         // create jdbc con obj
2720         Class.forName("oracle.jdbc.driver.OracleDriver");
2721         con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","scott"
2722
2723         //disable auto commt mode
2724         con.setAutoCommit(false); //begins Tx
2725
2726         // create jdbc statement object
2727         st=con.createStatement();
2728
2729         // add queries of transferMoney operation to batch processing
2730
2731         //query for withdraw operation
2732         st.addBatch("update jdbc_account set balance=balance-"+amt+" where acno="+sr
2733                                         //query for deposite operation
2734         st.addBatch("update jdbc_account set balance=balance+"+amt+" where acno="+de
2735
2736         // execute the queries of batch
2737         int res[]=st.executeBatch();
2738
2739         //write the code of Tx mgmt
2740         boolean flag=true;
2741
2742         for(int i=0;i<res.length;++i)
2743         {
2744             if(res[i]==0) // if any element value is 0
2745             {
2746                 flag=false;
2747                 break;
2748             }
2749         }
2750
2751         //commit or rollback the Tx
2752         if(flag==false)
2753         {
2754             con.rollback();
2755             System.out.println("Tx is rolledback(money not transferred)");
2756         }
2757         else
2758         {
2759             con.commit();
2760             System.out.println("Tx is committed(Money transferred)");
2761         }
2762
2763         //close jdbc objs
2764         st.close();
2765         con.close();
2766     }
2767     catch(Exception e)
2768     {
2769         try
2770         {
2771             con.rollback();
2772         }
2773     }
2774 }
```

```
2773                     catch(Exception e1){e1.printStackTrace(); }
2774             } //catch
2775         } //main
2776     } //class
2777 //>javac TxMgmtTest.java
2778 //>java TxMgmtTest
2779 =====
2780 App44 (Batch Processing by using PreparedStatement obj)
2781 =====
2782 -----psBatchTest.java-----
2783 //PsBatchTest.java
2784 package com.nt.jdbc;
2785 import java.sql.*;
2786 public class PsBatchTest
2787 {
2788     public static void main(String args[])throws Exception
2789     {
2790         // create jdbc con obj
2791         Class.forName("oracle.jdbc.driver.OracleDriver");
2792         Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "sc
2793
2794         //create PreparedStatement obj
2795         PreparedStatement ps=con.prepareStatement("insert into student values(?, ?, ?)");
2796         //add param values to batch
2797         ps.setInt(1,111);
2798         ps.setString(2,"raja");
2799         ps.setString(3,"hyd");
2800         ps.addBatch() //add 1st set param values to batch
2801
2802         ps.setInt(1,222);
2803         ps.setString(2,"ravi");
2804         ps.setString(3,"hyd1");
2805         ps.addBatch()//add 2nd set param values to batch
2806
2807         ps.setInt(1,333);
2808         ps.setString(2,"rakesh");
2809         ps.setString(3,"hyd2");
2810         ps.addBatch()//add 3rd set param values to batch
2811
2812         // execute the Query
2813         int res[] = ps.executeBatch();
2814
2815         //process the result
2816         int cnt=0;
2817         for(int i=0;i<res.length;++i)
2818         {
2819             cnt=cnt+res[i];
2820         }
2821
2822         System.out.println("no.of records that inserted:"+cnt);
2823
2824         //close jdbc objs
2825         ps.close();
2826         con.close();
2827     } //main
2828 } //class
2829 =====
2830 App45(program to make JDBC Application Flexible)
2831 =====
2832 -----DBDetails.properties-----
2833
2834 #properties file
2835 driver=oracle.jdbc.driver.OracleDriver
2836 url=jdbc:oracle:oci8:@xe
2837 user=scott
2838 password=tiger
```

```

2839 -----FlexibleJDBCTest.java-----
2840 //FlexibleJDBCTest.java
2841 package com.nt.jdbc;
2842 import java.sql.*;
2843 import java.io.*;
2844 import java.util.*;
2845
2846 public class FlexTest
2847 {
2848     public static void main(String[] args) throws Exception
2849     {
2850         //locate properties file
2851         FileInputStream fis=new FileInputStream("DBDetails.properties");
2852         //load the data of properties file to Properties class obj
2853         Properties p=new Properties();
2854         p.load(fis);
2855         // get jdbc details from Properties class obj
2856         String s1=p.getProperty("driver");
2857         String s2=p.getProperty("url");
2858         String s3=p.getProperty("user");
2859         String s4=p.getProperty("password");
2860
2861         //write jdbc code
2862         Class.forName(s1);
2863         Connection con=DriverManager.getConnection(s2,s3,s4);
2864         Statement st=con.createStatement();
2865         ResultSet rs=st.executeQuery("select * from student ");
2866         while(rs.next())
2867         {
2868             System.out.println(rs.getString(1)+" "+rs.getString(2)+" "+rs.getString(3)
2869         }
2870         rs.close();
2871         st.close();
2872         con.close();
2873
2874     }//main
2875 }//class
2876 =====
2877 App46 (on DriverManaged Jdbc Conn Pooling )(Client Side con pool)
2878 =====
2879 -----ConnPoolTest.java-----
2880 //ConnPoolTest.java (works with DriverManaged jdbc con pool)
2881 package com.nt.jdbc;
2882 import java.sql.*;
2883 import oracle.jdbc.pool.*; // (for OracleConnectionPoolDataSource (c))
2884 public class ConnPoolTest
2885 {
2886     public static void main(String args[]) throws Exception
2887     {
2888         // create jdbc datasource obj representing an empty driver managed con pool for oracle
2889         OracleConnectionPoolDataSource ds = new OracleConnectionPoolDataSource();
2890
2891         // give details to create jdbc con objects in the above empty con pool
2892         ds.setDriverType ("thin");
2893         ds.setServerName ("localhost");
2894         ds.setPortNumber (1521);
2895         ds.setServiceName ("xe");
2896         ds.setUser ("scott");
2897         ds.setPassword("tiger"); //-->now jdbc con pool is ready with jdbc con objs
2898
2899         // get one jdbc con obj from jdbc con pool through DataSource obj
2900         Connection con=ds.getConnection();
2901         // write jdbc persistance logic
2902         Statement st=con.createStatement();
2903         ResultSet rs=st.executeQuery("select * from student");
2904         while(rs.next())

```

```

2905     {
2906         System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
2907     }
2908     //close jdbc objects...
2909     rs.close();
2910     st.close();
2911     con.close(); //releases con obj back to Jdbc Con pool
2912 } //main
2913 } //class
2914 //addojdbc14.jar file to classpath
2915 //javac ConnPoolTest.java
2916 //java ConnPoolTest
2917 =====
2918 App47(SavePoint )
2919 =====
2920 -----SavePointTest.java-----
2921 //SavePointTest.java
2922 package com.nt.jdbc;
2923 import java.sql.*;
2924 public class SavePointTest
2925 {
2926     public static void main(String args[])throws Exception
2927     {
2928         //register jdbc driver and establish the connection
2929         Class.forName("oracle.jdbc.driver.OracleDriver");
2930         Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","scott","tiger");
2931
2932         //create statement obj
2933         Statement st = con.createStatement();
2934
2935         // Begin Tx
2936         con.setAutoCommit(false);
2937         //query1 (outside the save point area)
2938         st.executeUpdate("insert into student values(567,'xyz','hyd1')");
2939         // create Named Save Point("p1")
2940         Savepoint sp=con.setSavepoint("p1");
2941         //query2(      in save point area)
2942         st.executeUpdate("update student set saddr='new delhi1' where sno=111");
2943         //rollback upto savepoint(p1)
2944         con.rollback(sp);
2945         // commit Tx
2946         con.commit(); //end of the Tx
2947         //query1 will be committed but query2 will be rolled back
2948         //becoz the query2 is there in savepoint area..(p1)
2949     } //main
2950 } //class
2951 //javac SavePointTest.java
2952 //java SavePointTest
2953 =====
2954 App48 (interacting with Postgres SQL DB s/w)
2955 =====
2956 -----PostgreSQLApp.java-----
2957 //PostgreSQLApp.java
2958 package com.nt.jdbc;
2959 import java.sql.*;
2960 public class PostgreSQLApp
2961 {
2962     public static void main(String args[])throws Exception
2963     {
2964         //register jdbc driver with DriverManager Service
2965         Class.forName("org.postgresql.Driver");
2966
2967         //establish the connection
2968         Connection con=DriverManager.getConnection("jdbc:postgresql:mydb1","postgres","root");
2969         //or
2970         Connection con=DriverManager.getConnection("jdbc:postgresql://localhost:5432/mydb1","postgres");

```

```
2971      // create jdbc statement obj
2972      Statement st=con.createStatement();
2973      // send and execute SQL query
2974      ResultSet rs=st.executeQuery("select * from product");
2975      //process the jdbc ResultSet obj
2976      while(rs.next())
2977      {
2978          System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getFloat(3));
2979      }
2980
2981      // close jdbc objects
2982      rs.close();
2983      st.close();
2984      con.close();
2985  } //main
2986 } //class
2987
2988 //>javac PostgreSQLApp.java
2989 //>java PostgreSQLApp
2990 =====
2991 App49( Example Application Type3 JDBC driver)
2992 =====
2993 -----Type3Test1.java-----
2994 //Type3Test1.java
2995 package com.nt.jdbc;
2996 import java.sql.*;
2997 class Type3Test1
2998 {
2999     public static void main(String[] args) throws Exception
3000     {
3001         //register type jdbc driver
3002         Class.forName("ids.sql.IDSDriver");
3003         //Establish the connection
3004         //Connection con = DriverManager.getConnection("jdbc:ids://localhost:12/conn?dsn='or
3005         Connection con = DriverManager.getConnection("jdbc:ids://localhost:12/conn?dsn='accd
3006
3007         //create jdbc Statement obj
3008         Statement st = con.createStatement();
3009         // execute the SQL Query
3010         ResultSet rs=st.executeQuery("Select * from student");
3011         //process the ResultSet obj
3012         while(rs.next())
3013         {
3014             System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
3015         }
3016     }
3017     //close jdbc objs
3018     rs.close();
3019     st.close();
3020     con.close();
3021 }
3022 }
3023 =====
3024 App50 (Application on Type5 JDBC Driver)
3025 =====
3026 -----Type5Test.java-----
3027 //Type5Test.java
3028 package com.nt.jdbc;
3029 import java.sql.*;
3030 public class Type5Test
3031 {    public static void main(String[] args)throws Exception {
3032
3033         //loading jdbc driver class is optional here
3034         //Class.forName("com.ddtek.jdbc.oracle.OracleDriver");
3035
3036         // Establish the Connection
```

```

3037     String url = "jdbc:datadirect:oracle://localhost:1521;ServiceName=xe";
3038     Connection con = DriverManager.getConnection(url, "scott", "tiger");
3039
3040         //send and execute the query
3041         Statement st=con.createStatement();
3042         ResultSet rs=st.executeQuery("select * from student");
3043         while(rs.next())
3044         {
3045             System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
3046         }
3047         //close jdbc objs
3048         rs.close();
3049         st.close();
3050         con.close();
3051     }//main
3052 } //class
3053 //>javac Type5Test.java
3054 //>java TypeTest
3055
3056 =====
3057     Applications on Reflection API
3058 =====
3059 -----App1.java-----
3060 //App1.java (gives all classes of inheritance hierarchy for the given java class)
3061 public class App1
3062 {
3063     public static void main(String args[])throws Exception
3064     {
3065         // load the given class to application
3066         Class c=Class.forName(args[0]);
3067         // get super class of given class
3068         Class sc=c.getSuperclass();
3069         System.out.println("The inheritance of given class is");
3070         while(sc!=null)
3071         {
3072             System.out.println("\t\t"+sc.getName());
3073             c=sc;
3074             sc=c.getSuperclass();
3075         }//while
3076
3077     }//main
3078 } //class
3079 //>javac App1.java ( compile time)
3080 //>java App1 java.lang.String (runtime)
3081 //>java App1 java.awt.TextField
3082 -----App2.java-----
3083 //App2.java (gives interfaces implemented by the given java class)
3084 public class App2
3085 {
3086     public static void main(String args[])throws Exception
3087     {
3088         // load the given into Application
3089         Class c=Class.forName(args[0]);
3090         // get the interfaces implemented by given class
3091         Class inter[]=c.getInterfaces();
3092         // print interface names
3093         System.out.println("The interfaces implemented by "+args[0]);
3094         for(int i=0;i<inter.length;++i)
3095         {
3096             System.out.println("\t"+inter[i].getName());
3097         }//for
3098
3099     }//main
3100 } //class
3101 //>javac App2.java
3102 //>java App2 java.lang.String

```

```

3103 //>java App2 java.util.Date
3104 -----App4.java-----
3105 //App4.java (gives modifiers of the given java class)
3106 import java.lang.reflect.Modifier;
3107 public class App4
3108 {
3109     public static void main(String args[])throws Exception
3110     {
3111         // load the given in to Application
3112         Class c=Class.forName(args[0]);
3113         // get the modifiers of given class
3114         int x=c.getModifiers();
3115         System.out.println("x value is"+x);
3116         //print modifiers
3117         System.out.println("The modifiers of "+args[0]);
3118         if(Modifier.isPublic(x))
3119             System.out.println("\t\t public");
3120         if(Modifier.isFinal(x))
3121             System.out.println("\t\t final");
3122         if(Modifier.isAbstract(x))
3123             System.out.println("\t\t abstract");
3124     }//main
3125 }//class
3126 //>javac App4.java
3127 //>java App4 java.lang.String
3128 //>java App4 java.lang.Integer
3129 -----App5.java-----
3130 // App5.java (gives the modifiers of the given java class)
3131 import java.lang.reflect.*;
3132
3133 public class App5
3134 {
3135
3136     public static void main(String args[])throws Exception
3137     {
3138         //load the given class into Application
3139         Class c=Class.forName(args[0]);
3140         // get all the fields of given java class
3141         Field f[]=c.getDeclaredFields();
3142         //dipslay member variable details
3143         System.out.println("The Fields of "+args[0]);
3144         for(int i=0;i<f.length;++i)
3145         {
3146             // get modifiers of each field
3147             int x=f[i].getModifiers();
3148             if(Modifier.isPublic(x))
3149                 System.out.print("\t\t public");
3150             if(Modifier.isPrivate(x))
3151                 System.out.print("\t\t private");
3152             if(Modifier.isProtected(x))
3153                 System.out.print("\t\t protected");
3154             if(Modifier.isStatic(x))
3155                 System.out.print("\t\t static");
3156             if(Modifier.isFinal(x))
3157                 System.out.print("\t\t final");
3158             if(Modifier.isVolatile(x))
3159                 System.out.print("\t\t volatile");
3160             if(Modifier.isTransient(x))
3161                 System.out.print("\t\t transient");
3162             String ftype=f[i].getType().getName();
3163             String fname=f[i].getName();
3164             System.out.println("    "+ftype+"    "+fname);
3165         }//for
3166     }//main
3167 }//class
3168 //>javac App5.java

```

```
3169 //>java App5 java.lang.String
3170 //>java App5 java.lang.Integer
3171 -----App6.java-----
3172 //App6.java (gives all the constructors of given java class)
3173 import java.lang.reflect.*;
3174
3175 public class App6
3176 {
3177     public static void main(String args[])throws Exception
3178     {
3179         //load the given class in to app
3180         Class c=Class.forName(args[0]);
3181         // get the constructors of given java class
3182         Constructor cons[]=c.getDeclaredConstructors();
3183         //print constructor details
3184         System.out.println("the constructors of "+args[0]);
3185         for(int i=0;i<cons.length;++i)
3186         {
3187             //get modifiers of each constructor
3188             int x=cons[i].getModifiers();
3189             if(Modifier.isPublic(x))
3190                 System.out.print("\t public");
3191             if(Modifier.isProtected(x))
3192                 System.out.print("\t protected");
3193             if(Modifier.isPrivate(x))
3194                 System.out.print("\t private");
3195             // get name of the each constructor
3196             String name=cons[i].getName();
3197             //get parameter types of the each constructor
3198             Class params[]=cons[i].getParameterTypes();
3199             System.out.print(" "+name+"(");
3200             for(int k=0;k<params.length;++k)
3201             {
3202                 System.out.print(params[k].getName()+" ");
3203             }
3204             System.out.println(")");
3205         }
3206     }
3207 } //main
3208 //javac App6.java
3209 //java App6 java.lang.String
3210 //java App6 java.util.Date
3211
3212
3213
```

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 spphoto 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.

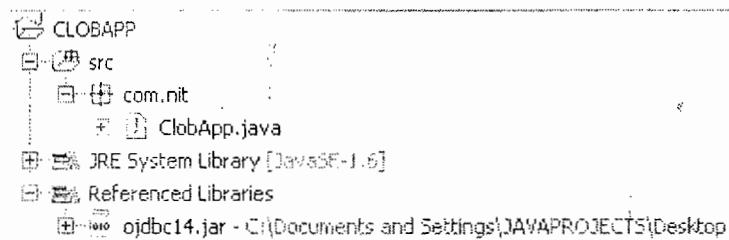
The **oracle.sql.CLOB** class includes the following methods:

- **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 terms, 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 BlobRetrieveApp {
    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 /

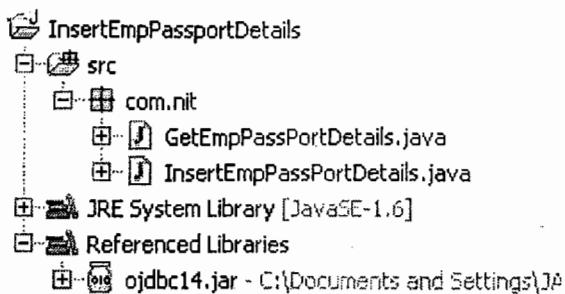
Type created.

```
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 (...), getClob(...),...

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
- e. 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. PreparedStatement 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 :

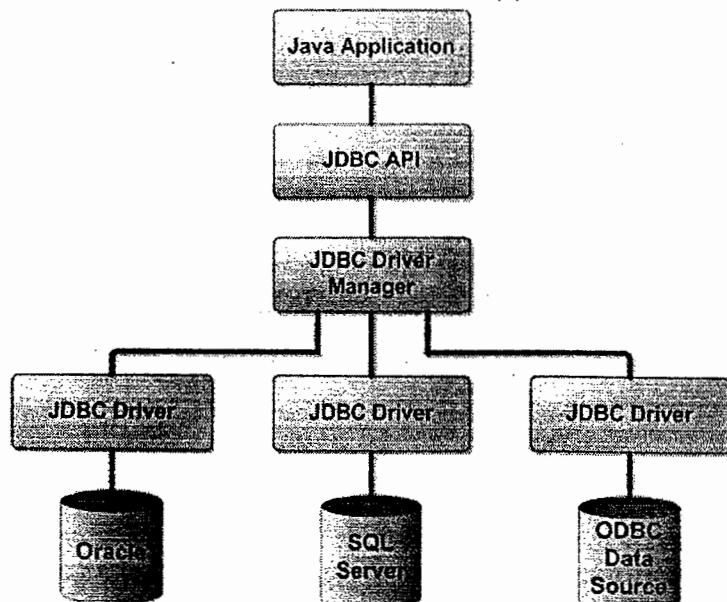
- Load the RDBMS specific JDBC driver because this driver actually communicates with the database (Incase of JDBC 4.0 this is automatically loaded).
- Open the connection to database which is then used to send SQL statements and get results back.
- Create JDBC Statement object. This object contains SQL 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.

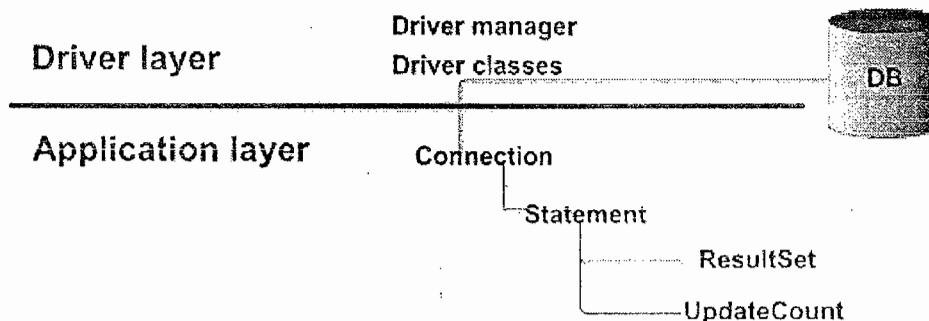


Figure: JDBC Application

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?

Statement	PreparedStatement
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 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.
- **Type 2: 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?

TYPE_SCROLL_INSENSITIVE	TYPE_SCROLL_SENSITIVE
An insensitive resultset is like the snapshot of the data in the database when query was executed.	A sensitive resultset does NOT represent a snapshot of data, rather it contains points to those rows which satisfy the query condition.
After we get the resultset the changes made to data are not visible through the resultset, and hence they are known as insensitive.	After we obtain the resultset if the data is modified then such modifications are visible through resultset.
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. Servlets 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.

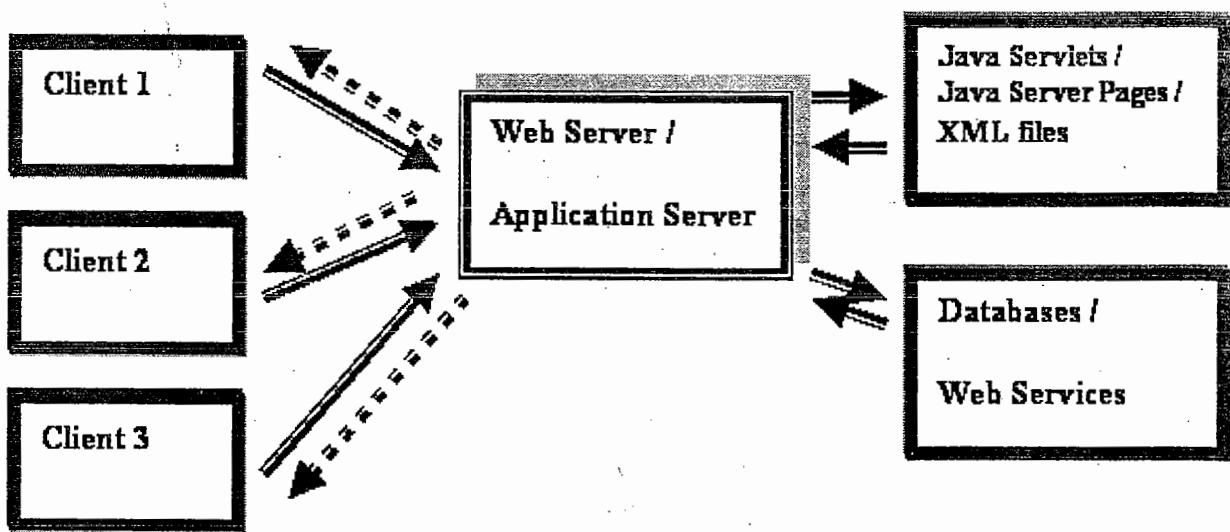


Figure – Server Side Programming (architecture)

2. Advantages of Server Side Programs

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

- All programs reside in one machine called the Server. Any number of remote machines (called clients) can access the server programs.
- 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.
- 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 check 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:

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

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 HTTPRequests 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 servets 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"

- 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
- There is one ServletContext object per "web application" per Java Virtual Machine

How to Access ServletContext Object?

- Within your servlet code, call getServletContext()
- Within your servlet filter code, call getServletContext()
- 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?

- 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
- Yet, HTTP is stateless
- HttpSession maintains client state
 - Used by Servlets to set and get the values of session scope attributes

How to Get HttpSession?

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

What is Servlet Request?

- Contains data passed from client to servlet
- 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
 - **Example:** javax.servlet.request.X509Certificate attribute for HTTPS
- Servlet set application-specific attribute
 - void setAttribute(java.lang.String name, java.lang.Object 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
 - . `Add To Cart`
 - . `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 is 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

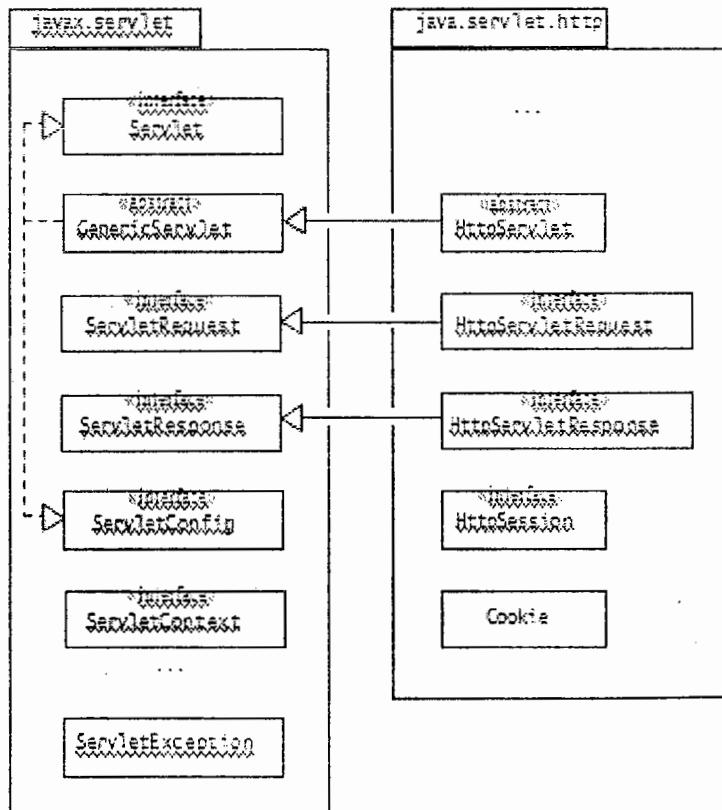
Handling Errors

- . 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



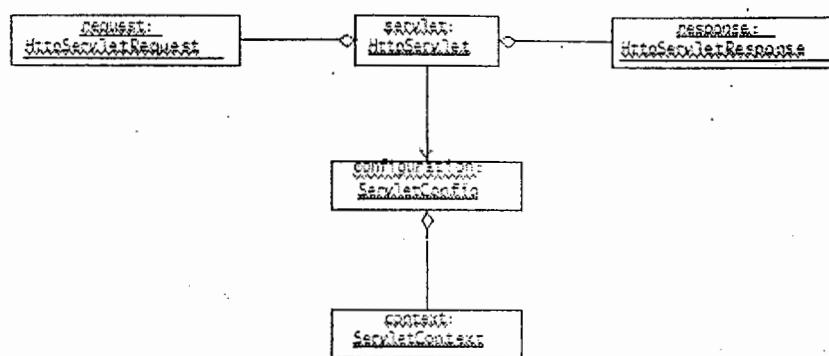
The `javax.servlet` Package

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

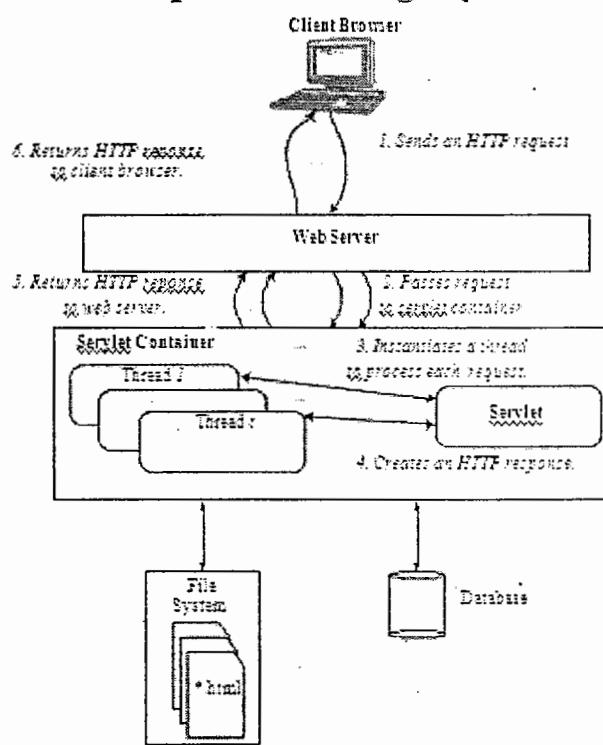
Class/Interface	Description
<code>abstract class HttpServlet</code>	This abstract class extends the <code>GenericServlet</code> class and is used for implementing HTTP servlets, i.e. servlets which use HTTP for requests and responses. In particular, it provides stubs for the <code>doGet(), doPost(), head()</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.

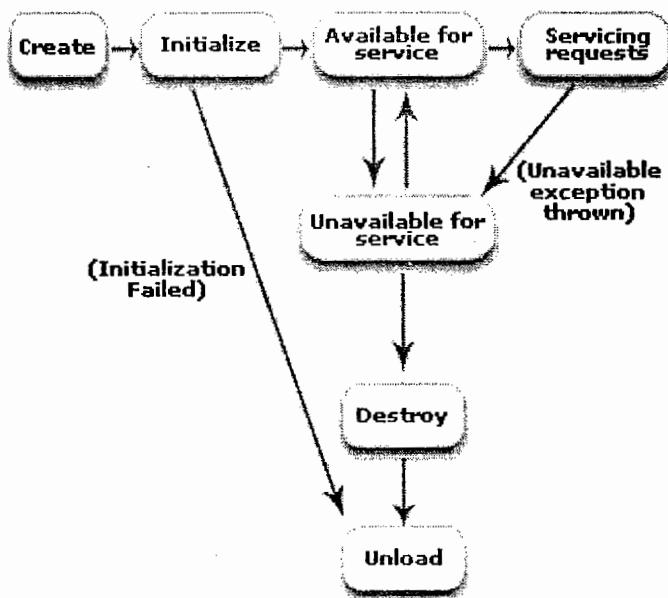


Figure:Life Cycle of a Servlet

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()

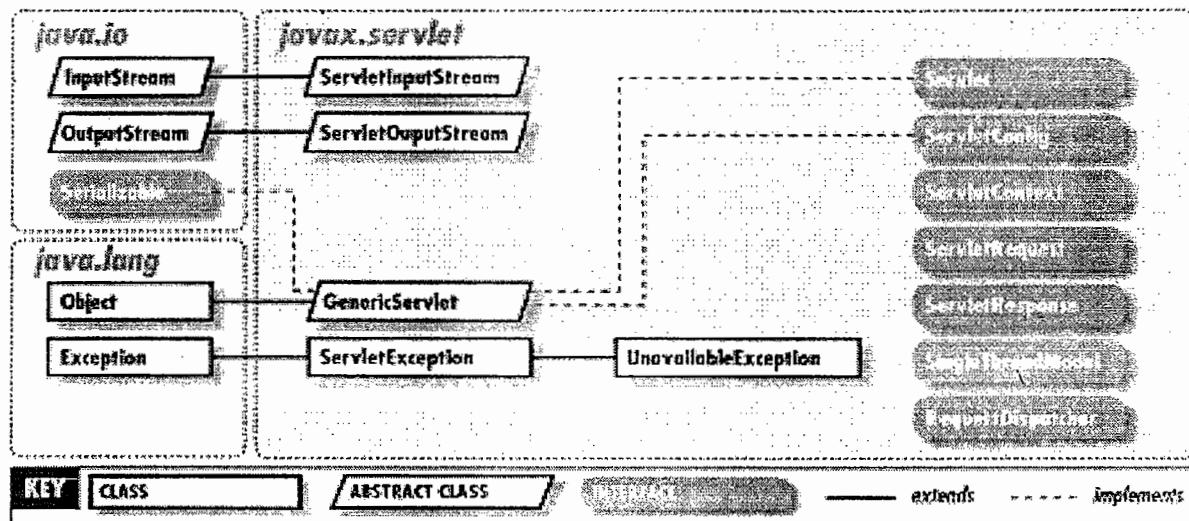
1. init():- init() method would be executed by the container automatically as soon as an object of the servlet is created. object of the servlet would be created only once. Thus init() method would be executed only once i.e when the object of the servlet is created for the first time.

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

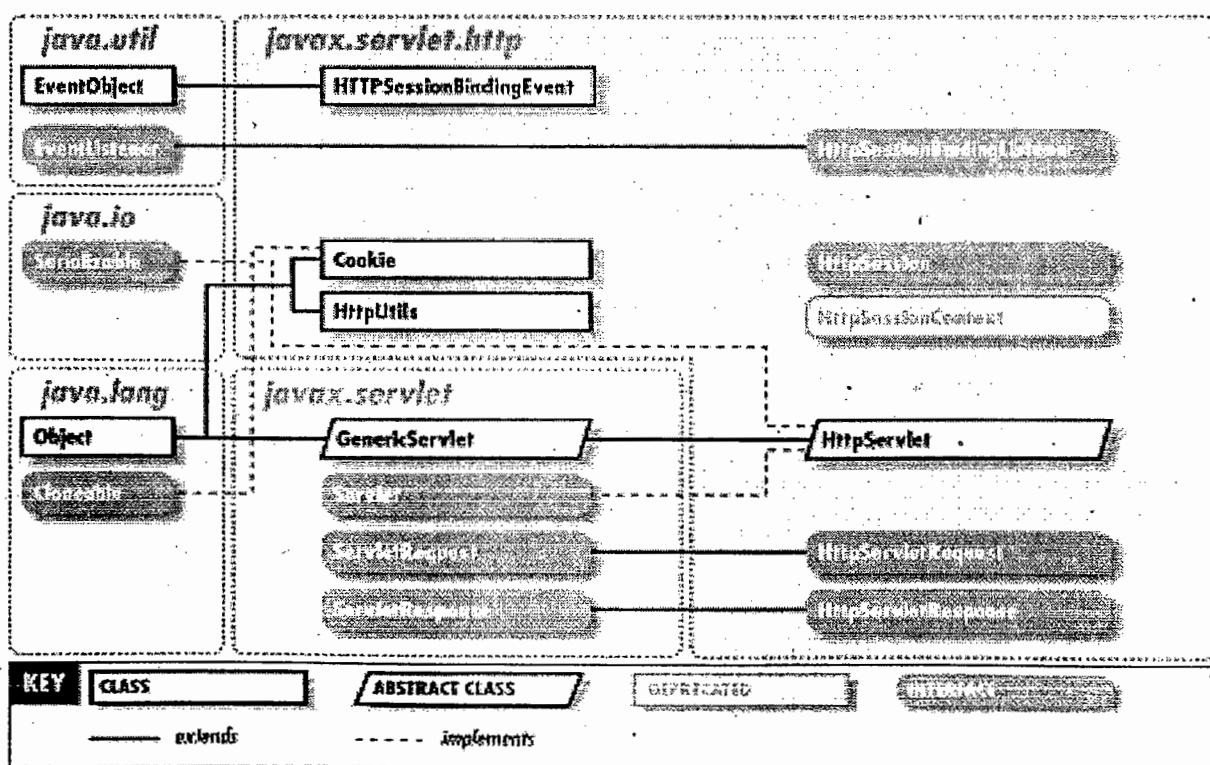
(i.e coming from the client) in the form of `ServletRequest` object and address of the client (from where the request is received) in the form of `ServletResponse` object as arguments.

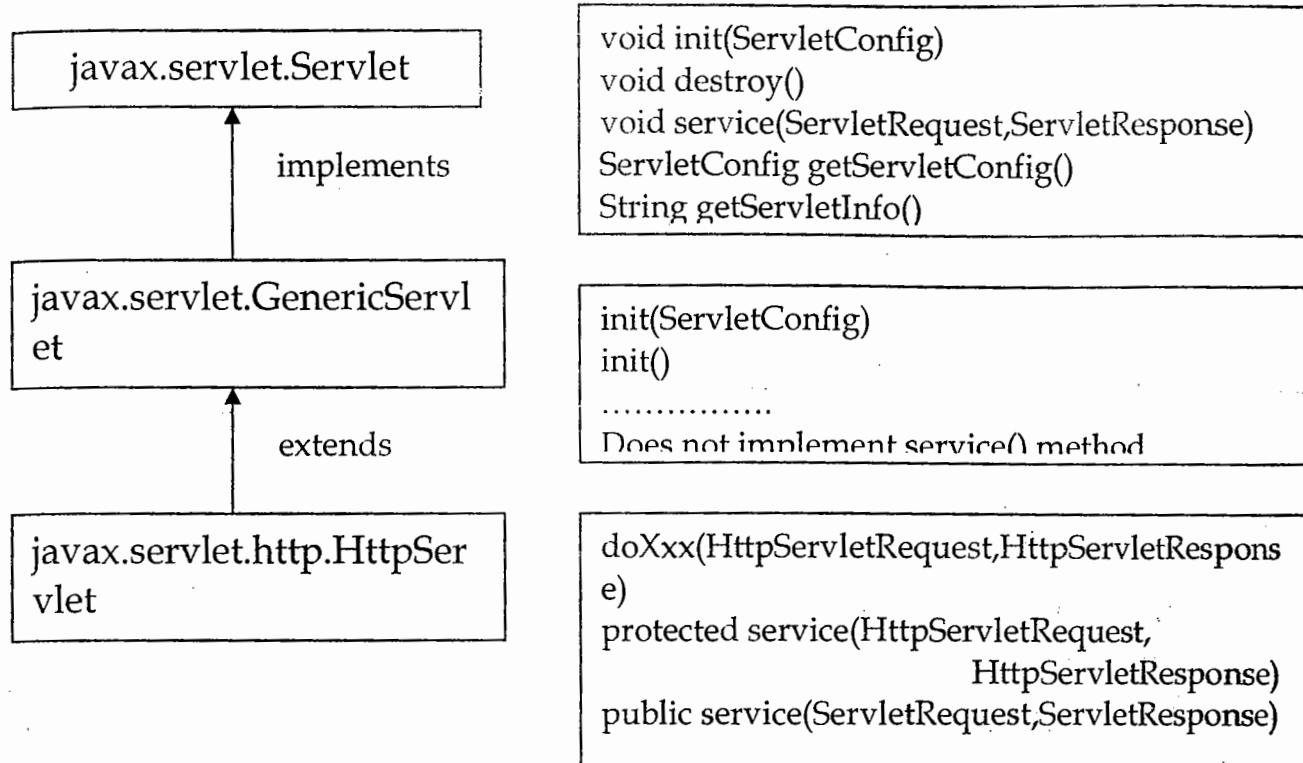
3. `destroy()`:- `destroy()` method would be executed before object of the servlet is deleted. Container maintains every servlet object for a certain period of time even if no request is coming to that servlet. After certain period of time container deletes object of the servlet before deleting or destroying 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
    .....
}

//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 Testsrv extends GenericServlet/HttpServlet

{

ServletConfig cg;

 public void init(ServletConfig cg)

{

 this.cg = cg; //(**5**) explicit initialization logic of ServletConfig object

 //our servlet program related initialiaztin 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 (Our servlet 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.



If you place both init(),inti(-) methods in our servlet program which init method will be exectued?



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 provide in javax.servlet.GenericServlet class. Generally, it is suggested that init(javax.servlet.ServletConfig) method present in javax.servlet.GenericSevlet class should be given a chance to get excecuted, 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 llogic of servlet program like creating JDBC connection object. He always keeps this logic in init() Method even thoug 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.

- GET (default)
- POST
- HEAD
- PUT
- DELETE
- TRACE
- 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.

Understanding Http

javax.servlet.GenericServlet

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

- ✓ GenericServlet defines a generic, protocol-independent servlet.
- ✓ GenericServlet gives a blueprint and makes writing servlet easier.
- ✓ GenericServlet provides simple versions of the lifecycle methods init and destroy and of the methods in the ServletConfig interface.
- ✓ GenericServlet implements the log method, declared in the ServletContext interface.
- ✓ 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

- ✓ HttpServlet defines a HTTP protocol specific servlet.
- ✓ 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:

- ✓ doGet, if the servlet supports HTTP GET requests
- ✓ doPost, for HTTP POST requests
- ✓ doPut, for HTTP PUT requests
- ✓ doDelete, for HTTP DELETE requests
- ✓ init and destroy, to manage resources that are held for the life of the servlet
- ✓ 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	doDelete(HttpServletRequest req, HttpServletResponse resp) Called by the server (via the service method) to allow a servlet to handle a DELETE request.
protected void	doGet(HttpServletRequest req, HttpServletResponse resp) Called by the server (via the service method) to allow a servlet to handle a GET request.
protected void	doHead(HttpServletRequest req, HttpServletResponse resp) Receives an HTTP HEAD request from the protected service method and handles the request.
protected void	doOptions(HttpServletRequest req, HttpServletResponse resp) Called by the server (via the service method) to allow a servlet to handle a OPTIONS request.
protected void	doPost(HttpServletRequest req, HttpServletResponse resp) Called by the server (via the service method) to allow a servlet to handle a POST request.

	POST request.
protected void	<u>doPut(HttpServletRequest req, HttpServletResponse resp)</u> Called by the server (via the service method) to allow a servlet to handle a PUT request.
protected void	<u>doTrace(HttpServletRequest req, HttpServletResponse resp)</u> Called by the server (via the service method) to allow a servlet to handle a TRACE request.
protected long	<u>getLastModified(HttpServletRequest req)</u> Returns the time the HttpServletRequest object was last modified, in milliseconds since midnight January 1, 1970 GMT.
protected void	<u>service(HttpServletRequest req, HttpServletResponse resp)</u> Receives standard HTTP requests from the public service method and dispatches them to the doXXX methods defined in this class.
void	<u>service(ServletRequest req, ServletResponse res)</u> Dispatches client requests to the protected service method.

Skeleton of a HTTP Seiylet

```

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 translates 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://www.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)

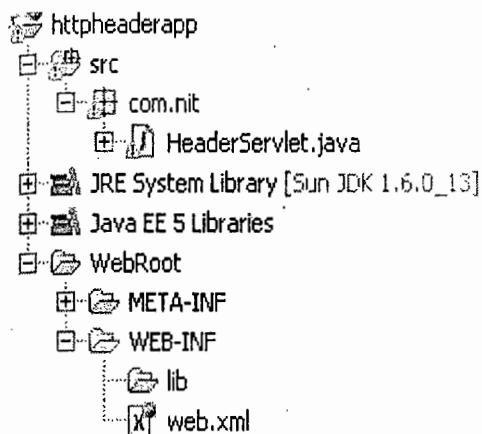
Content-Length: 230

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 **HttpRequest** 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 **getHeaderNames()** method of the request object which will return the Enumeration of the headers. Now to retrieve all the headers from the Enumeration use the **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

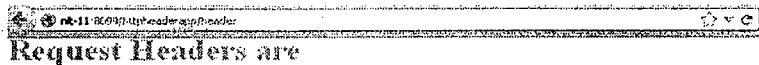
```

<?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

: nit-11:8099

user-agent

: Mozilla/5.0 (Windows NT 5.1; rv:23.0) Gecko/20100101 Firefox/23.0

accept

: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

accept-language

: en-US,en;q=0.5

accept-encoding

: gzip, deflate

connection

: keep-alive

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

1xx	Informational
<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
122	URI is longer than a maximum of 2083 characters
2xx	Success
<u>200</u>	standard response for successful HTTP requests
<u>201</u>	request has been fulfilled, new resource created
<u>202</u>	
<u>203</u>	
<u>204</u>	request accepted, processing pending

<u>205</u>	request processed, information may be from another source
<u>206</u>	request processed, no content returned
<u>207</u>	request processed, no content returned, reset document view
<u>208</u>	partial resource return due to request header
<u>226</u>	XML, can contain multiple separate responses
<u>3xx</u>	results previously returned
<u>300</u>	request fulfilled, response is instance-manipulations
<u>302</u>	Redirection
<u>303</u>	multiple options for the resource delivered this and all future requests directed to the given URI
<u>304</u>	
<u>305</u>	temporary response to request found via alternative URI
<u>306</u>	permanent response to request found via alternative URI
<u>307</u>	resource has not been modified since last requested
<u>308</u>	
<u>4xx</u>	content located elsewhere, retrieve from there
<u>400</u>	
<u>401</u>	subsequent requests should use the specified proxy
<u>402</u>	connect again to different uri as provided
<u>403</u>	resumable HTTP Requests
<u>404</u>	Client Error
<u>405</u>	request cannot be fulfilled due to bad syntax
<u>406</u>	authentication is possible but has failed
<u>407</u>	payment required, reserved for future use
<u>408</u>	server refuses to respond to request
<u>409</u>	requested resource could not be found
<u>410</u>	request method not supported by that resource
<u>411</u>	content not acceptable according to the Accept headers
<u>412</u>	client must first authenticate itself with the proxy

<u>413</u>	server timed out waiting for the request
<u>414</u>	request could not be processed because of conflict
<u>415</u>	resource is no longer available and will not be available again
<u>416</u>	request did not specify the length of its content
<u>417</u>	server does not meet request preconditions
<u>418</u>	request is larger than the server is willing or able to process
<u>422</u>	URI provided was too long for the server to process
<u>423</u>	server does not support media type
<u>424</u>	client has asked for unprovable portion of the file
<u>426</u>	server cannot meet requirements of Expect request-header field
<u>428</u>	I'm a teapot
<u>429</u>	Twitter rate limiting
<u>431</u>	request unable to be followed due to semantic errors
<u>444</u>	resource that is being accessed is locked
<u>449</u>	request failed due to failure of a previous request
<u>450</u>	client should switch to a different protocol
<u>451</u>	origin server requires the request to be conditional
5xx	user has sent too many requests in a given amount of time
<u>500</u>	server is unwilling to process the request
<u>501</u>	server returns no information and closes the connection
<u>502</u>	request should be retried after performing action
<u>503</u>	Windows Parental Controls blocking access to webpage
<u>504</u>	The server cannot reach the client's mailbox
<u>505</u>	connection closed by client while HTTP server is processing
<u>506</u>	
<u>507</u>	

<u>508</u>	Server Error generic error message
<u>509</u>	server does not recognise method or lacks ability to fulfill
<u>510</u>	server received an invalid response from upstream server
<u>511</u>	server is currently unavailable
<u>511</u>	gateway did not receive response from upstream server
<u>598</u>	server does not support the HTTP protocol version
<u>599</u>	content negotiation for the request results in a circular reference
	server is unable to store the representation
	server detected an infinite loop while processing the request
	bandwidth limit exceeded
	further extensions to the request are required
	client needs to authenticate to gain network access
	network read timeout behind the proxy
	network connect timeout behind the proxy

```
1 =====
2 App1(First web 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 public class DateSrv extends GenericServlet
12 {
13     public void service(ServletRequest req,ServletResponse res) throws
14         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-app>
30     <servlet>
31         <servlet-name>abc</servlet-name>
32         <servlet-class>com.nt.DateSrv</servlet-class>
33     </servlet>
34     <servlet-mapping>
35         <servlet-name>abc</servlet-name>
36         <url-pattern>/test1</url-pattern>
37     </servlet-mapping>
38 </web-app>
39
40 -----
41 App2 (webapplication having multipe servlet prgs)
42 =====
43 ----- HtmlSrv.java -----
44 //HtmlSrv.java
45 package com.nt;
46 import javax.servlet.*;
47 import javax.servlet.http.*;
48 import java.io.*;
49
50 public class HtmlSrv extends HttpServlet
51 {
52
53     public void service(HttpServletRequest req,HttpServletResponse res) throws
54         ServletException,IOException
55     {
56         // getPrintWriter obj
57         PrintWriter pw=res.getWriter();
58         // set response content type
59         res.setContentType("text/html");
60
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         //close stream obj.  
68         pw.close();  
69     }//service(-,-)  
70 } //class  
71 -----WordSrv.java-----  
72 //WordSrv.java  
73 package com.nt;  
74 import javax.servlet.*;  
75 import javax.servlet.http.*;  
76 import java.io.*;  
77  
78  
79 public class WordSrv extends HttpServlet  
80 {  
81  
82     public void service(HttpServletRequest req,HttpServletResponse res) throws  
ServletException,IOException  
83     {  
84         // getPrintWriter obj  
85         PrintWriter pw = res.getWriter();  
86         // set response content type  
87         res.setContentType("application/msword");  
88  
89         //write logic to generate output(webpage)  
90         pw.println("<table border='0'>");  
91         pw.println("<tr><th>Player</th><th>Role</th></tr>");  
92         pw.println("<tr><td>Dhoni</td><td>Captain</td></tr>");  
93         pw.println("<tr><td>Sachin</td><td>All Rounder</td></tr>");  
94         pw.println("</table>");  
95  
96         //close stream obj  
97         pw.close();  
98     }//service(-,-)  
99 } //class  
100 -----XmlSrv.java-----  
101 package com.nt;  
102 import javax.servlet.*;  
103 import javax.servlet.http.*;  
104 import java.io.*;  
105  
106 public class XmlSrv extends HttpServlet  
107 {  
108  
109     public void service(HttpServletRequest req,HttpServletResponse res) throws  
ServletException,IOException  
110     {  
111         // getPrintWriter obj  
112         PrintWriter pw = res.getWriter();  
113         // set response content type  
114         res.setContentType("text/xml");  
115  
116         //write logic to generate output(webpage)  
117         pw.println("<table border='0'>");  
118         pw.println("<tr><th>Player</th><th>Role</th></tr>");  
119         pw.println("<tr><td>Dhoni</td><td>Captain</td></tr>");  
120         pw.println("<tr><td>Sachin</td><td>All Rounder</td></tr>");  
121         pw.println("</table>");  
122  
123         //close stream obj  
124         pw.close();  
125     }//service(-,-)  
126 } //class  
127 -----ExcelSrv.java-----  
128 //ExcelSrv.java
```

```
129 package com.nt;
130 import javax.servlet.*;
131 import javax.servlet.http.*;
132 import java.io.*;
133
134 public class ExcelSrv extends HttpServlet
135 {
136
137     public void service(HttpServletRequest req,HttpServletResponse res) throws
138         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 -----
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>
```

```
194     <url-pattern>/xmlurl</url-pattern>
195   </servlet-mapping>
196
197 </web-app>
198 -----
199 =====
200 App3(Html-Servlet Communication using hyperlinks)
201 =====
202 -----page.html-----
203 <center><h1> Welcome </h1></center> <br>
204 <a href="http://localhost:3030/WishApp/wurl">
205   Generate Wish Msg
206 </a>
207 -----web.xml-----
208 <web-app>
209   <servlet>
210     <servlet-name>wish</servlet-name>
211     <servlet-class>com.nt.WishSrv</servlet-class>
212   </servlet>
213   <servlet-mapping>
214     <servlet-name>wish</servlet-name>
215     <url-pattern>/wurl</url-pattern>
216   </servlet-mapping>
217 </web-app>
218 -----WishSrv.java-----
219 //WishSrv.java
220 package com.nt;
221 import javax.servlet.*;
222 import javax.servlet.http.*;
223 import java.io.*;
224 import java.util.*;
225
226 public class WishSrv extends HttpServlet
227 {
228   public void service(HttpServletRequest req,HttpServletResponse res) throws
229   ServletException,IOException
230   {
231     // get PrintWriter
232     PrintWriter pw=res.getWriter();
233     // set content Type
234     res.setContentType("text/html");
235     //get System Date and Time
236     Calendar cl=Calendar.getInstance();
237     //get current hour of the day
238     int h=cl.get(Calendar.HOUR_OF_DAY);
239     // generate wish message
240     if(h<=12)
241       pw.println("<h1><font color='yellow'> Good morning </font></h1>");
242     else if(h<=16)
243       pw.println("<h1><font color='red'> Good Afternoon </font></h1>");
244     else if(h<=20)
245       pw.println("<h1><font color='green'> Good Evening </font></h1>");
246     else
247       pw.println("<h1><font color='black'> Good Night </font></h1>");
248
249     //add hyperlink
250     pw.println("<a href='http://localhost:3030/WishApp/page.html'>home</a>");
251
252     //close stream
253     pw.close();
254   } //service(-,-)
255 } //class
256 //javac -d . WishSrv.java
257 =====
258 App4 (Html to Servlet Communication using form page)
259 =====
```

```

259 -----input.html-----
260 <form action="vturl" method="get">
261     Name : <input type="text" name="pname"><br>
262     Age: <input type="text" name="page"><br>
263     <input type="hidden" name="vflag" value="no"/>
264     <input type="submit" value="Check Voting Eligibility">
265 </form>
266 -----VoterSrv.java-----
267 //VoterSrv.java
268 package com.nt;
269 import javax.servlet.*;
270 import javax.servlet.http.*;
271 import java.io.*;
272
273 public class VoterSrv extends HttpServlet
274 {
275     protected void process(HttpServletRequest req,HttpServletResponse res)
276             throws ServletException,IOException
277     {
278         System.out.println("VoterSrv:process(-,-)");
279         //general settings
280         PrintWriter pw=res.getWriter();
281         res.setContentType("text/html");
282         //read form data given by form page
283         String name=req.getParameter("pname");
284         String tage=req.getParameter("page").trim();
285         int age=Integer.parseInt(tage);
286
287         //write processing logic (B.logic)
288         if(age>=18)
289             pw.println("<h1><font color='cyan'>" +name +" u r eligible to vote");
290         else
291             pw.println("<h1><font color='brown'>" +name +" u r not eligible to vote");
292
293         //add graphical link
294         pw.println("<a href='input.html'><img src='tips.gif'></a>");
295         //close stream
296         pw.close();
297     }//doGet(-,-)
298
299     protected void doGet(HttpServletRequest req,HttpServletResponse res)
300             throws ServletException,IOException{
301         System.out.println("VoterSrv:doGet(-,-)");
302         process(req,res);
303     }//doPost(-,-)
304
305
306     protected void doPost(HttpServletRequest req,HttpServletResponse res)
307             throws ServletException,IOException{
308         System.out.println("VoterSrv:doPost(-,-)");
309         process(req,res);
310     }//doPost(-,-)
311
312 }//class
313 //javac -d . VoterSrv.java
314 -----web.xml-----
-----  

315 <web-app>
316     <servlet>
317         <servlet-name>abc</servlet-name>
318         <servlet-class>com.nt.VoterSrv</servlet-class>
319     </servlet>
320     <servlet-mapping>
321         <servlet-name>abc</servlet-name>
322         <url-pattern>/vturl</url-pattern>
323     </servlet-mapping>
```

```

324 <welcome-file-list>
325   <welcome-file>input.html</welcome-file>
326 </welcome-file-list>
327 </web-app>
328 =====
329 App5 (web application having both Client side and Server Side form validations)
330 =====
331 -----input.html-----
332 <h1 style="color:red"><center> Check Voting Eligibility </center></h1>
333 <script language="JavaScript" src="Validation.js">
334   </script>
335 <form action="vturl" method="get" onsubmit="return validate(this)">
336   Name : <input type="text" name="pname"> <span id="nameErr"></span><br>
337   Age: <input type="text" name="page"><span id="ageErr"></span><br>
338     <input type="hidden" name="vflag" value="no"/>
339     <input type="submit" value="Check Voting Eligibility">
340 </form>
341 -----Validation.js-----
342 function validate(frm){
343   document.getElementById("nameErr").innerHTML="";
344   document.getElementById("ageErr").innerHTML="";
345   //read form data
346   var name=frm.pname.value;
347   var age=frm.page.value;
348   // change value of hidden box to "yes" indicating Client side form validations are done
349   frm.vflag.value="yes";
350   alert("Client side form validations .....");
351   // Write Client side form validation logic
352   if(name==""){ //required rule
353     document.getElementById("nameErr").innerHTML="Person name is mandatory";
354     document.getElementById("nameErr").style.color="red";
355     frm.pname.focus();
356     return false;
357   }
358   if(age==""){ //required rule
359     document.getElementById("ageErr").innerHTML="Person age is mandatory";
360     document.getElementById("nameErr").style.color="red";
361     frm.page.focus();
362     return false;
363   }
364   else{ // check weather age is numeric value
365     if(isNaN(age)){
366       document.getElementById("ageErr").innerHTML="Person age must numerice
367       value";
368       document.getElementById("ageErr").style.color="red";
369       frm.page.focus();
370       frm.page.value="";
371       return false;
372     }
373     //else
374     return true;
375   } //function
376 -----VoterSrv.java-----
377 -----
378 //VoterSrv.java
379 package com.nt;
380 import javax.servlet.*;
381 import javax.servlet.http.*;
382 import java.io.*;
383
384 public class VoterSrv extends HttpServlet
385 {
386   protected void process(HttpServletRequest req,HttpServletResponse res)
387     throws ServletException,IOException
388   {
389     System.out.println("VoterSrv:process(-,-)");

```

```

388         //general settings
389         PrintWriter pw=res.getWriter();
390         res.setContentType("text/html");
391         //read form data given by form page
392         String name=req.getParameter("pname");
393         String tage=req.getParameter("page").trim();
394         int age=Integer.parseInt(tage);
395
396         //write processing logic (B.logic)
397         if(age>=18)
398             pw.println("<h1><font color='cyan'>" +name +" u r eligible to vote");
399         else
400             pw.println("<h1><font color='brown'>" +name +" u r not eligible to vote");
401
402         //add graphical link
403         pw.println("<a href='input.html'><img src='tips.gif'></a>");
404         //close stream
405         pw.close();
406     } //doGet(-,-)
407
408     protected void doGet(HttpServletRequest req,HttpServletResponse res)
409         throws ServletException,IOException{
410         System.out.println("VoterSrv:doGet(-,-)");
411         process(req,res);
412     } //doPost(-,-)
413
414
415     protected void doPost(HttpServletRequest req,HttpServletResponse res)
416         throws ServletException,IOException{
417         System.out.println("VoterSrv:doPost(-,-)");
418         process(req,res);
419     } //doPost(-,-)
420
421 } //class
422 //javac -d . VoterSrv.java
423 -----web.xml-----
424 <web-app>
425   <servlet>
426     <servlet-name>abc</servlet-name>
427     <servlet-class>com.nt.VoterSrv</servlet-class>
428   </servlet>
429   <servlet-mapping>
430     <servlet-name>abc</servlet-name>
431     <url-pattern>/vturl</url-pattern>
432   </servlet-mapping>
433   <welcome-file-list>
434     <welcome-file>input.html</welcome-file>
435   </welcome-file-list>
436 </web-app>
437 =====
438 App6(Working with diff types form comps)
439 =====
440 -----Form.html-----
441 <h1><center> Register u r Details <h1></center>
442 <form action="formurl">
443   <table border="1">
444     <tr>
445       <td>Name</td>
446       <td><input type="text" name="tname" value="" /></td>
447     </tr>
448     <tr>
449       <td>Age</td>
450       <td><input type="password" name="tage" value="" /> </td>
451     </tr>
452     <tr>
```

```

453           <td>Gender</td>
454           <td>
455               <input type="radio" name="gen" value="M" checked="checked" /> Male
456               <input type="radio" name="gen" value="F" />FeMale
457           </td>
458       </tr>
459       <tr>
460           <td>Address: </td>
461           <td>
462               <textarea name="taddress" rows="4" cols="20">
463                   enter address
464               </textarea>
465           </td>
466       </tr>
467       <tr>
468           <td>Marital Status</td>
469           <td><input type="checkbox" name="ms" value="married" /> Married</td>
470       </tr>
471       <tr>
472           <td>Qualification</td>
473           <td>
474               <select name="qlfy">
475                   <option value="B.Tech">Engg</option>
476                   <option value="MBBS">Medical</option>
477                   <option value="B.A">Arts</option>
478               </select>
479           </td>
480       </tr>
481       <tr>
482           <td>Courses:</td>
483           <td><select name="crs" size="3" multiple="multiple">
484               <option value="java">JAVA pkg</option>
485               <option value=".net">.NET pkg</option>
486               <option value="testing">Testing pkg</option>
487           </select>
488           </td>
489       </tr>
490       <tr>
491           <td>Hobies</td>
492           <td>
493               <input type="checkbox" name="hb" value="read" checked/> Reading
494               <input type="checkbox" name="hb" value="stamps"/> Stamp Collection
495               <input type="checkbox" name="hb" value="roaming" /> Travelling
496           </td>
497       </tr>
498       <tr>
499           <td colspan="2">
500               <input type="submit" value="submit" />
501               <input type="reset" value="cancel" />
502           </td>
503       </tr>
504   </table>
505 </form>
506 -----web.xml-----
507 <servlet>
508     <servlet-name>abc</servlet-name>
509     <servlet-class>com.nt.FormSrv</servlet-class>
510 </servlet>
511 <servlet-mapping>
512     <servlet-name>abc</servlet-name>
513     <url-pattern>/formurl</url-pattern>
514 </servlet-mapping>
515 <welcome-file-list>
516     <welcome-file>Form.html</welcome-file>
517     </welcome-file-list>
518 </web-app>

```

```
519 -----FormSrv.java-----
520 package com.nt;
521 import java.io.IOException;
522 import java.io.PrintWriter;
523 import javax.servlet.ServletException;
524 import javax.servlet.http.HttpServlet;
525 import javax.servlet.http.HttpServletRequest;
526 import javax.servlet.http.HttpServletResponse;
527
528 public class FormSrv extends HttpServlet {
529
530     protected void processRequest(HttpServletRequest request, HttpServletResponse response)
531     throws ServletException, IOException {
532         //get PrintWriter obj
533         PrintWriter pw=response.getWriter();
534         // set response content type
535         response.setContentType("text/html");
536         // read from data
537         String name=request.getParameter("tname");
538         int age=Integer.parseInt(request.getParameter("tage"));
539         String gender=request.getParameter("gen");
540         String ms=request.getParameter("ms");
541         String addrs=request.getParameter("taddress");
542         String qlfy=request.getParameter("qlfy");
543         String crs[]=request.getParameterValues("crs");
544         String hb[]=request.getParameterValues("hb");
545         // write request processing logic
546         if(gender.equalsIgnoreCase("M"))
547         {
548             if(age<=5)
549                 pw.println(name+" u r baby boy");
550             else if (age<=12)
551                 pw.println(name+" u r smalll boy");
552             else if(age<=19)
553                 pw.println(name+" u r teenage boy");
554             else if(age<=35)
555                 pw.println(name+" u r young man ");
556             else if(age<=50)
557                 pw.println(name+" u r middle aged man ");
558             else
559                 pw.println(name+" u r Budda");
560         }//if
561         else if(gender.equalsIgnoreCase("F"))
562         {
563             if(age<=5)
564                 pw.println(name+" u r baby girl");
565             else if (age<=12)
566                 pw.println(name+" u r small girl");
567             else if(age<=19)
568                 pw.println(name+" u r teenage girl");
569             else if(age<=35)
570                 pw.println(name+" u r young woman ");
571             else if(age<=50)
572                 pw.println(name+" u r middle aged woman ");
573             else
574                 pw.println(name+" u r old woman");
575         }
576
577         pw.println("<br>name="+name);
578         pw.println("<br>age="+age);
579         pw.println("<br>Gender="+gender);
580         pw.println("<br>Marital Status="+ms);
581         pw.println("<br>Address="+addrs);
582         pw.println("<br> Qualification="+qlfy);
583         pw.println("<br> Courses");
584         if(crs!=null)
```

```
585     {
586         for(int i=0;i<crs.length;++i)
587             pw.println(crs[i]+"...");
588     }
589
590     pw.println("<br> Hobies");
591     if(hb!=null)
592     {
593         for(int i=0;i<hb.length;++i)
594             pw.println(hb[i]+"...");
595     }
596 }
597
598
599 protected void doGet(HttpServletRequest request, HttpServletResponse response)
600 throws ServletException, IOException {
601     processRequest(request, response);
602 }
603
604 protected void doPost(HttpServletRequest request, HttpServletResponse response)
605 throws ServletException, IOException {
606     processRequest(request, response);
607 }
608 }
=====
610 App7) (Html -Servlet communication uisng java script)
611 =====
612 -----Form.html-----
613 <form name="frm" action="cps" method="get">
614     check prime or not? : <input type="text" name="t1" onblur="frm.submit()"><br>
615 </form>
616 -----CheckPrimeSrv.java-----
617 package com.nt;
618
619 import java.io.IOException;
620 import java.io.PrintWriter;
621 import javax.servlet.ServletException;
622 import javax.servlet.ServletOutputStream;
623 import javax.servlet.http.HttpServlet;
624 import javax.servlet.http.HttpServletRequest;
625 import javax.servlet.http.HttpServletResponse;
626
627 public class CheckPrimeSrv extends HttpServlet {
628
629     protected void processRequest(HttpServletRequest req, HttpServletResponse res)
630         throws ServletException, IOException {
631
632         // General settings
633         // PrintWriter pw=res.getWriter();
634         ServletOutputStream sos=res.getOutputStream();
635         ServletOutputStream sos1=res.getOutputStream();
636
637         res.setContentType("text/html");
638         // read form data
639         int value=Integer.parseInt(req.getParameter("t1"));
640         //check weather given value is primeNumber or not
641         boolean flag=false;
642         for(int i=2;i<value;++i){
643             if(value%i==0){
644                 flag=true;
645                 break;
646             } //if
647         } //for
648
649         if(flag==true)
650             sos.println(value+" is not a prime number");
```

```

651     else
652         sos.println(value+" is prime number");
653
654     //close steam
655     sos.close();
656 }
657
658 @Override
659 protected void doGet(HttpServletRequest request, HttpServletResponse response)
660     throws ServletException, IOException {
661     processRequest(request, response);
662 }
663
664 @Override
665 protected void doPost(HttpServletRequest request, HttpServletResponse response)
666     throws ServletException, IOException {
667     processRequest(request, response);
668 }
669
670 }//class
-----web.xml-----
671 <web-app>
672   <servlet>
673     <servlet-name>check</servlet-name>
674     <servlet-class>com.nt.CheckPrimeSrv</servlet-class>
675   </servlet>
676
677   <servlet-mapping>
678     <servlet-name>check</servlet-name>
679     <url-pattern>/cps</url-pattern>
680   </servlet-mapping>
681
682     <welcome-file-list>
683       <welcome-file>Form.html</welcome-file>
684     </welcome-file-list>
685   </web-app>
686 =====
687 App8) (Working with multiple hyperlinks)
688 =====
-----links.html-----
689 <a href="linkurl?s1=link1">All Countries </a>
690 <br>
691 <br>
692 <a href="linkurl?s1=link2">All Languages </a>
693 <br>
694 <br>
695 <a href="linkurl?s1=link3">Sys Date </a>
696 -----LinksSrv.java-----
697 package com.nt;
698
700
701 import java.io.IOException;
702 import java.io.PrintWriter;
703 import java.util.Arrays;
704 import java.util.Date;
705 import java.util.Locale;
706 import javax.servlet.ServletException;
707 import javax.servlet.http.HttpServlet;
708 import javax.servlet.http.HttpServletRequest;
709 import javax.servlet.http.HttpServletResponse;
710
711 public class LinksSrv extends HttpServlet {
712     @Override
713     protected void doGet(HttpServletRequest req, HttpServletResponse res)
714         throws ServletException, IOException {
715         // General settings
716         PrintWriter pw=res.getWriter();

```

```

717     res.setContentType("text/html");
718     //read additional req param (s1) value
719     String pval=req.getParameter("s1");
720
721     if(pval.equals("link1")){
722         Locale locale[] = Locale.getAvailableLocales();
723         for(Locale loc:locale){
724             pw.println(loc.getDisplayCountry()+" ");
725         }
726     }
727     else if(pval.equals("link2")){
728         Locale locale[] = Locale.getAvailableLocales();
729         for(Locale loc:locale){
730             pw.println(loc.getDisplayLanguage()+" ");
731         }
732     }
733     else
734     {
735         pw.println("Date and Time"+new Date());
736     }
737 } //doGet(-,-)
738
739 @Override
740 protected void doPost(HttpServletRequest req, HttpServletResponse res)
741     throws ServletException, IOException {
742     doGet(req,res);
743 }
744
745 }
746 -----web.xml-----
747 <web-app>
748   <servlet>
749     <servlet-name>links</servlet-name>
750     <servlet-class>com.nt.LinksSrv</servlet-class>
751   </servlet>
752
753   <servlet-mapping>
754     <servlet-name>links</servlet-name>
755     <url-pattern>/linkurl</url-pattern>
756   </servlet-mapping>
757
758   <welcome-file-list>
759     <welcome-file>links.html</welcome-file>
760   </welcome-file-list>
761 </web-app>
762
763 =====
764 App9 >>>> (Handling multiple submit Buttons and hyperlinks )
765 =====
766 -----Form.html-----
767 <form action="purl">
768   Value 1 : <input type="text" name="t1"><br></br>
769   Value 2 : <input type="text" name="t2"><br></br>
770   <input type="submit" name="s1" value="add">
771   <input type="submit" name="s1" value="sub">
772   <input type="submit" name="s1" value="mul">
773   <br><br>
774   <a href="purl?s1=link1"> System properties </a> &ampnbsp &ampnbsp
775   <a href="purl?s1=link2"> Date and Time </a> &ampnbsp &ampnbsp
776 </form>
777 -----ProcessSrv.java-----
778 package com.nt;
779 import java.io.IOException;
780 import java.io.PrintWriter;
781 import java.util.Date;
782 import javax.servlet.ServletException;

```

```
783 import javax.servlet.http.HttpServlet;
784 import javax.servlet.http.HttpServletRequest;
785 import javax.servlet.http.HttpServletResponse;
786
787 public class ProcessSrv extends HttpServlet {
788
789     protected void processRequest(HttpServletRequest req, HttpServletResponse res) throws
790     ServletException, IOException
791     {
792         //General settings
793         PrintWriter pw = res.getWriter();
794         res.setContentType("text/html");
795
796         //read s1 req param value
797         String pval = req.getParameter("s1");
798         if(pval.equals("link1")){
799             pw.println("Sys Properties=====>"+System.getProperties());
800         }
801         else if(pval.equals("link2")){
802             pw.println("Date and Time : "+new Date());
803         }
804         else if(pval.equals("add")){
805             int v1=Integer.parseInt(req.getParameter("t1"));
806             int v2=Integer.parseInt(req.getParameter("t2"));
807             pw.println("Sum: "+(v1+v2));
808         }
809         else if(pval.equals("sub")){
810             int v1=Integer.parseInt(req.getParameter("t1"));
811             int v2=Integer.parseInt(req.getParameter("t2"));
812             pw.println("Sub: "+(v1-v2));
813         }
814         else{
815             int v1=Integer.parseInt(req.getParameter("t1"));
816             int v2=Integer.parseInt(req.getParameter("t2"));
817             pw.println("Mul: "+(v1*v2));
818         }
819         //close stream
820         pw.close();
821     }//processRequest(-,-)
822
823     protected void doGet(HttpServletRequest request, HttpServletResponse response)
824         throws ServletException, IOException {
825         processRequest(request, response);
826     }
827
828     protected void doPost(HttpServletRequest request, HttpServletResponse response)
829         throws ServletException, IOException {
830         processRequest(request, response);
831     }
832 } //class
-----web.xml-----
-----
833 <web-app>
834     <servlet>
835         <servlet-name>process</servlet-name>
836         <servlet-class>com.nt.ProcessSrv</servlet-class>
837     </servlet>
838
839     <servlet-mapping>
840         <servlet-name>process</servlet-name>
841         <url-pattern>/purl</url-pattern>
842     </servlet-mapping>
843
844     <welcome-file-list>
845         <welcome-file>Form.html</welcome-file>
846     </welcome-file-list>
```

```
847 </web-app>
848 =====
849 App10>>>>>>>>>Example App on servlet to DB s/w communication>>>>>>..
850 =====
851 -----input.html-----
852 <h1><center> Get Emp Details </center></h1>
853 <form action="dburl" method="get">
854     Employee no: <input type="text" name="teno"/><br>
855     <input type="submit" value="GetDetails">
856 </form>
857 -----web.xml-----
858 <web-app>
859     <servlet>
860         <servlet-name>abc</servlet-name>
861         <servlet-class>com.nt.DBSrv</servlet-class>
862         <load-on-startup>1</load-on-startup>
863     </servlet>
864
865     <servlet-mapping>
866         <servlet-name>abc</servlet-name>
867         <url-pattern>/dburl</url-pattern>
868     </servlet-mapping>
869
870     <welcome-file-list>
871         <welcome-file>input.html</welcome-file>
872     </welcome-file-list>
873 </web-app>
874 -----DBSrv.java-----
875 //DBSrv.java
876 package com.nt;
877 import javax.servlet.*;
878 import javax.servlet.http.*;
879 import java.io.*;
880
881 import java.sql.*;
882
883 public class DBSrv extends HttpServlet
884 {
885     private Connection con;
886     private PreparedStatement ps;
887     public void init(){
888         try{
889             //register jdbc driver
890             Class.forName("oracle.jdbc.driver.OracleDriver");
891             // Establish the connection
892             con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","scott","tiger","System","m
893             anager");
894             //create Jdbc PreparedStatement obj
895             ps=con.prepareStatement("select empno,ename,job,sal from emp where empno=?");
896             }//try
897             catch(Exception e)
898             { e.printStackTrace(); }
899         }//init()
900
901         public void doGet(HttpServletRequest req,HttpServletResponse res) throws
902             ServletException,IOException
903         {
904             try{
905                 //General settings
906                 PrintWriter pw=res.getWriter();
907                 res.setContentType("text/html");
908                 // read form data
909                 int no=Integer.parseInt(req.getParameter("teno"));
910                 //set param value to SQL Query
911                 ps.setInt(1,no);
912                 // execute the SQL Query
```

```

911     ResultSet rs=ps.executeQuery();
912     //process the ResultSet
913     if(rs.next())
914     {
915         pw.println("<br> Employee no"+rs.getInt(1));
916         pw.println("<br> Employee name"+rs.getString(2));
917         pw.println("<br> Employee Desg :" +rs.getString(3));
918         pw.println("<br> Employee Salary:" +rs.getFloat(4));
919     }//if
920     else
921     {
922         pw.println("<br> No Employee Found ");
923     }
924     }//try
925     catch(Exception e)
926     { e.printStackTrace(); }
927 } //doGet(-,-)

928

929 public void doPost(HttpServletRequest req,HttpServletResponse res) throws
ServletException,IOException
930 {
931     doGet(req,res);
932 } //doPost(-,-)

933

934 public void destroy(){
935 //close jdbc objs
936     try{
937         if(ps!=null)
938             ps.close();
939         }//try
940         catch(Exception e)
941         { e.printStackTrace(); }

942     try{
943         if(con!=null)
944             con.close();
945         }//try
946         catch(Exception e)
947         { e.printStackTrace(); }
948     }//destroy()
949 } //class
950 //javac -d . DBSrv.java

951
952
953
954 =====
955 App11)Servlet-DB s/w communication (Using init param values)
956 -----
957 -----input.html-----
958 <h1> <center> Get Emp Details </center></h1>
959 <form action="dburl" method="get">
960     Employee no: <input type="text" name="teno"><br>
961     <input type="submit" value="getEmpDetails">
962 </form>
963 -----web.xml-----
964 <web-app>
965     <servlet>
966         <servlet-name>db</servlet-name>
967         <servlet-class>DBSrv</servlet-class>
968         <init-param>
969             <param-name>driver</param-name>
970             <param-value>oracle.jdbc.driver.OracleDriver</param-value>
971         </init-param>
972
973         <init-param>
974             <param-name>dburl</param-name>
975             <param-value>jdbc:oracle:thin:@localhost:1521:xe</param-value>

```

```

976             </init-param>
977
978         <init-param>
979             <param-name>dbuser</param-name>
980             <param-value>scott</param-value>
981         </init-param>
982
983         <init-param>
984             <param-name>dbpwd</param-name>
985             <param-value>tiger</param-value>
986         </init-param>
987     <load-on-startup>1</load-on-startup>
988   </servlet>
989
990   <servlet-mapping>
991       <servlet-name>db</servlet-name>
992       <url-pattern>/dburl</url-pattern>
993   </servlet-mapping>
994   <welcome-file-list>input.html</welcome-file-list>
995   <welcome-file>
996 </web-app>
997 -----DBSrv.java-----
998 //DBSrv.java (show servlet to DB s/w communication based on approach1)
999 package com.nt;
1000 import javax.servlet.*;
1001 import javax.servlet.http.*;
1002 import java.io.*;
1003 import java.sql.*;
1004
1005 public class DBSrv extends HttpServlet
1006 {
1007     Connection con;
1008     PreparedStatement ps;
1009     public void init()
1010     {
1011         try
1012         {
1013             //get Access to ServletConfig obj
1014             ServletConfig cg=getServletConfig();
1015             // read init param values from web.xml
1016             String s1=cg.getInitParameter("driver");
1017             String s2=cg.getInitParameter("dburl");
1018             String s3=cg.getInitParameter("dbuser");
1019             String s4=cg.getInitParameter("dbpwd");
1020
1021             // create jdbc con obj
1022             Class.forName(s1);
1023             con=DriverManager.getConnection(s2,s3,s4);
1024
1025             // create jdbc PreparedStatement obj.....
1026             ps=con.prepareStatement("select ename,sal from emp where empno=?");
1027         }//try
1028         catch(Exception e)
1029         {
1030             e.printStackTrace();
1031         }
1032     }//init
1033
1034     public void doGet(HttpServletRequest req,HttpServletResponse res) throws
1035     ServletException,IOException
1036     {
1037         try
1038         {
1039             // read form data from form page
1040             int no=Integer.parseInt(req.getParameter("teno"));

```

```

1041          //get PrintWriter obj
1042          PrintWriter pw=res.getWriter();
1043          // set response content type
1044          res.setContentType("text/html");
1045
1046          // write jdbc code.....
1047          //set value to parameter of the qry
1048          ps.setInt(1,no);
1049
1050          // execute the query
1051          ResultSet rs=ps.executeQuery();
1052
1053          //process ResultSet obj and display emp details....
1054          if(rs.next())
1055          {
1056              pw.println("<br><b><i> Emp name is: </i></b>" +rs.getString(1));
1057              pw.println("<br><b><i> Emp Salary is: </i></b>" +rs.getFloat(2));
1058          }//if
1059
1060
1061          //close ResultSet obj
1062          rs.close();
1063
1064          //close stream obj
1065          pw.close();
1066      }//try
1067      catch(Exception e)
1068      {
1069          e.printStackTrace();
1070      }
1071  }//doGet(-,-)
1072  public void doPost(HttpServletRequest req,HttpServletResponse res) throws
ServletException,IOException
1073  {
1074      doGet(req,res);
1075  }
1076
1077  public void destroy()
1078  {
1079      try
1080      {
1081          if(ps!=null)
1082              ps.close();
1083      }
1084      catch(Exception e1)
1085      {
1086          e1.printStackTrace();
1087      }
1088      try
1089      {
1090          if(con!=null)
1091              con.close();
1092      }
1093      catch(Exception e2)
1094      {
1095          e2.printStackTrace();
1096      }
1097  }//destroy()
1098 }//class
1099 =====
1100 App12) Servlet-DB s/w communication (using context param values)
1101 =====
1102 -----input.html-----
1103 <form action="dburl" method="get">
1104     Employee no: <input type="text" name="teno"><br>
1105     <input type="submit" value="getEmpDetails">

```

```
1106 </form>
1107 -----web.xml-----
1108 <web-app>
1109     <context-param>
1110         <param-name>driver</param-name>
1111         <param-value>oracle.jdbc.driver.OracleDriver</param-value>
1112     </context-param>
1113     <context-param>
1114         <param-name>dburl</param-name>
1115         <param-value>jdbc:oracle:thin:@localhost:1521:xe</param-value>
1116     </context-param>
1117     <context-param>
1118         <param-name>dbuser</param-name>
1119         <param-value>scott</param-value>
1120     </context-param>
1121     <context-param>
1122         <param-name>dbpwd</param-name>
1123         <param-value>tiger</param-value>
1124     </context-param>
1125
1126     <servlet>
1127         <servlet-name>db</servlet-name>
1128         <servlet-class>com.nt.DBsrv</servlet-class>
1129         <load-on-startup>1</load-on-startup>
1130     </servlet>
1131
1132     <servlet-mapping>
1133         <servlet-name>db</servlet-name>
1134         <url-pattern>/dburl</url-pattern>
1135     </servlet-mapping>
1136 </web-app>
1137 -----DBsrv.java-----
1138 //DBsrv.java (show servlet to DB s/w communication based on approach1)
1139 package com.nt;
1140 import javax.servlet.*;
1141 import javax.servlet.http.*;
1142 import java.io.*;
1143 import java.sql.*;
1144
1145 public class DBsrv extends HttpServlet
1146 {
1147     Connection con;
1148     PreparedStatement ps;
1149     public void init()
1150     {
1151         try
1152         {
1153             //get Access to ServletContext obj
1154             ServletContext sc=getServletContext();
1155             // read context param values from web.xml
1156             String s1=sc.getInitParameter("driver");
1157             String s2=sc.getInitParameter("dburl");
1158             String s3=sc.getInitParameter("dbuser");
1159             String s4=sc.getInitParameter("dbpwd");
1160
1161             // create jdbc con obj
1162             Class.forName(s1);
1163             con=DriverManager.getConnection(s2,s3,s4);
1164
1165             // create jdbc PreparedStatement obj.....
1166             ps=con.prepareStatement("select ename,sal from emp where empno=?");
1167         } //try
1168         catch(Exception e)
1169         {
1170             e.printStackTrace();
1171         }
1172 }
```

```
1172     }//init
1173
1174     public void doGet(HttpServletRequest req,HttpServletResponse res) throws
1175         ServletException,IOException
1176     {
1177         try
1178         {
1179             // read form data from form page
1180             int no=Integer.parseInt(req.getParameter("teno"));
1181
1182             //get PrintWriter obj
1183             PrintWriter pw=res.getWriter();
1184             // set response content type
1185             res.setContentType("text/html");
1186
1187             // write jdbc code.....
1188             //set value to parameter of the qry
1189             ps.setInt(1,no);
1190
1191             // execute the query
1192             ResultSet rs=ps.executeQuery();
1193
1194             //process ResultSet obj and display emp details....
1195             if(rs.next())
1196             {
1197                 pw.println("<br><b><i> Emp name is: </i></b>" +rs.getString(1));
1198                 pw.println("<br><b><i> Emp Salary is: </i></b>" +rs.getFloat(2));
1199             }
1200
1201             //close ResultSet obj
1202             rs.close();
1203
1204             //close stream obj
1205             pw.close();
1206         } //try
1207         catch(Exception e)
1208         {
1209             e.printStackTrace();
1210         }
1211     } //doGet(-,-)
1212     public void doPost(HttpServletRequest req,HttpServletResponse res) throws
1213         ServletException,IOException
1214     {
1215         doGet(req,res);
1216     }
1217
1218     public void destroy()
1219     {
1220         try
1221         {
1222             if(ps!=null)
1223                 ps.close();
1224         }
1225         catch(Exception e1)
1226         {
1227             e1.printStackTrace();
1228         }
1229         try
1230         {
1231             if(con!=null)
1232                 con.close();
1233         }
1234         catch(Exception e2)
1235         {
1236             e2.printStackTrace();
1237         }
1238     }
1239 }
```

```
1236     }
1237     } //destroy()
1238 } //class
1239 -----
1240 =====
1241 App13) Applet to Servlet Communiation
1242 =====
1243 -----Main.html-----
1244 <frameset rows="30,*">
1245   <frame src="Form.html" name="f1">
1246   <frame name="f2">
1247 </frameset>
1248 -----Form.html-----
1249 <applet code="WishApplet.class" width="400" height="400"/>
1250 -----
1251 -----WishApplet.java-----
1252 //WishApplet.java
1253 package com.nt;
1254 import java.applet.*;
1255 import java.awt.event.*;
1256 import java.awt.*;
1257 import java.net.*;
1258
1259
1260 public class WishApplet extends Applet implements ActionListener
1261 {
1262     Label l1;
1263     TextField tf1;
1264     Button btn;
1265
1266     public WishApplet()
1267     {
1268         setSize(300,400);
1269         setBackground(Color.RED);
1270         //add comps
1271         l1=new Label("name");
1272         add(l1);
1273         tf1=new TextField(20);
1274         add(tf1);
1275         btn=new Button("send");
1276         add(btn);
1277         //register listener
1278         btn.addActionListener(this);
1279         setLayout(new FlowLayout());
1280     }
1281
1282     public void actionPerformed(ActionEvent ae)
1283     {
1284         try
1285         {
1286             //frame request url
1287             String qstr="?prname="+tf1.getText().replace(' ','+');
1288             URL myurl =new URL("http://localhost:3030/AppletServletApp/wurl"+qstr);
1289             // get AppletContext obj
1290             AppletContext apc = getAppletContext();
1291             //send request
1292             apc.showDocument(myurl,"f2");
1293         }
1294         catch(Exception e)
1295         {
1296             e.printStackTrace();
1297         }
1298     } //method
1299 } //class
1300 -----web.xml-----
1301 <web-app>
```

```

1302      <servlet>
1303          <servlet-name>XYZ</servlet-name>
1304          <servlet-class>WishSrv</servlet-class>
1305      </servlet>
1306      <servlet-mapping>
1307          <servlet-name>XYZ</servlet-name>
1308          <url-pattern>/wurl</url-pattern>
1309      </servlet-mapping>
1310      <welcome-file-list>
1311          <welcome-file>Main.html</welcome-file>
1312      </welcome-file-list>
1313  </web-app>
1314  -----WishSrv.java-----
1315 //WishSrv.java
1316 package com.nt;
1317 import javax.servlet.*;
1318 import java.io.*;
1319 import javax.servlet.http.*;
1320 import java.util.*;
1321
1322 public class WishSrv extends HttpServlet
1323 {
1324     public void doGet(HttpServletRequest req,HttpServletResponse res) throws
1325         ServletException,IOException
1326     {
1327         // get PrintWriter
1328         PrintWriter pw=res.getWriter();
1329         // set content type
1330         res.setContentType("text/html");
1331         //read form data (applet data)
1332         String name=req.getParameter("prname");
1333         // process request and geneate response
1334         Calendar cl=Calendar.getInstance(); //gives current date and time
1335         int h=cl.get(Calendar.HOUR_OF_DAY);
1336         if(h<=12)
1337             pw.println("<b>Good morning:</b>"+name);
1338         else if(h<=16)
1339             pw.println("<b>Good Afternoon:</b>"+name);
1340         else if(h<=20)
1341             pw.println("<b>Good Evening:</b>"+name);
1342         else
1343             pw.println("<b> Good Night </b>");
1344
1345         //close stream obj
1346         pw.close();
1347     } //doGet(-,-)
1348     public void doPost(HttpServletRequest req,HttpServletResponse res) throws
1349         ServletException,IOException
1350     {
1351         doGet(req,res);
1352     }
1353 } //class
1354 =====
1355 App14(Servlet-DB Communication with the support of Server Managed JDBCConnectionPool)
1356 -----
1357 <html>
1358     <head><title>Displays the table from database</title></head>
1359     <body bgcolor="#fffffa9">
1360         <form action="poolurl" method="GET">
1361             <center>
1362                 <h2>Display Table</h2>
1363                 <br><br><br>
1364                 <table>
1365                     <tr>

```

```
1366 <td><b>Enter the table name</b>&nbsp;&nbsp;</td>
1367 <td><input type="text" name="table"></td>
1368 </tr>
1369 <tr>
1370 <td align="right">
1371 <input type=submit value="Display">&nbsp;&nbsp;
1372 </td>
1373 <td>
1374 <input type=reset value=cancel>
1375 </td>
1376 </tr>
1377 </table>
1378 </center>
1379 </form>
1380 </body>
1381 </html>
-----web.xml-----
-->
1383 <web-app>
1384     <welcome-file-list>
1385         <welcome-file>index.htm</welcome-file>
1386     </welcome-file-list>
1387     <servlet>
1388         <servlet-name>conPool</servlet-name>
1389         <servlet-class>com.nt.ConnPoolServlet</servlet-class>
1390     </servlet>
1391     <servlet-mapping>
1392         <servlet-name>conPool</servlet-name>
1393         <url-pattern>/poolurl</url-pattern>
1394     </servlet-mapping>
1395 </web-app>
-----ConnPoolServlet.java-----
-->
1397 package com.nt;
1398 import java.io.PrintWriter;
1399 import java.io.IOException;
1400
1401 import javax.servlet.ServletException;
1402 import javax.servlet.http.HttpServlet;
1403 import javax.servlet.http.HttpServletRequest;
1404 import javax.servlet.http.HttpServletResponse;
1405
1406 import java.sql.Connection;
1407 import java.sql.Statement;
1408 import java.sql.DriverManager;
1409 import java.sql.ResultSet;
1410 import java.sql.ResultSetMetaData;
1411 import javax.sql.DataSource;
1412
1413 import javax.naming.InitialContext;
1414
1415 public class ConnPoolServlet extends HttpServlet
1416 {
1417     public void doGet(HttpServletRequest req,HttpServletResponse res)
1418             throws ServletException,IOException
1419     {
1420         res.setContentType("text/html");
1421         PrintWriter out;
1422         out=res.getWriter();
1423         String tableName=req.getParameter("table");
1424         try
1425         {
1426             //get con obj from jdbc con poo
1427             Connection con=makeConnection();
1428             //create Statement obj
1429             Statement st=con.createStatement();
```

```
1430 //execute the SQL query
1431     ResultSet rs=st.executeQuery("select * from "+tableName);
1432 //Use ResultMetata to get DB table values along col names
1433     ResultSetMetaData rsmd=rs.getMetaData();
1434     int columnCount=rsmd.getColumnCount();
1435
1436     out.println("<html><body bgcolor=#fffffa9>");
1437     out.println("<center><h3>The Details of the table "+tableName);
1438     out.println("<br><br><table cellspacing=1 bgcolor=#00ff00>");
1439     out.println("<tr bgcolor=#fffffa9>");
1440         //print col names
1441         for(int col=1;col<=columnCount;col++)
1442     {
1443         out.println("<th>"+rsmd.getColumnLabel(col)+"&ampnbsp&ampnbsp</th>");
1444
1445     }//for
1446     out.println("</tr>");
1447 //print col values
1448 while(rs.next())
1449 {
1450     out.println("<tr bgcolor=#fffffa9>");
1451
1452         for(int i=1;i<=columnCount;i++){
1453             out.println("<td>"+rs.getString(i)+"&ampnbsp&ampnbsp</td>");
1454         }//for
1455
1456         out.println("</tr>");
1457     }//while
1458
1459     out.println("</table><br>");
1460     out.println("To view another table<b><a href=index.htm>Click here</a></b>");
1461     out.println("</body></html>");
1462 }//try
1463 catch(Exception e)
1464 {
1465     out.println("<html><body bgcolor=#fffffa9><center><br><br>");
1466     out.println("<h3>Table Doesn't Exist in Database</h3>");
1467     out.println("<br>To view another table <a href=index.htm>Click here</a>");
1468     out.println("</body></html>");
1469 }//catch
1470 }//doGet(-,-)
1471
1472 public void doPost(HttpServletRequest req,HttpServletResponse res)
1473                     throws ServletException,IOException
1474 {
1475     doGet(req,res);
1476 }
1477
1478 public Connection makeConnection()
1479 {
1480     Connection con=null;
1481     try
1482     { //get InitialContext obj
1483         InitialContext ic=new InitialContext();
1484         // get DataSource obj ref from jndi registry
1485         DataSource ds=(DataSource)ic.lookup("DsJndi");
1486         // get jdbc con obj from jdbc con pool
1487         con= ds.getConnection();
1488     }//try
1489     catch(Exception e)
1490     {
1491         System.out.println(e);
1492     }//catch
1493     return con;
1494 } //makeConnection()
```

```

1495 }//class
1496 =====
1497 App15 ) Application on DAO + VO +BO +DTO
1498 =====
1499 -----Form.html-----
1500 <h1><center> Get Student Result </center></h1>
1501
1502 <form action="student">
1503     Number: <input type="text" name="sno"><br>
1504     Name : <input type="text" name="sname"><br>
1505     Marks1 :<input type="text" name="m1"><br>
1506     Marks2 :<input type="text" name="m2"><br>
1507     Marks3 :<input type="text" name="m3"><br>
1508     <input type="submit" value="Get Result"/>
1509 </form>
1510 -----web.xml-----
1511 <web-app>
1512     <servlet>
1513         <servlet-name>ss</servlet-name>
1514             <servlet-class>com.nt.web.StudentServlet</servlet-class>
1515         </servlet>
1516         <servlet-mapping>
1517             <servlet-name>ss</servlet-name>
1518             <url-pattern>/student</url-pattern>
1519         </servlet-mapping>
1520         <welcome-file-list>
1521             <welcome-file>Form.html</welcome-file>
1522         </welcome-file-list>
1523     </web-app>
1524 -----StudentServlet.java-----
1525 package com.nt.web;
1526
1527 import java.io.IOException;
1528 import java.io.PrintWriter;
1529
1530 import javax.servlet.ServletException;
1531 import javax.servlet.http.HttpServlet;
1532 import javax.servlet.http.HttpServletRequest;
1533 import javax.servlet.http.HttpServletResponse;
1534
1535 import com.nt.dto.StudentDTO;
1536 import com.nt.service.StudentService;
1537 import com.nt.vo.StudentVO;
1538
1539 public class StudentServlet extends HttpServlet {
1540
1541
1542     @Override
1543     protected void doGet(HttpServletRequest req, HttpServletResponse res)
1544         throws ServletException, IOException {
1545         // general settings
1546         PrintWriter pw=res.getWriter();
1547         res.setContentType("text/html");
1548         //read form data
1549         int no=Integer.parseInt(req.getParameter("sno"));
1550         String name=req.getParameter("sname");
1551         int m1=Integer.parseInt(req.getParameter("m1"));
1552         int m2=Integer.parseInt(req.getParameter("m2"));
1553         int m3=Integer.parseInt(req.getParameter("m3"));
1554         //create VO class obj
1555         StudentVO vo=new StudentVO(no,name,m1,m2,m3);
1556         //Convert VO class obj to DTO class obj
1557         StudentDTO dto=new StudentDTO(vo.getSno(),
1558                                         vo.getsName(),
1559                                         vo.getM1(),
1560                                         vo.getM2());

```

```
1561          // use SErvie class b.logic
1562          StudentService service=new StudentService();
1563          String result=service.generateResult(dto);
1564          // Present the result
1565          pw.println("<h2> Student Result is"+result);
1566
1567          //close stream
1568          pw.close();
1569      }//doGet(-,-)
1570
1571      @Override
1572      protected void doPost(HttpServletRequest req, HttpServletResponse res)
1573          throws ServletException, IOException {
1574          doGet(req,res);
1575      }//doPost(-,-)
1576  }//class
1577  -----StudentVO.java-----
1578 package com.nt.vo;
1580
1581 public class StudentVO {
1582     private int sno;
1583     private String sname;
1584     private int m1,m2,m3;
1585     public StudentVO(){
1586         System.out.println("StudentVO:0-param constructor");
1587     }
1588
1589     public StudentVO(int sno, String sname, int m1, int m2, int m3) {
1590         System.out.println("StudentVO:5-param constructor");
1591         this.sno = sno;
1592         this.sname = sname;
1593         this.m1 = m1;
1594         this.m2 = m2;
1595         this.m3 = m3;
1596     }
1597
1598     public int getSno() {
1599         return sno;
1600     }
1601     public void setSno(int sno) {
1602         this.sno = sno;
1603     }
1604     public String getSname() {
1605         return sname;
1606     }
1607     public void setSname(String sname) {
1608         this.sname = sname;
1609     }
1610     public int getM1() {
1611         return m1;
1612     }
1613     public void setM1(int m1) {
1614         this.m1 = m1;
1615     }
1616     public int getM2() {
1617         return m2;
1618     }
1619     public void setM2(int m2) {
1620         this.m2 = m2;
1621     }
1622     public int getM3() {
1623         return m3;
1624     }
1625     public void setM3(int m3) {
1626         this.m3 = m3;
1627     }
1628 }
```

```
1627     }
1628 }
1629 -----
1630 package com.nt.dto;
1631
1632 import java.io.Serializable;
1633
1634 public class StudentDTO implements Serializable {
1635     private int sno;
1636     private String sname;
1637     private int m1,m2,m3;
1638     public StudentDTO(){
1639         System.out.println("StudentDTO:0-param constructor");
1640     }
1641
1642     public StudentDTO(int sno, String sname, int m1, int m2, int m3) {
1643         System.out.println("StudentDTO:5-param constructor");
1644         this.sno = sno;
1645         this.sname = sname;
1646         this.m1 = m1;
1647         this.m2 = m2;
1648         this.m3 = m3;
1649     }
1650
1651     public int getSno() {
1652         return sno;
1653     }
1654     public void setSno(int sno) {
1655         this.sno = sno;
1656     }
1657     public String getSname() {
1658         return sname;
1659     }
1660     public void setSname(String sname) {
1661         this.sname = sname;
1662     }
1663     public int getM1() {
1664         return m1;
1665     }
1666     public void setM1(int m1) {
1667         this.m1 = m1;
1668     }
1669     public int getM2() {
1670         return m2;
1671     }
1672     public void setM2(int m2) {
1673         this.m2 = m2;
1674     }
1675     public int getM3() {
1676         return m3;
1677     }
1678     public void setM3(int m3) {
1679         this.m3 = m3;
1680     }
1681 }
1682 -----
1683 package com.nt.service;
1684
1685 import com.nt.bo.StudentBO;
1686 import com.nt.dao.StudentDAO;
1687 import com.nt.dto.StudentDTO;
1688
1689 public class StudentService {
1690
1691     public String generateResult(StudentDTO dto){
1692         // write B.logic
```

```
1693         int total=dto.getM1()+dto.getM2()+dto.getM3();
1694         float avg=total/3.0f;
1695         String result=null;
1696         if(avg<=35)
1697             result="fail";
1698         else
1699             result="pass";
1700         // prepare BO obj having persistent data
1701         StudentBO bo=new StudentBO(dto.getSno(),dto.getSname(),total,avg,result);
1702
1703         //create Service class obj
1704         StudentDAO dao=new StudentDAO();
1705         int status=dao.insert(bo);
1706         if(status==0)
1707             return "Student Registration Failed";
1708         else
1709             return "Student Registration succeeded with no"+dto.getSno();
1710
1711     }//generateResult(-)
1712
1713 } //class
-----StudentBO.java-----
1715 package com.nt.bo;
1716
1717 public class StudentBO {
1718     private int sno;
1719     private String sname;
1720     private int total;
1721     private float avg;
1722     private String result;
1723
1724     public StudentBO() {
1725         System.out.println("StudentBO:0-param constructor");
1726     }
1727     public StudentBO(int sno, String sname, int total, float avg, String result) {
1728         System.out.println("StudentBO:5-param constructor");
1729         this.sno = sno;
1730         this.sname = sname;
1731         this.total = total;
1732         this.avg = avg;
1733         this.result = result;
1734     }
1735
1736
1737     public int getSno() {
1738         return sno;
1739     }
1740
1741     public void setSno(int sno) {
1742         this.sno = sno;
1743     }
1744
1745     public String getSname() {
1746         return sname;
1747     }
1748
1749     public void setSname(String sname) {
1750         this.sname = sname;
1751     }
1752
1753     public int getTotal() {
1754         return total;
1755     }
1756
1757     public void setTotal(int total) {
```

```
1759         this.total = total;
1760     }
1761
1762     public float getAvg() {
1763         return avg;
1764     }
1765
1766     public void setAvg(float avg) {
1767         this.avg = avg;
1768     }
1769
1770     public String getResult() {
1771         return result;
1772     }
1773
1774     public void setResult(String result) {
1775         this.result = result;
1776     }
1777
1778
1779 }
1780 -----StudentDAO.java-----
1781 package com.nt.dao;
1782
1783 import java.sql.Connection;
1784 import java.sql.PreparedStatement;
1785
1786 import javax.naming.InitialContext;
1787 import javax.sql.DataSource;
1788
1789 import com.nt.bo.StudentBO;
1790
1791 public class StudentDAO {
1792     private static final String STUDENT_INSERT_QRY="INSERT INTO STUDENT_TAB
1793     VALUES(?,?,?,?,?)";
1794
1795     public int insert(StudentBO bo){
1796         try{
1797             //get the con obj from jdbc con pool
1798             InitialContext ic=new InitialContext();
1799             // get DataSoruce obj ref from jndi registry
1800             DataSource ds=(DataSource)ic.lookup("DsJndi");
1801             //get con obj from jdbc con pool
1802             Connection con=ds.getConnection();
1803             //create PreparedStatement obj
1804             PreparedStatement ps=con.prepareStatement(STUDENT_INSERT_QRY);
1805             ps.setInt(1,bo.getSno());
1806             ps.setString(2,bo.getSname());
1807             ps.setInt(3,bo.getTotal());
1808             ps.setFloat(4,bo.getAvg());
1809             ps.setString(5,bo.getResult());
1810             //Execute the Query
1811             int result=ps.executeUpdate();
1812             return result;
1813         }
1814         catch(Exception e){
1815             e.printStackTrace();
1816             return 0;
1817         }
1818     }//insert
1819 }
1820 =====
1821 App16)Example App on rd.forward(-,-) and rd.include(-,-)
1822 =====
1823 -----input.html-----
```

```
1824 <form action="dburl" method="get">
1825     <b>Employee no </b><input type="text" name="tno"><br>
1826     <input type="submit" value="GetEmpDetails">
1827 </form>
1828 -----Footer.html-----
1829 <hr><br><br><br>
1830 <center><i><b>&copy; copy rights are reserved 2010-11</b></i></center>
1831 -----web.xml-----
1832 <web-app>
1833     <context-param>
1834         <param-name>driver</param-name>
1835         <param-value>oracle.jdbc.driver.OracleDriver</param-value>
1836     </context-param>
1837     <context-param>
1838         <param-name>url</param-name>
1839         <param-value>jdbc:oracle:thin:@localhost:1521:xe</param-value>
1840     </context-param>
1841     <context-param>
1842         <param-name>dbuser</param-name>
1843         <param-value>scott</param-value>
1844     </context-param>
1845     <context-param>
1846         <param-name>dbpwd</param-name>
1847         <param-value>tiger</param-value>
1848     </context-param>
1849
1850     <servlet>
1851         <servlet-name>ABC</servlet-name>
1852         <servlet-class>com.nt.DBSrv</servlet-class>
1853     </servlet>
1854
1855     <servlet-mapping>
1856         <servlet-name>ABC</servlet-name>
1857         <url-pattern>/dburl</url-pattern>
1858     </servlet-mapping>
1859
1860     <servlet>
1861         <servlet-name>E</servlet-name>
1862         <servlet-class>com.nt.ErrSrv</servlet-class>
1863     </servlet>
1864
1865     <servlet-mapping>
1866         <servlet-name>E</servlet-name>
1867         <url-pattern>/eurl</url-pattern>
1868     </servlet-mapping>
1869
1870     <servlet>
1871         <servlet-name>H</servlet-name>
1872         <servlet-class>com.nt.HeaderSrv</servlet-class>
1873     </servlet>
1874
1875     <servlet-mapping>
1876         <servlet-name>H</servlet-name>
1877         <url-pattern>/Headurl</url-pattern>
1878     </servlet-mapping>
1879
1880
1881     <welcome-file-list>
1882         <welcome-file>input.html</welcome-file>
1883     </welcome-file-list>
1884
1885 </web-app>
1886 ----- DBSrv.java-----
1887 //DBSrv.java(Main Servlet prg)
1888 package com.nt;
1889 import javax.servlet.*;
```

```
1890 import javax.servlet.http.*;
1891
1892 import java.io.*;
1893 import java.sql.*;
1894
1895 public class DBSrv extends HttpServlet
1896 {
1897     Connection con=null;
1898     PreparedStatement ps1;
1899     public void init()
1900     {
1901         try
1902         {
1903             //get Access to ServletConfig obj
1904             ServletContext sc=getServletContext();
1905             // reading context parameter values of web.xml
1906             String s1=sc.getInitParameter("driver");
1907             String s2=sc.getInitParameter("url");
1908             String s3=sc.getInitParameter("dbuser");
1909             String s4=sc.getInitParameter("dbpwd");
1910             //create jdbc con object.....
1911             Class.forName(s1);
1912             con=DriverManager.getConnection(s2,s3,s4);
1913         }//try
1914         catch(SQLException se)
1915         {
1916             se.printStackTrace();
1917         }
1918         catch(Exception e)
1919         {
1920             e.printStackTrace();
1921         }
1922     }//init()
1923
1924     public void doGet(HttpServletRequest req,HttpServletResponse res) throws
1925     ServletException,IOException
1926     {
1927         // get PrintWriter obj
1928         PrintWriter pw=res.getWriter();
1929         // set response content type
1930         res.setContentType("text/html");
1931         System.out.println("start of req processing logic in DBSrv prg");
1932
1933         try
1934         {
1935             //include header content from HeaderSrv
1936             RequestDispatcher rd1=req.getRequestDispatcher("Headurl");//pointing to HeadSrv
1937             servlet prg
1938             rd1.include(req,res);//includes Header logic's Html output
1939
1940             // read form data from form page
1941             int no=Integer.parseInt(req.getParameter("tno"));
1942
1943             //write jdbc code to manipulate DB table data...
1944             ps1=con.prepareStatement("select ename,sal from emp where empno=?");
1945             ps1.setInt(1,no);
1946             //execute the Query
1947             ResultSet rs=ps1.executeQuery();
1948             //Process the Result
1949             if(rs.next())
1950             {
1951                 pw.println("Employee name is:"+rs.getString(1));
1952                 pw.println("Employee Salary is:"+rs.getInt(2));
1953             }
1954
1955             //close ResultSet obj
```

```

1954         rs.close();
1955         System.out.println("end of req processing logic in DBSrv prg");
1956     //include FooterContent from Footer.html
1957         RequestDispatcher rd2=req.getRequestDispatcher("Footer.html");
1958         rd2.include(req,res); //includes Footer logic's Html output
1959
1960     }//try
1961     catch(Exception e)
1962     {
1963         RequestDispatcher rd=req.getRequestDispatcher("eurl"); //pointing ErrSrv
1964         servlet prg
1965         rd.forward(req,res);//forwards the request
1966     }
1967 }//doGet(-,-)

1968 public void doPost(HttpServletRequest req,HttpServletResponse res) throws
1969 ServletException,IOException
1970 {
1971     try
1972     {
1973         doGet(req,res);
1974     }
1975     catch(Exception e)
1976     {
1977         e.printStackTrace();
1978     }
1979 }//doPost(-,-)

1980
1981 public void destroy()
1982 {
1983     try
1984     {
1985         if(ps1!=null)
1986             ps1.close();
1987     }
1988     catch(SQLException se)
1989     {
1990         se.printStackTrace();
1991     }
1992     try
1993     {
1994         if(con!=null)
1995             con.close();
1996     }
1997     catch(SQLException se)
1998     {
1999         se.printStackTrace();
2000     }
2001
2002 } //destroy
2003 } //class
-----ErrSrv.java-----
2004 package com.nt;
2005 import javax.servlet.*;
2006 import javax.servlet.http.*;
2007 import java.io.*;
2008
2009
2010 public class ErrSrv extends HttpServlet
2011 {
2012
2013     public void service(HttpServletRequest req,HttpServletResponse res) throws
2014     ServletException,IOException
2015     {
2016         PrintWriter pw=res.getWriter();
2017         res.setContentType("text/html");

```

```

2017 //include header content from HeaderSrv
2018 RequestDispatcher rd1=req.getRequestDispatcher("Headurl");//pointing to HeadSrv
servlet prg
2019 rd1.include(req,res);//includes Header logic's Html output
2020
2021
2022 pw.println("<font color=red size=2>Internal problem</font>");
2023 pw.println("<br><br><a href='input.html'>home</a>");
2024 System.out.println("Service(-,-) in ErrSrv prg");
2025
2026 //include FooterContent from Footer.html
2027 RequestDispatcher rd2=req.getRequestDispatcher("Footer.html");
2028 rd2.include(req,res); //includes Footer logic's Html output
2029
2030 }
2031 } //class
2032 -----
2033 -----HeaderSrv.java-----
2034 package com.nt;
2035 import javax.servlet.*;
2036 import javax.servlet.http.*;
2037 import java.io.*;
2038
2039 public class HeaderSrv extends HttpServlet
2040 {
2041
2042     public void service(HttpServletRequest req,HttpServletResponse res) throws
ServletException,IOException
2043     {
2044         PrintWriter pw=res.getWriter();
2045         res.setContentType("text/html");
2046
2047         // writer Header logic
2048         pw.println("<center><font color=red size=7>HCL Technologies </font></center><br>");
2049         pw.println("<br><br><br><hr>");
2050     }//service(-,-)
2051 } //class
2052 -----
2053 =====
2054 App16>>>Example App on rd.include(-,-) to get communication b/w two web applications of same
server(weblogic)
2055 =====
2056 WaOne (First App)
2057 -----Form.html-----
2058
2059 <form action="furl" method="get">
2060     Value: <input type="text" name="t1"><br>
2061     <input type="submit" value="GetSquare&Cube">
2062 </form>
2063
2064 -----web.xml-----
2065 <web-app>
2066     <servlet>
2067         <servlet-name>abc</servlet-name>
2068         <servlet-class>com.nt.FirstSrv</servlet-class>
2069     </servlet>
2070     <servlet-mapping>
2071         <servlet-name>abc</servlet-name>
2072         <url-pattern>/furl</url-pattern>
2073     </servlet-mapping>
2074 </web-app>
2075 -----FirstSrv.java-----
2076 //FirstSrv.java (Servlet prg)
2077 package com.nt;
2078 import javax.servlet.*;
2079 import javax.servlet.http.*;

```

```
2080 import java.io.*;
2081 public class FirstSrv extends HttpServlet
2082 {
2083     public void doGet(HttpServletRequest req,HttpServletResponse res) throws
2084         ServletException,IOException
2085     {
2086         // General settings
2087         res.setContentType("text/html");
2088         PrintWriter pw=res.getWriter();
2089         //read form data
2090         int val=Integer.parseInt(req.getParameter("t1"));
2091         //Display Square value
2092         pw.println("<br>FSrv: square value is"+(val*val));
2093
2094         // include the response of SSrv prg belonging to WaTwo web application
2095         //....get WaOne web application's ServletContext obj
2096         ServletContext sc1=getServletContext();
2097         // get WaTwo web application's ServletContext obj
2098         ServletContext sc2=sc1.getServletContext("WaTwo");
2099         // get RequestDispatcher obj pointing to SSrv prg of WaTwo
2100         RequestDispatcher rd=sc2.getRequestDispatcher("/surl");
2101         //include response
2102         rd.include(req,res);
2103
2104         //close stream
2105         pw.close();
2106     }//doGet(-,-)
2107
2108     public void doPost(HttpServletRequest req,HttpServletResponse res) throws
2109         ServletException,IOException
2110     {
2111         doGet(req,res);
2112     }//doPost(-,-)
2113 } //class
-----
2114 -----SecondSrv.java-----
2115 // SecondSrv.java
2116 package com.nt;
2117 import javax.servlet.*;
2118 import javax.servlet.http.*;
2119 import java.io.*;
2120
2121 public class SecondSrv extends HttpServlet
2122 {
2123     public void doGet(HttpServletRequest req,HttpServletResponse res) throws
2124         ServletException,IOException
2125     {
2126         // General settings
2127         PrintWriter pw=res.getWriter();
2128         res.setContentType("text/html");
2129         // read form data
2130         int val=Integer.parseInt(req.getParameter("t1"));
2131         pw.println("<br> Cube value is"+(val*val*val));
2132
2133     }//doGet(-,-)
2134     public void doPost(HttpServletRequest req,HttpServletResponse res) throws
2135         ServletException,IOException
2136     {
2137         doGet(req,res);
2138     }//doPost(-,-)
2139 } //class
-----
2140 <web-app>
2141     <servlet>
```

```

2142      <servlet-name>abc</servlet-name>
2143          <servlet-class>com.nt.SecondSrv</servlet-class>
2144      </servlet>
2145      <servlet-mapping>
2146          <servlet-name>abc</servlet-name>
2147          <url-pattern>/surl</url-pattern>
2148      </servlet-mapping>
2149  </web-app>
2150 -----
2151 =====
2152 App17>>>>>>Example App res.sendRedirect(-)(redirecting request from Servlet prg of one
2153 Server to other Servlet prg of
2154 Another Server)
2155 -----
2156 FirstApp
2157 -----Form.html-----
2158 <form action="billurl" >
2159     Item Name: <input type="text" name="t1"><br>
2160     Item Price : <input type="text" name="t2"><br>
2161     Item Qty : <input type="text" name="t3"><br>
2162     <input type="submit" value="Get Bill Details">
2163 </form>
2164 -----web.xml-----
2165
2166 <web-app>
2167   <servlet>
2168     <servlet-name>abc</servlet-name>
2169     <servlet-class>com.nt.BillSrv</servlet-class>
2170   </servlet>
2171
2172   <servlet-mapping>
2173       <servlet-name>abc</servlet-name>
2174       <url-pattern>/billurl</url-pattern>
2175   </servlet-mapping>
2176 </web-app>
2177 -----
2178 -----BillSrv.java-----
2179 //BillSrv.java
2180 package com.nt;
2181
2182 import javax.servlet.*;
2183 import javax.servlet.http.*;
2184 import java.io.*;
2185
2186
2187 public class BillSrv extends HttpServlet
2188 {
2189 public void doGet(HttpServletRequest req,HttpServletResponse res) throws
2190 ServletException, IOException
2191 {
2192     //General settings
2193     PrintWriter pw = res.getWriter();
2194     res.setContentType("text/html");
2195
2196     //read form data
2197     String name = req.getParameter("t1");
2198     float price = Float.parseFloat(req.getParameter("t2"));
2199     float qty = Float.parseFloat(req.getParameter("t3"));
2200     //calc Bill Amt
2201     float bamt = price * qty;
2202
2203     if(bamt >= 50000)
2204     {
2205         System.out.println("before res.sendRedirect(-) in BillSrv");
2206         res.sendRedirect("http://localhost:7879/SecondApp/discounturl?bill="+bamt+"&iname="+name);
2207     }
2208 }
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225

```

```
2206     System.out.println("after res.sendRedirect() in BillSrv");
2207     }//if
2208     else
2209     {
2210         pw.println("<br> From BillSrv prg <br>");
2211         pw.println("<br> Item name:"+name+" Price:"+price+" Qty :" +qty+"Bill Amt:"+amt);
2212     }//else
2213
2214     //close stream
2215     pw.close();
2216 } //doGet(-,-)
2217
2218 public void doPost(HttpServletRequest req,HttpServletResponse res) throws
ServletException,IOException
2219 {
2220     doGet(req,res);
2221 } //doPost(-,-)
2222 } //class
2223 -----
2224 SecondApp
2225 -----DiscountSrv.java-----
2226 //DiscountSrv.java
2227 package com.nt;
2228 import javax.servlet.*;
2229 import javax.servlet.http.*;
2230 import java.io.*;
2231
2232 public class DiscountSrv extends HttpServlet
2233 {
2234     public void doGet(HttpServletRequest req,HttpServletResponse res) throws
ServletException,IOException
2235     {
2236         // General settings
2237         PrintWriter pw = res.getWriter();
2238         res.setContentType("text/html");
2239
2240         //read additional request params given by BillSrv prg (source prg)
2241         float amt = Float.parseFloat(req.getParameter("bill"));
2242         String name = req.getParameter("iname");
2243
2244         System.out.println("From DiscountSrv: doGet(-,-) method");
2245
2246         //Calc discount Amt
2247         float discount = amt * 0.3f;
2248         float finalamt = amt - discount;
2249
2250         // Display Details
2251         pw.println("<br> Item name :" + name);
2252         pw.println("<br> Bill Amt :" + amt);
2253         pw.println("<br> Duscount :" + discount + " Final Amount:" + finalamt);
2254         pw.println("<br> From DiscountSrv prg");
2255         //close stream
2256         pw.close();
2257     } //doGet(-,-)
2258     public void doPost(HttpServletRequest req,HttpServletResponse res) throws
ServletException,IOException
2259     {
2260         doGet(req,res);
2261     }
2262
2263 } //class
2264 -----web.xml-----
2265
2266 <web-app>
2267   <servlet>
2268     <servlet-name>xyz</servlet-name>
```

```
2269 <servlet-class>com.nt.DiscountSrv</servlet-class>
2270 </servlet>
2271
2272 <servlet-mapping>
2273   <servlet-name>xyz</servlet-name>
2274     <url-pattern>/discounturl</url-pattern>
2275   </servlet-mapping>
2276 </web-app>
2277
2278 =====
2279 App18 (Using res.sendRedirect(-) to redirect the request to internet websites)
2280 =====
2281 -----input.html-----
2282 <form action="searchurl" method="get">
2283   Search String: <input type="text" name="tsearch"><br>
2284   SearchEngg : <input type="radio" name="engg" value="google" checked>Google
2285           <input type="radio" name="engg" value="bing" >Bing
2286           <input type="radio" name="engg" value="yahoo" >YahooSearch
2287   <br>
2288   <input type="submit" value="search">
2289 </form>
2290 -----web.xml-----
2291 <web-app>
2292   <servlet>
2293     <servlet-name>abc</servlet-name>
2294     <servlet-class>com.nt.SearchSrv</servlet-class>
2295   </servlet>
2296
2297   <servlet-mapping>
2298     <servlet-name>abc</servlet-name>
2299     <url-pattern>/searchurl</url-pattern>
2300   </servlet-mapping>
2301 </web-app>
2302 -----SearchSrv.java-----
2303 //SeachSrv.java
2304 package com.nt;
2305 import javax.servlet.*;
2306 import javax.servlet.http.*;
2307 import java.io.*;
2308 import java.util.*;
2309
2310 public class SearchSrv extends HttpServlet
2311 {
2312
2313     public void doGet(HttpServletRequest req,HttpServletResponse res) throws
2314     ServletException,IOException
2315     {
2316         // general work
2317         PrintWriter pw=req.getWriter();
2318         res.setContentType("text/html");
2319
2320         // read form data
2321         String ss=req.getParameter("tsearch");
2322         String sengg=req.getParameter("engg");
2323
2324         // frame req urls to perform redirection
2325         String url=null;
2326
2327         if(sengg.equals("google"))
2328         {
2329             url="+ss;
2330         }
2331         else if\(sengg.equals\("yahoo"\)\)
2332         {
2333             url="+ss;
2334         }
```

```
2334         . . . else { url="http://www.bing.com/search?q="+ss; }
2335         . . .
2336         . . .
2337         . . .
2338         // perform send Redirection
2339         res.sendRedirect(url);
2340     } //doGet(-,-)
2341     public void doPost(HttpServletRequest req,HttpServletResponse res) throws
2342         ServletException,IOException {
2343     {
2344         doGet(req,res);
2345     }
2346     } //calss
2347 =====
2348 App18>>>>>>>Application on request/servletcontext/session_attributes>>>>>>>>>>>>>
2349 -----Srv1.java-----
2350 package com.nt;
2351
2352 import java.io.IOException;
2353
2354 import javax.servlet.RequestDispatcher;
2355 import javax.servlet.ServletContext;
2356 import javax.servlet.ServletException;
2357 import javax.servlet.http.HttpServlet;
2358 import javax.servlet.http.HttpServletRequest;
2359 import javax.servlet.http.HttpServletResponse;
2360 import javax.servlet.http.HttpSession;
2361
2362 public class Srv1 extends HttpServlet {
2363
2364 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
2365 ServletException, IOException {
2366     //create request attribute
2367     request.setAttribute("attr1","val1");
2368     //create Session attribute
2369     HttpSession ses=request.getSession();
2370     ses.setAttribute("attr2","val2");
2371     //create ServletContext attribute
2372     ServletContext sc=getServletContext();
2373     sc.setAttribute("attr3","val3");
2374
2375     //forward request to Srv2 prg
2376     RequestDispatcher rd=request.getRequestDispatcher("/Srv2");
2377     rd.forward(request,response);
2378 }
2379 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
2380 ServletException, IOException {
2381     {
2382         doGet(request,response);
2383     }
2384 } -----Srv2.java-----
2385 package com.nt;
2386
2387 import java.io.IOException;
2388 import java.io.PrintWriter;
2389
2390 import javax.servlet.RequestDispatcher;
2391 import javax.servlet.ServletContext;
2392 import javax.servlet.ServletException;
2393 import javax.servlet.http.HttpServlet;
2394 import javax.servlet.http.HttpServletRequest;
2395 import javax.servlet.http.HttpServletResponse;
2396 import javax.servlet.http.HttpSession;
```

```

2397 public class Srv2 extends HttpServlet {
2398
2399     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
2400         ServletException, IOException {
2401         // General settings
2402         PrintWriter pw=response.getWriter();
2403         response.setContentType("text/html");
2404
2405         //read request attribute value
2406         pw.println("Srv2:attr1 (request) value"+request.getAttribute("attr1"));
2407         //read session attribute value
2408         HttpSession ses=request.getSession();
2409         pw.println("<br>Srv2:attr2 (session) value"+ses.getAttribute("attr2"));
2410         //read ServletContext attribuite value
2411         ServletContext sc=getServletContext();
2412         pw.println("<br>Srv2:attr3 (sc) value"+sc.getAttribute("attr3"));
2413
2414         //forward request to Srv3 prg
2415         RequestDispatcher rd=request.getRequestDispatcher("/Srv3");
2416         rd.forward(request,response);
2417     }
2418     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
2419         ServletException, IOException {
2420         doGet(request,response);
2421     }
2422     -----Srv3.java-----
2423     package com.nt;
2424     import java.io.IOException;
2425     import java.io.PrintWriter;
2426
2427     import javax.servlet.RequestDispatcher;
2428     import javax.servlet.ServletContext;
2429     import javax.servlet.ServletException;
2430     import javax.servlet.annotation.WebServlet;
2431     import javax.servlet.http.HttpServlet;
2432     import javax.servlet.http.HttpServletRequest;
2433     import javax.servlet.http.HttpServletResponse;
2434     import javax.servlet.http.HttpSession;
2435
2436     public class Srv3 extends HttpServlet {
2437
2438         protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
2439             ServletException, IOException {
2440
2441             // General settings
2442             PrintWriter pw=response.getWriter();
2443             response.setContentType("text/html");
2444
2445             //read request attribute value
2446             pw.println("Srv3:attr1 (request) value"+request.getAttribute("attr1"));
2447
2448             //read session attribute value
2449             HttpSession ses=request.getSession();
2450             pw.println("<br>Srv3:attr2 (session) value"+ses.getAttribute("attr2"));
2451
2452             //read ServletContext attribuite value
2453             ServletContext sc=getServletContext();
2454             pw.println("<br>Srv3:attr3 (sc) value"+sc.getAttribute("attr3"));
2455
2456         }
2457         protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
2458             ServletException, IOException {
2459             doGet(request,response);
2460         }

```

```
2459 }  
2460 -----Srv4.java-----  
2461 package com.nt;  
2462 import java.io.IOException;  
2463 import java.io.PrintWriter;  
2464  
2465 import javax.servlet.RequestDispatcher;  
2466 import javax.servlet.ServletContext;  
2467 import javax.servlet.ServletException;  
2468 import javax.servlet.annotation.WebServlet;  
2469 import javax.servlet.http.HttpServlet;  
2470 import javax.servlet.http.HttpServletRequest;  
2471 import javax.servlet.http.HttpServletResponse;  
2472 import javax.servlet.http.HttpSession;  
2473  
2474 public class Srv4 extends HttpServlet {  
2475  
2476 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws  
ServletException, IOException {  
2477  
2478     // General settings  
2479     PrintWriter pw=response.getWriter();  
2480     response.setContentType("text/html");  
2481  
2482     //read request attribute value  
2483     pw.println("Srv4:attr1 (request) value"+request.getAttribute("attr1"));  
2484  
2485     //read session attribute value  
2486     HttpSession ses=request.getSession();  
2487     pw.println("<br>Srv4:attr2 (session) value"+ses.getAttribute("attr2"));  
2488  
2489     //read ServletContext attribuite value  
2490     ServletContext sc=getServletContext();  
2491     pw.println("<br>Srv4:attr3 (sc) value"+sc.getAttribute("attr3"));  
2492  
2493     }  
2494 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws  
ServletException, IOException {  
2495  
2496     doGet(request,response);  
2497 }  
2498 }  
2499 -----web.xml-----  
2500 <web-app>  
2501     <servlet>  
2502         <servlet-name>ABC</servlet-name>  
2503         <servlet-class>com.nt.Srv1</servlet-class>  
2504     </servlet>  
2505  
2506     <servlet-mapping>  
2507         <servlet-name>ABC</servlet-name>  
2508         <url-pattern>/s1url</url-pattern>  
2509     </servlet-mapping>  
2510  
2511     <servlet>  
2512         <servlet-name>XYZ</servlet-name>  
2513         <servlet-class>com.nt.Srv2</servlet-class>  
2514     </servlet>  
2515  
2516     <servlet-mapping>  
2517         <servlet-name>XYZ</servlet-name>  
2518         <url-pattern>/s2url</url-pattern>  
2519     </servlet-mapping>  
2520  
2521     <servlet>  
2522         <servlet-name>mno</servlet-name>
```

```

2523 <servlet-class>com.nt.Srv3</servlet-class>
2524 </servlet>
2525
2526 <servlet-mapping>
2527   <servlet-name>mno</servlet-name>
2528   <url-pattern>/s3url</url-pattern>
2529 </servlet-mapping>
2530 <servlet>
2531   <servlet-name>jkr</servlet-name>
2532   <servlet-class>com.nt.Srv4</servlet-class>
2533 </servlet>
2534
2535 <servlet-mapping>
2536   <servlet-name>jkr</servlet-name>
2537   <url-pattern>/s4url</url-pattern>
2538 </servlet-mapping>
2539
2540 </web-app>
2541 -----
2542 =====
2543 App19>>>>>>>Example App on Annotations based servlet
2544 =====
2545 -----input.html-----
2546 <form action="test1" method="get">
2547   Name :<input type="text" name="tname"><br>
2548   <input type="submit" value="send">
2549 </form>
2550 -----FormSrv.java-----
2551 package com.nt;
2552 import java.io.IOException;
2553 import java.io.PrintWriter;
2554
2555 import javax.servlet.ServletException;
2556 import javax.servlet.annotation.WebServlet;
2557 import javax.servlet.http.HttpServlet;
2558 import javax.servlet.http.HttpServletRequest;
2559 import javax.servlet.http.HttpServletResponse;
2560
2561 /**
2562 * Servlet implementation class FormSrv
2563 */
2564 @WebServlet(
2565   name = "abc",
2566   urlPatterns = {
2567     "/test2",
2568     "/test1"
2569   })
2570 public class FormSrv extends HttpServlet {
2571   @Override
2572   public void doGet(HttpServletRequest req,HttpServletResponse res) throws
2573   ServletException,IOException
2574   {
2575     // get PrintWriter obj
2576     PrintWriter pw=res.getWriter();
2577     // set response content type
2578     res.setContentType("text/html");
2579
2580     // read form data
2581     String name=req.getParameter("tname");
2582
2583     //display response
2584     pw.println("Hello Mr./Miss./Mrs"+name+" Good morning ");
2585
2586     //close stream
2587     pw.close();
2588   } //doGet(-,-)

```

```
2588     protected void doPost(HttpServletRequest request, HttpServletResponse response)
2589         throws ServletException, IOException {
2590             doGet(request, response);
2591         }
2592     }
2593     -----
2594     =====
2595     App26( Basic App on Filters)
2596     =====
2597     -----MyFilter.java-----
2598
2599 package com.nt;
2600 import javax.servlet.*;
2601 import java.io.IOException;
2602 import java.io.PrintWriter;
2603 public class MyFilter implements Filter
2604 {
2605     public MyFilter()
2606     {
2607         System.out.println("Inside constructor of Filter class");
2608     }
2609     public void init (FilterConfig fcon) throws ServletException
2610     {
2611         System.out.println("Inside init() method of Filter class");
2612     }
2613     public void doFilter(ServletRequest req,ServletResponse res,FilterChain fc)
2614                                     throws IOException,ServletException
2615     {
2616         System.out.println("inside dofilter() of Filter class");
2617         int x=Integer.parseInt(req.getParameter("first"));
2618         int y=Integer.parseInt(req.getParameter("second"));
2619
2620         if ((x<0 ||y<0))
2621         {
2622             PrintWriter pw=res.getWriter();
2623             pw.println("<html> <head>");
2624             pw.println("<head><title>From Filter !!!</title></head>");
2625             pw.println("<body> sorry !!! ur input should be only positive numbers ! ! .");
2626             pw.println(" . ");
2627             pw.println("</body></html>");
2628         }//if
2629         else
2630         {
2631             fc.doFilter(req,res);
2632         }//else
2633     }//doFilter
2634
2635     public void destroy()
2636     {
2637         System.out.println("inside destroy() of Filter class");
2638     }
2639 } //class
2640 -----web.xml-----
2641 <web-app>
2642     <filter>
2643         <filter-name>CheckFilter</filter-name>
2644         <filter-class>com.nt.MyFilter</filter-class>
2645     </filter>
2646     <filter-mapping>
2647         <filter-name>CheckFilter</filter-name>
2648         <url-pattern>/ser/testurl</url-pattern>
2649     </filter-mapping>
2650
```

```

2651      <servlet>
2652          <servlet-name>myser</servlet-name>
2653          <servlet-class>com.nt.MyServlet</servlet-class>
2654      </servlet>
2655      <servlet-mapping>
2656          <servlet-name>myser</servlet-name>
2657          <url-pattern>/ser/testurl</url-pattern>
2658      </servlet-mapping>
2659  </web-app>
2660  -----MyServlet.java-----
2661
2662 package com.nt;
2663 import javax.servlet.*;
2664 import javax.servlet.http.*;
2665 import java.io.IOException;
2666 import java.io.PrintWriter;
2667
2668 public class MyServlet extends HttpServlet
2669 {
2670     public void doGet(HttpServletRequest req,HttpServletResponse res)
2671             throws ServletException,
2672             IOException
2673     {
2674         System.out.println("inside doGet() of Servlet class");
2675         int x=Integer.parseInt(req.getParameter("first"));
2676         int y=Integer.parseInt(req.getParameter("second"));
2677         int z=x+y;
2678         PrintWriter pw=res.getWriter();
2679         pw.println("<html>");
2680         pw.println("<head><title>From servlet !!!</title></head>");
2681         pw.println("<body> result is"+ z+"</body></html>");
2682         pw.close();
2683     }
2684     public void doPost(HttpServletRequest req,HttpServletResponse res)
2685             throws ServletException,
2686             IOException
2687     {
2688         doGet(req,res);
2689     }
2690
2691 } //class
2692 =====
2693 App27(App to apply multipe Filters on one Servlet prg)
2694 =====
2695 -----index.html-----
2696
2697 <html>
2698     <body>
2699         <center>
2700             <form name=f1 action="loginurl">
2701                 <b> Enter usern name </b> <input type=text name=uname
2702                     size=10 > <br>
2703                 <b> Enter password </b> <input type=password name=pwd
2704                     size=10 > <br>
2705                 </b> <input type=submit value=send size=10> <br>
2706             </form>
2707             This is the index page of the web application.
2708         </body>
2709     </html>
2710 -----web.xml-----
2711
2712 <web-app>
2713     <filter>
2714         <filter-name>Auth</filter-name>

```

```

2710      <filter-class>com.nt.AuthFilter</filter-class>
2711  </filter>
2712  <filter-mapping>
2713      <filter-name>Auth</filter-name>
2714      <url-pattern>/loginurl</url-pattern>
2715  </filter-mapping>
2716  <filter>
2717      <filter-name>pTest</filter-name>
2718      <filter-class>com.nt.PerfTestFilter</filter-class>
2719  </filter>
2720  <filter-mapping>
2721      <filter-name>pTest</filter-name>
2722      <url-pattern>/loginurl</url-pattern>
2723  </filter-mapping>
2724  <servlet>
2725      <servlet-name>main</servlet-name>
2726      <servlet-class>com.nt.MainSrv</servlet-class>
2727  </servlet>
2728  <servlet-mapping>
2729      <servlet-name>main</servlet-name>
2730      <url-pattern>/loginurl</url-pattern>
2731  </servlet-mapping>
2732 </web-app>
2733 -----AuthFilter.java-----
2734 package com.nt;
2735 import java.sql.*;
2736 import javax.servlet.*;
2737 import javax.servlet.http.*;
2738 import java.io.IOException;
2739
2740 public class AuthFilter implements Filter
2741 {
2742     private FilterConfig f = null;
2743     private ServletContext sc = null;
2744
2745     public void init(FilterConfig filterConfig)
2746     {
2747         f = filterConfig;
2748         sc = f.getServletContext();
2749     } //init()
2750
2751     public void doFilter(ServletRequest request,
2752                         ServletResponse response,
2753                         FilterChain chain)
2754     throws IOException, ServletException
2755     {
2756         HttpServletRequest hreq = (HttpServletRequest) request;
2757         String un = hreq.getParameter("uname");
2758         String pwd = hreq.getParameter("pwd");
2759
2760         sc.log("in filter :" + hreq.getRequestURI());
2761         sc.log("in filter :" + hreq.getServletPath());
2762         sc.log("uname = " + un + "password = " + pwd);
2763
2764         try
2765         {
2766             DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
2767             Connection con=DriverManager.getConnection
2768                     ("jdbc:oracle:thin:@localhost:1521:xe","scott","tiger");
2769             PreparedStatement ps=con.prepareStatement
2770                     ("SELECT * FROM USERLIST WHERE USERNAME=? AND
2771                         PASSWORD=?");
2772             ps.setString(1,un);
2773             ps.setString(2,pwd);
2774             ResultSet rs=ps.executeQuery();

```

```

2774         if(rs.next())
2775         {
2776             /* here is the code to check for uname/pwd */
2777             sc.log("user name and passwords are given and they are correct");
2778             chain.doFilter(request,response);
2779             return;
2780         }//if
2781     else
2782     {
2783         sc.log("Trying to dispaly the same login page");
2784         RequestDispatcher rd=sc.getRequestDispatcher("/one.html");
2785         rd.forward(request,response);
2786         sc.log("after forwarding request to login page");
2787         return;
2788     }//else
2789 }//try
2790 catch(SQLException se)
2791 {
2792     se.printStackTrace();
2793 }
2794 catch(Exception e)
2795 {
2796     e.printStackTrace();
2797 }
2798 }//doFilter()
2799
2800 public void destroy()
2801 {
2802     f=null;
2803 } //destroy()
2804 } //class
2805 -----one.html-----
2806 <html>
2807 <body>
2808 <center><b><font color=red> u must enter valid uname && password</font> </b>
2809 <form name=f1 action="loginurl">
2810     <b> Enter usern name </b> <input type=text name=username size=10 > <br>
2811     <b> Enter password </b> <input type=password name=password size=10 > <br>
2812     </b>, <input type=submit value=send size=10> <br>
2813 </form>
2814 </body>
2815 </html>
2816 -----PerfTestFilter.java-----
2817 //Filter to check the time taken to process a request. We can use such a filter
2818 //while optimizing the code to improve the speed.
2819 package com.nt;
2820 import java.io.*;
2821 import javax.servlet.*;
2822 import javax.servlet.http.*;
2823
2824 public final class PerfTestFilter implements Filter
2825 {
2826     FilterConfig fg=null;
2827     public void init(FilterConfig filterConfig) throws ServletException
2828     {
2829         fg= filterConfig;
2830     } //init()
2831
2832     public void doFilter(ServletRequest request,ServletResponse response,FilterChain chain)
2833     throws IOException, ServletException
2834     {
2835         // get request trapping time
2836         long beforeProcess = System.currentTimeMillis();
2837

```

```

2838         chain.doFilter(request, response);
2839     // get response trapping time
2840     long afterProcess = System.currentTimeMillis();
2841     // find out the difference
2842     long diff = afterProcess - beforeProcess;
2843     //get HttpServletRequest obj
2844     HttpServletRequest hreq = (HttpServletRequest) request;
2845     // write msg to log file
2846     fg.getServletContext().log("Total Time for processing "+ hreq.getRequestURI()+" :"
2847     "+ diff);
2848     } //doFilter()
2849
2850     public void destroy()
2851     {
2852         fg=null;
2853     } //destroy()
2854
2855 } //class
2856 -----MainSrv.java-----
2857 package com.nt;
2858 import javax.servlet.*;
2859 import javax.servlet.http.*;
2860 import java.io.*;
2861
2862 public class MainSrv extends HttpServlet
2863 {
2864     public void doGet(HttpServletRequest request, HttpServletResponse response)
2865     throws ServletException, java.io.IOException
2866     {
2867         PrintWriter out;
2868         response.setContentType("text/html");
2869         out = response.getWriter();
2870
2871         out.println("<HTML><HEAD><TITLE>");
2872         out.println("</TITLE>MainSrv</HEAD><BODY>");
2873         try
2874         {
2875             Thread.sleep(60000);
2876         }
2877         catch(Exception e)
2878         {
2879             e.printStackTrace();
2880         }
2881
2882         out.println(" <h1><center> Login Successfull <h1>");
2883         out.println("</BODY>");
2884
2885         out.close();
2886     } //doGet()
2887     public void doPost(HttpServletRequest request, HttpServletResponse response)
2888     throws ServletException, java.io.IOException{
2889     doGet(req,res);
2890     }
2891 } //class
2892 =====
2893 App28(File Uploading using java zoom api)
2894 =====
2895 -----upload1.html-----
2896 <form method="post" action="uplurl" enctype="multipart/form-data">
2897     Select File1: <input type="file" name="f1"><br><br>
2898     Select File2: <input type="file" name="f2"><br><br>
2899     <input type="Submit" value="Upload">
2900 </form>
2901 -----web.xml-----
2902 <web-app>

```



```
2969 =====
2970 App29[On Programmatic Security]
2971 -----login.html-----
2972 <center>
2973   <form action="adminurl">
2974     Username : <input type=text name="tuname"> <br>
2975     Password : <input type=password name="tpwd"><br>
2976     <input type=submit value="login">
2977   </form>
2978 </center>
2979 -----web.xml-----
2980 <web-app>
2981   <servlet>
2982     <servlet-name>s1</servlet-name>
2983     <servlet-class>com.nt.AdminServlet</servlet-class>
2984   </servlet>
2985   <servlet-mapping>
2986     <servlet-name>s1</servlet-name>
2987     <url-pattern>/adminurl</url-pattern>
2988   </servlet-mapping>
2989
2990   <servlet>
2991     <servlet-name>s2 </servlet-name>
2992     <servlet-class>com.nt.LoginServlet</servlet-class>
2993   </servlet>
2994   <servlet-mapping>
2995     <servlet-name>s2</servlet-name>
2996     <url-pattern>/loginurl</url-pattern>
2997   </servlet-mapping>
2998 </web-app>
2999 -----LoginServlet.java-----
3000 package com.nt;
3001 import javax.servlet.*;
3002 import javax.servlet.http.*;
3003 import java.io.*;
3004 public class LoginServlet extends HttpServlet
3005 {
3006     public void doGet(HttpServletRequest req,HttpServletResponse res) throws
3007         ServletException,IOException
3008     {
3009         HttpSession ses = req.getSession(true);
3010         String un = req.getParameter("tuname");
3011         String pw = req.getParameter("tpwd");
3012         if(un.equals("admin") && pw.equals("admin"))
3013         {
3014             ses.setAttribute("loggedin","ok");
3015             String up = (String)ses.getAttribute("target");
3016             RequestDispatcher rd = req.getRequestDispatcher(up);
3017             rd.forward(req,res);
3018         }
3019         else
3020         {
3021             RequestDispatcher rd = req.getRequestDispatcher("login.html");
3022             rd.forward(req,res);
3023         }
3024     }
3025 -----AdminServlet.java-----
3026 package com.nt;
3027 import javax.servlet.*;
3028 import javax.servlet.http.*;
3029 import java.io.*;
3030 public class AdminServlet extends HttpServlet
3031 {
3032     public void doGet(HttpServletRequest req, HttpServletResponse res) throws
3033         ServletException,IOException
```

```

3033     {
3034         HttpSession ses=req.getSession(true);
3035         String str =(String) ses.getAttribute("loggedin");
3036         if(str==null)
3037         {
3038             ses.setAttribute("target","/adminurl");
3039             RequestDispatcher rd = req.getRequestDispatcher("/loginurl");
3040             rd.forward(req,res);
3041         }
3042         else
3043         {
3044             PrintWriter pw = res.getWriter();
3045             pw.println("<font color=red size=7> From AdminServlet prg </font>");
3046             pw.close();
3047         }
3048     }
3049 }
3050 =====
3051 Applications Declarative Web Security
3052 App30: (ON BASIC/DIGEST Model)
3053 -----FirstSrv.java-----
3054 package com.nt;
3055 import javax.servlet.*;
3056 import javax.servlet.http.*;
3057 import java.io.*;
3058 public class FirstSrv extends HttpServlet
3059 {
3060     public void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
3061     IOException
3062     {
3063         PrintWriter pw=resp.getWriter();
3064         pw.println("page successfully displayed");
3065         pw.println("<center><b>the user name is "+req.getRemoteUser()+"</center></b>");
3066         pw.println("<center><b>the auth type    is "+req.getAuthType()+"</center></b>");
3067         pw.close();
3068     }// doGet(-,-)
3069     public void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
3070     IOException
3071     {
3072         doGet(req,res);
3073     }
3074 } // class
3075 -----web.xml-----
3076 <web-app>
3077     <servlet>
3078         <servlet-name>servlet1</servlet-name>
3079         <servlet-class>com.nt.FirstSrv</servlet-class>
3080     </servlet>
3081
3082     <servlet-mapping>
3083         <servlet-name>servlet1</servlet-name>
3084         <url-pattern>/srv1</url-pattern>
3085     </servlet-mapping>
3086
3087     <security-constraint>
3088         <web-resource-collection>
3089             <web-resource-name>Authentication</web-resource-name>
3090             <url-pattern>/srv1</url-pattern>
3091             <http-method>GET</http-method>
3092         </web-resource-collection>
3093
3094         <auth-constraint>
3095             <role-name>manager</role-name>
3096         </auth-constraint>
3097
3098     </security-constraint>
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3180
3181
3182
3183
3184
3185
3186
3187
3188
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3289
3290
3291
3292
3293
3294
3295
3296
3297
3298
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3310
3311
3312
3313
3314
3315
3316
3317
3318
3319
3320
3321
3322
3323
3324
3325
3326
3327
3328
3329
3330
3331
3332
3333
3334
3335
3336
3337
3338
3339
3340
3341
3342
3343
3344
3345
3346
3347
3348
3349
3350
3351
3352
3353
3354
3355
3356
3357
3358
3359
3360
3361
3362
3363
3364
3365
3366
3367
3368
3369
3370
3371
3372
3373
3374
3375
3376
3377
3378
3379
3380
3381
3382
3383
3384
3385
3386
3387
3388
3389
3390
3391
3392
3393
3394
3395
3396
3397
3398
3399
3400
3401
3402
3403
3404
3405
3406
3407
3408
3409
3410
3411
3412
3413
3414
3415
3416
3417
3418
3419
3420
3421
3422
3423
3424
3425
3426
3427
3428
3429
3430
3431
3432
3433
3434
3435
3436
3437
3438
3439
3440
3441
3442
3443
3444
3445
3446
3447
3448
3449
3450
3451
3452
3453
3454
3455
3456
3457
3458
3459
3460
3461
3462
3463
3464
3465
3466
3467
3468
3469
3470
3471
3472
3473
3474
3475
3476
3477
3478
3479
3480
3481
3482
3483
3484
3485
3486
3487
3488
3489
3490
3491
3492
3493
3494
3495
3496
3497
3498
3499
3500
3501
3502
3503
3504
3505
3506
3507
3508
3509
3510
3511
3512
3513
3514
3515
3516
3517
3518
3519
3520
3521
3522
3523
3524
3525
3526
3527
3528
3529
3530
3531
3532
3533
3534
3535
3536
3537
3538
3539
3540
3541
3542
3543
3544
3545
3546
3547
3548
3549
3550
3551
3552
3553
3554
3555
3556
3557
3558
3559
3560
3561
3562
3563
3564
3565
3566
3567
3568
3569
3570
3571
3572
3573
3574
3575
3576
3577
3578
3579
3580
3581
3582
3583
3584
3585
3586
3587
3588
3589
3590
3591
3592
3593
3594
3595
3596
3597
3598
3599
3600
3601
3602
3603
3604
3605
3606
3607
3608
3609
3610
3611
3612
3613
3614
3615
3616
3617
3618
3619
3620
3621
3622
3623
3624
3625
3626
3627
3628
3629
3630
3631
3632
3633
3634
3635
3636
3637
3638
3639
3640
3641
3642
3643
3644
3645
3646
3647
3648
3649
3650
3651
3652
3653
3654
3655
3656
3657
3658
3659
3660
3661
3662
3663
3664
3665
3666
3667
3668
3669
3670
3671
3672
3673
3674
3675
3676
3677
3678
3679
3680
3681
3682
3683
3684
3685
3686
3687
3688
3689
3690
3691
3692
3693
3694
3695
3696
3697
3698
3699
3700
3701
3702
3703
3704
3705
3706
3707
3708
3709
3710
3711
3712
3713
3714
3715
3716
3717
3718
3719
3720
3721
3722
3723
3724
3725
3726
3727
3728
3729
3730
3731
3732
3733
3734
3735
3736
3737
3738
3739
3740
3741
3742
3743
3744
3745
3746
3747
3748
3749
3750
3751
3752
3753
3754
3755
3756
3757
3758
3759
3760
3761
3762
3763
3764
3765
3766
3767
3768
3769
3770
3771
3772
3773
3774
3775
3776
3777
3778
3779
3780
3781
3782
3783
3784
3785
3786
3787
3788
3789
3790
3791
3792
3793
3794
3795
3796
3797
3798
3799
3800
3801
3802
3803
3804
3805
3806
3807
3808
3809
3810
3811
3812
3813
3814
3815
3816
3817
3818
3819
3820
3821
3822
3823
3824
3825
3826
3827
3828
3829
3830
3831
3832
3833
3834
3835
3836
3837
3838
3839
3840
3841
3842
3843
3844
3845
3846
3847
3848
3849
3850
3851
3852
3853
3854
3855
3856
3857
3858
3859
3860
3861
3862
3863
3864
3865
3866
3867
3868
3869
3870
3871
3872
3873
3874
3875
3876
3877
3878
3879
3880
3881
3882
3883
3884
3885
3886
3887
3888
3889
3890
3891
3892
3893
3894
3895
3896
3897
3898
3899
3900
3901
3902
3903
3904
3905
3906
3907
3908
3909
3910
3911
3912
3913
3914
3915
3916
3917
3918
3919
3920
3921
3922
3923
3924
3925
3926
3927
3928
3929
3930
3931
3932
3933
3934
3935
3936
3937
3938
3939
3940
3941
3942
3943
3944
3945
3946
3947
3948
3949
3950
3951
3952
3953
3954
3955
3956
3957
3958
3959
3960
3961
3962
3963
3964
3965
3966
3967
3968
3969
3970
3971
3972
3973
3974
3975
3976
3977
3978
3979
3980
3981
3982
3983
3984
3985
3986
3987
3988
3989
3990
3991
3992
3993
3994
3995
3996
3997
3998
3999
4000
4001
4002
4003
4004
4005
4006
4007
4008
4009
4010
4011
4012
4013
4014
4015
4016
4017
4018
4019
4020
4021
4022
4023
4024
4025
4026
4027
4028
4029
4030
4031
4032
4033
4034
4035
4036
4037
4038
4039
4040
4041
4042
4043
4044
4045
4046
4047
4048
4049
4050
4051
4052
4053
4054
4055
4056
4057
4058
4059
4060
4061
4062
4063
4064
4065
4066
4067
4068
4069
4070
4071
4072
4073
4074
4075
4076
4077
4078
4079
4080
4081
4082
4083
4084
4085
4086
4087
4088
4089
4090
4091
4092
4093
4094
4095
4096
4097
4098
4099
4100
4101
4102
4103
4104
4105
4106
4107
4108
4109
4110
4111
4112
4113
4114
4115
4116
4117
4118
4119
4120
4121
4122
4123
4124
4125
4126
4127
4128
4129
4130
4131
4132
4133
4134
4135
4136
4137
4138
4139
4140
4141
4142
4143
4144
4145
4146
4147
4148
4149
4150
4151
4152
4153
4154
4155
4156
4157
4158
4159
4160
4161
4162
4163
4164
4165
4166
4167
4168
4169
4170
4171
4172
4173
4174
4175
4176
4177
4178
4179
4180
4181
4182
4183
4184
4185
4186
4187
4188
4189
4190
4191
4192
4193
4194
4195
4196
4197
4198
4199
4200
4201
4202
4203
4204
4205
4206
4207
4208
4209
4210
4211
4212
4213
4214
4215
4216
4217
4218
4219
4220
4221
4222
4223
4224
4225
4226
4227
4228
4229
4230
4231
4232
4233
4234
4235
4236
4237
4238
4239
4240
4241
4242
4243
4244
4245
4246
4247
4248
4249
4250
4251
4252
4253
4254
4255
4256
4257
4258
4259
4260
4261
4262
4263
4264
4265
4266
4267
4268
4269
4270
4271
4272
4273
4274
4275
4276
4277
4278
4279
4280
4281
4282
4283
4284
4285
4286
4287
4288
4289
4290
4291
4292
4293
4294
4295
4296
4297
4298
4299
4300
4301
4302
4303
4304
4305
4306
4307
4308
4309
4310
4311
4312
4313
4314
4315
4316
4317
4318
4319
4320
4321
4322
4323
4324
4325
4326
4327
4328
4329
4330
4331
4332
4333
4334
4335
4336
4337
4338
4339
4340
4341
4342
4343
4344
4345
4346
4347
4348
4349
4350
4351
4352
4353
4354
4355
4356
4357
4358
4359
4360
4361
4362
4363
4364
4365
4366
4367
4368
4369
4370
4371
4372
4373
4374
4375
4376
4377
4378
4379
4380
4381
4382
4383
4384
4385
4386
4387
4388
4389
4390
4391
4392
4393
4394
4395
4396
4397
4398
4399
4400
4401
4402
4403
4404
4405
4406
4407
4408
4409
4410
4411
4412
4413
4414
4415
4416
4417
4418
4419
4420
4421
4422
4423
4424
4425
4426
4427
4428
4429
4430
4431
4432
4433
4434
4435
4436
4437
4438
4439
4440
4441
4442
4443
4444
4445
4446
4447
4448
4449
4450
4451
4452
4453
4454
4455
4456
4457
4458
4459
4460
4461
4462
4463
4464
4465
4466
4467
4468
4469
4470
4471
4472
4473
4474
4475
4476
4477
4478
4479
4480
4481
4482
4483
4484
4485
4486
4487
4488
4489
4490
4491
4492
4493
4494
4495
4496
4497
4498
4499
4500
4501
4502
4503
4504
4505
4506
4507
4508
4509
4510
4511
4512
4513
4514
4515
4516
4517
4518
4519
4520
4521
4522
4523
4524
4525
4526
4527
4528
4529
4530
4531
4532
4533
4534
4535
4536
4537
4538
4539
4540
4541
4542
4543
4544
4545
4546
4547
4548
4549
4550
4551
4552
4553
4554
4555
4556
4557
4558
4559
4560
4561
4562
4563
4564
4565
4566
4567
4568
4569
4570
4571
4572
4573
4574
4575
4576
4577
4578
4579
4580
4581
4582
4583
4584
4585
4586
4587
4588
4589
4590
4591
4592
4593
4594
4595
4596
4597
4598
4599
4600
4601
4602
4603
4604
4605
4606
4607
4608
4609
4610
4611
4612
4613
4614
4615
4616
4617
4618
4619
4620
4621
4622
4623
4624
4625
4626
4627
4628
4629
4630
4631
4632
4633
4634
4635
4636
4637
4638
4639
4640
4641
4642
4643
4644
4645
4646
4647
4648
4649
4650
4651
4652
4653
4654
4655
4656
4657
4658
4659
4660
4661
4662
4663
4664
4665
4666
4667
4668
4669
4670
4671
4672
4673
4674
4675
4676
4677
4678
4679
4680
4681
4682
4683
4684
4685
4686
4687
4688
4689
4690
4691
4692
4693
4694
4695
4696
4697
4698
4699
4700
4701
4702
4703
4704
4705
4706
4707
4708
4709
4710
4711
4712
4713
4714
4715
4716
4717
4718
4719
4720
4721
4722
4723
4724
4725
4726
4727
4728
4729
4720
4731
4732
4733
4734
4735
4736
4737
4738
4739
4730
4741
4742
4743
4744
4745
4746
4747
4748
4749
4740
4751
4752
4753
4754
4755
4756
4757
4758
4759
4750
4761
4762
4763
4764
4765
4766
4767
4768
4769
4760
4771
4772
4773
4774
4775
4776
4777
4778
4779
4770
4781
4782
4783
4784
4785
4786
4787
4788
4789
4780
4791
4792
4793
4794
4795
4796
4797
4798
4799
4790
4800
4801
4802
4803
4804
4805
4806
4807
4808
4809
4800
4810
4811
4812
4813
4814
4815
4816
4817
4818
4819
4810
4820
4821
4822
4823
4824
4825
4826
4827
4828
4829
4820
4830
4831
4832
4833
4834
4835
4836
4837
4838
4839
4830
4840
4841
4842
4843
4844
4845
4846
4847
4848
4849
4840
4850
4851
4852
4853
4854
4855
4856
4857
4858
4859
4850
4860
4861
4862
4863
4864
4865
4866
4867
4868
4869
4860
4870
4871
4872
4873
4874
4875
4876
4877
4878
4879
4870
4880
4881
4882
4883
4884
4885
4886
4887
4888
4889
4880
4890
4891
4892
4893
4894
4895
4896
4897
4898
4899
4890
4900
4901
4902
4903
4904
4905
4906
4907
4908
4909
4900
4910
4911
4912
4913
4914
4915
4916
4917
4918
4919
4910
4920
4921
4922
4923
4924
4925
4926
4927
4928
4929
4920
4930
4931
4932
4933
4934
4935
4936
4937
4938
4939
4930
4940
4941
4942
4943
4944
4945
4946
4947
4948
4949
4940
4950
4951
4952
4953
4954
4955
4956
4957
4958
4959
4950
4960
4961
4962
4963
4964
4965
4966
4967
4968
4969
4960
4970
4971
4972
4973
4974
4975
4976
4977
4978
4979
4970
4980
4981
4982
4983
4984
4985
4986
4987
4988
4989
4980
4990
4991
4992
4993
4994
4995
4996
4997
4998
4999
4990
5000
5001
5002
5003
5004
5005
5006
5007
5008
5009
5000
5010
5011
5012
5013
5014
5015
5016
5017
5018
5019
5010
5020
5021
5022
5023
5024
5025
5026
5027
5028
5029
5020
5030
5031
5032
5033
5034
5035
5036
5037
5038
5039
5030
5040
5041
5042
5043
5044
5045
5046
5047
5048
5049
5040
5050
5051
5052
5053
5054
5055
5056
5057
5058
5059
5050
5060
5061
5062
5063
5064
5065
5066
5067
5068
5069
5060
5070
5071
5072
5073
5074
5075
5076
5077
5078
5079
5070
5080
5081
5082
5083
5084
5085
5086
5087
5088
5089
5080
5090
5091
5092
5093
5094
5095
5096
5097
5098
5099
5090
5100
5101
5102
5103
5104
5105
5106
5107
5108
5109
5100
5110
5111
5112
5113
5114
5115
5116
5117
5118
```

```
3097 <login-config>
3098   <auth-method>BASIC</auth-method>
3099   <realm-name>myrealm</realm-name>
3100 </login-config>
3101 </web-app>
3102 =====
3103 WebApplication on Declarative Security
3104 App31: (ON FORM Model)
3105 -----login.jsp-----
3106 <html>
3107   <head>
3108     <title>Security WebApp login page</title>
3109   </head>
3110   <body bgcolor="#cccccc">
3111     <blockquote>
3112       <h2>Please enter your user name and password:</h2>
3113       <p>
3114         <form method="POST" action="j_security_check">
3115           <table border=1>
3116             <tr>
3117               <td>Username:</td>
3118               <td><input type="text" name="j_username"></td>
3119             </tr>
3120             <tr>
3121               <td>Password:</td>
3122               <td><input type="password" name="j_password"></td>
3123             </tr>
3124             <tr>
3125               <td colspan=2 align=right><input type=submit
3126                               value="Submit"></td>
3127             </tr>
3128           </table>
3129         </form>
3130       </blockquote>
3131     </body>
3132 </html>
3133 -----login_fail.jsp-----
3134 <html>
3135   <head>
3136     <title>Login failed</title>
3137   </head>
3138   <body bgcolor="#ffffff">
3139     <blockquote>
3140       <h2>Sorry, your user name and password were not recognized.</h2>
3141       <p><b>
3142         <a href="login.jsp">Return to welcome page</a> </b>
3143       </blockquote>
3144     </body>
3145 </html>
3146 -----edit.jsp-----
3147 <html>
3148 <body>
3149 <%
3150 out.println("<center><b>the user name is "+request.getRemoteUser()+"</center></b>");
3151 %>
3152 <center><b>login successfull... welcome to home page</center></b>
3153 </body>
3154 -----web.xml-----
3155 <!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
3156 "http://java.sun.com/dtd/web-app_2_3.dtd">
3157 <web-app>
3158
3159   <servlet>
3160     <servlet-name>servlet2</servlet-name>
3161     <jsp-file>/edit.jsp</jsp-file>
3162   </servlet>
```

```
3163 <servlet-mapping>
3164     <servlet-name> servlet2</servlet-name>
3165     <url-pattern>/test2</url-pattern>
3166 </servlet-mapping>
3167
3168 <security-constraint>
3169     <web-resource-collection>
3170         <web-resource-name>Authentication1</web-resource-name>
3171         <url-pattern>/test2</url-pattern>
3172         <http-method>GET</http-method>
3173     </web-resource-collection>
3174     <auth-constraint>
3175         <role-name>manager</role-name>
3176     </auth-constraint>
3177 </security-constraint>
3178
3179 <login-config>
3180     <form-login-config>
3181         <form-login-page>/login.jsp</form-login-page>
3182         <form-error-page>/login_fail.jsp</form-error-page>
3183     </form-login-config>
3184     <auth-method>FORM</auth-method>
3185     <realm-name>myrealm</realm-name>
3186 </login-config>
3187 </web-app>
3188
3189 =====
3190 =====
3191 App32) Example App Servlet wrappers
3192 =====
3193 -----Login.html-----
3194 <form action="loginurl">
3195     User : <input type="text" name="user"> @gmail.com<br>
3196     Password : <input type="password" name="pwd"><br>
3197     <input type="submit" value="Login">
3198 </form>
3199 -----web.xml-----
3200 <?xml version="1.0" encoding="UTF-8"?>
3201 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3202   xmlns="http://xmlns.jcp.org/xml/ns/javaee"
3203   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
3204   http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
3205   id="WebApp_ID" version="3.1">
3206     <display-name>WrapperApp</display-name>
3207     <welcome-file-list>
3208       <welcome-file>Login.html</welcome-file>
3209     </welcome-file-list>
3210
3211     <servlet>
3212       <servlet-name>login</servlet-name>
3213       <servlet-class>com.nt.servlet.LoginSry</servlet-class>
3214     </servlet>
3215
3216     <servlet-mapping>
3217       <servlet-name>login</servlet-name>
3218       <url-pattern>/loginurl</url-pattern>
3219     </servlet-mapping>
3220
3221     <filter>
3222       <filter-name>loginF</filter-name>
3223       <filter-class>com.nt.filter.LoginFilter</filter-class>
3224     </filter>
3225     <filter-mapping>
3226       <filter-name>loginF</filter-name>
```

```
3226     <url-pattern>/loginurl</url-pattern>
3227     </filter-mapping>
3228
3229 </web-app>
3230 -----LoginSrv.java-----
3231
3232 package com.nt.servlet;
3233
3234 import java.io.IOException;
3235 import java.io.PrintWriter;
3236
3237 import javax.servlet.ServletException;
3238 import javax.servlet.http.HttpServlet;
3239 import javax.servlet.http.HttpServletRequest;
3240 import javax.servlet.http.HttpServletResponse;
3241
3242 public class LoginSrv extends HttpServlet {
3243
3244     @Override
3245     protected void doGet(HttpServletRequest req, HttpServletResponse res)
3246         throws ServletException, IOException {
3247         //general settings
3248         PrintWriter pw=res.getWriter();
3249         res.setContentType("text/html");
3250         //read form data
3251         String uname=req.getParameter("user");
3252         String pwd=req.getParameter("pwd");
3253         //write Validation logic
3254         if(uname.equals("raja@gmail.com")&& pwd.equals("rani")){
3255             pw.println("<h1> Valid Credentials </h1>");
3256         }
3257         else{
3258             pw.println("<h1> InValid Credentials </h1>");
3259         }
3260         pw.println("<br>"+uname+" "+pwd);
3261         //close stream
3262         pw.close();
3263     }//doGet(-,-)
3264
3265     @Override
3266     protected void doPost(HttpServletRequest req, HttpServletResponse res)
3267         throws ServletException, IOException {
3268         doGet(req,res);
3269     }
3270 -----LoginFilter.java-----
```

```
3271 package com.nt.filter;
3272
3273 import java.io.IOException;
3274 import java.io.PrintWriter;
3275
3276 import javax.servlet.Filter;
3277 import javax.servlet.FilterChain;
3278 import javax.servlet.FilterConfig;
3279 import javax.servlet.ServletException;
3280 import javax.servlet.ServletRequest;
3281 import javax.servlet.ServletResponse;
3282 import javax.servlet.http.HttpServlet;
3283 import javax.servlet.http.HttpServletRequest;
3284
3285 import com.nt.wrappers.MyRequest;
3286 import com.nt.wrappers.MyResponse;
3287
3288 public class LoginFilter implements Filter {
```

```

3290     @Override
3291     public void destroy() {
3292         // TODO Auto-generated method stub
3293     }
3294
3295     @Override
3296     public void doFilter(ServletRequest req, ServletResponse res,
3297                         FilterChain fc) throws IOException, ServletException {
3298         //Create Custom Request obj
3299         HttpServletRequest hreq=(HttpServletRequest)req;
3300         MyRequest mreq=new MyRequest(hreq);
3301
3302         //create Custom Response obj
3303         HttpServletResponse hres=(HttpServletResponse)res;
3304         MyResponse mres=new MyResponse(hres);
3305
3306         fc.doFilter(mreq,mres);
3307         //Get Servlet response
3308         String result=mres.toString().toUpperCase();
3309         //add signature at the end
3310         PrintWriter pw=res.getWriter();
3311         pw.println(result+"<br><br>-----<br> Naresh IT, Ammeerpet.");
3312
3313
3314     }
3315
3316
3317     @Override
3318     public void init(FilterConfig arg0) throws ServletException {
3319         // TODO Auto-generated method stub
3320     }
3321 }
3322
-----MyRequest.java-----
3323
3324 package com.nt.wrappers;
3325
3326 import javax.servlet.http.HttpServletRequest;
3327 import javax.servlet.http.HttpServletRequestWrapper;
3328
3329 public class MyRequest extends HttpServletRequestWrapper {
3330     HttpServletRequest request;
3331     public MyRequest(HttpServletRequest request) {
3332         super(request);
3333         this.request=request;
3334     }
3335
3336     @Override
3337     public String getParameter(String name) {
3338         String pval=request.getParameter(name);
3339         if(name.equals("user")){
3340             if(pval.indexOf("@gmail.com")==-1){
3341                 pval=pval+"@gmail.com";
3342                 return pval;
3343             }
3344         }
3345         return pval;
3346     }
3347 }
3348
-----MyResponse.java-----
3349
3350 package com.nt.wrappers;
3351
3352 import java.ioCharArrayWriter;
3353 import java.io.IOException;
3354 import java.io.PrintWriter;

```

```
3354  
3355 import javax.servlet.http.HttpServletResponse;  
3356 import javax.servlet.http.HttpServletResponseWrapper;  
3357  
3358 public class MyResponse extends HttpServletResponseWrapper {  
3359  
3360     HttpServletResponse response;  
3361     CharArrayWriter charWriter;  
3362  
3363     public MyResponse(HttpServletResponse response) {  
3364         super(response);  
3365         this.response=response;  
3366         charWriter=new CharArrayWriter();  
3367     }  
3368  
3369     @Override  
3370     public PrintWriter getWriter() throws IOException {  
3371         PrintWriter writer=new PrintWriter(charWriter);  
3372         return writer;  
3373     }  
3374     @Override  
3375     public String toString() {  
3376         return charWriter.toString();  
3377     }  
3378 }  
3379  
3380  
3381
```

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 :	<input type="text" value="Raaj"/>
Age :	<input type="text" value="23"/>
Father's Name:	<input type="text" value="Ramesh"/>
Marital Status:	<input type="text" value="married"/>
Spouse Name:	<input type="text"/>
No. of Children:	<input type="text"/>
When to Marry:	<input type="text" value="After 2 Years"/>
Why to Marry :	<input type="text" value="To Get Personal Friend"/>
<input type="button" value="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.

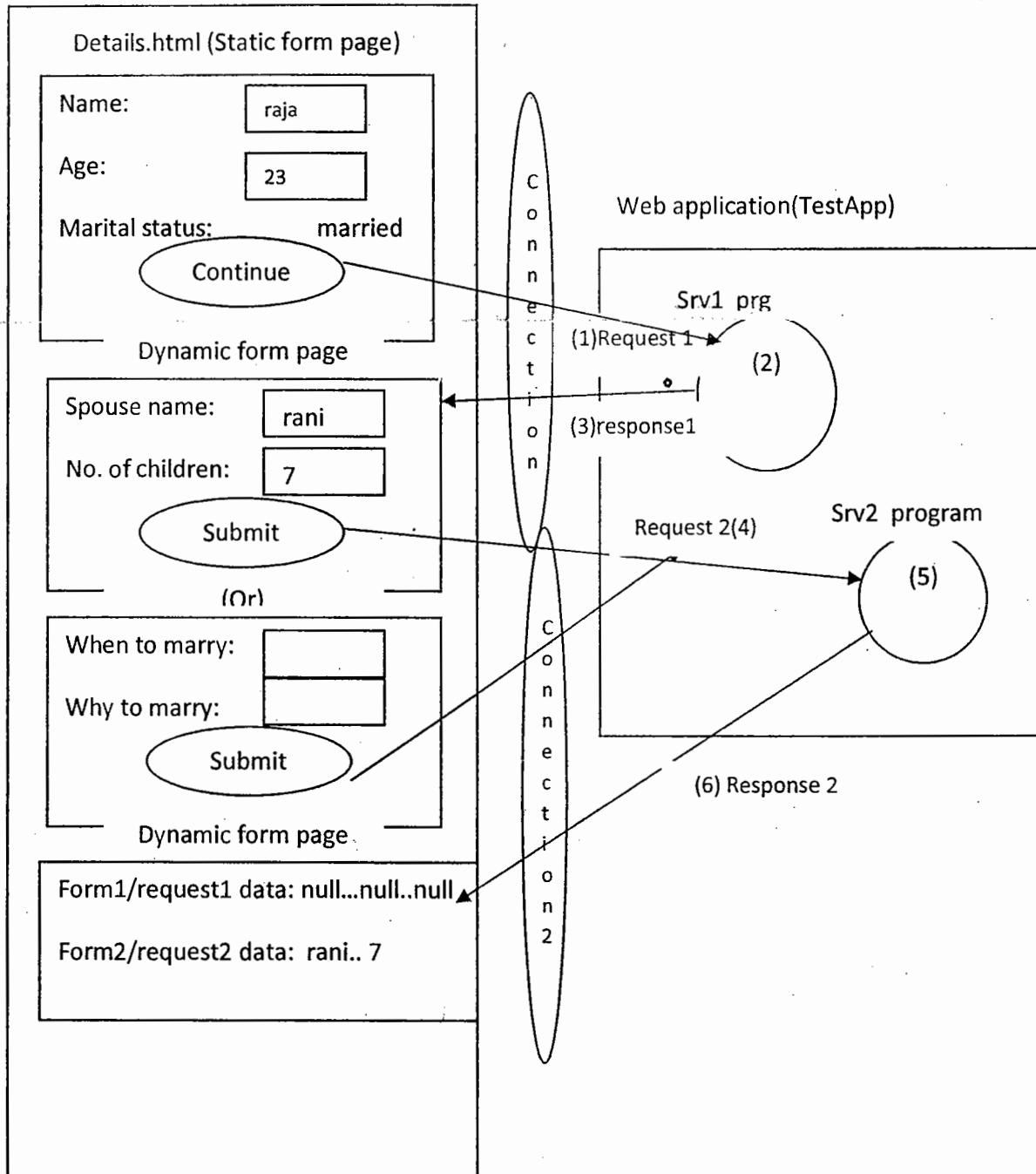
To overcome this problem ask common question in first form page which is static form page and generate second form page dynamically based on the content given in first form page, then end user will not be forced to read and ignore certain questions of form pages.

Set of continuous 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.

Browser window



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 demonistrate 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
{
```

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 to 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 accessible while processing another connection related request given by client. The above whole discussion says 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:

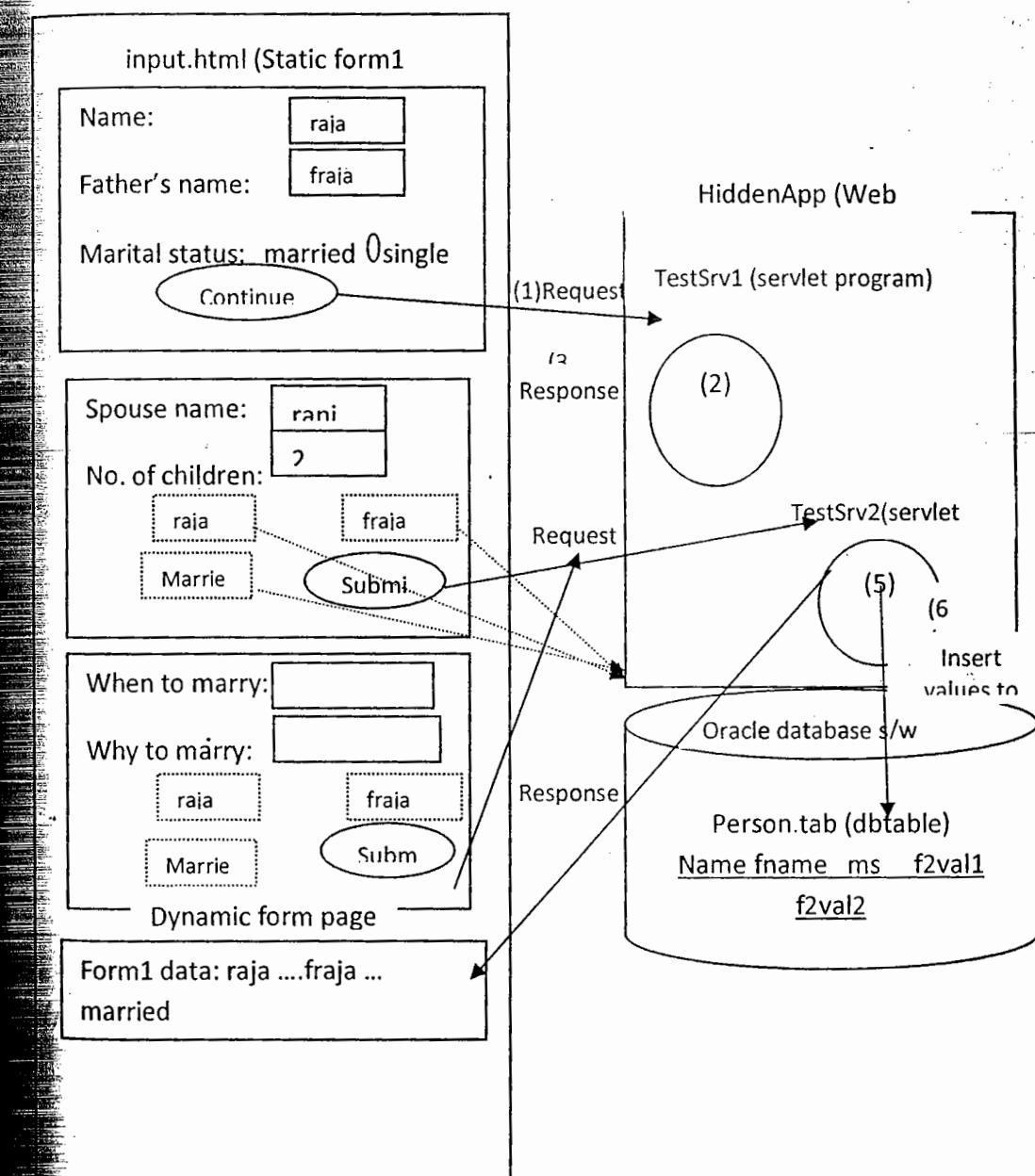
```
<input type="hidden" name="h1" value="hello">
    Hidden box name → hidden box value/
    Or                               request parameter value
    Request parameter Name
```

In servlet program:

```
String val1=req.getParameter("h1"); //gives "hello"
```

Session management or session tracking both are same.

Browser window



Example :

App21(SessionTracking using HiddenForm Fields)

-----input.html-----

```
<form action="t1url" method="get">
    Name: <input type="text" name="tname"><br>
    Father name: <input type="text" name="tfname"><br>
    Marital Status <input type="radio" name="ms" value="married">Married
    &nbsp;&nbsp;&nbsp;
    <input type="radio" name="ms" value="single">Single <br>

    <input type="submit" value="continue">
</form>
```

-----web.xml-----

```
<web-app>
    <servlet>
        <servlet-name>abc1</servlet-name>
        <servlet-class>com.nt.TestSrv1</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>abc1</servlet-name>
        <url-pattern>/t1url</url-pattern>
    </servlet-mapping>

    <servlet>
        <servlet-name>xyz1</servlet-name>
        <servlet-class>com.nt.TestSrv2</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>xyz1</servlet-name>
        <url-pattern>/t2url</url-pattern>
    </servlet-mapping>
</web-app>
```

-----TestSrv1.java-----

```
//TestSrv1.java (generates dynamice form page)
package com.nt;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class TestSrv1 extends HttpServlet
{
    public void service(HttpServletRequest req,HttpServletResponse res) throws
    ServletException,IOException
```

```

{
    // general code
    PrintWriter pw=res.getWriter();
    res.setContentType("text/html");

    // read form1 data
    String pname=req.getParameter("tname");
    String pfname=req.getParameter("tfname");
    String pms=req.getParameter("ms");

    //design and send dynamic form page to browser window
    // based on the marital status....
    if(pms.equals("single"))
    {
        pw.println("<form action='t2url' method='get'>");
        pw.println("why do u want to marry <input type='text' name='f2t1'><br>");
        pw.println("when do u want to marry <input type='text' name='f2t2'><br>");
        // add form1/req1 data to dynamic form page as hidden box values
        pw.println("<input type='hidden' name='hname' value='"+pname+">");
        pw.println("<input type='hidden' name='hfname' value='"+pfname+">");
        pw.println("<input type='hidden' name='hms' value='"+pms+">");

        pw.println("<input type='submit' value='submit'>");
        pw.println("</form>");
    }
    else
    {
        pw.println("<form action='t2url' method='get'>");
        pw.println("Spouse name<input type='text' name='f2t1'><br>");
        pw.println("no.of children <input type='text' name='f2t2'><br>");
        // add form1/req1 data to dynamic form page as hidden box values
        pw.println("<input type='hidden' name='hname' value='"+pname+">");
        pw.println("<input type='hidden' name='hfname' value='"+pfname+">");
        pw.println("<input type='hidden' name='hms' value='"+pms+">");

        pw.println("<input type='submit' value='submit'>");
        pw.println("</form>");
    }
}

//close the stream obj
pw.close();
}//service
//class

```

TestSrv2.java-----

```

//TestSrv2.java
package com.nt;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.sql.*;

public class TestSrv2 extends HttpServlet

```

```

}

    public void service(HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException
    {
        // general code
        PrintWriter pw = res.getWriter();
        res.setContentType("text/html");

        // read form1 data
        String name = req.getParameter("hname");
        String fname = req.getParameter("hfname");
        String ms = req.getParameter("hms");
        // read form2 data
        String f2val1 = req.getParameter("f2t1");
        String f2val2 = req.getParameter("f2t2");

        // write form1/req1,from2/req2 values to DB table as record
        try
        {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
            PreparedStatement ps = con.prepareStatement("insert into person_tab
values(?,?,?,?,?)");
            ps.setString(1, name);
            ps.setString(2, fname);
            ps.setString(3, ms);
            ps.setString(4, f2val1);
            ps.setString(5, f2val2);

            ps.executeUpdate();

            pw.println("<br>Record Inserted<br>");
        }
        //try
        catch(Exception e)
        {e.printStackTrace(); }

        // display form1 ,form2 data on the browser window
        pw.println("<br> form1 data is "+name+" "+fname+".... "+ms);
        pw.println("<br> form2 data is "+f2val1+" "+f2val2);

        //close stream obj
        pw.close();
    }//service
}//class

```

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.

state?

What is the difference between session management/session tracking and management?

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, Http session, url rewriting kind of techniques will be utilized.

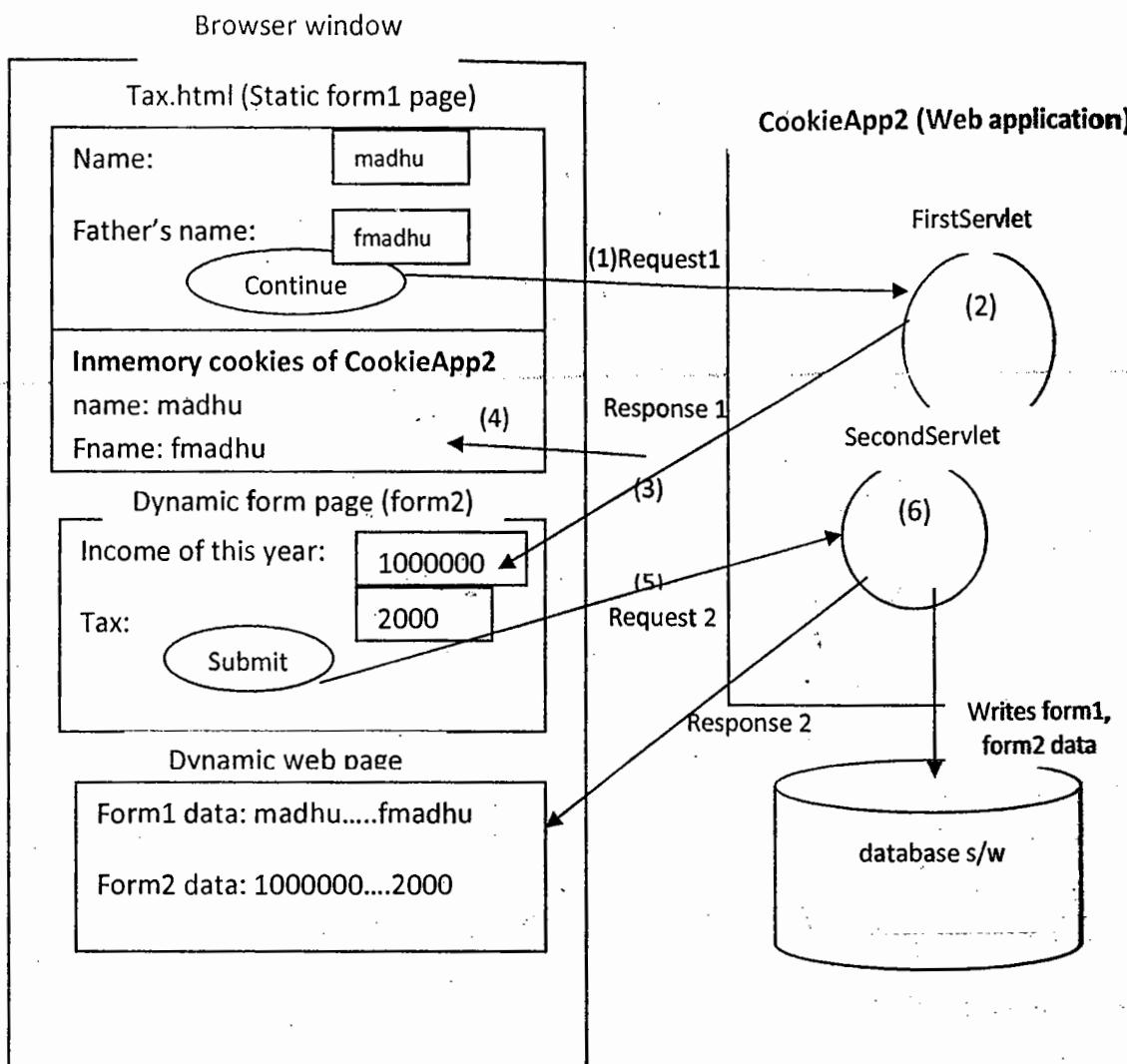
The session started between browser window and web application is specific to each browser window (client). Multiple clients start multiple sessions with single web application on one per client basis. Session and session related data always specific to one client so only that client can use session 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.

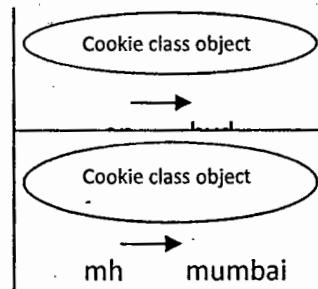
To modify cookie value:

```
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
```

Ck[](array of type cookie class



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

Window Username no. of cookies
Web application name

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.

Examples:

App22(Basic Example on Cookies)

SetCookiesSrv.java

```
package com.nt;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SetCookiesSrv extends HttpServlet {

    public void doGet(HttpServletRequest request,HttpServletResponse response)
        throws ServletException,IOException {

        Cookie cookie1,cookie2,cookie3,cookie4;
        // default maxage is -1, indicating cookie
        // 2 in-memory cookies applies only to current browsing session
        cookie1=new Cookie("ts","Hyderabad");
        cookie2=new Cookie("tn","Chennai");
        response.addCookie(cookie1);
        response.addCookie(cookie2);

        // Cookie is valid for an hour, regardless of whether
        // user quits browser, reboots computer or whatever
        // 2 persistent cookies
        cookie3=new Cookie("kr","Bangalore");
    }
}
```

```

cookie4=new Cookie("mh","mumbai");
cookie3.setMaxAge(3600);
cookie4.setMaxAge(3600);

response.addCookie(cookie3);
response.addCookie(cookie4);

//get PrintWriter obj
PrintWriter pw=res.getWriter();
response.setContentType("text/html");

pw.println("Successful in setting cookies");
System.out.println("Successful in setting cookies....");
}
//doGet()
}
//class
-----ShowCookiesSrv.java-----

```

```

// program to work with cookies
package com.nt;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ShowCookiesSrv extends HttpServlet {

    public void doGet(HttpServletRequest request,HttpServletResponse response)
        throws ServletException,IOException {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();
        String title="Active cookies";

        out.println("<html> <head> <title>" + title);
        out.println("</title> </head> <body>");
        out.println("<table border=1 align=\"center\">");
        out.println("<tr> <td> Cookie Name</td>");
        out.println("<td> Cookie Value</td></tr>");

        Cookie[] ck=request.getCookies();
        if(ck!=null) {
            for (int i=0;i<ck.length;i++) {
                out.println("<tr><td>" + ck[i].getName() + "</td>"
                + "<td>" + ck[i].getValue() + "</td></tr>");
            }
        }
        out.println("</table></body></html>");
    }
    else {
        System.out.println("No cookies present....");
        pw.println("<br><b>no cookies present.....</b>");
    }
}

```

```

        } //else

    } //doGet()
} //class
----- web.xml -----
<web-app>
    <servlet>
        <servlet-name>s1</servlet-name>
        <servlet-class>com.nt.SetCookiesSrv</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>s1</servlet-name>
        <url-pattern>/test1</url-pattern>
    </servlet-mapping>

    <servlet>
        <servlet-name>s2</servlet-name>
        <servlet-class>com.nt.ShowCookiesSrv</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>s2</servlet-name>
        <url-pattern>/test2</url-pattern>
    </servlet-mapping>
</web-app>
-----
```

App23(SessionTracking using Cookies)

```

----- Tax.html -----
<html>
    <body bgcolor="#ffffff">
        <form action="furl" method=GET>
            Name <input type=text name=name> <BR>
            Father name <input type=text name=Fname>
            <input type=submit value = continue>
        </form>
    </font>
    </body>
</html>
```

FirstServlet.java

```

package com.nt;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

public class FirstServlet extends HttpServlet

{

```

public void doGet (HttpServletRequest      request,HttpServletResponse  response)
                  throws ServletException, IOException
{
    // set response content type
    response.setContentType("text/html");
    // get PrintWriter obj
    PrintWriter out = response.getWriter();

    //get values submitted by the form1
    String s1 = request.getParameter("name");
    String s2 = request.getParameter("Fname");

    // create cookies to store the form1 data
    Cookie ck1 = new Cookie("name",s1);
    Cookie ck2 = new Cookie("fname",s2);

    response.addCookie(ck1);
    response.addCookie(ck2);

    //now we need to generate the second form dynamically from here
    out.print("<form action='surl' method=get>");
    out.print("income for this year <input type=text name=income> <BR>");
    out.print(" Tax<input type=text name=tax>");
    out.print("<br><br> <BR><BR><input type=submit value = submit>");
    out.print("</form>");
    //close streams
    out.close();
}

public void doPost (HttpServletRequest      request,HttpServletResponse  response)
                   throws ServletException, IOException
    doGet(request,response);
}

```

-----SecondServlet.java-----

```

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

public class SecondServlet extends HttpServlet
{
    public void doGet (HttpServletRequest      request,HttpServletResponse  response)
                      throws ServletException, IOException
    {
        // set content type
        response.setContentType("text/html");

```

```

// get PrintWriter obj
PrintWriter out = response.getWriter();

//get values submitted by the form2
String income = request.getParameter("income");
String tax = request.getParameter("tax");

//read form1/req1 data from cookie
Cookie ck[] = request.getCookies();
String name=null, fname=null;
if(ck!=null)
{
    name=ck[0].getValue();
    fname=ck[1].getValue();
}//if

// use jdbc code to store form1/req1 and form2/req2 values in Db table
try{
    //register jdbc driver and establish the connection
    Class.forName("oracle.jdbc.driver.OracleDriver");
    Connection
    con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
    //create PreparedStatement obj pointing to SQL Query
    PreparedStatement ps=con.prepareStatement("insert into tax_tab
values(?,?,?,?,?)");
    //set values to Query Params
    ps.setString(1, name);
    ps.setString(2, fname);
    ps.setString(3, income);
    ps.setString(4, tax);
    //execute the Query
    ps.executeUpdate();
    out.println("<br>Record has been Inserted");
}//try
catch(Exception e)
{
    e.printStackTrace();
}

//display both form1,form2 data
out.print("Form2 data"+income+"...."+tax+"<br>");
out.print("Form1 data"+name+"...."+fname+"<br>");
//close streams
out.close();
}//doGet(-,-)
public void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
}

```

```

    doGet(request,response);
}
}//class

```

-----web.xml-----

```

<web-app>

    <servlet>
        <servlet-name>FF1</servlet-name>
        <servlet-class>com.nt.FirstServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>FF1</servlet-name>
        <url-pattern>/furl</url-pattern>
    </servlet-mapping>

    <servlet>
        <servlet-name>SF2</servlet-name>
        <servlet-class>com.nt.SecondServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>SF2</servlet-name>
        <url-pattern>/surl</url-pattern>
    </servlet-mapping>
</web-app>

```

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).

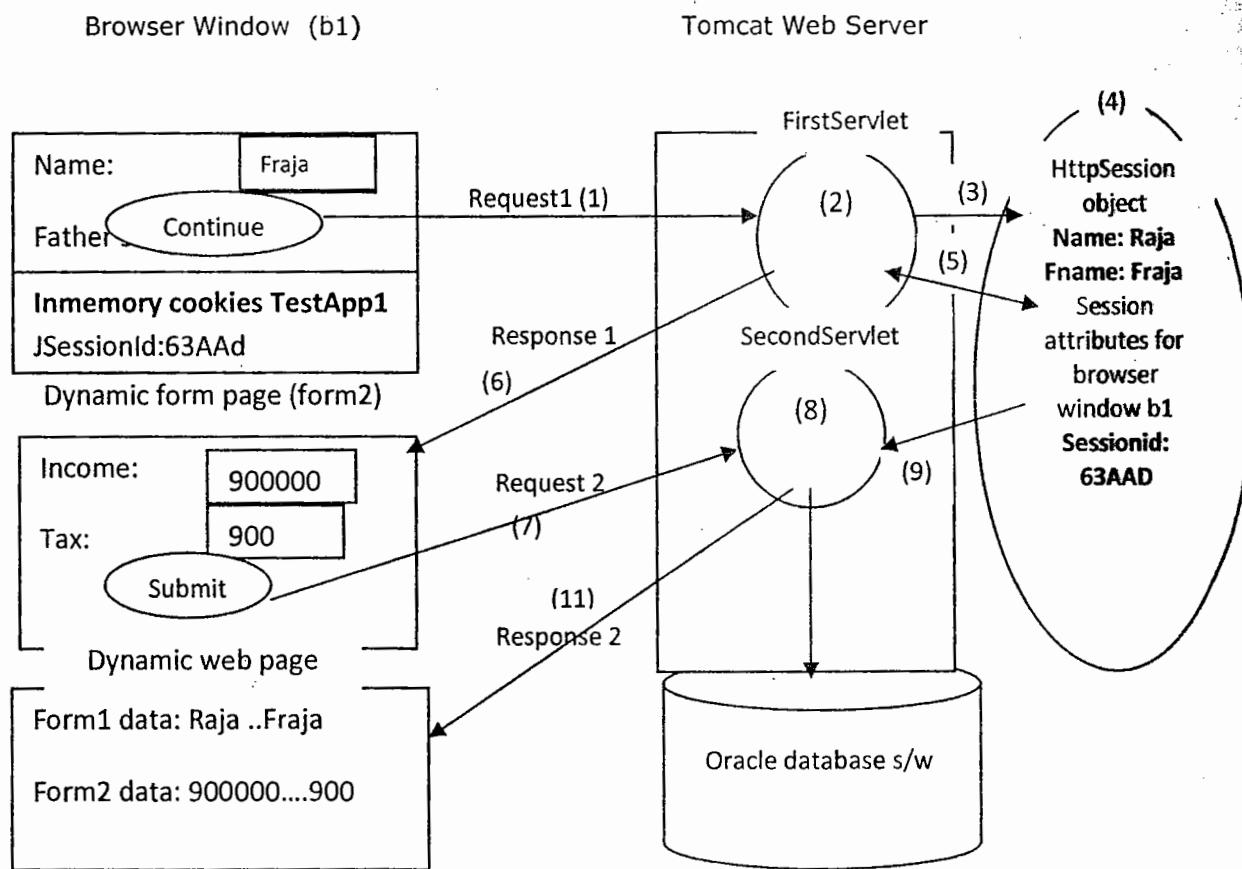
Point to remember:

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(SessionTracking).
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 1st 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 Http session

```
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 ii=(Integer)ses.getAttribute("age");
```

```
String name=(String)ses.getValue("name");
```

To remove existing session attributes:

ses.removeAttribute(-) or ses.removeAttribute(-) methods.

```
ses.removeAttribute("name");
```

```
ses.removeAttribute("age");or ses.removeAttribute("name");
```

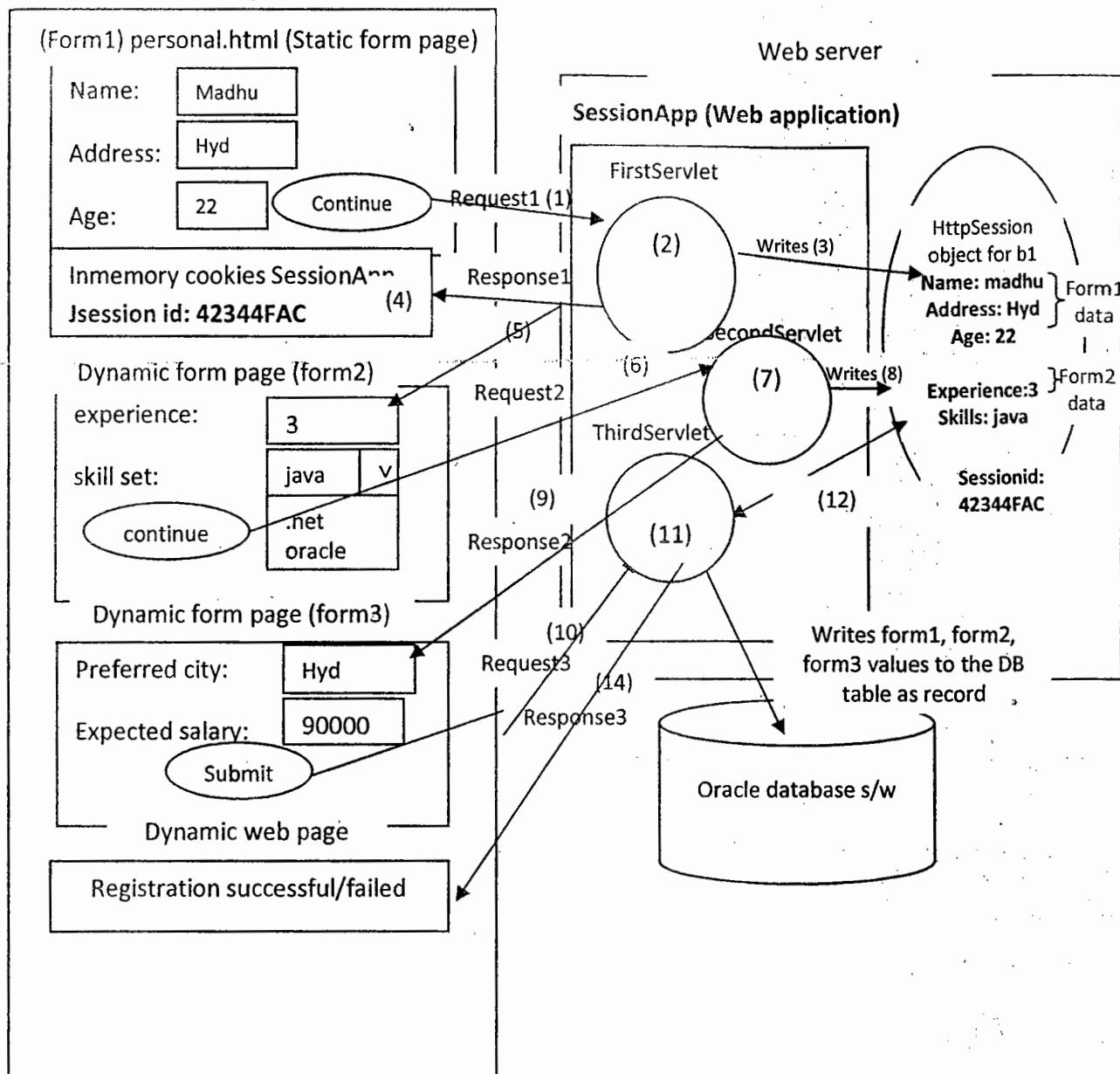
Point to remember:

ses.putValue(),ses.getValue(),ses.removeAttribute() 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.

Browser window(b1)



Example:

App24(Session Tracking using HttpSession with Cookies)**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 cookies </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-----
<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 service(HttpServletRequest req,
                        HttpServletResponse res) throws ServletException, IOException
    {
        //general settings
        res.setContentType("text/html");
        PrintWriter pw = res.getWriter();
        //read form1/request1 data
        String name=req.getParameter("name");
        String address=req.getParameter("address");
        String age=req.getParameter("age");
        //create Session for browser window
        HttpSession session = req.getSession(true);
        //keep form1/req1 data in session attributes
        session.setAttribute("name", name);
        session.setAttribute("Addr", address);
        session.setAttribute("age", age);

        // generate form2 dynamically
        pw.println("<BODY BGCOLOR=cyan>");
        pw.println("<CENTER><H1><FONT COLOR=red>Provide Your Exp
& Skills</FONT></H1></CENTER>\"");
        pw.println("<FORM ACTION='surl' METHOD=GET>");
        pw.println("<TABLE ALIGN=CENTER>");
        pw.println("<TR>");
```

--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 service(HttpServletRequest req, HttpServletResponse res)
            throws ServletException, IOException
    {
        //general settings
        res.setContentType( "text/html" );
        PrintWriter pw = res.getWriter( );

        //read form2/request2 data
        String exp=req.getParameter("exp");
        String skils=req.getParameter("skils");
        // get acces to Session obj
        HttpSession session = req.getSession(false);
```

```

// keep form2/req2 data in session attributes
session.setAttribute("exp", exp );
session.setAttribute("skils", skils );

// Generate form3 dynamically
pw.println("<BODY BGCOLOR=cyan>");
pw.println("<CENTER><H1><FONT COLOR=red>Provide City &
Salary</FONT></H1></CENTER>");

pw.println("<FORM ACTION='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>");
//close stream
pw.close();
} // 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
    {
        //general settings

```

```
res.setContentType( "text/html" );
PrintWriter pw = res.getWriter( );
// read form3/request3 data
String city=req.getParameter("city");
String sal=req.getParameter("sal");
// Get access to Session obj
HttpSession session = req.getSession(false);
// read form1/req1 and form2/req2 data from session attributes
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");

// insert form1/req1 , form2/req2 and form3 and req3 values as record
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,skils);
    pst.setString(6,city);
    pst.setString(7,sal);
    int i = pst.executeUpdate();
    //invalidate the session
    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>");
    }

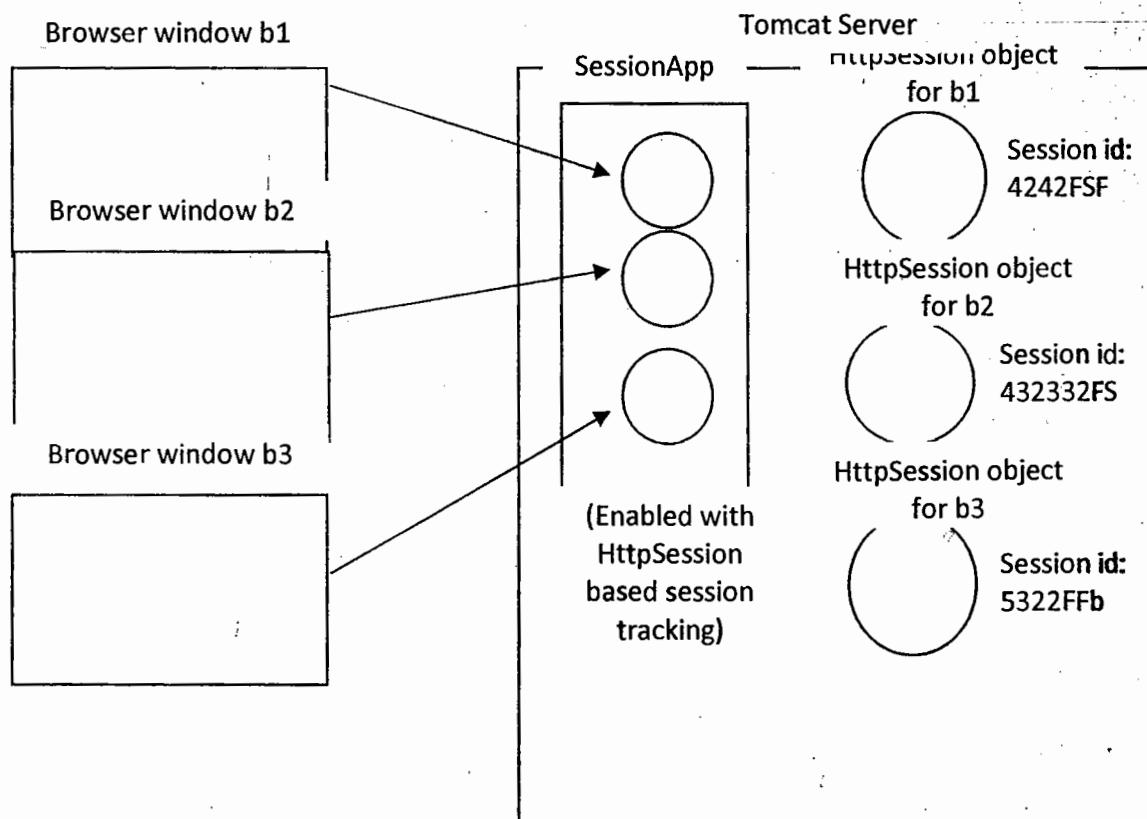
} // try
catch(Exception e)
{
    e.printStackTrace();
    pw.println("<BODY BGCOLOR=cyan>");
    pw.println("<CENTER><H1><FONT COLOR=red>Try
Again</FONT></H1></CENTER>");
    pw.println("<a href= personal.html>Home</a>");
}

} // doGet(-,-)
public void doPost(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException
doGet(req,res);
}
} // class

```

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.



What happens if underlying web server is restarted in the middle of the session that is started between browser window and web server?

 Surprisingly, session will be continued after restarting the server. Reason is, when server is restarted/shutted 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>");
        pw.println("<TABLE ALIGN= CENTER>");
        pw.println("<TR>");
        pw.println("<TD>");
        pw.println("<H2><FONT COLOR=BLUE>Enter Number of Years Exp :");
        pw.println("<INPUT TYPE=TEXT NAME=exp SIZE=6>");


```

-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>");
} // service

```

```

public void doPost(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException
doGet(req,res);
}
} // 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
    {
        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,skils);
    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, www.sblife.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?**

**A** 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 setStauts 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 requests 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

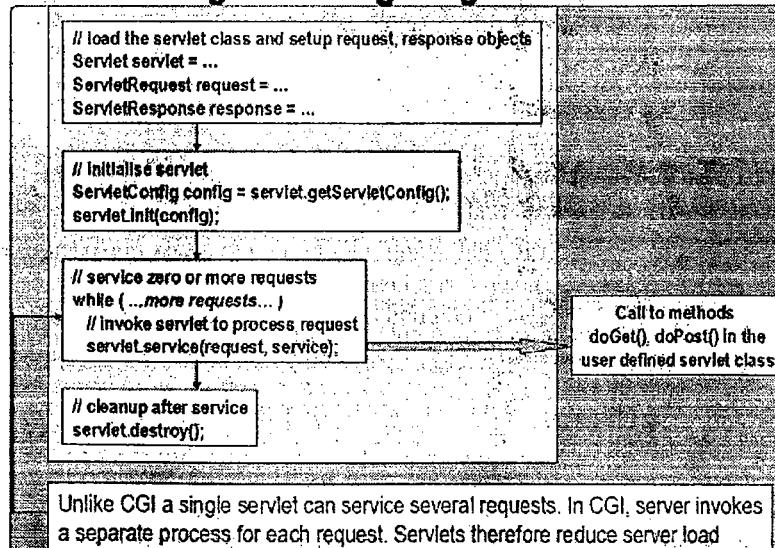
#### 5. What are the phases of the servlet life cycle?

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.

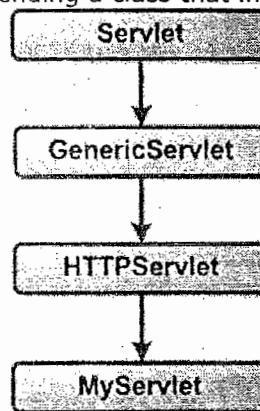
## 8. How a Servlet is unloaded?

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 POST 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 ?**

# SERVLET EXAMPLE

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

```
public class ServWelcome extends HttpServlet
```

```
{ public void doGet(HttpServletRequest request, HttpServletResponse response)
```

```
throws IOException, ServletException
```

```
{
```

```
response.setContentType("text/html");
PrintWriter out = response.getWriter();
```

```
out.print("<HTML>");
```

```
out.print("<HEAD><TITLE>First Servlet Program</TITLE></HEAD>");
```

```
out.print("<BODY>");
```

```
out.print("<H1>Welcome to Servlets</H1>");
```

```
out.print("</BODY>");
```

```
out.print("</HTML>");
```

```
out.close();
```

```
}
```

```
}
```

Servlets are not part of the standard SDK,  
they are part of the J2EE

Servlets normally extend HttpServlet

The response to be sent to the client

Details of the HTTP request from the client

Set the response type to text/html (this is  
normal)

This HTML text is  
sent to the client

Don't forget to close  
the connection with  
the client

**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?

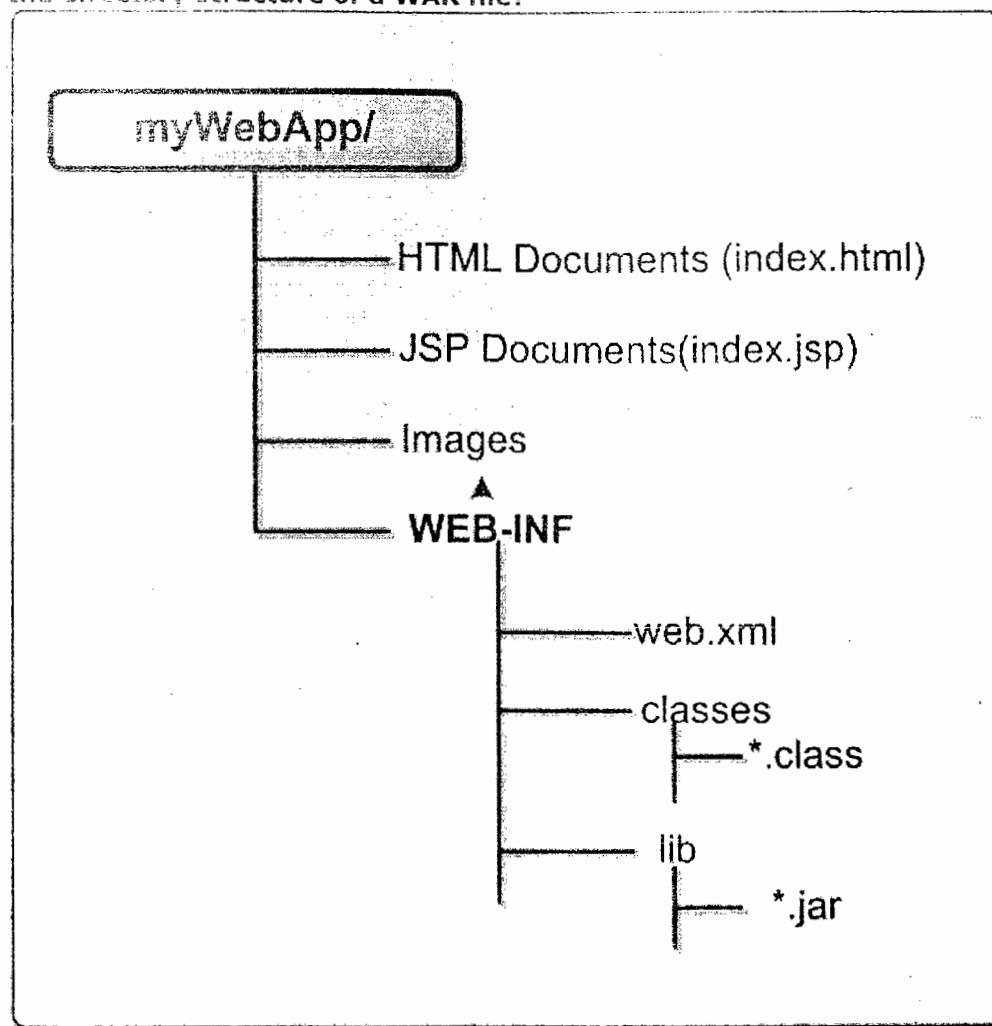
| <code>forward()</code>                                                                                                                                   | <code>sendRedirect()</code>                                                                                                                                                                                                           |
|----------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 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.<br>(Variables need to be passed by via the <code>session</code> 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?

| <code>include()</code>                                                                                                                                                                        | <code>forward()</code>                                                                                                                                                                                        |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| The RequestDispatcher <code>include()</code> 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 <code>forward()</code> 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 <code>include()</code> method is often used to include common "boilerplate" text or template markup that may be included by many servlets.                                                | The <code>forward()</code> 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?

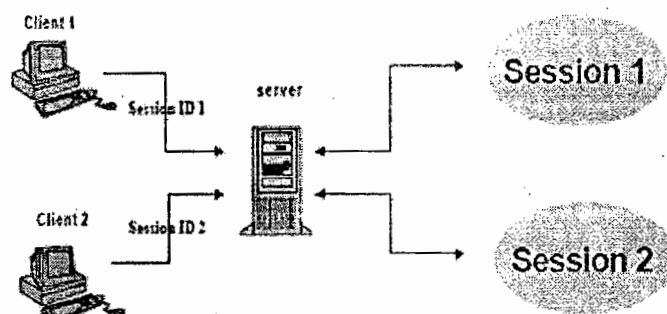
The <load-on-startup> element of a deployment descriptor is used to load a servlet file when the server starts instead of waiting for the first request. It is also used to specify the order in which the files are to be loaded. The <load-on-startup> element is written in the deployment descriptor as follows:

```
<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 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=123456789>

#### 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:

- Programatically : 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:

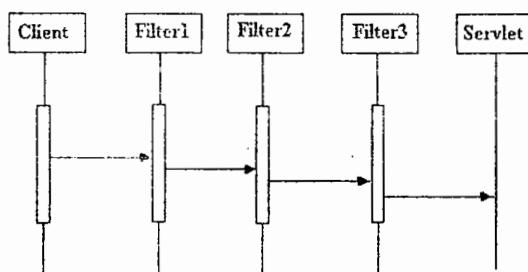
- Security checks
- Modifying the request or response
- Data compression
- Logging and auditing
- Response compression

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 `ServletRequest` and `ServletResponse` 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 development 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 devided 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" %>

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 the 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**.

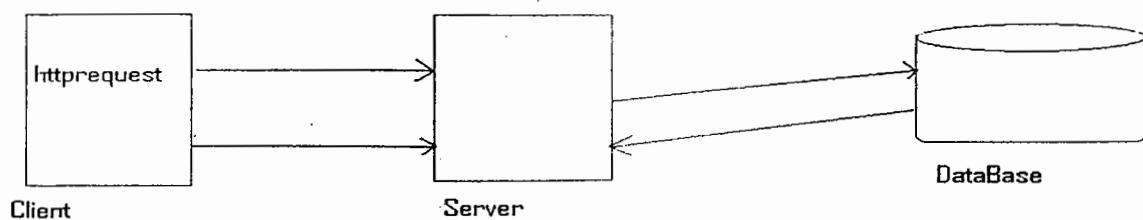
- **jsp:useBean**

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

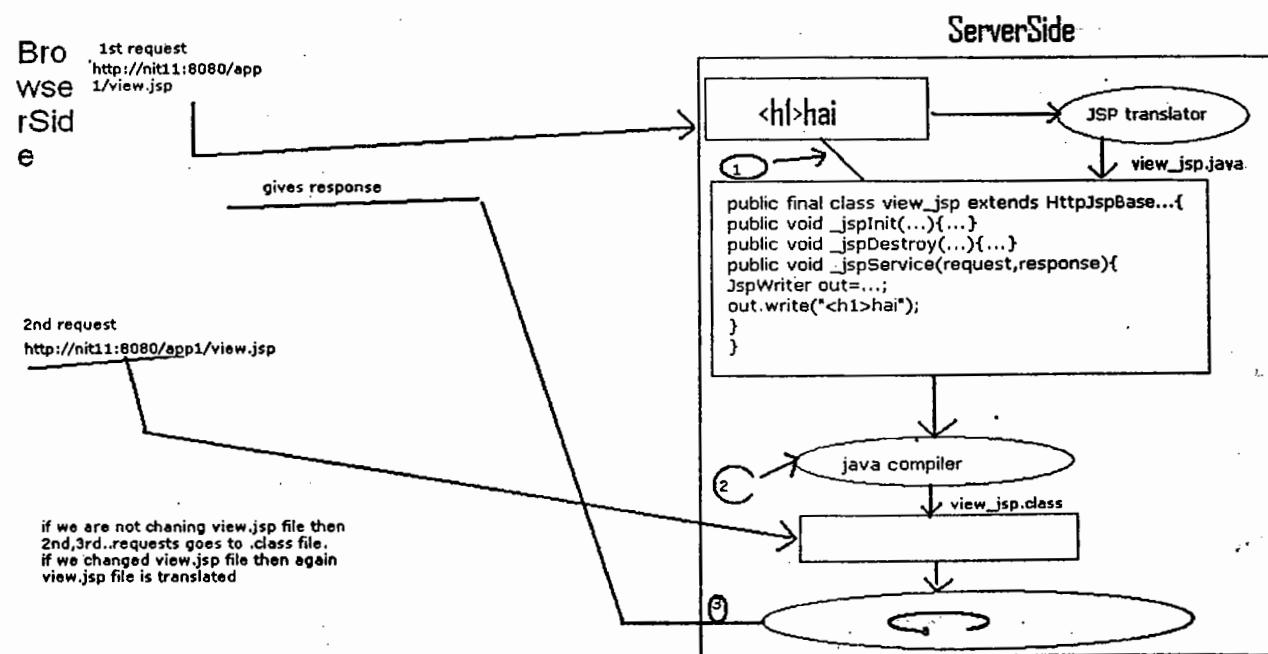
phase5: Initialization phase

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----> web container loads and execute jsp life cycle methods.

1-->JSP translation phase

2-->compilation phase

3-->request 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.

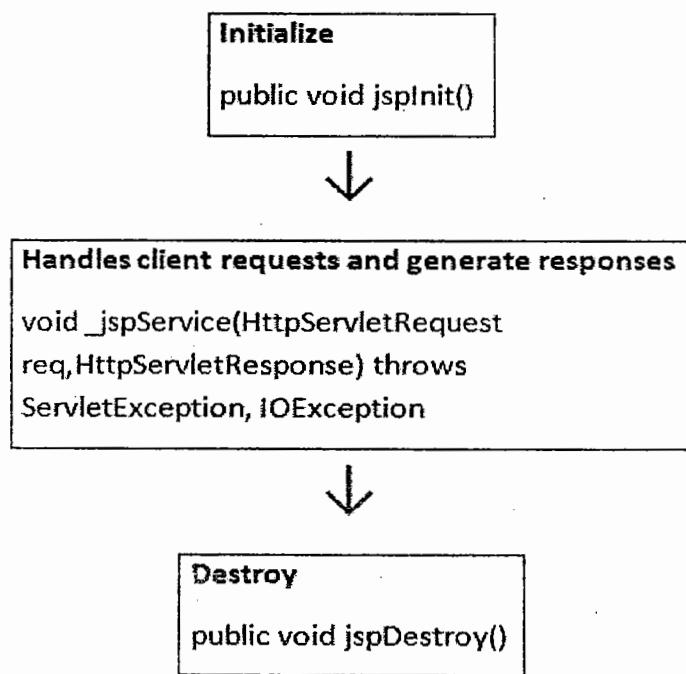
It will call doGet or doPost() method of servlet created.

**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()%>

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("
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();
 }
%>
```

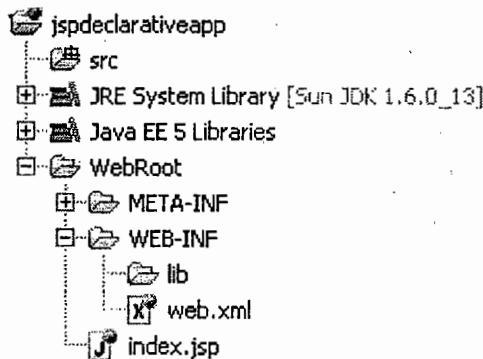
Hello! The time is now <%=getDate() %>

```
</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") + "
" + "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 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 scriptlet `<% a = 100; %>` will change the value of the class-level instance variable from 300 to 100.

The second scriptlet, `<% 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 scriptlet `<% 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



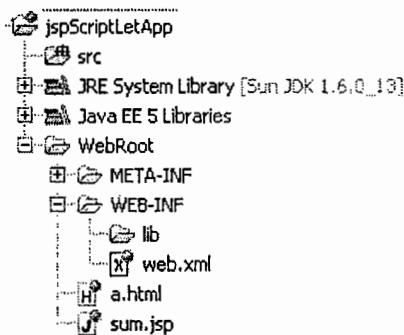
#### web.xml

```

<?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:
<%=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"/>

Num2:<input type="text" name="param2"/>

<input type="submit" value="click"/>

</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:

<b>Objects</b>	<b>Description</b>
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 <b>defined by the</b> translated servlet class.
exception	The <b>Exception</b> object allows the exception data to be accessed by <b>designated JSP</b> .

**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"/>

num2: <input type="text" name=" param2"/>

<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 <%=<br/>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.

**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"](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 it's 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 it's different attributes and set values of the attributes as per requirements.

Syntax of the declaration of the **page** directive with it's 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, it's 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 it's 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:

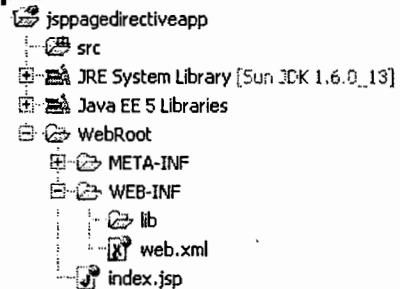
`<%@ page [attribute="value" attribute="value".....] %>`

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 language="java" only valid value for this attribute is java.	<code>language="java"</code>	<code>&lt;%@page language="java"%&gt;</code>
<b>Import</b>	This attribute defines the list of packages, each separated by comma .	<code>import="package.class"</code>	<code>&lt;%@page import="java.util.* , java.io.*"%&gt;</code>
<b>Extends</b>	Indicates the superclass of servlet when jsp translated in extends="package.class" servlet.	<code>extends="com.Connection"</code>	<code>&lt;%@page extends="com.Connection"%&gt;</code>
<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.	<code>session="true false"</code>	<code>&lt;%@page session="true"%&gt;</code>
<b>Buffer</b>	specifies the buffer size for out and default is 8kb.	<code>buffer="sizekb none"</code>	<code>&lt;%@ page buffer="8kb"%&gt;</code>

<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

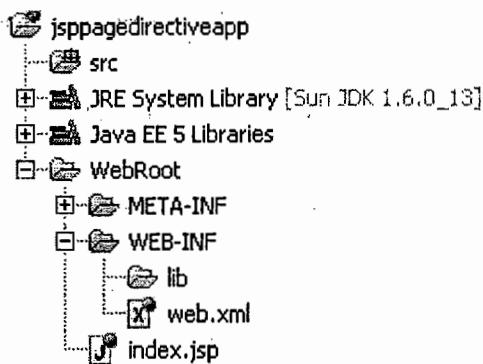
```

<%@ 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****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>

 <%
 String name="Student";
 out.println("Hai " + name + "!");
 %>

 </body>
</html>
```

**Include Directive**

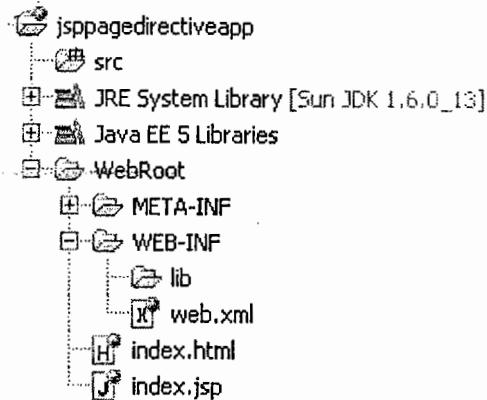
The **include** directive is used to include a file during the translation phase. This directive tells the container to merge the content of other external files with the current JSP during the translation phase. You may code *include* directives anywhere in your JSP page. The general usage form of this 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">
<marquee>Welcome to NareshIT</marquee>
 </body>
</html>

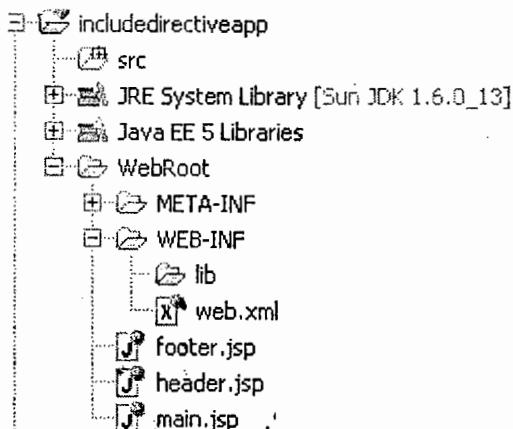
```

### **index.jsp**

```

<%@include file="index.html" %>
<body>
<marquee>Register Here...</marquee>
</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>

footer.jsp
<html>
<head></head>

<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 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 setContentType="image/gif" %>
- C ) <%@ page contentType="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 errorPage="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.



Element	Description	Legend
HTML Comment	Creates a comment that is sent to the client in the viewable page source.	All tags are case sensitive. A pair of single quotes is equivalent to a pair of double quotes. Spaces are not allowed between an equals sign and an attribute value.
Hidden Comment	Documents the JSP file, but is not sent to the client.	
Declaration	Declares variables or methods valid in the page scripting language.	<%! declaration; [ 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" ] [ import="package.class   package.*" ] ... " ] [ session="true   false" ] [ buffer="none   8KB   sizeKB" ] [ autoFlush="true   false" ] [ isThreadSafe="true   false" ] [ info="text" ] [ info="relativeURL" ] [ contentType="mimeType" [ ; charset=characterSet ] [ ; charset=UTF-8   ISO-8859-1 ] [ ; 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:tagName attribute="value"+ ... /> <tagPrefix:tagName attribute="value"+ ... > other tags </tagPrefix:tagName>

More information:  
<http://java.sun.com/products/jsp/>



Element	Description	Code Snippet	Type	Implicit Objects
<jsp:forwards>	Forwards a client request to an HTML file, JSP file, or servlet for processing.	<jsp:forward page=" <i>[relativeURL   &lt;%= expression %&gt;]</i> " { />   [ <jsp:param name="parameterName" value=" <i>[parameterValue   &lt;%= expression %&gt;]</i> " /> ]+ </jsp:forward> }		
<jsp:getProperty>	Gets the value of a Bean property so that you can display it in a result page.	<jsp:include page=" <i>[relativeURL   &lt;%= expression %&gt;]</i> " flush="true" { />   [ <jsp:param name="parameterName" value=" <i>[parameterValue   &lt;%= expression %&gt;]</i> " /> ]+ </jsp:include> }		
<jsp:include>	Includes a static file or sends a request to a dynamic file.	<jsp:plugin type="bean applet" code=" <i>classFileName</i> " database="classFileDirectoryName" [ name="instanceName" ] [ archive="URIToArchive,..."] [ align="bottom top middle left right" ] [ height="displayPixels" ] [ width="displayPixels" ] [ hspace="leftRightPixels" ] [ vspace="topBottomPixels" ] [ jreversion="JREVersionNumber   1.1" ] [ nspluginurl="URLToPlugin" ] [ iepluginurl="URLToPlugin" ] > [ <jsp:params>   <jsp:param name="parameterName" value=" <i>[parameterValue   &lt;%= expression %&gt;]</i> " /> ]+ </jsp:params>   <jsp:fallback> <jspt:message for user </jspt:fallback> ] </jsp:plugin>		
<jsp:plugin>	Downloads plug-in software to the Web browser to execute an applet or Bean.	<jsp:setProperty name="beanInstanceName" [ property="propertyName" [ param="parameterName" ]   property="propertyName" value=" <i>[string   &lt;%= expression %&gt;]</i> " ] /> <jsp:useBean id="beanInstanceName" scope="page request session application" [ class="package.class"   type="package.class" ]   type="package.class"   beanName="packageName" [ type="package.class   <%= expression %>"   type="package.class"   />   > other elements </jspt:useBean> ]	Scope	Some Useful Methods (see class or interface for others)
<jsp:setProperty>	Sets a property value or values in a Bean.	Request		getAttribute, getParameter, getParameterNames, getParameterValues Not typically used by JSP page authors.
<jsp:useBean>	Locates or instantiates a Bean with a specific name and scope.	Page		findAttribute, getAttribute, getAttributesScope, getAttributeNamesInScope
		Session		getID, getValue, getValueNames, putValue
		Application		getMimeType, getRealPath
		Page		clear, clearBuffer, flush, getBufferSize, getRemaining
		Page		getInitParameter, getInitParameterNames
		Page		Not typically used by JSP page authors.
		Page		getMessage, getLocalizedMessage, printStackTrace, toString

```
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 definatons 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
125 ex4:
126 <jsp:declaration>int xyz=10; </jsp:declaration>
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
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
```

```
:65
:66
:67
:68
:69
:70
:71
:72
:73
:74
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 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 =====
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 =====
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.calss" %>
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 =====
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 =====
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 =====
349 The extends Attribute
350 <%@ page extends="package.class" %>
351 =====
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 "sname" 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
```

```
1 =====
2 App1(example using scripting tags)
3 =====
4 -----First.jsp-----
5 <%! public String generateWishMsg(String uname){
6 // get Sys Date
7 java.util.Calendar cl=java.util.Calendar.getInstance();
8 // get current hour of the day (24 hrs format)
9 int h=cl.get(java.util.Calendar.HOUR_OF_DAY);
10 // generate wish msg
11 if(h<=12)
12 return "Good morning"+uname;
13 else if(h<=16)
14 return "Good Afternoon:"+uname;
15 else if(h<=20)
16 return "Good Evening:"+uname;
17 else
18 return "Good Night:"+uname;
19 }
20 %>
21 <h1> Welcome to Jsp </h1>

22 Date and time is : <%=new java.util.Date() %>

23
24 <%
25 String name=request.getParameter("uname");
26 %>
27

28 Wish message : <%=generateWishMsg(name) %>
29 -----
30 App2)Html----->Jsp ----->DB s/w Communication
31 -----Input.html-----
32 <form action="dburl" method="get">
33 Number : <input type="text" name="tsno">

34 Name : <input type="text" name="tsname">

35 Address : <input type="text" name="tsadd">

36 <input type="submit" value="register" name="s1">

37 </form>
38 Get All Student Details
39 -----web.xml-----
40 <web-app>
41 <servlet>
42 <servlet-name>abc</servlet-name>
43 <jsp-file>/DBJsp.jsp</jsp-file>
44 <init-param>
45 <param-name>driver</param-name>
46 <param-value>oracle.jdbc.driver.OracleDriver</param-value>
47 </init-param>
48 <init-param>
49 <param-name>url</param-name>
50 <param-value>jdbc:oracle:thin:@localhost:1521:orcl</param-value>
51 </init-param>
52 <init-param>
53 <param-name>dbuser</param-name>
54 <param-value>scott</param-value>
55 </init-param>
56 <init-param>
57 <param-name>dbpwd</param-name>
58 <param-value>tiger</param-value>
59 </init-param>
60 <load-on-startup>2</load-on-startup>
61 </servlet>
62
63 <servlet-mapping>
64 <servlet-name>abc</servlet-name>
65 <url-pattern>/dburl</url-pattern>
66 </servlet-mapping>
```

```

67 </web-app>
68 -----DBJsp.jsp-----
69 <%@page import="java.sql.*" %>
70
71 <%
72 Connection con;
73 PreparedStatement ps1,ps2;
74 public void jspInit()
75 {
76 System.out.println("DBJsp: jspInit() method");
77 try{
78 //get Access to ServletConfig obj
79 ServletConfig cg=getServletConfig();
80 // read init param values
81 String s1=cg.getInitParameter("driver");
82 String s2=cg.getInitParameter("url");
83 String s3=cg.getInitParameter("dbuser");
84 String s4=cg.getInitParameter("dbpwd");
85 // create jdbc connection ,PreparedStatement objs
86 Class.forName(s1);
87 con=DriverManager.getConnection(s2,s3,s4);
88 ps1=con.prepareStatement("insert into student values(?, ?, ?)");
89 ps2=con.prepareStatement("select * from student");
90 }//try
91 catch(Exception e)
92 { e.printStackTrace(); }
93 } //jspInit()
94 %>
95
96 <%
97 System.out.println("DBJsp: from scriptlet ");
98 //read s1 req param value
99 String pval=request.getParameter("s1");
100 if(pval.equals("register")) // when submit btn is clicked
101 {
102 //read form data
103 int no=Integer.parseInt(request.getParameter("tsno"));
104 String name=request.getParameter("tsname");
105 String addrs=request.getParameter("tsadd");
106 //set form data to the params of insert query
107 ps1.setInt(1,no);
108 ps1.setString(2,name);
109 ps1.setString(3,addrs);
110 //execute the Query
111 int result=ps1.executeUpdate();
112 //process the results
113 if(result==0)
114 { %
115 Registration Failed
116 %
117 else
118 { %
119 Registration Success
120 %
121 } //else
122 else //when hyperlink is clicked
123 {
124 //execute the query
125 ResultSet rs=ps2.executeQuery();
126 // get ResultSet MetaData obj
127 ResultSetMetaData rsmd=rs.getMetaData();
128 int cnt=rsmd.getColumnCount(); %
129 <table>
130 <tr>
131 <% //print col names
132 for(int i=1;i<=cnt;++i)

```

```
133 { %>
134 <td> <%=rsmd.getColumnLabel(i) %></td>
135 //for %
136
137 </tr>
138 <%
139 //print col vlaue
140 while(rs.next())
141 { %
142 <tr>
143 <% for(int i=1;i<=cnt;++i)
144 { %
145 <td><%=rs.getString(i) %></td>
146 <%}//for
147 }//while
148 rs.close();
149 } //else
150 %
151 <%! public void jspDestroy()
152 {
153 System.out.println("DBJsp:jspDestroy()");
154 try
155 {
156 //close jdbc objs
157 try{
158 if(ps1!=null)
159 ps1.close();
160 }catch(Exception e)
161 { e.printStackTrace(); }
162 try{
163 if(ps2!=null)
164 ps2.close();
165 }
166 catch(Exception e)
167 { e.printStackTrace(); }
168 try{
169 if(con!=null)
170 con.close();
171 }
172 catch(Exception e)
173 { e.printStackTrace(); }
174 } //try
175 catch(Exception e)
176 { e.printStackTrace(); }
177 } //jspDestroy()
178 %
179 =====
180 App3(Example on Directive Tags)
181 =====
182 -----DirectiveEx1.html-----
183 <form action="DirectiveEx1.jsp">
184 <pre>
185 Name : <input type="text" name="name"/>
186
187 age : <input type="text" name="value"/>
188
189 <input type="submit" name="s1" value="Add">
190 <input type="submit" name="s1" value="remove">
191 <input type="submit" name="s1" value="View"/>
192
193 </form>
194 -----DirectiveEx1.jsp-----
195 <%-
196 Page Directive
197 --%>
198 '
```

```
199 <%@page import="java.util.*"%>
200 <%!
201 Hashtable ht=new Hashtable();
202 %>
203
204 <%
205 String s=request.getParameter("s1");
206
207 if (s.equals("Add"))
208 {
209 ht.put(request.getParameter("name"),request.getParameter("value"));
210 %>
211
212 Value Added
213
214 <%
215 }
216 else if (s.equals("remove"))
217 {
218 ht.remove(request.getParameter("name"));
219 %>
220
221 Value Removed
222
223 <%
224 }
225 else
226 {
227 %>
228
229 <%@include file="ViewJsp.jsp"%>
230
231 <%
232 }
233
234 session.setAttribute("Ht",ht);
235
236 System.out.println(ht.toString());
237 %>
238 -----ViewJsp.jsp-----
239 <%@page import="java.util.*" errorPage="Error.jsp" session="true"%>
240
241 <html>
242 <body>
243 <table>
244 <tr>
245 <th>Name</th>
246 <th>Value</th>
247 </tr>
248
249 <%
250 Hashtable ht=(Hashtable)session.getAttribute("Ht");
251
252 Enumeration names=ht.keys();
253
254 while (names.hasMoreElements())
255 {
256 String name= (String) names.nextElement();
257
258 String value= (String) ht.get(name);
259 %>
260 <tr>
261 <td><%=name%></td>
262 <td><%=value%></td>
263 </tr>
264 <%
```

```
265 } //while
266 %>
267 </table>
268 </body>
269 </html>
270 -----Error.jsp-----
271 <%-
272 Error Page
273 --%>
274
275 <%@page isErrorPage="true"%>
276
277 Output From Error.jsp
278

279 Error :
280 <%= exception.toString()
281 %>
282

283 Desc
284 <%= exception.getMessage()
285 %>
286 =====
287 App4) Using both Directive Include and Action Include
288 =====
289 -----page1.jsp-----
290 <table width="100%" height="100%">
291 <tr height="30%">
292 <td colspan="2"><jsp:include page="/header"/></td>
293 </tr>
294 <tr height="60%">
295 <td width="30%"><%@include file="LeftContent.html" %></td>
296 <td width="70%"><jsp:include page="home.jsp"/></td>
297 </tr>
298 <tr height="10%">
299 <td colspan="2"><%@include file="Footer.html" %></td>
300 </tr>
301 </table>
302 -----page2.jsp-----
303 <table width="100%" height="100%">
304 <tr height="30%">
305 <td colspan="2"><jsp:include page="/header"/></td>
306 </tr>
307 <tr height="60%">
308 <td width="30%"><%@include file="LeftContent.html" %></td>
309 <td width="70%"><jsp:include page="Weather.jsp"/></td>
310 </tr>
311 <tr height="10%">
312 <td colspan="2"><%@include file="Footer.html" %></td>
313 </tr>
314 </table>
315 -----page3.jsp-----
316 <table width="100%" height="100%">
317 <tr height="30%">
318 <td colspan="2"><jsp:include page="/header"/></td>
319 </tr>
320 <tr height="60%">
321 <td width="30%"><%@include file="LeftContent.html" %></td>
322 <td width="70%"><jsp:include page="Sports.jsp"/></td>
323 </tr>
324 <tr height="10%">
325 <td colspan="2"><%@include file="Footer.html" %></td>
326 </tr>
```

```

331 </table>
332 -----LeftContent.jsp-----
333 home
334

335 Weather
336

337 Sports
338

339 -----home.jsp-----
340

341

342 <center> Welcome to HINDU</center>
343

344 <hr>
345 <pre><h3> TRAINS DERAILED In MP </pre></h3>
346 NewsOnline 5th august: Two trains derailed into
347 River at Harda in MP. According reliable sources
348 nearly 30+ people dead and still counting is go in.
349
350 -----Footer.html-----
351

352 <i><center> © All rights reserved</i>
353
354 -----Sports.jsp-----
355
356 <h1><center> SPORTS</center></h1>
357 <h2> U MUMA is leading in PRO Kabaddi</h2>
358 NewsOnline: <%=new java.util.Date() %>: The Season2 of ProKabaddi
359 really started well and U MUMBA team is leading
360 in the points table .
361 -----Weather.jsp-----
362
363 <h1><center> Weather Report</center></h1>
364
365 <h3>Hot Rainy Season in India </h3>
366 No rains in india, people are suffering to drinking
367 water.
368 <hr>
369 <%=new java.util.Date() %>
370 =====
371 App5)Example App <jsp:forward>
372 =====
373 -----Form.html-----
374 <form action="Bill.jsp" >
375 Name :<input type="text" name="t1">

376 Price :<input type="text" name="t2">

377 Qty : <input type="text" name="t3">

378 <input type="submit" value="GetBill"/>
379 </form>
380 -----Bill.jsp-----
381 <% //read form data
382 String name=request.getParameter("t1");
383 int price=Integer.parseInt(request.getParameter("t2"));
384 int qty=Integer.parseInt(request.getParameter("t3"));
385 //calc Bill amt
386 int bamt=price*qty;
387 if(bamt>=50000){ %
388 <jsp:forward page="Discount.jsp">
389 <jsp:param name="bill" value="<%=bamt %>" />
390 </jsp:forward>
391 <% } //if
392 else
393 { %
394 from Bill.jsp

395 Name = <%=name %>

396 Price = <%=price %>

```

```
397 Qty =<%=qty %>

398 Bill Amt = <%=bamt %>

399 <% }//else %>
400 -----Discount.jsp-----
401 <% //read form data
402 String name= request.getParameter("t1");
403 int price=Integer.parseInt(request.getParameter("t2"));
404 int qty=Integer.parseInt(request.getParameter("t3"));
405 //read additional req param value given by Bill.jsp
406 int bamt=Integer.parseInt(request.getParameter("bill"));
407
408 //Give 30% discount on Bill Amt
409 float discount=bamt*0.3f;
410 float famt=bamt-discount;
411 %>
412
413

414 From Discount.jsp
415

416 Item name : <%=name %>

417 Price : <%=price %>

418 Qty : <%=qty %>

419 Bill Amt : <%=bamt %>

420 Discount : <%=discount %>

421 Final Bill Amt: <%=famt %>
422 =====
423 App6(Application on <jsp:plugin>
424 =====
425 -----TestApplet.java-----
426 import java.applet.*;
427 import java.awt.*;
428 import java.awt.event.*;
429 public class TestApplet extends Applet
430 {
431 public void paint(Graphics g)
432 {
433
434 setBackground(Color.red);
435 g.drawString("this is from applet to test",50,50);
436 }
437 }
438 -----plug.jsp-----
439 <html>
440 <title> Plugin example </title>
441 <body bgcolor="white">
442 <jsp:plugin type="applet" code="TestApplet.class"
443 codebase="/jsp-plug" width="300" height="300" >
444 <jsp:fallback>
445 Plugin tag OBJECT or EMBED not supported by browser.
446 </jsp:fallback>
447 </jsp:plugin>
448 </body>
449 </html>
450 =====
451 App7(jsp----->java bean)
452 =====
453 -----StudentBean.java-----
454 package com.nt;
455
456 public class StudentBean
457 {
458 private int sno;
459 private String sname;
460 private String result;
461 public StudentBean()
462 {
```

```
463 System.out.println("inside constructor");
464 }
465 public void setSno(int sno)
466 {
467 this.sno = sno;
468 }
469 public int getSno()
470 {
471 return sno;
472 }
473 public void setSname(String sname)
474 {
475 this.sname = sname;
476 }
477 public String getSname()
478 {
479 return sname;
480 }
481 public void setResult(String result)
482 {
483 this.result = result;
484 }
485 public String getResult()
486 {
487 return result;
488 }
489 }
-----SetValues.jsp-----
490 <%@ page import="com.nt.StudentBean"%>
491
492 <jsp:useBean id="stud" class="com.StudentBean" scope="application"></jsp:useBean>
493
494 <jsp:setProperty name="stud" property="sno" value="101"/>
495 <jsp:setProperty name="stud" property="sname" value="raja"/>
496 <jsp:setProperty name="stud" property="result" value="pass"/>
-----GetValues.jsp-----
497 <%@ page import="com.nt.StudentBean"%>
498
499 <jsp:useBean id="stud" class="com.nt.StudentBean" scope="application"></jsp:useBean>
500
501 <html>
502 <head><title>jsp prg</title></head>
503 <body>
504 student number is<jsp:getProperty name="stud" property="sno"/>

505 student name is <jsp:getProperty name="stud" property="sname"/>

506 student result is <jsp:getProperty name="stud" property="result"/>
507 </body>
508 </html>
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
```



```
595 attributes
596
597 to modify pageContext attribute values
598 -----
599 pageContext.setAttribute("attr1","val1");
600 --> modifies page scope "attr1" attribute value
601 pageContext.setAttribute("attr2","new val2",pageContext.SESSION_SCOPE);
602 --> modifies the session scope "attr2" attribute value.
603
604 to read pageContext attribute values
605 -----
606 String s1=(String)pageContext.getAttribute("attr1");
607 --> reads page scope "attr1" attribute value
608 String s2=(String)pageContext.getAttribute("attr2",PageContext.SESSION_SCOPE);
609 --> reads session scope "attr2" attribute value
610 to find pageContext attribute
611 -----
612 String s1=(String) pageContext.findAttribute("attr1");
613 ->searchs and reads attr1 attribute value in multiples scopes
614 like pagescope,request scope,session scope,application scope
615 to remove pageContext attribute
616 -----
617 pageContext.removeAttribute("attr1");
618 -->removes the page scope "attr1" attribute
619 pageContext.removeAttribute("attr2",PageContext.SESSION_SCOPE);
620 -->removes the session scope "attr2" attribute
621 =====
622 App9 (Application on pageContext Attributes)
623 =====
624 -----A.jsp-----
625 <% //create attributes
626 pageContext.setAttribute("attr1","val1",PageContext.REQUEST_SCOPE);
627 pageContext.setAttribute("attr2","val2",PageContext.SESSION_SCOPE);
628 pageContext.setAttribute("attr3","val3",PageContext.APPLICATION_SCOPE); %>
629 <!-- forward request to B.jsp -->
630 <jsp:forward page="B.jsp"/>
631 -----B.jsp-----
632 from B.jsp
633
request attribute attr1 value is = <%=pageContext.findAttribute("attr1")%>
634
session attribute attr2 value is = <%=pageContext.findAttribute("attr2") %>
635
application attribute attr3 value is = <%=pageContext.findAttribute("attr3") %>
636 <!-- forward the request to C.jsp -->
637 <jsp:forward page="C.jsp"/>
638
639 -----C.jsp-----
640 from C.jsp
641
642
request attribute attr1 value is = <%=pageContext.findAttribute("attr1")%>
643
session attribute attr2 value is = <%=pageContext.findAttribute("attr2") %>
644
application attribute attr3 value is = <%=pageContext.findAttribute("attr3") %>
645
646 -----D.jsp-----
647 from D.jsp
648
649
request attribute attr1 value is = <%=pageContext.findAttribute("attr1")%>
650
session attribute attr2 value is = <%=pageContext.findAttribute("attr2") %>
651
application attribute attr3 value is = <%=pageContext.findAttribute("attr3") %>
652
653
654
655
```

## 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 scriplets 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 java.lang.Boolean.valueOf(String)
byte or Byte	As indicated in java.lang.Byte.valueOf(String)
char or Character	As indicated in java.lang.Character.valueOf(String)
double or Double	As indicated in java.lang.Double.valueOf(String)
int or Integer	As indicated in java.lang.Integer.valueOf(String)
float or Float	As indicated in java.lang.Float.valueOf(String)
long or Long	As indicated in java.lang.Long.valueOf(String)

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	<code>doStartTag</code> , <code>doEndTag</code> , <code>release</code>
Attributes	<code>doStartTag</code> , <code>doEndTag</code> , <code>set/getAttribute1...N</code>
Body, No Interaction	<code>doStartTag</code> , <code>doEndTag</code> , <code>release</code>
Body, Interaction	<code>doStartTag</code> , <code>doEndTag</code> , <code>release</code> , <code>doInitBody</code> , <code>doAfterBody</code>

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:

```
<tlt: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>
 <rexprvalue>true|false|yes|no</rexprvalue>
</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 -----Test.jsp-----
5 <%@ taglib uri="demo" prefix="start"%>
6 <html>
7 <head>
8 <title>Tag Test !!!....</title>
9 </head>
10 <body>
11 <center><h2>
12 <start:example/>
13

14 <start:prime />
15

16 <start:prime n="30" />
17

18 <start:display font="Garamond" >
19
JAVA powered by
20 </start:display>
21

22 <start:display font="calibri" size="100">
23 HCL Technologies
24 </start:display>
25 </h2></center>
26 </body>
27 </html>-----web.xml-----
28 <web-app>
29 <jsp-config>
30 <taglib>
31 <taglib-uri>demo</taglib-uri>
32 <taglib-location>/WEB-INF/first.tld</taglib-location>
33 </taglib>
34 </jsp-config>
35 </web-app>
36 -----first.tld-----
37 <?xml version="1.0" encoding="UTF-8"?>
38 <!DOCTYPE taglib
39 PUBLIC "-//Sun Microsystems, Inc./DTD JSP Tag Library 1.2//EN"
40 "http://java.sun.com/dtd/web-jsptaglibrary_2_1.dtd">
41
42 <taglib>
43 <tlib-version>1.0</tlib-version>
44 <jsp-version>2.1</jsp-version>
45 <short-name>start</short-name>
46 <description>
47 An example Tag Library
48 </description>
49 <tag>
50 <name>example</name>
51 <tag-class>tags.ExampleTag</tag-class>
52 <body-content>EMPTY</body-content>
53 <description>Simplest example: inserts one line of output</description>
54 </tag>
55 <tag>
56 <name>prime</name>
57 <tag-class>tags.PrimeTag</tag-class>
58 <body-content>EMPTY</body-content>
59 <description>Prime numbers generation</description>
60
61 <attribute>
62 <name>n</name>
63 <required>false</required>
64 </attribute>
65 </tag>
66 <tag>
```

```
67 <name>display</name>
68 <tag-class>tags.DisplayTag</tag-class>
69 <body-content>Jsp</body-content>
70 <description>Displaying text in different fonts</description>
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 </taglib>
81
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"+
);
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;
181 }
182 public void setSize(int fontSize)
183 {
184 this.size=fontSize;
185 }
186
187 public int doStartTag()
188 {
189 System.out.println("Inside doStartTag() of DisplayTag !!!....");
190 String status=null;
191 JspWriter out=pageContext.getOut();
192 ServletRequest req=pageContext.getRequest();
193 try
194 {
195 out.print("<table border=\"0\">");
196 out.print("<tr><th><span style= \" font-size:"+size+"px; " + "font-family:");
197 }
198 catch(IOException ioe)

```

```
199 {
200 System.out.println("Error in ExampleTag: "+ioe);
201 }
202 status=req.getParameter("print");
203 if(status==null)
204 return(SKIP_BODY);
205 else
206 if(status.equalsIgnoreCase("YES"))
207 return(EVAL_BODY_INCLUDE);
208 else
209 return(SKIP_BODY);
210 }//doStartTag()
211
212 public int doEndTag()
213 {
214 System.out.println("Inside doEndTag()");
215 try
216 {
217 JspWriter out=pageContext.getOut();
218 out.print("</TABLE>");
219 }
220 catch(IOException ie)
221 {
222 ie.printStackTrace();
223 }
224 return EVAL_PAGE;
225 }//doEndTag()
226 }//class
```

# 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();

**String(param name)**

name

course

mail

**String[](param value)**

{NIT}

{java,sql}

{hr@nareshit.com,info@nit.com}

Ex:\${param.x}

It prints the value associates with request parameter x.

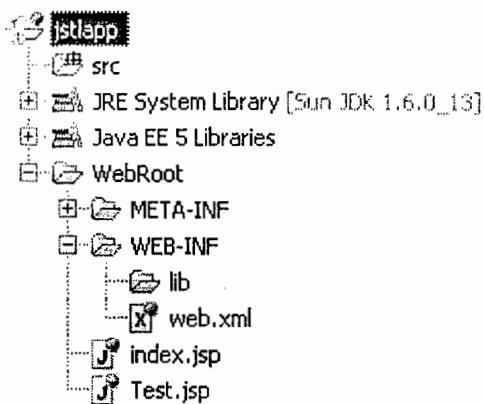
If the parameters is associated with multiple values.then it prints the first value.

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>
LoginForm</legend>
<table border="2" cellpadding="2" bordercolor="orange">
<tr><td>Name:</td>
<td><input type="text" name="name"/></td></tr>
<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}

${param.age}

${param.food}

${paramValues.name}

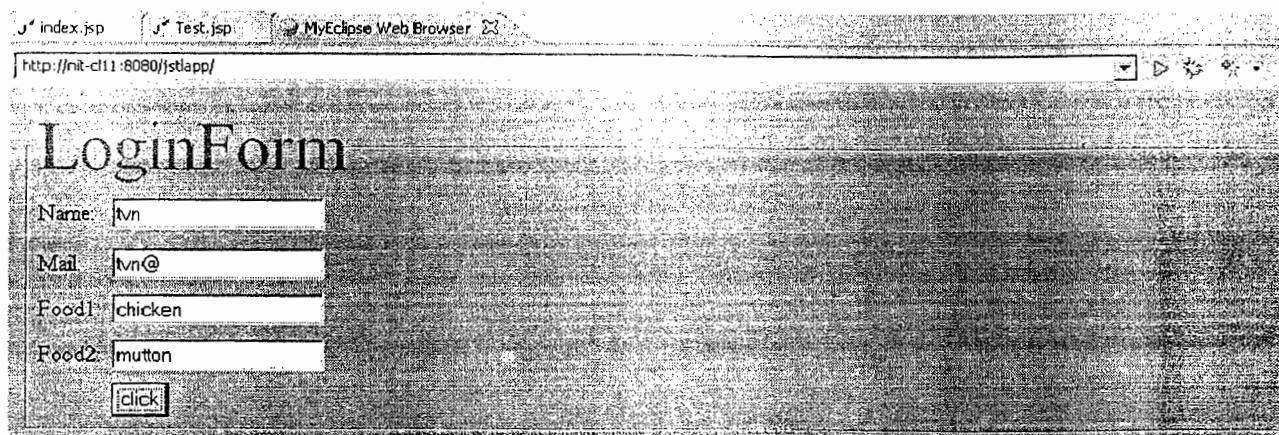
${paramValues.name[0]}

${paramValues.food[0]}

${paramValues.food[100]}

${paramValues.food[1]}

</h1>
```

**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\_13]

+ 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}

 ${cookie.JSESSIONID.name}

 ${cookie.JSESSIONID.value}

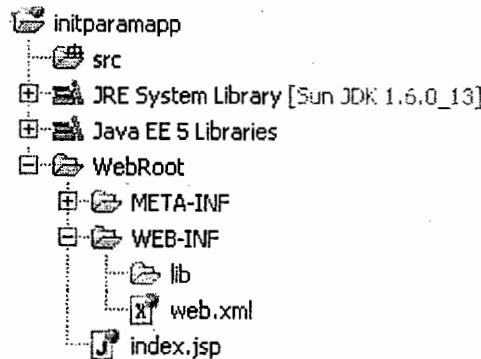
```

**init param:-**

by using this implicit object we can access context initialization parameters  
 initparameters----→application.getInitParameter();

**example**

```
 ${initparam.user}----→NITAdmin
 ${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
 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}

 ${initParam.pwd}

```

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 available operators in EL.

- 1)property access operator(.)
- 2)collection access operator([])
- 3)Arithmetic operators
- 4)Relational operators
- 3)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"} → Number Format Exception
- 7) {" "+"3"} → Number Format Exception

**2) '-' operator:**

$\$\{a-b\}$   
All rules is exactly same as '+' operator  
 $\$\{10-3\} \rightarrow 7$   
 $\$\{"abc"-3\} \rightarrow$  Number Format Exception  
 $\$\{"abc"-3\} \rightarrow -3$

**3) '\*' operator:-**

$\$\{a*b\}$   
All rules are exactly similar to "+" operator  
 $\$\{19*2\} \rightarrow 38$

**4) '/' operator:-**

$\$\{a/b\}$  or  $\$\{a \text{ div } b\}$   
All rules are exactly same as "+" operator  
 $\$\{10/2\} \rightarrow 5$   
 $\$\{10/0\} \rightarrow \text{infinity}$   
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.

$\$\{10%3\} \rightarrow 1$   
 $\$\{10%0\} \rightarrow \text{ArithmaticException}$   
 $\$\{10.0/0\} \rightarrow \text{NaN (Not any Number)}$

**Relational Operators:-**

**> or gt**  
 $\${2>1} \rightarrow \text{true}$

**< or lt**

$\${3<4} \rightarrow \text{true}$   
**== or eq**  
 $\${abcd==abcd} \rightarrow \text{true}$   
 $\{"abcd"=="abc"} \rightarrow \text{false}$   
**!= or ne**  
**>= or ge**  
**<= or le**

**Logical Operators:-**

**and | &&**  
 $\${\text{true and false}} \rightarrow \text{false}$   
**or | ||**  
 $\${\text{true or false}} \rightarrow \text{true}$   
**not | !**  
 $\${\text{not true}} \rightarrow \text{false}$

**Conditional Operator:-**

$\{(3<4) ? \text{"yes"} : \text{"no"}\} \rightarrow \text{yes}$   
 $\{(\text{true} == \text{false}) ? \text{"Right"} : \text{"wrong"}\} \rightarrow \text{wrong}$

**empty Operator:-**

$\${\text{empty object}}$   
 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.  
 $\${\text{empty abcd}} \rightarrow \text{true}$   
 $\${\text{empty "abcd"}} \rightarrow \text{false}$   
 $\${\text{empty null}} \rightarrow \text{true}$

JSP Expression Language (EL) supports most of the arithmetic and logical operators supported by Java. Below is the list of most frequently used operators:

<b>Operator</b>	<b>Description</b>
.	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 ge	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 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.  
 $\${(5*5) \text{ ne } 25}$
- 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.  
 $\${'cat' \text{ gt } 'cap'}$
- 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>

## 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

<u>parseDate</u>	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" %>
```

<u>transaction</u>	Provides nested database action elements with a shared Connection, set up to execute all statements as one transaction.
<u>query</u>	Executes the SQL query defined in its body or through the sql attribute.
<u>update</u>	Executes the SQL update defined in its body or through the sql attribute.
<u>param</u>	Sets a parameter in an SQL statement to the specified value.
<u>dateParam</u>	Sets a parameter in an SQL statement to the specified java.util.Date value.
<u>setDataSource</u>	Creates a simple DataSource suitable only for prototyping.

**JSTL XML:****Standard Syntax:**

```
<% taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
```

<u>choose</u>	Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by <when> and <otherwise>
<u>out</u>	Like <%= ... >, but for XPath expressions.
<u>if</u>	XML conditional tag, which evaluates its body if the supplied XPath expression evaluates to 'true' as a boolean
<u>forEach</u>	XML iteration tag.
<u>otherwise</u>	Subtag of <choose> that follows <when> tags and runs only if all of the prior conditions evaluated to 'false'
<u>param</u>	Adds a parameter to a containing 'transform' tag's Transformer
<u>parse</u>	Parses XML content from 'source' attribute or 'body'
<u>set</u>	Saves the result of an XPath expression evaluation in a 'scope'
<u>transform</u>	Conducts a transformation given a source XML document and an XSLT stylesheet
<u>when</u>	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	<u>contains</u> ( java.lang.String, java.lang.String)	Tests if an input string contains the specified substring.
boolean	<u>containsIgnoreCase</u> ( java.lang.String, java.lang.String)	Tests if an input string contains the specified substring in a case insensitive way.
boolean	<u>endsWith</u> ( java.lang.String, java.lang.String)	Tests if an input string ends with the specified suffix.

java.lang.String	<b><u>escapeXml( java.lang.String)</u></b>	Escapes characters that could be interpreted as XML markup.
int	<b><u>indexOf( java.lang.String, java.lang.String)</u></b>	Returns the index within a string of the first occurrence of a specified substring.
java.lang.String	<b><u>join( java.lang.String[], java.lang.String)</u></b>	Joins all elements of an array into a string.
int	<b><u>length( java.lang.Object)</u></b>	Returns the number of items in a collection, or the number of characters in a string.
java.lang.String	<b><u>replace( java.lang.String, java.lang.String, java.lang.String)</u></b>	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( java.lang.String, java.lang.String)</u></b>	Splits a string into an array of substrings.
boolean	<b><u>startsWith( java.lang.String, java.lang.String)</u></b>	Tests if an input string starts with the specified prefix.
java.lang.String	<b><u>substring( java.lang.String, int, int)</u></b>	Returns a subset of a string.
java.lang.String	<b><u>substringAfter( java.lang.String, java.lang.String)</u></b>	Returns a subset of a string following a specific substring.
java.lang.String	<b><u>substringBefore( java.lang.String, java.lang.String)</u></b>	Returns a subset of a string before a specific substring.
java.lang.String	<b><u>toLowerCase( java.lang.String)</u></b>	Converts all of the characters of a string to lower case.
java.lang.String	<b><u>toUpperCase( java.lang.String)</u></b>	Converts all of the characters of a string to upper case.
java.lang.String	<b><u>trim( java.lang.String)</u></b>	Removes white spaces from both ends of a string.

```
1 -----
2 -----EL Application-----
3 -----Test1.jsp-----
4 <%@page isELIgnored="false"%>
5
6 Sum is : ${4+5}

7 4>5 ? ${4>5}

8 request param uname value : ${param.uname}

9 Current browser name : ${header['user-agent']}

10 <%
11 request.setAttribute("course","java");
12 %>
13 Request attribute vlaue : ${requestScope.course}

14 Cookie name ${cookie.JSESSIONID.name}

15 Cookie value ${cookie.JSESSIONID.value}

16 -----
17 JSTL Applications
18 -----
19 -----ex1.jsp-----
20 <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
21
22 <c:set var="var1" value="hello" scope="request"/>
23 First: <c:out value="${var1}" />
24

25 <c:remove var="var1" scope="request"/>
26 Secohd <c:out value="${var1}" />
27 -----ex2.jsp-----
28 <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
29 <HTML>
30 <HEAD>
31 <TITLE>Example on JSTL Coretags</TITLE>
32 </HEAD>
33 <BODY BGCOLOR="#FFFFFF">
34 <c:set var="var1" value="Welcome" />
35
36 <c:if test="${!empty param.uname}">
37 <c:out value="${var1}" />
38 <c:out value="${param.uname}" />
39 </c:if>
40 </BODY>
41 </HTML>
42 -----ex3.jsp-----
43 <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
44
45 <c:catch var="e">
46 <jsp:scriptlet>
47 int a=Integer.parseInt("10a");
48 </jsp:scriptlet>
49 </c:catch>
50 <c:out value="${e}" />
51 -----ex4.jsp-----
52 <%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
53
54 how are u?
55 <c:import url="abc.html" /> <!-- gets abc.html file output content -->
56
57 <a href=<c:url value="http://localhost:3030/JSTLApp1/abc.html"/>> go
58 -----ex5.jsp-----
59 <%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
60 <c:redirect url="http://www.google.com"/>
61 -----ex6.jsp-----
```

```
67 <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
68 <HTML>
69 <HEAD>
70 <TITLE>Example on JSTL Coretags</TITLE>
71 </HEAD>
72 <BODY BGCOLOR="#FFFFFF">
73 <c:choose>
74 <c:when test="${param.p>0}">
75 p is +ve
76 </c:when>
77 <c:when test="${param.p<0}">
78 p is -ve
79 </c:when>
80 <c:otherwise>
81 p is zero
82 </c:otherwise>
83 </c:choose>
84
85 </BODY>
86 </HTML>
87 -----ex7.jsp-----
88 <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
89 <HTML>
90 <HEAD>
91 <TITLE>Example on JSTL Coretags</TITLE>
92 </HEAD>
93 <BODY BGCOLOR="#FFFFFF">
94 </BODY>
95 List of parameters

96 <c:forEach var="p" items="${paramValues}">
97 parameter name = <c:out value="${p.key}" />

98 values =
99 <c:forEach var="pv" items="${p.value}">
100 <c:out value="${pv}" />

101 </c:forEach>
102 </c:forEach>
103 </HTML>
104
105 -----ex8.jsp-----
106 <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
107 <HTML>
108 <HEAD>
109 <TITLE>Example on JSTL Coretags</TITLE>
110 </HEAD>
111 <BODY BGCOLOR="#FFFFFF">
112
113 List of students

114 <c:set var="str" value="raja,ravi,jani,rakesh" scope="page" />
115
116 <c:forTokens var="sname" items="${str}" delims="," >
117 <c:out value="${sname}" />

118 </c:forTokens>
119
120 </BODY>
121 </HTML>
122
123 -----ex9.jsp-----
124 <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
125 <HTML>
126 <HEAD>
127 <TITLE>Example on JSTL Coretags</TITLE>
128 </HEAD>
129 <BODY BGCOLOR="#FFFFFF">
130 <table>
131 <c:forEach var="x" begin="1" end="10" step="1">
132 <tr>
```

```
133 <td><c:out value="2 * ${x} = "/></td>
134 <td><c:out value="${2 * x}" /></td>
135 </tr>
136 </c:forEach>
137 </table>
138
139 </BODY>
140 </HTML>
141 -----ex10.jsp-----
142 <%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>
143 <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
144
145 <!-- set Datasource -->
146 <sql:setDataSource var="ds" driver="oracle.jdbc.driver.OracleDriver"
147 url="jdbc:oracle:thin:@localhost:1521:orcl" user="scott" password="tiger" />
148 <!-- Create ResultSet -->
149 <sql:query var="res_set" dataSource="${ds}" sql="select * FROM emp" />
150
151 <HTML>
152 <HEAD>
153 <TITLE>Example on JSTL tags</TITLE>
154 </HEAD>
155 <BODY BGCOLOR="#FFFFFF">
156
157 <c:forEach var="row" items="${res_set.rows}">
158 <c:out value="${row.ename}" />
159 <c:out value="${row.job}" />
160

161 </c:forEach>
162
163 </BODY>
164 </HTML>
165 -----ex11.jsp-----
166 <%@ taglib uri="http://java.sun.com/jstl/sql" prefix="sql" %>
167 <%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
168
169 <!-- set DataSource -->
170 <sql:setDataSource var="ds" driver="oracle.jdbc.driver.OracleDriver"
171 url="jdbc:oracle:thin:@localhost:1521:orcl" user="scott" password="tiger" />
172
173 <HTML>
174 <HEAD>
175 <TITLE>Example on JSTL tags</TITLE>
176 </HEAD>
177 <BODY BGCOLOR="#FFFFFF">
178
179 <sql:transaction dataSource="${ds}">
180 <sql:update>
181 update emp SET comm =comm + ? WHERE empno = ?
182 <sql:param value="1000"/>
183 <sql:param value="7499"/>
184 </sql:update>
185 </sql:transaction>
186 Record updated.
187 </BODY>
188 </HTML>
189 -----ex12.jsp-----
190 <%@ taglib uri="http://java.sun.com/jstl/sql" prefix="sql" %>
191 <%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
192
193 <!-- set DataSource -->
194 <sql:setDataSource var="ds" driver="oracle.jdbc.driver.OracleDriver"
195 url="jdbc:oracle:thin:@localhost:1521:orcl" user="scott" password="tiger" />
196
197 <HTML>
198 <HEAD>
```

```

199 <TITLE>Example on JSTL tags</TITLE>
200 </HEAD>
201 <BODY BGCOLOR="#FFFFFF">
202 <sql:transaction dataSource="${ds}">
203
204 <sql:update>
205 delete from emp WHERE empno = ?
206 <sql:param value="7499"/>
207 </sql:update>
208 </sql:transaction>
209
210 Record Deleted
211
212 </BODY>
213 </HTML>
214 -----ex13.jsp (Formatting tags)-----
215 <%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
216 <%@taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
217
218 <fmt:setLocale value="fr_FR"/>
219
220 <!-- Displays message collected from the properties file -->
221 <fmt:setBundle basename="myfile"/>
222 <fmt:message var="msg" key="wishMsg"/>
223 <h1> <c:out value="${msg}" /></h1>
224
225 <!-- Formats the number -->
226 <fmt:formatNumber var="price" value="4343444343.55"/>
227 Price is : <c:out value="${price}" />

228
229 <!-- Formats the date -->
230 <jsp:useBean id="dt" class="java.util.Date"/>
231 <fmt:formatDate var="fdt" value="${dt}"/>
232 Date value : <c:out value="${fdt}" />
233 -----ex14.jsp(JSTL functions)-----
234 <%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
235 <%@taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
236
237 <c:set var="msg" value="heilo how are u?"/>
238
239 ${fn:length(msg)}

240 ${fn:toUpperCase(msg)}

241 ${fn:toLowerCase(msg)}

242 ${fn:substring(msg,4,20)}

243 ${fn:contains(msg,"are")}

244 ${fn:startsWith(msg,"hello")}

245 ${fn:endsWith(msg,"u?")}

246 -----ex15.jsp (JSTL Xml tags)-----
247 <%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
248 <%@taglib uri="http://java.sun.com/jsp/jstl/xml" prefix="x" %>
249
250 <!-- loads file -->
251 <c:import var="file" url="orders.xml"/>
252 <!-- parses the file -->
253 <x:parse xml="${file}" var="doc"/>
254

255 printing all item dealis

256 <x:forEach var="ord" select="$doc/orders/order">
257 Name: <x:out select="$ord/item"/>

258 price: <x:out select="$ord/price"/>

259 </x:forEach>
260

261 printing items whose price >= 500

262 <x:forEach var="ord" select="$doc/orders/order">
263 <x:if select="$ord/price>=500">
264 Name: <x:out select="$ord/item"/>


```

```
265 price: <x:out select="$ord/price"/>

266 </x:if>
267 </x:forEach>
268 -----
269 <orders>
270 <order>
271 <item>table</item>
272 <price>1000</price>
273 </order>
274 <order>
275 <item>chair</item>
276 <price>300</price>
277 </order>
278 </orders>
279
280
281
282
283
284
285
286
287
288
289
290
291
292 =====
```

## 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?**

- A. One      B. Two  
C. Three      D. 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", "ablin", 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 TagSupport 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 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 setContentType="image/gif" %>
- C ) <%@ page contentType="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 errorPage="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.  
 C. Two numbers gets printed.  
 E. Four numbers gets printed.

- B. One number gets printed.  
 D. Three 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 byviewing the page source from your Web browser.

JSP Syntax

<!-- comment [ <%= expression %> ] -->

**Example 1**

<!-- This is a commnet sent to client on

<%= (new java.util.Date()).toLocaleString() %>

-->

**Q.Displays in the page source:**

<!-- This is a commnet 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 colent 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 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.

**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 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.

**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 within 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 methods 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 in turn extends the Servlet interface of the javax.servlet package. The generated servlet class thus implements all the methods of these three interfaces. The JspPage interface declares only two methods - 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 interface specifically for the JSP pages serving HTTP requests. This interface declares one method \_jspService().

The jspInit()- The container calls the jspInit() to initialize the 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 to take the instance out of service. It is the last method called in 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 scriplet 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 scriplet 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
```

**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 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.

### **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 occurs 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="<%=(new java.util.Date()).toLocaleString()%>" />
```

### **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.





```
133 cmd.focus();
134 return false;
135 }
136 return true;
137 }
138 function isNumber(cmd)
139 {
140 if(isNaN(cmd.value))
141 {
142 alert("Stu_ID should be numeric value");
143 cmd.focus();
144 return false;
145 }
146 return true;
147 }
148 -----InsertDemo.java-----
149 import javax.servlet.*;
150 import javax.servlet.http.*;
151 import java.io.*;
152 import java.sql.*;
153
154 public class InsertDemo extends HttpServlet
155 {
156 public void doPost(HttpServletRequest req,HttpServletResponse res)
157 {
158 Connection con = null;
159 PreparedStatement ps = null;
160 int rs = 0;
161 try
162 {
163 PrintWriter pw = res.getWriter();
164 res.setContentType("text/html");
165 pw.println("<html><form target = 'display'>");
166 int sid = Integer.parseInt(req.getParameter("stu_id"));
167 String sname = req.getParameter("stu_name");
168 String sadd = req.getParameter("stu_add");
169 Class.forName("oracle.jdbc.driver.OracleDriver");
170 con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","s
171 System.out.println("connection");
172 ps = con.prepareStatement("insert into student_info values(?, ?, ?)");
173 ps.setInt(1,sid);
174 ps.setString(2,sname);
175 ps.setString(3,sadd);
176 rs = ps.executeUpdate();
177 if(rs!=1)
178 pw.println("<h2>Record is problem</h2>");
179 else
180 pw.println("<h2 style = 'position:absolute;left:50;top:50'>One Recor
181
182 ps.close();
183 con.close();
184 }
185 catch(Exception e)
186 {
187 e.printStackTrace();
188 }
189 }
190 public void doGet(HttpServletRequest req,HttpServletResponse res)
191 {
192 try
193 {
194 doPost(req,res);
195 }
196 catch(Exception e)
197 {
198 e.printStackTrace();
199 }
200 }
201 }
202 }
```

```
199 }
200 }
201 }
202 -----Delete.jsp-----
203 <html>
204 <script language = 'javascript'>
205 function isValid()
206 {
207 if(f.stu_id.value == "")
208 {
209 alert(f.stu_id.name + " " + "Field Missed");
210 f.stu_id.focus();
211 return false;
212 }
213 if(isNaN(f.stu_id.value))
214 {
215 alert("Stu_ID should be numeric value");
216 f.stu_id.focus();
217 return false;
218 }
219 return true;
220 }
221 </script>
222 <body>
223 <form name = f action = './delete' method = 'post' onSubmit ='return isValid()'%gt;
224 <table border = 1 cellpadding = 7 cellspacing = 7 align = center>
225 <caption><i><u>Delete Records</u></i></caption>
226 <tr>
227 <th>Student ID<th><input type = "text" name = "stu_id"></tr>
228 <tr>
229 <th>click<th><input type = "submit" value = " DELETE "></tr>
230 </table>
231 </form>
232 </body>
233 </html>
234 -----DeleteDemo.java-----
235 import javax.servlet.*;
236 import javax.servlet.http.*;
237 import java.io.*;
238 import java.sql.*;
239
240 public class DeleteDemo extends HttpServlet
241 {
242 public void doPost(HttpServletRequest req,HttpServletResponse res)
243 {
244 Connection con = null;
245 PreparedStatement ps = null;
246 int rs = 0;
247 try
248 {
249 PrintWriter pw = res.getWriter();
250 res.setContentType("text/html");
251 pw.println("<html><form target = 'display'>");
252 int sid = Integer.parseInt(req.getParameter("stu_id"));
253 Class.forName("oracle.jdbc.driver.OracleDriver");
254 con = DriverManager.getConnection
255 ("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
256 ps = con.prepareStatement("delete from student_info where stu_id=?");
257 ps.setInt(1,sid);
258 rs = ps.executeUpdate();
259 if(rs!=1)
260 pw.println("<h2>Student id is problem</h2>");
261 else
262 pw.println("<h2 style = 'position:absolute;left:50;top:50'>One Record Deleted</h2>");
```

```
265 ps.close();
266 con.close();
267 }
268 catch(Exception e)
269 {
270 e.printStackTrace();
271 }
272 }
273 public void doGet(HttpServletRequest req,HttpServletResponse res)
274 {
275 try{
276 doPost(req,res);
277 }
278 catch(Exception e)
279 {
280 e.printStackTrace();
281 }
282 }
283 }
284 -----Update.jsp-----
285 <html>
286 <script src = "insert.js" language = "javascript">
287 </script>
288 <body>
289 <form name = f action = './update' method = 'post' onSubmit = "return isValid(this)">
290 <table border = 1 cellpadding = 7 cellspacing = 7 align = center>
291 <caption><i><u>Update Records</u></i></caption>
292 <tr>
293 <th>Student ID<th><input type = "text" name = "stu_id"></tr>
294 <tr>
295 <th>Student NAME<th><input type = "text" name = "stu_name"></tr>
296 <tr>
297 <th>Student Address<th><input type = "text" name = "stu_add"></tr>
298 <tr>
299 <th>click<th><input type = "submit" value = " UPDATE "></tr>
300 </table>
301 </form>
302 </body>
303 </html>
304 -----UpdateDemo.java-----
305 import javax.servlet.*;
306 import javax.servlet.http.*;
307 import java.io.*;
308 import java.sql.*;
309
310 public class UpdateDemo extends HttpServlet
311 {
312 public void doPost(HttpServletRequest req,HttpServletResponse res)
313 {
314 Connection con = null;
315 PreparedStatement ps = null;
316 int rs = 0;
317 try
318 {
319 PrintWriter pw = res.getWriter();
320 res.setContentType("text/html");
321 pw.println("<html><form target = 'display'>");
322 int sid = Integer.parseInt(req.getParameter("stu_id"));
323 String sname = req.getParameter("stu_name");
324 String sadd = req.getParameter("stu_add");
325 Class.forName("oracle.jdbc.driver.OracleDriver");
326 con = DriverManager.getConnection
327 ("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
328 ps = con.prepareStatement
329 ("update student_info set stu_name=? ,stu_add=? where stu_id=?");
330 ps.setString(1,sname);
```

```

331 ps.setString(2,sadd);
332 ps.setInt(3,sid);
333 rs = ps.executeUpdate();
334 if(rs!=1)
335 pw.println("<h2>Student ID problem</h2>");
336 else
337 pw.println("<h2 style = 'position:absolute;left:50;top:50'>
338 One Record Updated</h2>");

339
340 ps.close();
341 con.close();
342 }
343 catch(Exception e)
344 {
345 e.printStackTrace();
346 }
347 }
348 public void doGet(HttpServletRequest req,HttpServletResponse res)
349 {
350 try
351 {
352 doPost(req,res);
353 }
354 catch(Exception e)
355 {
356 e.printStackTrace();
357 }
358 }
359 }
360 =====
361 (Mini Project2)
362 =====
363 -----Database Details-----
364 SQL>drop table STUDENT_INFO cascade;
365 SQL>create table STUDENT_INFO(STU_ID number,STU_NAME varchar2(30),STU_ADD varchar2(30));
366 -----
367 -----web.xml-----
368 <?xml version="1.0" encoding="ISO-8859-1"?>
369 <!DOCTYPE web-app
370 PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
371 "http://java.sun.com/dtd/web-app_2_3.dtd">
372 <web-app>
373 <servlet>
374 <servlet-name>First</servlet-name>
375 <servlet-class>StudentDemo</servlet-class>
376 </servlet>
377 <servlet-mapping>
378 <servlet-name>First</servlet-name>
379 <url-pattern>/studentdemo</url-pattern>
380 </servlet-mapping>
381
382 <welcome-file-list>
383 <welcome-file>StudentDetails.jsp</welcome-file>
384 </welcome-file-list>
385 </web-app>
386 -----StudentDetails.jsp-----
387 <%
388 String res =(String)request.getAttribute("msg");
389 String id = (String)request.getAttribute("no");
390 String name = (String)request.getAttribute("name");
391 String addr = (String)request.getAttribute("address");
392 %>
393
394 <html>
395 <script language = 'javascript'>
396

```

```
397 function isInsert()
398 {
399 f.setVal.value = "insert";
400 check();
401 }
402 function isDelete()
403 {
404 f.setVal.value = "delete";
405 check1();
406 }
407 function isUpdate()
408 {
409 f.setVal.value = "update";
410 check();
411 }
412 function isDisplay()
413 {
414 f.setVal.value = "display";
415 check1();
416 }
417
418 function check()
419 {
420 if(f.stu_id.value == "")
421 {
422 alert("Please provide Student ID");
423 f.stu_id.focus();
424 }
425 else if(isNaN(f.stu_id.value))
426 {
427 alert("Student ID should be numeric");
428 f.stu_id.value = "";
429 f.stu_id.focus();
430 }
431 else if(f.stu_name.value == "")
432 {
433 alert("Please provide Student Name");
434 f.stu_name.focus();
435 }
436 else if(f.stu_add.value == "")
437 {
438 alert("Please provide Student Address");
439 f.stu_add.focus();
440 }
441 else
442 f.submit();
443 }
444
445 function check1()
446 {
447 if(f.stu_id.value == "")
448 {
449 alert("Please provide Student ID");
450 f.stu_id.focus();
451 }
452 else if(isNaN(f.stu_id.value))
453 {
454 alert("Student ID should be numeric");
455 f.stu_id.value = "";
456 f.stu_id.focus();
457 }
458 else
459 f.submit();
460 }
461
462 function disp1_output()
```

```
463 {
464 var i = <%=id%>;
465 var n ='<%=name%>';
466 var a ='<%=addr%>';
467 f.stu_id.value = i;
468 f.stu_name.value=n;
469 f.stu_add.value = a;
470 }
471
472 function isReloading()
473 {
474 <%
475 if(res != null)
476 {
477 if(res.equals("1"))
478 out.println("alert('Information stored successfully')");
479 else if(res.equals("2"))
480 out.println("alert('Given Student ID already exists')");
481 else if(res.equals("3"))
482 out.println("alert('Information deleted successfully')");
483 else if(res.equals("4"))
484 out.println("alert('Given Student ID is Invalid')");
485 else if(res.equals("5"))
486 out.println("alert('Information updated successfully')");
487 else if(res.equals("6"))
488 {
489 out.println("disp1_output()");
490 }
491 else if(res.equals("7"))
492 out.println("alert('Unknown problem occured.')");
493 }
494 %
495 }
496
497 </script>
498
499 <body onLoad='isReloading()'>
500 <form name=f action='./studentdemo' method='post'>
501
502 <center>
503
504
505 HCL TECHNOLOGIES
506 <hr color=pink width=100%>
507 </center>
508
509 <table border=1 cellpadding=7 cellspacing=7 align=center>
510
511 <caption>
512 <i><u>Student Table</u></i>
513 </caption>
514
515 <tr>
516 <th>Student ID</th>
517 <th><input type="text" name="stu_id"></th>
518 </tr>
519 <tr>
520 <th>Student Name</th>
521 <th><input type="text" name="stu_name"></th>
522 </tr>
523 <tr>
524 <th>Student Address</th>
525 <th><input type="text" name="stu_add"></th>
526 </tr>
527 <tr>
528 <th colspan=4>
```

```
529 <input type='button' value='Insert' onClick='isInsert()'>
530 <input type='button' value='Delete' onClick='isDelete()'>
531 <input type='button' value='Update' onClick='isUpdate()'>
532 <input type='button' value='Display' onClick='isDisplay()'>
533 </tr>
534 </table>
535
536 <input type='text' name='setVal' readonly
537 style='visibility:hidden'>
538 </form>
539 </body>
540 </html>
541 -----StudentDemo.java-----
542 import javax.servlet.*;
543 import javax.servlet.http.*;
544 import java.io.*;
545 import beans.DbConnector;
546
547 public class StudentDemo extends HttpServlet
548 {
549 public void doPost(HttpServletRequest req,HttpServletResponse res)
550 {
551 HttpSession session = null;
552 int flag = 0;
553 int forward = 0;
554 int output = 0;
555
556 String store = null;
557 String value = null;
558 String dispval1,dispval2,dispval3;
559
560 DbConnector dbc;
561
562 try
563 {
564 session = req.getSession(true);
565 PrintWriter out = res.getWriter();
566 res.setContentType("text/html");
567
568 int s_id = Integer.parseInt(req.getParameter("stu_id"));
569 String s_name = req.getParameter("stu_name");
570 String s_add = req.getParameter("stu_add");
571
572 String checkAction = req.getParameter("setVal");
573
574 dbc = new DbConnector(s_id,s_name,s_add);
575
576 if(checkAction.equals("insert"))
577 output = dbc.insert();
578 else if(checkAction.equals("delete"))
579 output = dbc.delete();
580 else if(checkAction.equals("update"))
581 output = dbc.update();
582 else if(checkAction.equals("display"))
583 {
584 output = dbc.display();
585
586 dispval1 = String.valueOf(dbc.getID());
587 dispval2 = dbc.getName();
588 dispval3 = dbc.getAdd();
589
590 req.setAttribute("no", dispval1);
591 req.setAttribute("name", dispval2);
592 req.setAttribute("address", dispval3);
593 }
594 }
```

```
595 value = String.valueOf(output);
596 req.setAttribute("msg", value);
597 RequestDispatcher rd = req.getRequestDispatcher("StudentDetails.jsp");
598 rd.forward(req,res);
599 } // try
600 catch(Exception e)
601 {
602 e.printStackTrace();
603 }
604 } // doPost()
605 public void doGet(HttpServletRequest req,HttpServletResponse res)
606 {
607 try{
608 doPost(req,res);
609 }
610 catch(Exception e)
611 {
612 e.printStackTrace();
613 }
614 } // doGet()
615 } // class
-----DbConnector.java-----
616 package beans;
617
618 import java.io.*;
619 import java.sql.*;
620
621 public class DbConnector
622 {
623 private Connection con = null;
624
625 private int id;
626 private String name;
627 private String add;
628
629 public DbConnector()
630 {
631 }
632
633
634 public DbConnector(int id, String name, String add)
635 {
636 this.id = id;
637 this.name = name;
638 this.add = add;
639
640 con = getConnection();
641 }
642
643 public void setID(int id)
644 {
645 this.id = id;
646 }
647
648 public int getID()
649 {
650 return id;
651 }
652
653 public void setName(String name)
654 {
655 this.name = name;
656 }
657
658 public String getName()
659 {
660 return name;
```

```
661 }
662
663 public void setAdd(String add)
664 {
665 this.add = add;
666 }
667
668 public String getAdd()
669 {
670 return add;
671 }
672
673 public Connection getConnection()
674 {
675 try
676 {
677 Class.forName("oracle.jdbc.driver.OracleDriver");
678 con=DriverManager.getConnection
679 ("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
680 }catch(Exception e)
681 {
682 e.printStackTrace();
683 }
684 return con;
685 }
686
687 public boolean exists(int id)
688 {
689 PreparedStatement ps = null;
690 try
691 {
692 ps = con.prepareStatement("SELECT COUNT(*) FROM STUDENT_INFO " +
693 " WHERE STU_ID = ?");
694 ps.setInt(1, id);
695
696 ResultSet rs = ps.executeQuery();
697 rs.next();
698
699 if(rs.getInt(1) == 0)
700 return false;
701 else
702 return true;
703 }
704 catch(Exception e)
705 {
706 e.printStackTrace();
707 return false;
708 }
709 finally
710 {
711 if(ps != null)
712 {
713 try
714 {
715 ps.close();
716 }
717 catch(Exception e)
718 {
719 e.printStackTrace();
720 }
721 }
722 }
723 } // exists()
724
725 public int insert()
726 {
```

```
727 PreparedStatement ps = null;
728 try
729 {
730 boolean alreadyPresent = exists(this.getID());
731
732 if(! alreadyPresent)
733 {
734 ps = con.prepareStatement("INSERT INTO STUDENT_INFO VALUES(?, ?, ?)");
735 ps.setInt(1, id);
736 ps.setString(2, name);
737 ps.setString(3, add);
738 ps.executeUpdate();
739 return 1;
740 }
741 else
742 return 2;
743 }
744 catch(Exception e)
745 {
746 e.printStackTrace();
747 return 7;
748 }
749 finally
750 {
751 if(ps != null)
752 {
753 try
754 {
755 ps.close();
756 }
757 catch(Exception e)
758 {
759 e.printStackTrace();
760 }
761 }
762 }
763 } // insert()
764
765 public int delete()
766 {
767 PreparedStatement ps = null;
768 try
769 {
770 boolean alreadyPresent = exists(this.getID());
771
772 if(alreadyPresent)
773 {
774 ps = con.prepareStatement("DELETE FROM STUDENT_INFO WHERE STU_ID=?");
775 ps.setInt(1,id);
776 ps.executeUpdate();
777 return 3;
778 }
779 else
780 return 4;
781 }
782 catch(Exception e)
783 {
784 e.printStackTrace();
785 return 7;
786 }
787 finally
788 {
789 if(ps != null)
790 {
791 try
792 {
```

```
793 ps.close();
794 }
795 catch(Exception e)
796 {
797 e.printStackTrace();
798 }
799 }
800 } // delete()
801
802 public int update()
803 {
804 PreparedStatement ps = null;
805 try
806 {
807 boolean alreadyPresent = exists(this.getID());
808
809 if(alreadyPresent)
810 {
811 ps = con.prepareStatement("UPDATE STUDENT_INFO SET
812 STU_NAME=? ,STU_ADD=? WHERE STU_ID = ?");
813 ps.setString(1,name);
814 ps.setString(2,add);
815 ps.setInt(3,id);
816 ps.executeUpdate();
817 return 5;
818 }
819 else
820 return 4;
821 }
822 catch(Exception e)
823 {
824 e.printStackTrace();
825 return 7;
826 }
827 finally
828 {
829 if(ps != null)
830 {
831 try
832 {
833 ps.close();
834 }
835 catch(Exception e)
836 {
837 e.printStackTrace();
838 }
839 }
840 }
841 } // update()
842
843 public int display()
844 {
845 PreparedStatement ps = null;
846
847 boolean alreadyPresent = exists(this.getID());
848
849 try
850 {
851 if(alreadyPresent)
852 {
853 ps = con.prepareStatement("SELECT STU_NAME,STU_ADD FROM
854 STUDENT_INFO WHERE STU_ID=?");
855 ps.setInt(1,id);
856 ResultSet rs = ps.executeQuery();
857
858 }
```

```

859 if(rs.next())
860 {
861 String nm = rs.getString(1);
862 this.setName(nm);
863
864 String ad = rs.getString(2);
865 this.setAdd(ad);
866 }
867
868 rs.close();
869
870 return 6;
871 }
872 else
873 return 4;
874 }
875 catch(Exception e)
876 {
877 e.printStackTrace();
878 return 7;
879 }
880 finally
881 {
882 if(ps != null)
883 {
884 try
885 {
886 ps.close();
887 }
888 catch(Exception e)
889 {
890 e.printStackTrace();
891 }
892 }
893 }
894 } // display()
895 } // class
896 =====
897 PROJECT On BookSearch using mvc2 architecture (MinProject3)
898 =====
899 -----SQL.txt-----
900 CREATE TABLE SELECT_BOOKS (BOOKID VARCHAR2(10),
901 BOOKNAME VARCHAR2(30),
902 AUTHORNAME VARCHAR2(30),
903 STATUS VARCHAR2(10),
904 CATEGORY VARCHAR2(20)
905);
906 -----web.xml-----
907 <?xml version="1.0" encoding="ISO-8859-1"?>
908 <!DOCTYPE web-app
909 PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
910 "http://java.sun.com/dtd/web-app_2_3.dtd">
911
912 <web-app>
913 <welcome-file-list>
914 <welcome-file>Search.jsp</welcome-file>
915 </welcome-file-list>
916
917 <servlet>
918 <servlet-name>search</servlet-name>
919 <servlet-class>com.nt.controller.MainServlet</servlet-class>
920 </servlet>
921
922 <servlet-mapping>
923 <servlet-name>search</servlet-name>
924 <url-pattern>/controller</url-pattern>

```

```
925 </servlet-mapping>
926 </web-app>
927 -----Search.jsp-----
928 <html>
929
930 <script language = "javascript">
931
932 function isHtml()
933 {
934 f.source.value = "Html";
935 validate();
936 }
937
938 function isExcel()
939 {
940 f.source.value = "Excel";
941 validate();
942 }
943
944 function validate()
945 {
946 if(f.category.selectedIndex == '0')
947 {
948 alert("You should select Category !!!");
949 f.category.focus();
950 return false;
951 }
952 else
953 {
954 f.submit();
955 return true;
956 }
957 }
958
959 </script>
960
961 <body>
962 <form name=f action="controller" method="post">
963 <center>
964
965 Search for Books
966 <hr color=orange width=50%>
967 </center>
968
969 <table border=1 cellpadding=4 cellspacing=4 align=center bgcolor='lavender'>
970 <tr>
971 <th>Select Category</th>
972 <th>
973 <select name='category'>
974 <option selected value=''>Select a value</option>
975 <option value='java'>JAVA</option>
976 <option value='.net'>.NET</option>
977 <option value='jscript'>JavaScript</option>
978 </select>
979 </th>
980 </tr>
981 <tr>
982 <th><input type='button' value='Html Output' onClick='isHtml()'></th>
983 <th><input type='button' value='Excel Output' onClick='isExcel()'></th>
984 </tr>
985 </table>
986
987 <input type='text' name='source' readonly style='visibility:hidden'>
988 </form>
989 </body>
990 </html>
```

```

991 -----MainServlet.java-----
992 package com.nt.controller;
993 import javax.servlet.*;
994 import javax.servlet.http.*;
995 import java.io.*;
996 import java.util.*;
997 import com.nt.service.DbConnector;
998
999 public class MainServlet extends HttpServlet
1000 {
1001 public void doGet(HttpServletRequest req, HttpServletResponse res)
1002 {
1003 try
1004 {
1005 //read form data
1006 String cat = req.getParameter("category");
1007 String checkAction=req.getParameter("source");
1008 //Use Service class
1009 DbConnector dbc = new DbConnector();
1010 ArrayList al = dbc.search(cat);
1011 //keep results in request attributes
1012 req.setAttribute("list", al);
1013 req.setAttribute("category", cat);
1014 //decide the result page
1015 String target;
1016 if(checkAction.equalsIgnoreCase("Html"))
1017 target = "HtmlPrint.jsp";
1018 else
1019 target = "ExcelScreen.jsp";
1020
1021 RequestDispatcher rd = null;
1022 rd = req.getRequestDispatcher(target);
1023 if(rd != null)
1024 rd.forward(req,res);
1025 } // try
1026 catch(Exception e)
1027 {
1028 e.printStackTrace();
1029 }
1030 } // doGet(-,-)
1031 public void doPost(HttpServletRequest req, HttpServletResponse res)
1032 {
1033 doGet(requeste,response);
1034 } //doPost(-,-)
1035 } // class
1036 -----DbConnector.java-----
1037 package com.nt.service;
1038
1039 import java.io.*;
1040 import java.sql.*;
1041 import java.util.ArrayList;
1042 import com.nt.bo.BookBean;
1043
1044 public class DbConnector
1045 {
1046 private static final SEARCH_QRY="SELECT BOOKID, BOOKNAME, AUTHORNAME, STATUS FROM SELECT_BOOKS WHERE";
1047 public Connection getConnection()
1048 {
1049 Connection con = null;
1050 try
1051 {
1052 Class.forName("oracle.jdbc.driver.OracleDriver");
1053 con = DriverManager.getConnection
1054 ("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
1055 }
1056 catch(Exception e)
1057 {

```

```
1057 e.printStackTrace();
1058 }
1059 return con;
1060 }
1061
1062 public ArrayList<BookBean> search(String category)
1063 {
1064 Connection con = getConnection();
1065 PreparedStatement ps = null;
1066 ResultSet rs = null;
1067 ArrayList<BookBean> al = new ArrayList<BookBean>();
1068 try
1069 {
1070 //create PreparedStatement obj
1071 ps = con.prepareStatement(SEARCH_QRY);
1072 //set values to Query params
1073 ps.setString(1, category);
1074 //execute the Query
1075 rs = ps.executeQuery();
1076 //copy the records of ResultSet to ArrayList
1077 while(rs.next())
1078 {
1079 BookBean b = new BookBean();
1080 b.setBookId(rs.getString(1));
1081 b.setBookName(rs.getString(2));
1082 b.setAuthorName(rs.getString(3));
1083 b.setStatus(rs.getString(4));
1084 al.add(b);
1085 }
1086 rs.close();
1087 }catch(Exception e)
1088 {
1089 e.printStackTrace();
1090 }
1091 finally
1092 {
1093 if(ps != null)
1094 {
1095 try
1096 {
1097 ps.close();
1098 }
1099 catch(Exception e)
1100 {
1101 e.printStackTrace();
1102 }
1103 }
1104 if(con != null)
1105 {
1106 try
1107 {
1108 con.close();
1109 }
1110 catch(Exception e)
1111 {
1112 e.printStackTrace();
1113 }
1114 }
1115 } // finally
1116 return al;
1117 } // search()
1118 } // class
1119 -----BookBean.java-----
1120 package com.nt.bo;
1121
1122 public class BookBean implements java.io.Serializable
```

```
1123 { private String bookid;
1124 private String bookname;
1125 private String authornname;
1126 private String status;
1127
1128 public void setBookId(String bookid)
1129 {
1130 this.bookid = bookid;
1131 }
1132 public String getBookId()
1133 {
1134 return bookid;
1135 }
1136 public void setBookName(String bookname)
1137 {
1138 this.bookname = bookname;
1139 }
1140 public String getBookName()
1141 {
1142 return bookname;
1143 }
1144 public void setAuthorName(String authornname)
1145 {
1146 this.authornname = authornname;
1147 }
1148 public String getAuthorName()
1149 {
1150 return authornname;
1151 }
1152 public void setStatus(String status)
1153 {
1154 this.status = status;
1155 }
1156 public String getStatus()
1157 {
1158 return status;
1159 }
1160 } // BookBean class.
1161 -----HtmlPrint.jsp-----
1162 <%@page import="java.util.ArrayList, com.nt.bo.BookBean" %>
1163 <%
1164 ArrayList al = (ArrayList)request.getAttribute("list");
1165 String cat = (String)request.getAttribute("category");
1166 %>
1167
1168 <html>
1169
1170 <script language='javascript'>
1171 function showprint()
1172 {
1173 frames.focus();
1174 frames.print();
1175 }
1176 </script>
1177
1178 <body>
1179 <center><h2><u>
1180 Books belonging to category <%= cat.toUpperCase().%>
1181 <u></h2></center>
1182

1183
1184 <table border="1" width="100%">
1185 <tr>
1186 <th>Sno</th>
1187 <th>BookId</th>
1188 <th>BookName</th>
```



```

1255 -----DB script.txt-----
1256 SQL> desc employereg;
1257 Name Null? Type
1258 -----
1259 EMP_ID NOT NULL NUMBER(4)
1260 EMP_NAME VARCHAR2(10)
1261 EMP_ADD VARCHAR2(10)
1262 EMP_RESUME VARCHAR2(50)
1263 EMP_PIC VARCHAR2(50)
1264
1265 create table employereg (EMP_ID NUMBER(4) primary key,
1266 EMP_NAME VARCHAR2(10),
1267 EMP_ADD VARCHAR2(10),
1268 EMP_RESUME VARCHAR2(50),
1269 EMP_PIC VARCHAR2(50))
1270 -----Home.html-----
1271 <center>
1272 Register new Employee
1273

1274 view and Download Emp Details
1275 -----Register.jsp-----
1276 body bgcolor=wheat>
1277 <center><h1>Employee Registration Page</h1>
1278 <form action="reg" method="post" enctype="multipart/form-data">
1279
1280 <table>
1281 <tr>
1282 <td>Employee ID</td><td><input type=text name=tid></td>
1283 </tr>
1284 <tr>
1285 <td>Employee Name</td><td><input type=text name=tname></td>
1286 </tr>
1287 <tr>
1288 <td>Employee Add</td><td><input type=text name=tadd></td>
1289 </tr>
1290 <tr>
1291 <td>Employee photo</td><td><input type=file name=tphoto></td>
1292 </tr>
1293 <tr>
1294 <td>Employee resume</td><td><input type=file name=tresume></td>
1295 </tr>
1296 <tr>
1297 <td><input type=submit value=register></td>
1298 </tr>
1299 </form>
1300 </center>
1301 </body>
1302 -----web.xml-----
1303 <web-app>
1304
1305 <servlet>
1306 <servlet-name>reg</servlet-name>
1307 <servlet-class>com.nt.servlet.RegisterServlet</servlet-class>
1308 </servlet>
1309
1310 <servlet-mapping>
1311 <servlet-name>reg</servlet-name>
1312 <url-pattern>/reg</url-pattern>
1313 </servlet-mapping>
1314
1315 <welcome-file-list>
1316 <welcome-file>Home.html</welcome-file>
1317 </welcome-file-list>
1318
1319 </web-app>
1320 -----RegisterServlet.java-----

```

```
I21 //RegisterServlet.java
I22 package com.nt.servlet;
I23 import java.io.IOException;
I24 import java.io.PrintWriter;
I25 import java.sql.Connection;
I26 import java.sql.DriverManager;
I27 import java.sql.PreparedStatement;
I28 import java.util.ArrayList;
I29 import java.util.Vector;
I30
I31 import javax.servlet.ServletException;
I32 import javax.servlet.http.HttpServlet;
I33 import javax.servlet.http.HttpServletRequest;
I34 import javax.servlet.http.HttpServletResponse;
I35
I36 import javazoom.upload.MultipartFormDataRequest;
I37 import javazoom.upload.UploadBean;
I38 import javazoom.upload.UploadParameters;
I39
I40 public class RegisterServlet extends HttpServlet
I41 {
I42 public void doGet (HttpServletRequest req, HttpServletResponse res)
I43 throws ServletException, IOException
I44 {
I45 doPost(req,res);
I46 }
I47
I48 public void doPost (HttpServletRequest req, HttpServletResponse res)
I49 throws ServletException, IOException
I50 {
I51
I52 String resumePath="c:/store/resumes";
I53 String photoPath="c:/store/photos";
I54 PrintWriter out = res.getWriter();
I55 try
I56 {
I57 // get special request obj given by Java zoom api
I58 MultipartFormDataRequest nreq = new MultipartFormDataRequest(req);
I59 int eId=Integer.parseInt(nreq.getParameter("tid"));
I60 String eName=nreq.getParameter("tname");
I61 String eAdd=nreq.getParameter("tadd");
I62 /*String ePhoto=nreq.getParameter("tphoto"); shows Null
I63 String eResume=nreq.getParameter("tresume"); shows Null */
I64
I65 // settings related to Resume uploading
I66 UploadBean upb = new UploadBean();
I67 upb.setFolderstore(resumePath);
I68 upb.setOverwrite(false);
I69 upb.store(nreq,"tresume");// complets files uploading
I70
I71 // setting related photo uploading
I72 upb.setFolderstore(photoPath);
I73 upb.setOverwrite(false);
I74 upb.store(nreq,"tphoto"); complets files uploading
I75
I76 // get the file names of the uploaded files
I77 Vector history = upb.getHistory();
I78
I79 ArrayList <String> filesName=new ArrayList<String>();
I80 for (int i=0;i<history.size();i++)
I81 {
I82 UploadParameters up = (UploadParameters) history.elementAt(i);
I83 filesName.add(up.getFilename());
I84 }
I85
I86 // store the paths of the uploaded files to c:\store folder
```

```

1387 Class.forName("oracle.jdbc.driver.OracleDriver");
1388 Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","");
1389 PreparedStatement ps=con.prepareStatement("insert into EmployeeReg values(?,?,?,?,?,?)");
1390 ps.setInt(1,eId);
1391 ps.setString(2, eName);
1392 ps.setString(3, eAdd);
1393 ps.setString(4,resumePath+"/"+filesName.get(0));
1394 ps.setString(5,photoPath+"/"+filesName.get(1));
1395
1396 int i=ps.executeUpdate();
1397 if(i==1)
1398 out.println("<h1>Successfully uploaded and Stored in DataBase</h1>");
1399 else
1400 out.println("<h1>Failed in uploading</h1>");
1401
1402 }//try
1403 catch(Exception e)
1404 {
1405 out.println(e);
1406 }//catch
1407 }//doPost ()
1408 }//class
1409
1410 -----View.jsp-----
1411 <%@page import="java.io.File,java.util.*,java.sql.*"%>
1412
1413 <!-- Retrive records from DB table -->
1414 <H1>List of All files under C:\store</H1>
1415 <%
1416 Class.forName("oracle.jdbc.driver.OracleDriver");
1417 Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger")
1418 PreparedStatement ps=con.prepareStatement("select * from EmployeeReg");
1419 ResultSet rs=ps.executeQuery();
1420 %>
1421
1422 <!-- display employee details as html table content-->
1423 <body bgcolor=wheat>
1424 <table border=1>
1425 <tr><td>Employee Name</td><td>Employee Address</td><td>Employee Resume</td><td>Employee Photo</td></
1426
1427 <%
1428 while(rs.next())
1429 {
1430 %>
1431 <tr>
1432 <td><%=rs.getString(2)%></td>
1433 <td><%=rs.getString(3)%></td>
1434 <td><a href='Download.jsp?resumeId=<%=rs.getString(1)%>'>Download Here</td>
1435 <td><a href='Download.jsp?photoId=<%=rs.getString(1)%>'>Download Here</td>
1436 </tr>
1437 <%} %>
1438
1439 </table>
1440 </body>
1441 -----Download.jsp-----
1442 <%@page import="java.io.*,javax.servlet.*,javax.servlet.http.*,java.sql.*" %>
1443
1444 <!-- get the path of the file to downloaded -->
1445 <%
1446 String fileName="";
1447 String queryText="";
1448
1449 Class.forName("oracle.jdbc.driver.OracleDriver");
1450 Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
1451
1452 if(request.getParameter("resumeId")!=null)

```

```
1453 queryText="select emp_resume from employeereg where emp_id="+request.getParameter("resumeId");
1454 else
1455 queryText="select emp_pic from employeereg where emp_id="+request.getParameter("photoId");
1456
1457 PreparedStatement ps=con.prepareStatement(queryText);
1458
1459 ResultSet rs=ps.executeQuery();
1460 while(rs.next()){
1461 fileName=rs.getString(1);
1462 }
1463 // prepare settings to download the files
1464
1465 File f= new File(fileName);
1466 int length = 0;
1467 //get Servletoutput Stream
1468 ServletOutputStream op = response.getOutputStream();
1469 // get MIME type of file to downloaded
1470 String mimetype = application.getMimeType(fileName);
1471 // set the mime type of file to be downloaded as response set content type
1472 response.setContentType((mimetype != null) ? mimetype : "application/octet-stream");
1473 //set the file.length as response content length
1474 response.setContentLength((int)f.length());
1475
1476 response.setHeader("Content-Disposition", "attachment;filename="+fileName);
1477
1478 // complete file downloading using buffering
1479 byte[] buf = new byte[1024];
1480 DataInputStream in = new DataInputStream(new FileInputStream(f));
1481
1482 while ((in != null) && ((length = in.read(buf)) != -1))
1483 {
1484 op.write(buf,0,length);
1485 }
1486 //close streams
1487 in.close();
1488 op.flush();
1489 op.close();
1490
1491 %>
1492
```