

**SOFTWARE
DEVELOPMENT**

IT-IMS

ENGINEERING SERVICES

SOFT SKILLS



seed
Official Curriculum

Copyrights of SEED Infotech Ltd. | Official Curriculum

Java Web Component Developer Student Book and Lab Manual

Module Code : [M-ILT-Jav-00016-WCD-I-En]

Copyright @ Avani Publications.

Author : Mr. Chetan Natu

This edition has been printed and published in house by Avani Publications.

This book including interior design, cover design and icons may not be duplicated/reproduced or transmitted in anyway without the express written consent of the publisher, except in the form of brief excerpts quotations for the purpose of review. The information contained herein is for the personal use of the reader and may not be incorporated in any commercial programs, other books, databases or any kind of software without written consent of the publisher. Making copies of this book or any portion thereof for any purpose other than your own is a violation of copyright laws.

Limits of Liability/Disclaimer of Warranty : The author and publisher have used their best efforts in preparing this book. Avani Publications make no representation or warranties with respect to the accuracy or completeness of the contents of this book, and specifically disclaim any implied warranties of merchantability or fitness for any particular purpose. There are no warranties, which extend beyond the descriptions contained in this paragraph. No warranty may be created or extended by sales representatives or written sales materials. The accuracy and completeness of the information provided herein and the opinions stated herein are not guaranteed or warranted to produce any particular results and the advice and strategies contained herein may not be suitable for every individual. Author or Avani Publications shall not be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential or other damages.

Trademarks : All brand names and product names used in this book are trademarks, registered trademarks or trade names of their respective holders. Avani Publications is not associated with any product or vendor mentioned in this book.

Note: All API references are taken from Java documentation. (<http://docs.oracle.com/javase/7/docs/api/>)

Print Edition : January 2012

Issue No./ Date : 01 / Jan. 25, 2012 **Revision No. & Date :** 00 / Jan. 25, 2012

Avani Publications

Flat no 2, Ground Floor, Radhakrishna Apts., Prabhat Road, Lane No. 10, Erandawana Pune 411004, Maharashtra, India.

▶ ▶ ▶ **Contents**

Sr. No.	Chapter Name	Page No.
0	Enhancing Employability	iv
1.	Java Database Connectivity(JDBC)	1
2.	Advanced JDBC	43
3.	Introduction to Web Technologies	78
4.	HTML and JavaScript	102
5.	XML in Java	137
6.	Servlets	162
7.	Java Servlets (Session, Filters)	190
8.	JSP (Basic JSP, MVC and Action Tags)	220
9.	JSP (Custom Tags, EL and JSTL)	247
10.	Appendix A	277
11.	Appendix B	279
12.	Lab Manual	283
13.	Appendix C	312
14.	Appendix D	325

Enhancing Employability

Employability depends on the knowledge, skills and abilities that individuals possess, the way they use these to solve problems and contribute to the growth of the employing organization and the society. Employability skills are those skills that are necessary for getting, keeping and doing well in a job.

Employability can be defined as an equation.



Promise of this book is to help strengthen your “WCD programming skill” which can be classified under **Hard Skills** of employability equation.

Why learn ‘WCD’ programming?

With increasing awareness of the internet it becomes a necessity for any business to have a presence on the internet. Web application created using web components help in creating web applications reaching thousands of clients worldwide. Individuals seeking employment should learn Web Component Development for the following reasons:

- a) Java is a simple, platform independent and robust object-oriented programming language.
- b) Java web applications are used in various sectors including banking, insurance, retail, media, education, manufacturing etc.
- c) E-commerce, Gaming, Mobile, Media and many more types of applications are being developed using Java.
- d) The enterprise applications that use technologies like application servers, business process management (BPM), Portal Servers are either created using Java or can be extended using Java.

Frameworks like Struts, Spring, Hibernate are being widely adopted by industry to build applications rapidly. Knowledge of such frameworks is becoming essential for application developers.

- e) There is continuous demand for Web Component Developers in software industry in large numbers.

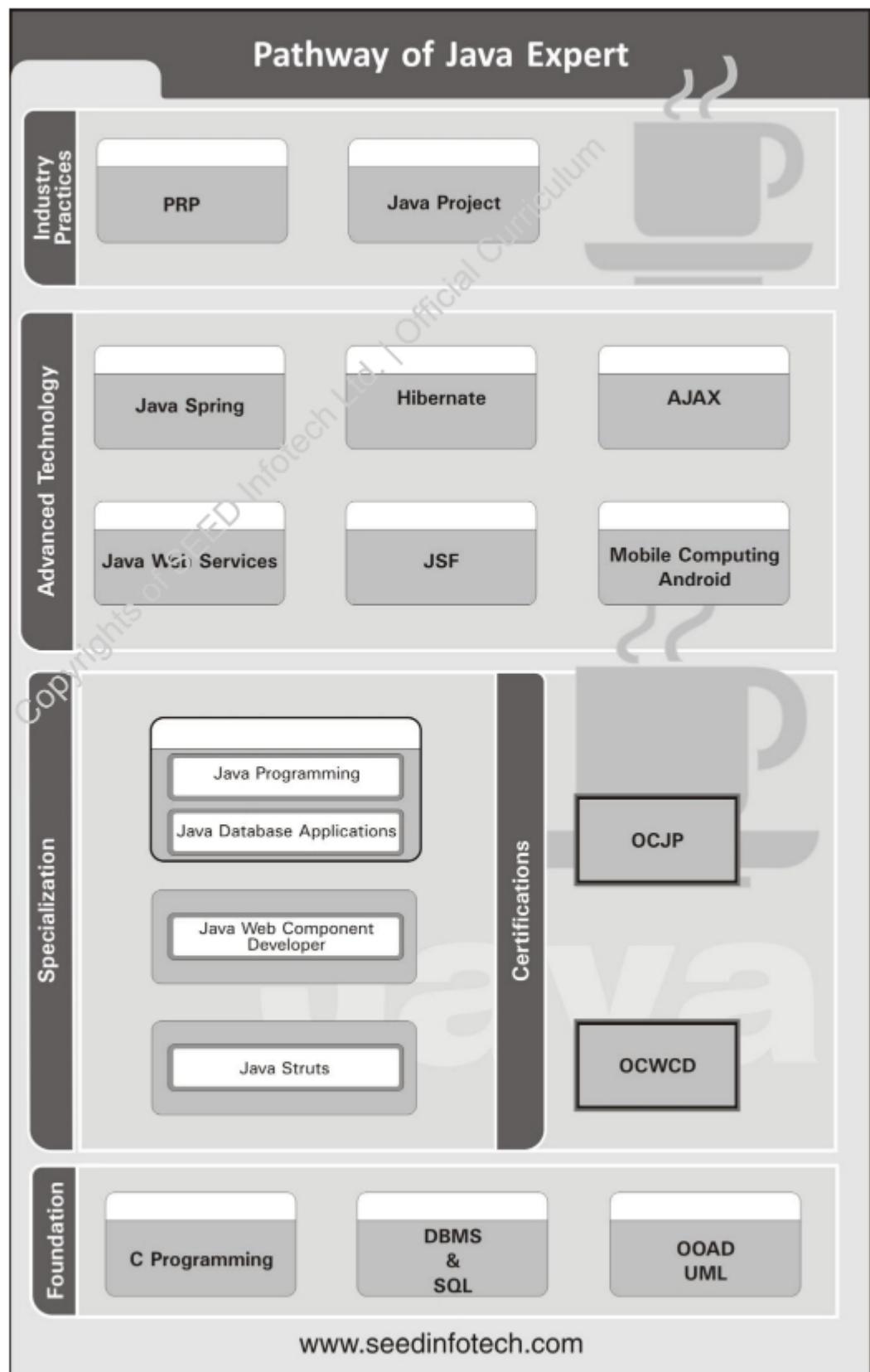
Role of this book

This book is a step by step guide to master WCD programming skills and would aid and reinforce the learning in the classroom. To master any programming language one needs hands-on practice along with clarity of concepts. This student book combined with lab manual is designed to serve the purpose.

The smart tips embedded in the form of Tech App, Interview Tip, Additional Reading, Best Practices, etc. would help increase your curiosity and also help you to become expert with knowledge of peripheral concepts.

We strive hard to make technical contents of this book completely error free. However some mistakes might have slipped our attention. We request the reader to send us any such errors you sight which will help us in improving this book further. Your issues or suggestions are welcome on email at

product-issues@seedinfotech.com



“Beyond Obvious” Icons

In the student book, we have included special icons (Beyond Obvious Icons) in the form of footnotes that are interspersed in the study material to give you the precise context of the concept you are learning. As you get accustomed to this way of learning, you will enjoy the fun of it which will make this learning highly productive!



This icon indicates a particular best practice which is followed while developing the applications, in design, in coding etc. Knowledge of best practices makes one a good developer or designer.



This icon indicates the points which are important from your technical interview. Before interview, you may visit these small tips as quick revision pointers.



This icon indicates the features which are new and available in new version JDK 7.



This icon indicates the features which are new and available in new version Servlet 3.



This icon suggests that it is good for the student to go through this additional reading material to bring more clarity to the concepts.



This icon indicates that this is a group discussion or group exercise. This is added for collaborative learning and problem solving. In the job environment one needs to take part in such discussions to arrive at the solution.



This icon indicates that more details are provided with respect to application of the technology/tool/concept which you are learning.

This is application of technology learned to solve the real world problem.



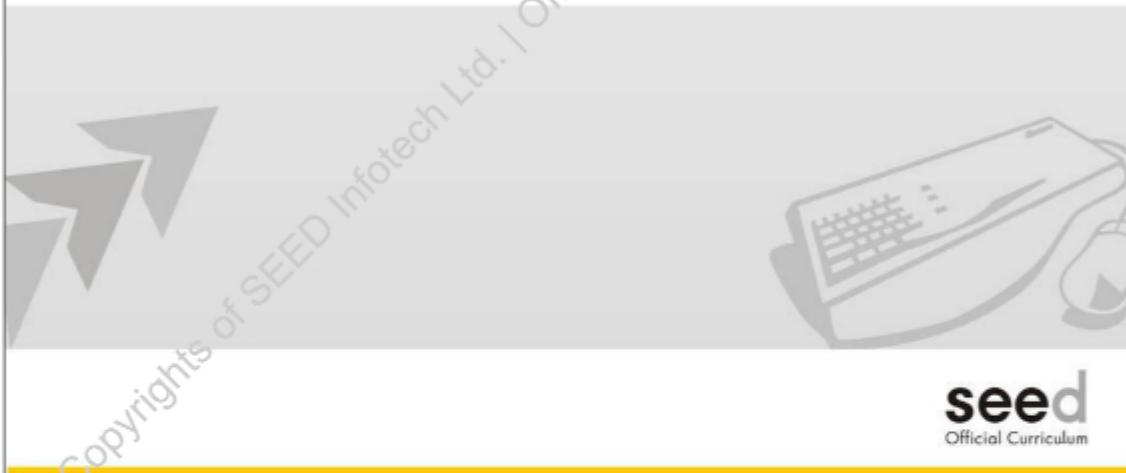
This icon indicates a important note or tip from the application design perspective.



This icon indicates quiz to be solved in the classroom. This is for reinforcement of what you have learnt by challengin you through the questions.

Chapter - 1

Java Database Connectivity(JDBC)



This chapter covers database connectivity using JDBC API. It includes SQL, Parameterized SQL, Stored Procedure, JDBC Drivers, Database Transactions, Result set metadata and Database metadata.

Objectives

At the end of this chapter you will be able to:

- Compare File versus Database as storage medium.
- List features provided by databases.
- List Types of SQL.
- Identify the need for JDBC.
- List various technologies used to access data from databases.
- State the algorithm for reading data from a database.
- Construct a Java program to execute an SQL query.
- List different types of JDBC drivers with their usage.
- Identify various parts of JDBC URL and construct it as per need.
- Use SQL to update the database records.

- Use JDBC construct to ensure ACID properties of transactions.
- Construct and use parameterized SQL.
- Invoke a given stored procedure.
- Use `ResultSetMetaData` class to fetch additional information about the result set like column names.
- List important methods of `DatabaseMetaData` class and their usage.

Storage: Files Vs Database

- A database is a system which helps to organize, store, and retrieve large amounts of data easily.



	File	Database
Data Redundancy	High	Low
Data Integrity	No	Yes
Data Ownership	File wise	Data item
Security Level	File	Data item
Multi-access	Difficult	Easier

Data is the most important part for any business application. Essentially, programs are created to manage this business data. Consider an example of data about all the employees in an organization. It is maintained in the files in different departments like HR, Accounts and the department to which the employee belongs to. In this case, same data is duplicated at multiple places. Extra computer storage is needed to store these duplicate copies of data. Additionally, when the data is stored at different locations, it is difficult to make the changes at all the locations at the same time. This leads to a problem called data integrity. Data integrity refers to the correctness of data maintained. For example, an employee leaves the organization and the accounts department is unaware about it, the salary processing will take place for that employee. Since each department being an owner may modify their own copy making the data inconsistent. Hence control over a group of records or data cannot be made uniform at the organizational level. File system supports security at file level. Access can be given to an entire file or denied to the entire file. This acts as a severe limitation when one wants to control data at record level.

A database is a system which helps to organize, store, and retrieve large amounts of data. All these limitations are overcome by using a database in which the data is stored at a central location. All applications can have uniform access to this data. Data duplication or redundancy is greatly reduced. Data integrity is maintained as the changes are done at one place in the database. All the applications can be given selective access to the data items depending on the requirement. As the data can be accessed at data item level rather than file level, security can be controlled at data item level. Modern business applications use databases to store the data instead of files. Relational database management systems or RDBMS are widely used for this purpose. Examples of leading RDBMS softwares are Oracle, SQL Server, IBM DB2, Sybase, MySQL etc.

Relational Database Management System



- Relational Databases or RDBMS stores data in the form of rows and columns.

3

A database is a system which helps to organize, store and retrieve large amounts of data easily. It is a set of related records and a set of programs to access this data. It is an entire system that helps to enter, store and manage data, so it called Database Management System (DBMS). Filing cabinet is used to file the papers. New papers can be added to it and old unwanted papers can be removed. Changes can be made to the existing papers. If any particular paper is required, it can be retrieved. For easy and fast retrieval, index can be maintained. Sequence can also be given these papers for easy maintenance and retrieval. DBMS is exactly like this filing cabinet which is electronic.

Relational Databases or RDBMS is a type of database management system which stores data in the form of rows and columns. There are many database objects like stored procedures, index, triggers, sequence, view, etc. as shown in the above diagram.

- **Structured Query Language (SQL):** SQL is used to perform these operations such as addition, deletion, updating and retrieval of data from the database.
- **Parameterized Query:** An SQL can be treated like a function in a programming language and passed parameters to it. This special type of SQL queries are called parameterized query. They execute faster than SQL.
- **Stored Procedure:** Stored procedure by definition is a segment of code which contains declarative or procedural SQL statements.
- **Index :** Index is an additional data created in the database which allows faster access to the rows through use of pointers. Indexing of a table typically speeds up the data access from that table. It is a sorted list of rows accompanied by location of the row.
- **Trigger:** Trigger is a PLSQL program unit associated with a specific table which gets fired automatically whenever any user or application tries to modify the data from the concerned table in a predefined way. Trigger can be used to apply business rules or to log all the changes in a particular data table.
- **Sequence:** Sequence helps to give a unique identity to particular entity.
- **View:** View is a logical or virtual table, based on a table or another view. View does not have its own data but shows data from the base tables. Views can be used to present relevant data from the same table to different users in different format (view).



Focus on Knowledge about various database objects and their usage.

Structured Query Language (SQL)

- Language to query the database
- Types of SQL
 - Data Definition Language (DDL)
 - Creating tables and other database objects, etc.
 - Data Manipulation Language (DML)
 - Inserting, Updating, Deleting the records
 - Data Control Language (DCL)
 - Granting privileges, revoking them

4

SQL is used to perform these operations such as addition, deletion, updating and retrieval of data from the database.

SQL has three sub parts.

- **Data Definition Language (DDL)** is a set of SQL commands used to create or modify and delete database structures or objects such as tables, views etc. and not actual data.
Examples of DDL commands are CREATE, ALTER, DROP.
- **Data Manipulation Language (DML)** is a set of SQL commands which are used to query, insert, update, and delete data stored in the database.
Examples of DML commands included are SELECT, INSERT, UPDATE, DELETE.
- **Data Control Language (DCL)** is a set of commands for controlling access to the data.
Examples of DCL commands included are GRANT, REVOKE.



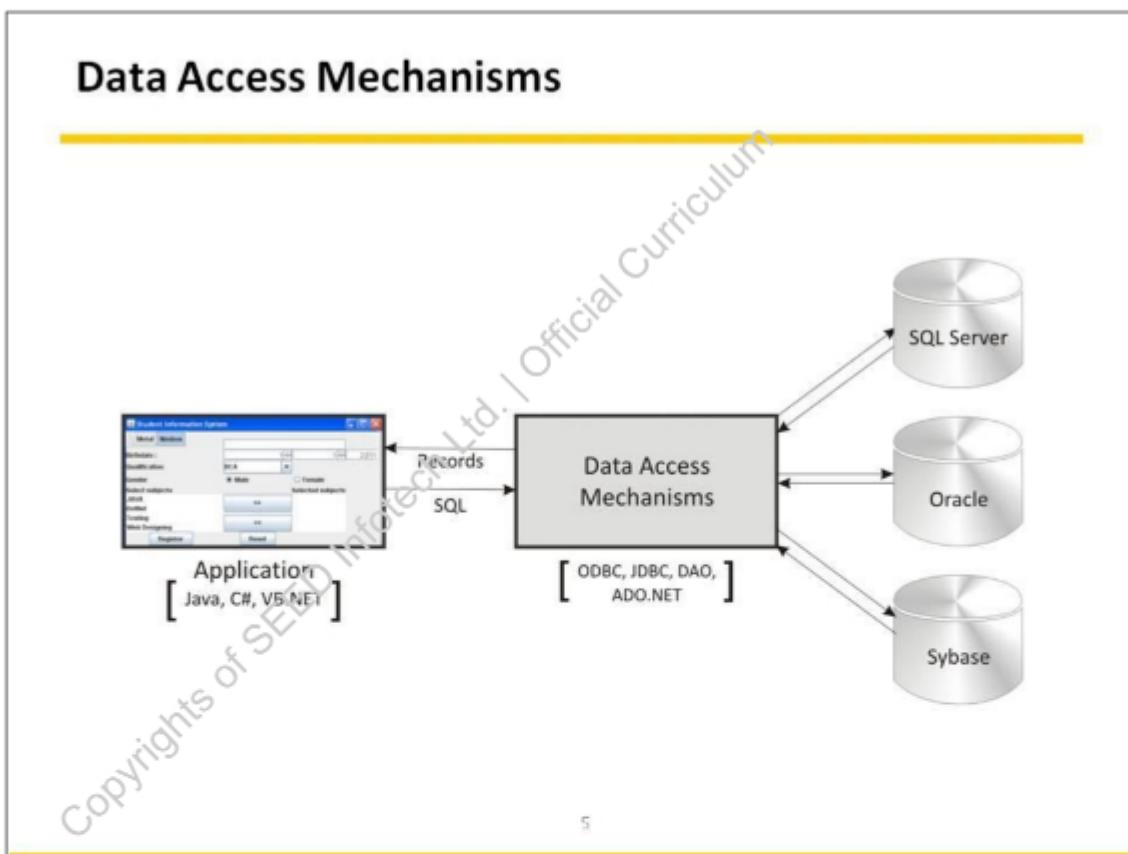
Group Exercise

Employee Table Structure

Name	Type
EMPLOYEE NO	NUMBER(4)
ENAME	VARCHAR2 (50)
SAL	NUMBER (5)

Write SQL Queries for following.

1. Select all employee names from employee table.
2. Delete employee with employee no = 30.
3. Add an employee with employee no 31, name joy and salary 20000.



SQL is used to access the database. Application can be written in any language, for example, Java, C#, VB.NET, C++, VB 6.0, etc. Consider an example of online reservation system, banking application, etc. These applications need to access the data or sometimes change the data in the database. Data access mechanisms are required to do this task. They can be ODBC (Open Database Connectivity), JDBC (Java Database Connectivity), DAO (Data Access Object), ActiveX Data Object (ADO) and ActiveX Data Object on .NET platform (ADO.NET).

These mechanisms provide support in the form of different objects that interact with lower level APIs (Application Programming Interfaces) to interact with the respective database. These lower level APIs are called drivers. They provide an interface to access the underlying database.

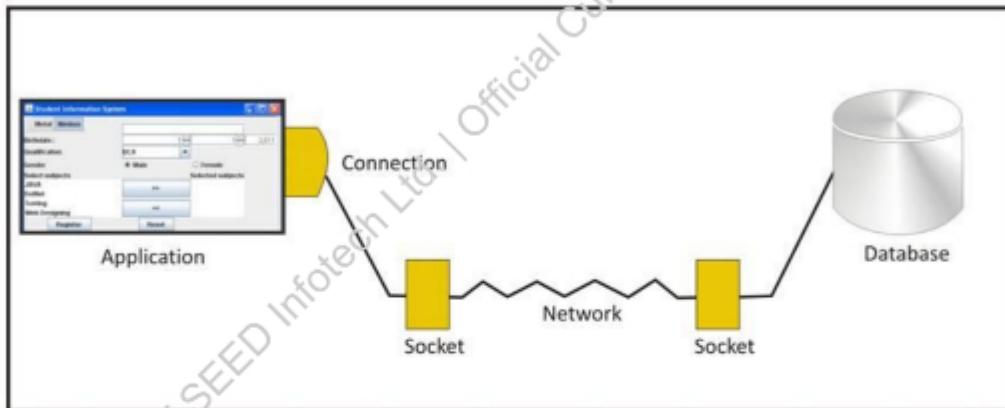
Accessing the Data: Steps

- 1 Create a SQL query.
- 2 Establish a connection.
- 3 Associate SQL with the connection.
- 4 Fire the query.
- 5 Get the result (set of rows)
- 6 Process the result.
- 7 Close the connection.

6

To access the data from database above are the typical steps. Let us understand each of these steps in more details. We already know how to create an SQL query.

Connection and JDBC Drivers



7

A Connection object represents a connection with a database. All operations executed on database are invoked through API of Connection object. It represents a session or a communication channel. Multiple connections can be opened with the same/different databases simultaneously. Connection object and its API completely hides complexity of socket communication between the Application program and the database. Connection class APIs are discussed in details in subsequent slides.

Types of JDBC Drivers

Driver is a program which allows the application to establish connection with the database by hiding complexity of handling network connection/sockets. Essentially Connection object uses the JDBC driver which is a Java class to make the connection and communicate with the database.

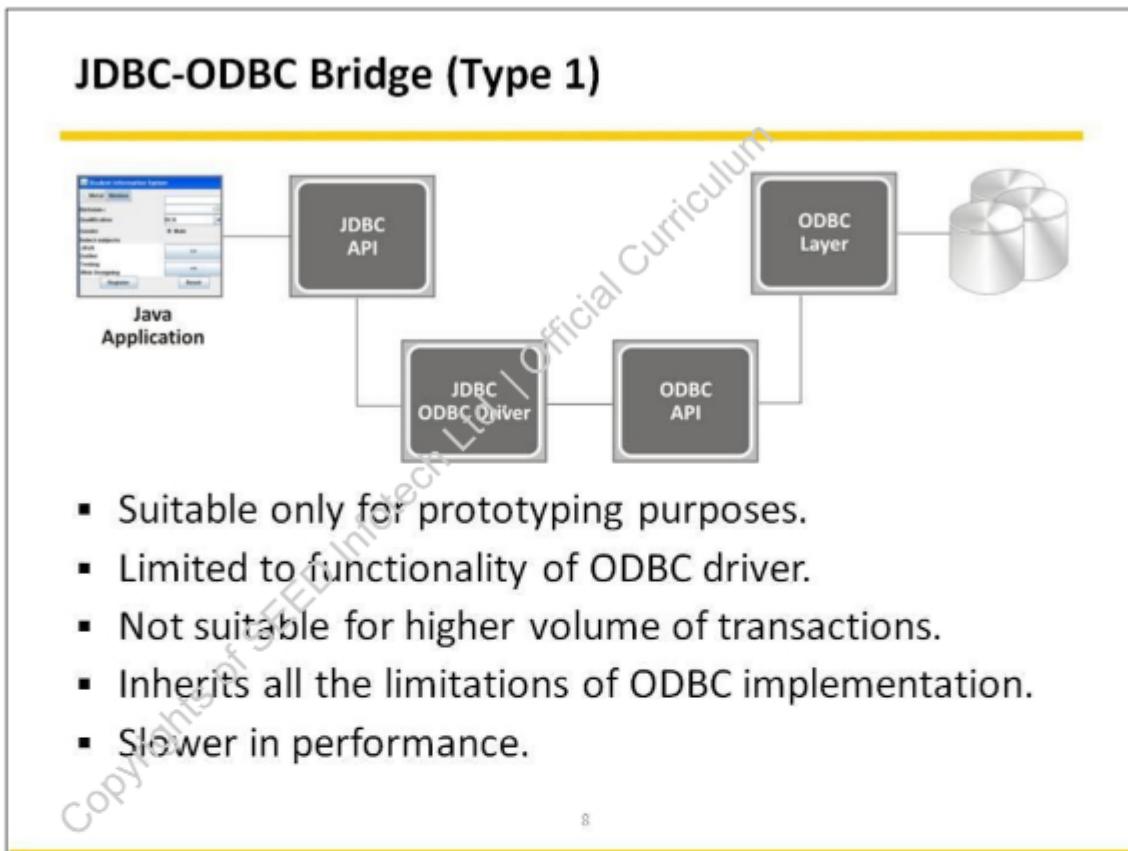
The JDBC drivers are of 4 types based on how they are created.

JDBC-ODBC Bridge	(Type 1)
Native-API partly Java Driver	(Type 2)
Net-Protocol All-Java Driver	(Type 3)
Native Protocol All-Java Driver	(Type 4)



Tech App

To get complete list of JDBC drivers use the following link
<http://industry.java.sun.com/products/jdbc/drivers>



Type 1 driver leverages ODBC driver implementation. It translates all JDBC calls to ODBC (open database connectivity) calls and sends them to the ODBC driver. It is a Java wrapper developed over ODBC API. It should be used only for prototyping and is not recommended for production environment.

Advantage

- Easily available as part of JDK.

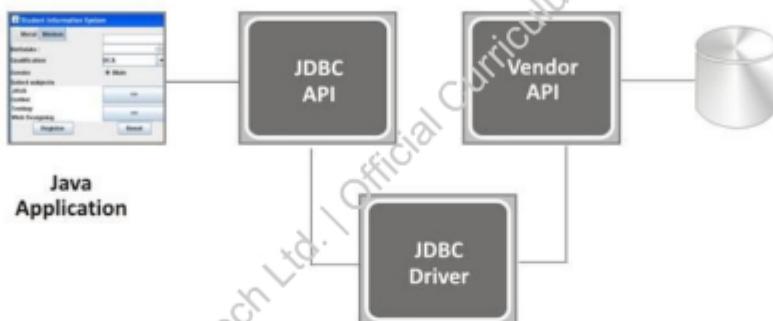
Disadvantages

- Limited to functionality of ODBC driver.
- Not suitable for higher volume of transactions.
- Slower in performance.

Example

- `sun.jdbc.odbc.JdbcOdbcDriver`

Native-API partly Java Driver (Type 2,



- Written partly in Java and partly in native code.
- Some platform-specific code in addition to Java library.
- Uses native 'C' language lib calls for Conversion.

9

Type 2 drivers use a mixture of Java implementation and vendor-specific native APIs to provide data access. Essentially a Java Native Interface (JNI) implementation is developed around the native driver code. This mandates a requirement of installation of native code API on every client that runs Java application. Native part of the code performs faster and efficient communication with the underlying database system.

Advantage

- Type 2 drivers typically offer better performance than the JDBC-ODBC Bridge as the layers of communication (tiers) are less than that of Type 1 and also it uses Native API which is Database specific.

Disadvantages

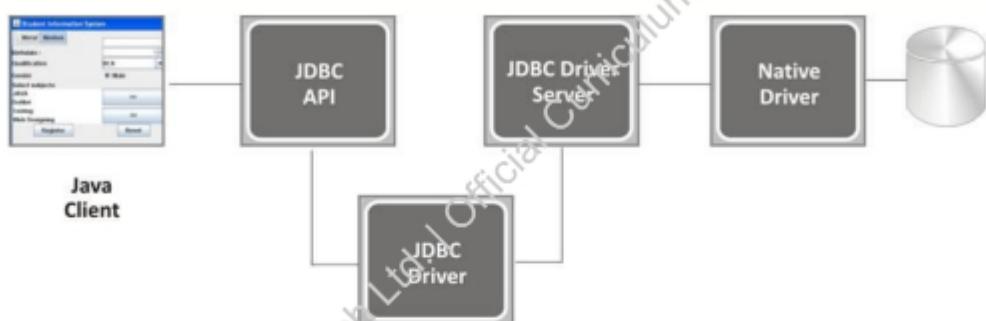
- Native API must be installed in the Client System and hence type 2 drivers cannot be used for the Internet.
- Like Type 1 drivers, it is not written in Java Language which forms a portability issue.

- If the database is changed the native API also has to be changed.
- Mostly obsolete now.
- Usually not thread safe.

Example

- `com.ibm.db2.hcc.DB2Driver`

Net-Protocol All-Java Driver (Type 3)



- Uses DB independent protocol to communicate DB-requests to a server component.
- Translates requests into a DB-specific protocol.
- Since client is independent of the actual DB, deployment is simpler and more flexible.

10

Advantages

- This driver is server-based. So, the vendor database is library not required for client machines. it is fully written in Java and hence portable. It is suitable for the web.
- There are many opportunities to optimize portability, performance, and scalability.
- The net protocol can be designed to make the client JDBC driver very small and fast to load.
- It 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.
- It is very flexible and allows access to multiple databases using one driver.
- It is the most efficient amongst all driver types.
- It essentially converts JDBC calls into vendor specific protocols thereby eliminating need of native code installation at each client.

Disadvantages

- It requires another server application to be installed and maintained. Traversing the record set may take longer, since the data comes through the backend server.
- In this case clients connect to database servers via an intermediate server component that acts as a gateway for multiple database servers.

Example

- `com.informix.jdbc.lfxDriver`

Native Protocol All-Java Driver (Type 4)



- JDBC calls are directly converted to network protocol used by the DBMS server.
- driver usually comes only from DB-vendor.

11

Type 4 drivers typically make direct socket connections to the databases. They convert the JDBC API calls to network calls using vendor-specific networking protocols. Type 4 drivers generally offer better performance than their Type 1 and Type 2 counterparts. These are preferred over other drivers, since there are no additional libraries or middleware to install. Almost all major database vendors provide Type 4 JDBC drivers.

Advantages

- Type 4 drivers are completely written in Java. So we can achieve platform independency. Number of translation layers is very less.
- Performance is good.
- Need not to install special software on the client or server. Further, these drivers can be downloaded dynamically.

Disadvantages

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

Example

- `oracle.jdbc.driver.OracleDriver`

JDBC URL: Locate the database

- Needed by drivers to locate, access and get other valid information about the databases
- Typical Syntax

`jdbc:[subprotocol]:[subname][attributes]`

- JDBC URL examples

`jdbc:odbc: dataSourceName`
`jdbc:db2:database_name`

A Java application may use multiple databases to store data. These databases can be accessed using different database drivers. The parameters required to obtain connection may be different. In this situation, how to identify a specific driver is a question. A JDBC URL is the answer. A database connection URL is a string that DBMS JDBC driver uses to connect to a database. It can contain information such as where to search for the database, the name of the database to connect to, and configuration properties. The exact syntax of a database connection URL is specified by your DBMS and JDBC driver documentation.

The standard syntax for JDBC URLs is shown below.

<code>jdbc:[subprotocol]:[subname][attributes]</code>

JDBC URL examples

ODBC Driver	<code>jdbc:odbc: dataSourceName</code> <code>jdbc:odbc:test</code>
DB2 Driver	<code>jdbc:db2:database_name</code> <code>jdbc:db2:MyDB</code>
Informix Driver	<code>jdbc:Informix-sqli://<host>:<port>/<database_name>:INFORMIXSERVER=<sid></code> <code>jdbc:Informix-sqli://localhost:1025/MyInformixDB:INFORMIXSERVER=MyDBServer</code>
Oracle thin Driver	<code>jdbc:oracle:thin:@host:port:service</code> <code>jdbc:oracle:thin:@localhost:1521:oracle10g</code>

Getting Connection

```
try {
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Properties p = new Properties();
    p.put("user","dba");
    p.put("password","sql");
    String url = "jdbc:odbc:mytable";
    Connection c =
    DriverManager.getConnection(url,p);
}
```

Class.forName takes a string class name and loads the necessary class dynamically at run time. When a driver is loaded, it registers itself with DriverManager using registerDriver() method. The getConnection() method of DriverManager class attempts to establish a connection to the given database URL. The DriverManager attempts to select an appropriate driver from the set of registered JDBC drivers.

Using Driver class

Code Example

```
String driverName="oracle.jdbc.driver.OracleDriver";
Driver driver= (Driver) Class.forName
(driverName).newInstance ();
Properties p=new Properties();
p.put ("user","java");
p.put ("password","java");
String
url="jdbc:oracle:thin:@192.168.1.100:1521:oracle9i";
```

```
con=driver.connect (url, p);
```

Getting connection using **System.properties**

Code Example

```
Properties props = new Properties ();
FileInputStream in = new FileInputStream
("database.properties");
props.load (in);
in.close();
String drivers = props.getProperty ("jdbc.drivers");
if (drivers!= null) System.setProperty("jdbc.drivers",
drivers);
String url = props.getProperty("jdbc.url");
String username = props.getProperty ("jdbc.username");
String password = props.getProperty ("jdbc.password");
```

Executing a Simple SQL Query

```
String url = "jdbc:odbc:MyDataSource"
Connection con = DriverManager.getConnection(
    url);
Statement stmt = con.createStatement();
String sql = "SELECT Last_Name FROM EMPLOYEES";
ResultSet rs = stmt.executeQuery(sql);
while(rs.next())
{
    System.out.println(rs.getString("Last_Name"))
}
}
```

Statement

The Statement object returns a java.sql.ResultSet object that encapsulates the results of query execution. This is an interface implemented by driver vendors. The result set can be scrolled using a cursor for reading the data values in the ResultSet.

Using Statement

1. Load the JDBC driver.
2. Get Connection.
3. Call CreateStatement.
4. Call executeQuery with SQL query as parameter.
5. Iterate over ResultSet to retrieve data.
6. Close the Statement object.
7. When you are finished using a Statement, call the method Statement.close to immediately release the resources it is using. When you call this method, its ResultSet objects are closed.

Closing the connection

Before JDK 7

```
try { // your database operation statements go  
    here  
}  
finally { if (con != null) con.close(); }
```

With JDK 7

```
try (Statement stmt = con.createStatement()) {  
} catch (SQLException e) {  
}
```

It is very important to close the connection once work is done. If connection object is kept open and its reference is lost, there will be a resource leak i.e. connection object continues to occupy memory but is not useable. If this continues for a longer period, database would run out of connections and application would crash. It is utmost important to manage resources like connection object.



Best Practice

Prior to JDK 7 and JDBC 4.1

When you work with the connection is finished, you need to make sure that Connection is closed properly. Following code snippet shows how to do it in a correct way. Call to `close()` method will release the connection with the database or will return the connection object to the pool is dependent on the implementation.

```
try { // your database operation
```

```
statements go here  
}  
finally { if (con != null) con.close(); }
```



In JDK 7 and JDBC 4.1

In JDBC 4.1, which is available in Java SE release 7 and later, you can use a try-with-resources statement to automatically close Connection, Statement, and ResultSet objects, regardless of whether an SQLException has been thrown. An automatic resource statement consists of a try statement and one or more declared resources.

```
try (Statement stmt =  
con.createStatement()) {  
} catch (SQLException e) {  
}
```

Executing DML Statement

```
String url = "jdbc:odbc:MyDataSource"
Connection con = DriverManager.getConnection(
    url);
Statement stmt = con.createStatement();
String sql="insert into employee(empno,
    empname,designation,salary)

    values(5,'rrr','manager',11000)";
int updateRowCount=stmt.executeUpdate(sql);
```

To execute a DML type of SQL a different method is provided called `executeUpdate()`. It executes the given SQL statement which may be INSERT, UPDATE or DELETE and returns the count of affected rows in the database.

Maintaining Data integrity: Commit

```

Begin Transaction
  unit operation 1
  unit operation 2
  unit operation n
if Successful then
  Commit
Else
  Rollback
  
```



```

Begin Transaction
  Cancel ticket Train1
  Book ticket Train2
if (successful) then
  Make changes permanent
Else
  do not cancel the
    ticket Train1
  
```

17

Transactions

The day to day (real-life) applications like electronic money transfer, reservation systems or accounting system involve a series of operations to complete a task. This series of operations needs to be treated as an atomic unit - 'a logical transaction' and we also need to have the facility for undoing changes under certain circumstances.

For example, in a railway reservation system request to book a ticket for a particular train against the cancellation of ticket for some other train. It is also specified that the existing ticket is to be cancelled only if we are able to book it for the other train.

1. This task involves two operations.
2. Cancellation of a ticket for one train and
3. Booking of a ticket for some other train.

These operations can be accomplished by executing a set of commands on the reservation database. If one operation in a set of operations fails it is not

acceptable here. Here the set of operations should either succeed or fail as a group and not as individual operations.

The transaction processing support provides a solution for such situations. A series of changes made using SQL commands like INSERT, UPDATE or DELETE can be grouped together using transactions.

In a transaction, when manipulating the data in the database, the changes made by us are not normally made permanent in the database until we give a command to do so.

In a transaction, the changes made are not normally made permanent in the database until we confirmed them. If changes are not confirmed, the database maintains its original state i.e. changes made during the database operations are undone.



Read more about ACID properties of transactions.

Additional Reading

Transactions in JDBC

The code block for transactions in JDBC is shown in the example below. By default the property called AutoCommit is true, hence every single DML statement is treated as a transaction and committed to the database. Hence when setAutoCommit(false) is called, it disables it and allow the transaction scope to contain more than one DML statements. Here this statement is equivalent to Begin Transaction. All the statements inside the try block are part of the transaction. If everything goes well, last statement to execute is Commit which indicates success of the transaction.

In case of any error, an exception would be thrown and catch block will be executed rolling back the entire transaction.

```

public void MoneyTransfer (Account from, Account to,
int amount) {
    try {
        con.setAutoCommit (false);
        from.balance=from.balance-amount; // withdraw from
one Account
        pst=con.prepareStatement ("update account set
balance=? Where
accno=?");
        pst.setInt (1, from.balance);
        pst.setInt (2, from.accountId);
        int i=pst.executeUpdate ();
        to.balance=to.balance+amount; // deposit in other
account
        pst=con.prepareStatement ("update account set
balance=? where
accno=?");
        pst.setInt (1, to.balance);
        pst.setInt (2, to.accountId);
        int ii=pst.executeUpdate ();
        con.commit (); // if both are successful keep the
new state

    } catch (Exception b) {
        try {
            con.rollback (); // if one of them fails undo
the other one
        } catch (SQLException e) {
            e.printStackTrace ();
        }
    }
}

```



Tech App

Transaction handling is an important part of database interactions while creating business applications. Understand the transaction mechanism in details.

Copyrights of SEED Infotech Ltd. | Official Curriculum

JDBC Connection API

- `close()`
- `commit()`
- `void setAutoCommit(boolean b)`
- `rollback()`
- `Statement createStatement()`
- `CallableStatement prepareCall(String sql)`
- `PreparedStatement
prepareStatement(String sql)`

Following are important APIs of Connection interface:

Method	Description
<code>public void close() throws SQLException</code>	Releases a Connection's database and JDBC resources immediately instead of waiting for them to be automatically released.
<code>public void commit() throws SQLException</code>	Makes all changes made since the previous commit/rollback permanent and releases any database locks currently held by the Connection. This method should be used only when auto-commit mode has been disabled.
<code>public void</code>	Sets this connection's auto-

<pre>setAutoCommit(boolean autoCommit) throws SQLException</pre>	<p>commit mode. If a connection is in auto-commit mode, then all its SQL statements will be executed and committed as individual transactions. Otherwise, its SQL statements are grouped into transactions that are terminated by a call to either the method commit or the method rollback</p>
<pre>public void rollback() throws SQLException</pre>	<p>Drops all changes made since the previous commit/rollback and releases any database locks currently held by this Connection. This method should be used only when auto-commit has been disabled.</p>
<pre>public Statement createStatement() throws SQLException</pre>	<p>Creates a Statement object for sending SQL statements to the database. SQL statements without parameters are normally executed using Statement objects. If the same SQL statement is executed many times, it is more efficient to use a PreparedStatement object.</p>
<pre>public CallableStatement prepareCall(String sql) throws SQLException</pre>	<p>Creates a CallableStatement object for calling database stored procedures. The CallableStatement object provides methods for setting up its IN and OUT parameters, and methods for executing the call to</p>

	a stored procedure
public PreparedStatement prepareStatement(String sql) throws SQLException	Creates a PreparedStatement object for sending parameterized SQL statements to the database. A SQL statement with or without IN parameters can be pre-compiled and stored in a PreparedStatement object. This object can then be used to efficiently execute this statement multiple times.

Copyrights of SEED Infotech Ltd. | Official Curriculum

Parameterized SQL

```
String SQL =  
    "select * from Employees where First_Name=?";  
  
PreparedStatement pstat=  
    con.prepareStatement(sql);  
  
pstat.setString(1, "John");  
  
ResultSet rs = pstat.executeQuery();  
  
pstat.clearParameters();
```

Parameterized SQL is an SQL in which can pass the parameters at runtime while you are executing it. Every database has a different way of implementation parameterized query. Since they are cached by the database, they improve the performance as compared to a non-parameterized query.

When database engine receives an SQL query it parses the query to find out any syntax errors, then execution plan is prepared to access the data in most efficient way. These steps are executed each time an SQL query is received. Databases usually have a statement cache which stores the execution plan. This arrangement facilitates reuse of execution plans for the statements which have been executed previously, merely by substituting different parameters for each successive execution.

PreparedStatement

PreparedStatement is Java way of performing parameterized query. The PreparedStatement object contains an SQL statement that has already been compiled. This means execution can be faster than that of Statement objects. It

may have one or more IN parameters whose value is not specified when the SQL statement is created. The statement has a question mark ("?") as a placeholder for each IN parameter. A value for each question mark must be supplied by the appropriate setXXX() method before the statement is executed, where XXX stands for data type. Being a subclass of Statement, PreparedStatement inherits all the functionality of Statement.



Use PreparedStatement for better performance instead of Statement object in Java applications.

Using PreparedStatement

1. Call `prepareStatement()` to create the object.
2. Set values for all the input parameters using `setXXX()` method. The first argument to the `setXXX()` methods is the ordinal position (index number) of the parameter to be set, and the second argument is the value to which the parameter is to be set. Call appropriate methods from `executeQuery`, `executeUpdate`.
3. Use `clearParameters` before using the same `PreparedStatement` object for successive call with different parameters.
4. In addition, it adds a whole set of methods which are needed for setting the values to be sent to the database in place of the placeholders for IN parameters. Also, the three methods `execute`, `executeQuery`, and `executeUpdate` are modified so that they take no argument. The Statement forms of these methods (the forms that take an SQL statement parameter) should never be used with a `PreparedStatement` object.
5. Before a `PreparedStatement` object is executed, the value of each '?' parameter must be set. This is done by calling a `setXXX` method, where XXX is the appropriate type for the parameter. For example, if the parameter has a Java type of `long`, the method to use is `setLong()`.

Stored Procedure

```
CREATE DEFINER='root'@'localhost' PROCEDURE
`sp_product`(IN pid INT,OUT price INT)
BEGIN
    SELECT price from product where product_id=pid;
    SET price=price*2;
    Update product set price=price where
product_id=pid;
END
```

By definition stored procedure is a segment of code which contains declarative or procedural SQL statements. It overcomes limitations of SQL. SQL has following limitations:

- SQL is a non-procedural language.
- SQL does not allow usage of variables and constants.
- Developer cannot control the flow of execution using SQL.
- SQL does not have ability to process data row by row in a loop.

Most of the RDBMS systems have their own programming language to create such procedures. Oracle's PL/SQL and SQL Server's Transact-SQL are examples of the same. These objects being tightly coupled with the underlying database are not easily portable across databases. Nonetheless they are faster as compared to SQL or parameterized SQL.

Above examples shows a simple stored procedure which accepts product ID and doubles its price. This stored procedure is already created. Let us see how to invoke it from A Java application.

Note: How to create a stored procedure for a particular database is out of scope of this course.

Invoking Stored Procedure

```
CallableStatement cstmt =  
con.prepareCall( "{ call sp_product(?) } ");  
cstmt.setInt(1,12);  
cstmt.registerOutParameter(2,Types.FLOAT);  
  
cstmt.execute();  
  
System.out.println(cstmt.getFloat(2));
```

Use `prepareCall()` method to create `CallableStatement` object. JDBC provides callable statement interface as a mechanism to call stored procedures. `prepareCall()` method of `Connection` object is used to create a `CallableStatement`. JDBC defines standard escape syntax to access stored procedures.

```
{call procedure_name[ (?, ?, ...) ]}
```

In the above syntax, each question mark (?) represents a placeholder for a procedure or a return value. Note that the parameters are optional. It is responsibility of JDBC driver to convert this escape syntax into the database's own stored procedure syntax.

1. Set input Parameters

Substitute input parameters by calling `setXXX()` methods. These are numbered from 1 to n, left to right.

2. Register OUT and IN/OUT parameters

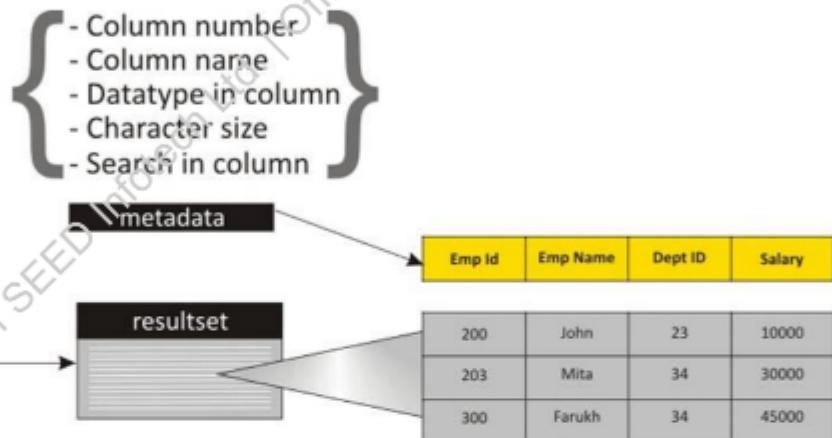
Register all the OUT and IN/OUT parameters using registerOutParameter() method. This method takes type of output as second argument.

3. Call execute() to Invoke the stored procedure.

Extract the OUT parameter values using the getXXX() methods of CallableStatement.

ResultSet MetaData

- ResultSet MetaData describes the structure of a `resultset` object

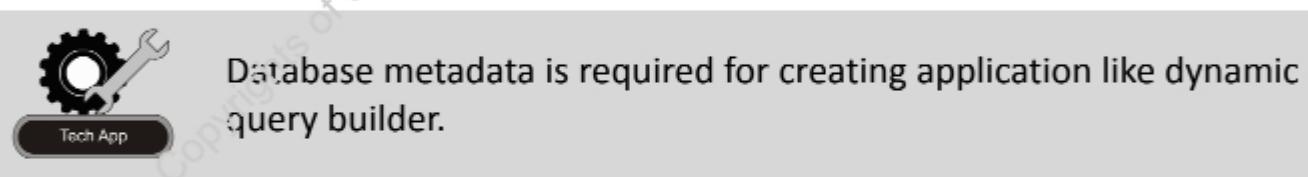


22

To retrieve the records or fields from a `resultSet` object, one needs to know the structure, sequence, column names, data type of columns etc. In most of the situations this information is known to the developer. Sometimes, it is necessary to get this information dynamically at runtime. `ResultSetMetaData` is used for this purpose. `ResultSetMetaData` interface provides information about the structure of a particular `ResultSet`. It can be obtained by calling `getResultSetMetadata()` method on any result set object. `ResultSetMetaData` makes it possible for the user to display or manipulate the data in the `resultset` without knowing its structure in advance. For example, a generic method can be developed which would accept any result set object as a parameter and display the contents of `ResultSet` in a Grid/Table fashion. The beauty of this mechanism is that there is no hard-coding of column names or their types.

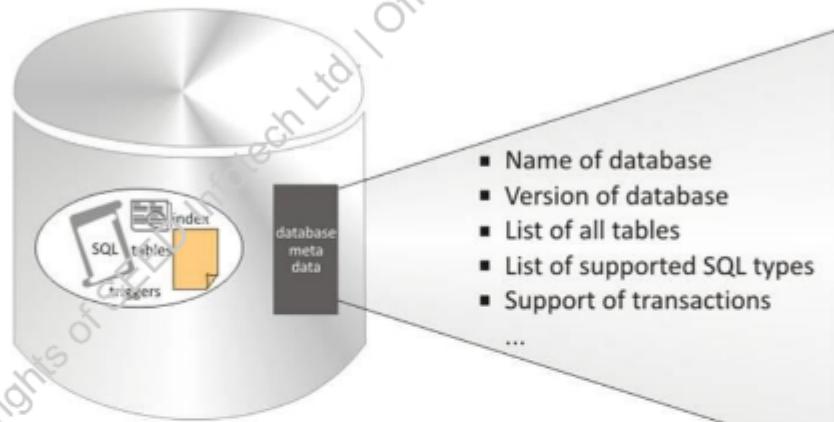
Following table shows key methods of `ResultSetMetaData` interface:

Method	Description
<code>int getColumnCount()</code>	Returns the number of columns in this <code>ResultSet</code> object.
<code>String getColumnLabel()</code>	Gets the designated column's suggested title for use in printouts and displays.
<code>String getColumnTypeName()</code>	Note that type names returned by <code>getColumnTypeName()</code> are database-specific (e.g., Oracle refers to a string value as a <code>VARCHAR</code> ; Microsoft Access calls it <code>TEXT</code>).



Database MetaData

- Database Metadata provides information about the database.



23

Business applications are developed with prior knowledge of the database and the table structure. Sometimes, it is necessary to know information about the table structures, features supported by database and so on dynamically at runtime. This is like SYS Table space provided by Oracle. This table space stores information about the data hence called meta-data. JDBC provides DatabaseMetaData interface to dynamically discover information about database configuration, system table structure, its capabilities, and so on. Using this interface one can create an application to read and manipulate data or other objects like indexes, and so on. independent of a particular database. This interface contains large number of methods.

Some of the important ones are listed below:

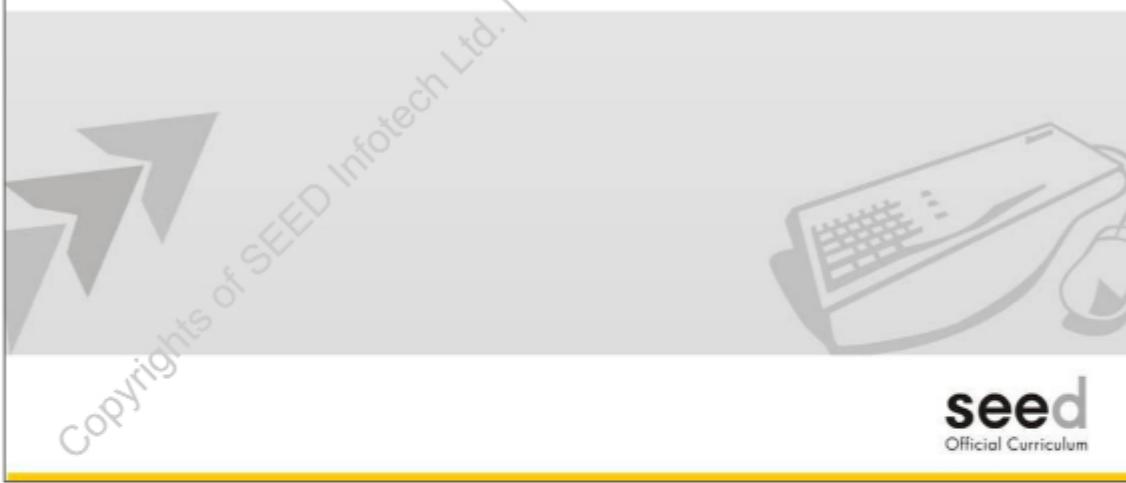
Method	Description
String getDatabaseProductName ()	Retrieves the name of this database product.
ResultSet getTables ()	Retrieves a description of the tables available in the given catalog
ResultSet getIndexInfo()	Retrieves a description of the given table's indices and statistics.
int getMaxConnections ()	Retrieves the maximum number of concurrent connections to this database that are possible.
ResultSet getTablePrivileges ()	Retrieves a description of the access rights for each table available in a catalog.



Database metadata can be used to create application which is similar to Microsoft Access or TOAD.

Chapter - 2

Advanced JDBC



seed
Official Curriculum

Typical database manipulations can be handled by simple JDBC mechanisms. This chapter covers the advanced features such as navigable and updatable ResultSets, Connection Pooling, DataSources, SQL exceptions and data manipulation using disconnected architecture. These features allow using the full power of database connectivity.

Objectives

At the end of this chapter you will be able to:

- List the types of ResultSet
- Construct an application with scrollable and updatable ResultSet.
- Construct an application for multiple queries with Batch Update.
- List SQL Exception categories.
- State the limitations of connected architecture.
- Demonstrate disconnected architecture with JDBC.

JDBC - Advanced Features

- JDBC 1.x provided core features necessary for working with databases
- To improve performance, usability and robustness few more components were added in latter versions of JDBC
 - Scrollable, Updatable ResultSets
 - Enhanced SQL Exceptions
 - Batch Updates
 - DataSource and ConnectionPool
 - Disconnected database connectivity

JDBC 2.x introduced optional packages to JDBC for providing enhanced performance and robustness. The basic API for executing the standard SQL queries was already available through the JDBC 1.x packages.

Performance Enhancement

- Batch Updates
- Data Source and Connection Pool

Increased Usability

- Scrollable and Updatable ResultSets
- Disconnected database connectivity

Robustness

- Enhanced SQL Exceptions



These features are available through the `javax.sql` package and not the standard `java.sql` package. The extra 'x' stands for eXtended.

ResultSet Navigation

- By default, ResultSet can be iterated in a forward direction only.
 - Backward movement and random access is not possible.
- Limitations
 - Time consuming task to retrieve a single record situated at the end of the ResultSet.
 - Once data is retrieved, it cannot be revisited again.
- Solution
 - Navigation becomes possible by providing the ResultSet with scrolling mechanism.

```
Statement stmt = con.createStatement(ResultSet.<type>,
                                     ResultSet.<update>)
```

ResultSet object obtained through the simple Statement object cannot be iterated upon in a random order. There are two parameters associated with the Statement object, namely Type and Concurrency, which control the scrolling and updation feature of the ResultSet object.

By default, a standard Statement object provides a ResultSet with only forward movement. If the current record pointer is positioned at row number 10, there is no way it can move to row number 9 or before.

When the requirement is to display the data retrieved by the query on the User Interface (UI), using ResultSet is the standard procedure. When the data has to be filtered according to the requirements, accessing a record at the end becomes a problem since the only way to reach the last record is to iterate through the individual records one-by-one. This process will have a major hit on the performance of the application.

To overcome this problem, the ResultSet object will have to be created with a scrollable type. The Scrollable ResultSet can be obtained as follows:

```
// Database Connectivity code  
Statement  
stmt=con.createStatement(ResultSet.TYPE_SCROLL_<scroll  
type>,ResultSet.<update>);  
ResultSet rs=stmt.executeQuery("Select empId, empName,  
empAddr, Age, Salary from empTable");  
//Code for further processing goes here
```



Interview Tip

By default, ResultSet is TYPE_FORWARD_ONLY and READ_ONLY.

For freshly created ResultSet object the current row pointer will be at 0. rs.next() is mandatory before accessing any data.

Scrollable ResultSet

- Making a statement “Scrollable” ensures all ResultSet objects created to be scrollable.
- Two types of Scrollable ResultSets available
 1. Scroll Sensitive
(ResultSet.TYPE_SCROLL_SENSITIVE)
 - Scrolling and database change sensitive objects.
 2. Scroll Insensitive
(ResultSet.TYPE_SCROLL_INSENSITIVE)
 - Scrolling but database change insensitive objects.

A Scrollable ResultSet ensures that the navigation within the ResultSet becomes dynamic, as both forward and backward movement becomes possible. This also facilitates jumping directly to a specific record within the ResultSet, as well as making a jump relative to the current position.

```
//scrollable and updatable resultset creation
rs.next();
if(rs.isAfterLast())
{
    System.out.println("You have reached end of
records)
    rs.last();
}
// to move to 3rd row and then 2 rows behind it.
rs.absolute(3);
// perform some operations
rs.relative(-2);
```

A negative value for relative positioning moves the pointer backwards. A scrollable ResultSet also has functionalities to check whether the iteration has crossed its boundaries and the program can then correct itself accordingly, and avoid throwing an exception.

Scroll Sensitive ResultSet

A Scroll Sensitive ResultSet is called so, because it is sensitive to changes being made to the underlying database. If more than one connection is manipulating the database and if the connection is of Scroll Sensitive type, changes made by another connection are generally visible to the client. Visibility of the changes is dependent upon the capabilities of the driver being used.

```
Statement  
stmt=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,  
    ResultSet.<update>);  
ResultSet rs=stmt.executeQuery("Select empid from  
emp");
```

Scroll Insensitive ResultSet

A Scroll Insensitive ResultSet has scrolling capabilities but it does not reflect the changes being made by other concurrent connection to the database.

```
Statement  
stmt=con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,  
    ResultSet.<update>);  
ResultSet rs=stmt.executeQuery("Select empid from  
emp");
```



Select * from <tablename> cannot be used with scrollable resultsets, but there is a workaround for this. Use individual column names to select the data or use aliases with the table names.

Scrollable ResultSets cannot handle stream data, hence scrollable resultsets should not be used when working with Blob/Clobs



Best Practice

Use Scroll sensitive ResultSet only if the situation demands it. The ResultSet needs to be kept fresh at all times, which means that any operation on the resultset initiates a call to the database to ensure the freshness of the data. This leads to unnecessary overheads.



Tech App

Scroll Sensitive ResultSets are a best fit for applications which use real-time data where updated data is of utmost concern and takes precedence over performance or speed. For example, reservation systems or applications monitoring stocks need instant update of the data, otherwise the application loses its meaning.

ResultSet Updating

- By default, ResultSet object is a read only cursor.
- It can be made updatable
 - Changes made against the ResultSet can be persisted in the database without firing a query.

```
Statement stmt = con.createStatement(ResultSet.<type>,
                                    ResultSet.<update>)
```

- Types of update
 - ResultSet.TYPE_READONLY
 - ResultSet.TYPE_UPDATABLE

By default, ResultSet can be used only for retrieval of the data (read-only) and not for dynamic updation as and when required. As discussed earlier, default ResultSet object obtained as follows:

```
//Database connectivity code
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("Select * from
empTable");
//Further processing
```

It creates a forward-only and read-only kind of a ResultSet, which is a static snapshot of a database at a given point of time. To change the data obtained using the ResultSet, separate queries will have to be written and executed through the code. This results in additional SQL programming burden and network utilization. Instead, Java provides a work-around for this.

An Updatable ResultSet obtained as follows:

```
// Database Connectivity code  
Statement stmt=con.createStatement(ResultSet.<scroll  
type>,ResultSet.UPDATABLE);  
ResultSet rs=stmt.executeQuery("Select * from  
empTable");  
//Further processing
```

It can be utilized to perform data manipulation operations without having to resort to writing SQL queries or having to initiate a network call.

```
//scrollable and updatable resultset creation  
//To update the Slogan field of sixth row  
rs.absolute(6);  
rs.updateString("Slogan","Beyond the Obvious");  
rs.updateRow();
```

```
//To add a new row  
rs.moveToInsertRow();  
rs.updateString("Name","Sony Corporation");  
rs.updateString(2,"make.believe");  
rs.insertRow();  
  
//to delete 10th record  
rs.absolute(10);  
rs.deleteRow();
```

The system takes care of porting the data behind the scene and the developer does not have to worry about writing or learning SQL statements. Data updation from the client side to the database server happens in a background process generally at the end of the transaction.

Dynamic Database Manipulation

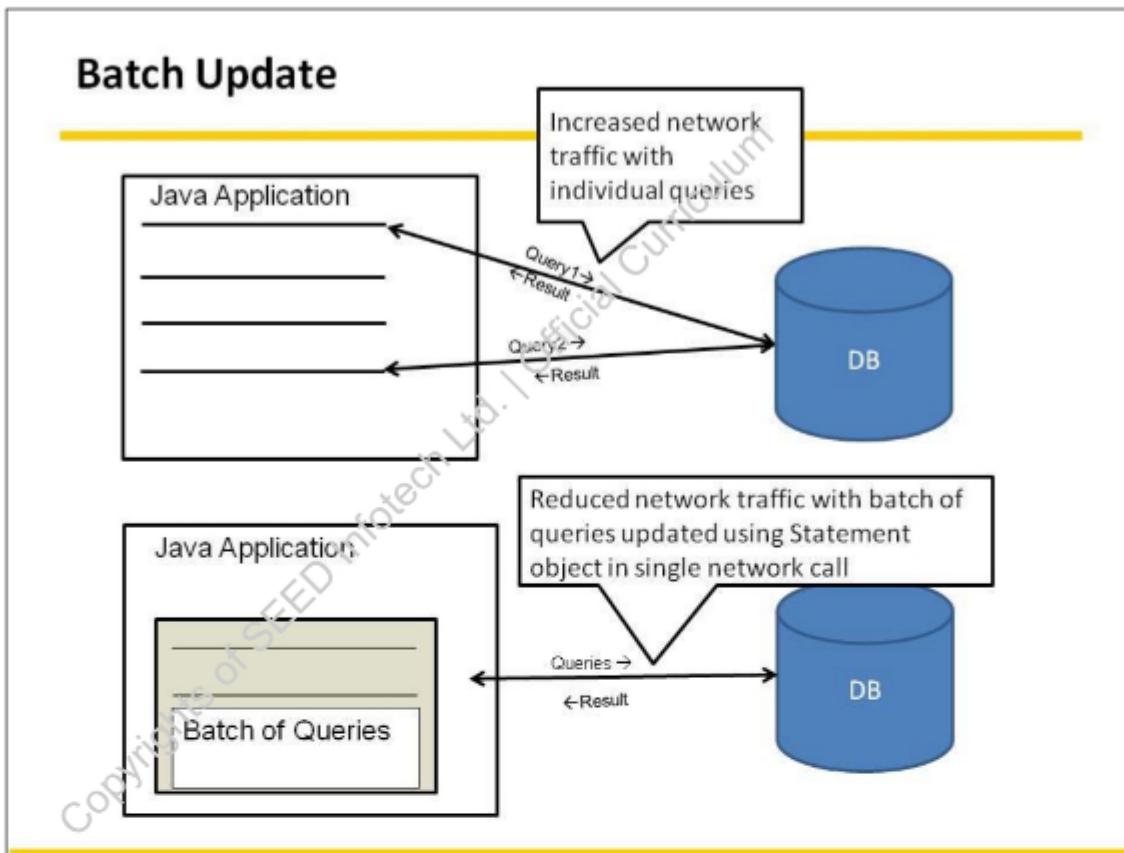
- Using a scrollable and updatable ResultSet the following operations can be executed easily
 - Move to First, Last record
 - Move to 'n'th record (Absolute number, relative)
 - Check if boundaries crossed
 - Add a record
 - Update a record
 - Remove a record

A scrollable and updatable ResultSet can be used to navigate and manipulate the database using the ResultSet object itself.

Following are some of the important methods for dynamically manipulating a database:

Method	Description
boolean absolute(int row)	Moves the cursor to the given row number in this ResultSet object.
void afterLast()	Moves the cursor to the end of this ResultSet object, just after the last row.
void beforeFirst()	Moves the cursor to the front of this ResultSet object, just before the first row.
void deleteRow()	Deletes the current row from this ResultSet object and from the underlying database.

<code>void insertRow()</code>	Inserts the contents of the insert row into this ResultSet object and into the database.
<code>boolean isAfterLast()</code>	Retrieves whether the cursor is after the last row in this ResultSet object.
<code>boolean isBeforeFirst()</code>	Retrieves whether the cursor is before the first row in this ResultSet object.
<code>boolean isFirst()</code>	Retrieves whether the cursor is on the first row of this ResultSet object.
<code>boolean isLast()</code>	Retrieves whether the cursor is on the last row of this ResultSet object.
<code>void moveToCurrentRow()</code>	Moves the cursor to the last used position, usually the current row.
<code>void moveToInsertRow()</code>	Moves the cursor to the insert row.
<code>boolean relative(int rows)</code>	Moves the cursor a relative number of rows, either positive or negative.



In a typical enterprise application scenario, a specific task may be executing multiple subtasks. Implementing such as task involves firing multiple statements corresponding to each of the tasks. Each statement in JDBC is an atomic execution unit. For every statement that is executed, a server-trip is mandatory; therefore firing multiple statements in succession leads to network bandwidth problems such as application performance and network utilisation.

To reduce the network bandwidth utilisation, there needs to be an alternate way of executing a statement that allows for combining multiple SQL statements into a single execution unit. JDBC provides a way of doing the same.

All the update statements required for completing a specific task are combined together into a batch of statements and then the statement is executed against the database.

```
// turn off autocommit
con.setAutoCommit(false);
```

```
Statement stmt = con.createStatement();
stmt.addBatch("INSERT INTO emp VALUES (1000, 'Joe
Jones')");
stmt.addBatch("INSERT INTO departments VALUES (260,
'Shoe')");
stmt.addBatch("INSERT INTO emp_dept VALUES (1000,
260)");

// submit a batch of update commands for execution

int[] updateCounts = stmt.executeBatch();
```

If the entire batch does not execute successfully, then a `BatchUpdateException` is thrown. It can be handled in the corresponding catch block. For example, if a statement deep within the batch fails, whether the entire batch should be failed or the statements prior to the failed query should be maintained, can be decided in the catch block associated with the `BatchUpdateException`.

Successful execution of the batch returns an array of integers having a size matching with number of queries added into the batch. Each subscript holds the value of `updateRowCount` (number of rows affected) corresponding to each successful query execution. In case of a failure, the size of the array indicates and pinpoints the exact query that failed.

The batch update facility can be applied with `Statement`, `PreparedStatement` as well as `CallableStatement`.



App Design

Select query does not work with batch updates, as the return value is always the number of rows affected in the table.



Best Practice

It is recommended that before running the batch update, the auto-commit feature of JDBC should be turned off. This will ensure a chance to either accept or reject partial transaction execution.

Why Connection Pooling?

- Java application needs to create a Connection object to connect to a database.
- Problems
 - Process is resource-heavy.
 - Only one user can use the Connection object at a time.
 - Results in a blockage, if multiple users wish to access the same connection object to attach to the database.
- Solution
 - Multiple connection objects with same characteristics can be created and held in a pool for use as-and-when required.

Database connectivity is generally obtained using a connection object created either at runtime within the module or in a special module providing a connection object. In a standalone application, where there is only one user and one data storage available this approach works fine; but in enterprise applications, where concurrency is of paramount importance to any operation, it becomes a problematic scenario. This is because more than one client needs to connect with the same source of data and reuse the same piece of code that generates connection object.

Connection objects are always dedicated to a client as it holds client-specific information and data. This presents a problem for concurrency because once the connection object is consumed by a specific user it is dedicated to that user and other clients have to wait for it to become free.

When the user releases the connection there is one more potential problem. The connection has to be cleaned of the user's configuration so as to allow another user to have a fresh connectivity. This is a very complicated and time consuming process.

Probably providing individual connection object to individual user could solve this problem but creating an object at runtime while the applications are running and destroying after scope is a resource heavy process, which should be avoided so far as possible.

Connection Pool

- Collection of database connections maintained and managed externally.
- Intermediary between the client's Connection object and the physical database connection.
- Client code accesses the connection pool using a component named `DataSource` to enhance flexibility.
- When connection object is closed, the physical connection to the connection pool is lost.
 - The connection object when released, goes back in the pool ready to serve another request.

As the requirements of all the users who wish to connect to the database is similar, multiple connection objects having similar configurations can be created and held at a specific location. Any user, who wishes to connect to the database, can find the collection of the connections and request for an object to be made available. As each user can have access to this shared space, the problem of having to wait for a connection is solved. The collection of objects can also be created before the application starts, when the processing requirements are relatively sparse.

This shared space where the collection of connection objects are kept is known as connection pool. It acts as a go-between for the client code's local connection object and the actual physical connection linking it to the database. When the connection is closed from the client's code, the Connection object on local heap is destroyed but the conduit to the database simply breaks the connection between the client and the database. It then reverts back to another connection object ready to serve another request within the pool.

The connection pool is almost always external to the code which is using it and hence it needs to be handled and managed separately, and because of this every client who wishes to use the connection pool has to search for the connection pool in the shared workspace.

This is normally achieved using a component called as **DataSource**.

Copyrights of SEED Infotech Ltd. | Official Curriculum

DataSource

- Provides a way to obtain the connection object, either from the pool or an isolated object, depending on the configuration.
- Created and registered using JNDI (Java Naming and Directory Interface) service
- Configured either programmatically (JDBC 2.0 optional packages) or declaratively (Application Server).
- Retrieved through JNDI lookup operation.

```
DataSource ds = (DataSource)
                org.apache.derby.jdbc.ClientDataSource();
// get the connection from the pool
Connection con = ds.getConnection();
```

DataSource is a standard way of connecting with a database using connection pool as a medium. DataSources come in two flavours - they can either be created programmatically or they can be configured on the server. Enterprise applications generally use the latter approach, while a standard client-server application may use the former one.

Programmatic creation involves having a vendor implementation of the DataSource object available.

```
//Initialize the DataSource with derby's client
DataSource
DataSource ds = (DataSource)
org.apache.derby.jdbc.ClientDataSource();

//populate the DataSource
ds.setPort(1527);
ds.setHost("localhost");
ds.setUser("APP")
```

```

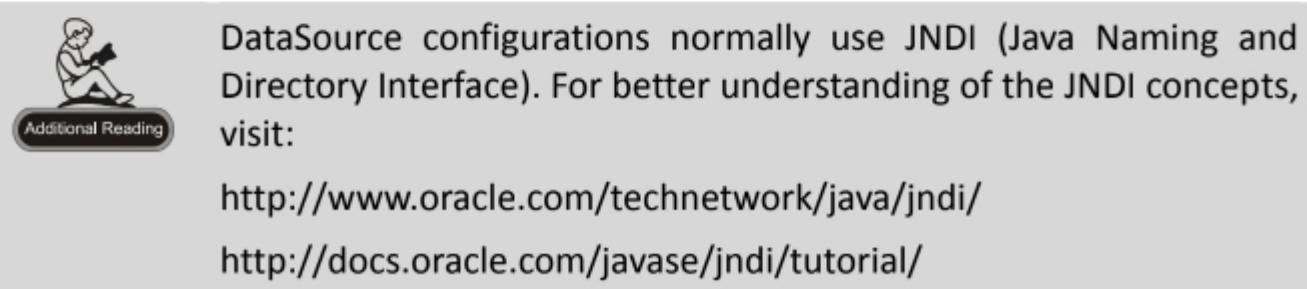
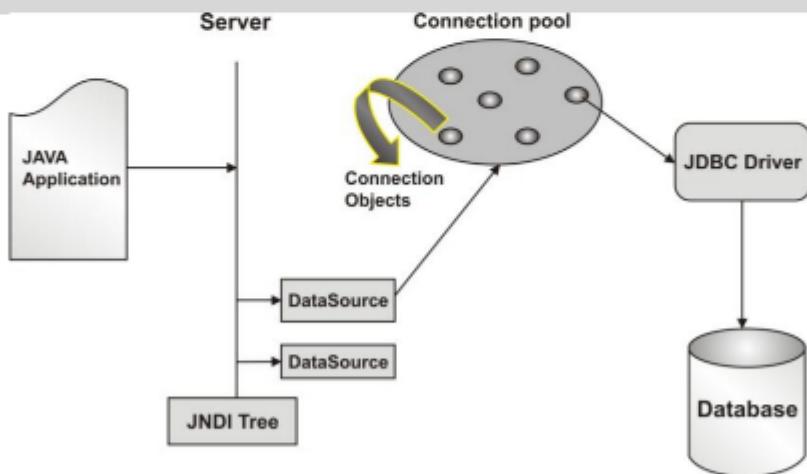
ds.setPassword("APP");

//Obtain the connection
Connection con = ds.getConnection();
//perform database operations

```

Declarative DataSource creation is a little complex and is handled in detail in later chapters. In abstract, it involves creating and binding a DataSource object using Java Naming and Directory Interface (JNDI). The client searches the JNDI tree using a virtual name associated with the data source. Internally this DataSource is wired to the connection pool which connects to the database.

All these layers of indirection ensure that the final application becomes extremely flexible and independent of database vendors. Changes at the database levels are abstracted out at the DataSource level and the client code never has to bother about what DB it is interacting with.



Iterable SQLException

- A chain of exception that provides information on a database access or other errors.
- Contains a string describing the error. This is used as the Java Exception message, available via the method `SQLException.getMessage()`.
- If more than one error occurs the exceptions are stacked into and referenced via this chain.
- If no more exceptions are chained, the `getNextException` method returns null.

Whenever an error occurs in the database server, a `SQLException` is raised. Error handling for JDBC works a little different than other APIs of Java. Frequently the cause and location of error is at the Database Server side, where the client code has no control, yet it is necessary to pin point the exact cause of error. In JDK 1.5 the first step towards this were taken by introducing a `ChainedException`, which could chain down into the root cause at the database level, but to achieve that, a developer had to write code as follows:

```
try{
    // Some Database Access Code
} catch(SQLException exception) {
    System.out.println(exception.getMessage());
    SQLException nextException =
        exception.getNextException();
}
```

```
while (nextException != null) {  
    System.out.println(nextException.getMessage());  
    nextException =  
nextException.getNextException();  
}  
}
```



With JDK 1.7, the same process has been simplified by making the SQLException class implement Iterable<Throwable> interface.

The same can now be done using the following:

```
try  
{  
// Some Database Access Code  
  
}  
catch(SQLException exception)  
{  
  
    for(Throwable throwable: exception)  
    {  
        System.out.println(throwable.getMessage());  
    }  
}
```

Following methods of the `SQLException` class are used to retrieve the text of the error message, the error code, and the `SQLSTATE` value:

Method	Description
<code>String getMessage()</code>	Returns a description of the error. <code>SQLException</code> inherits this method from the <code>java.util.Throwable</code> class.
<code>int getErrorCode()</code>	Returns an integer value that corresponds to the database server or error code.
<code>String getSQLState()</code>	Returns a string that describes the <code>SQLSTATE</code> value. The string follows the X/Open <code>SQLSTATE</code> conventions.

Enhanced SQL Exception Handling

- A lot of enhancement has been done regarding SQL Exception Handling :
 - Iterable SQL Exception
 - Concrete sub-classes for specific SQL Exception

Iterable SQL Exception makes development effort faster and easier. Chain of exception can be traversed faster and with lesser amount of coding.

Concrete Sub Classes help divide the `SQLException` hierarchy into three broad categories to help deal with them in a better way. They are:

1. Transient
2. Recoverable
3. Non-transient

Enhanced SQL Exception Handling

- The Program can recover from the exception without making any application specific change, than it is kind of **TransientException**.
- The Program can recover from exception when applying some Application Specific Recovery on this Transient Exception, than it is the **RecoverableException**.
- The exception that can never be recovered even though infinite number of tries are performed by an application programmer, than it is the **Non-Transient Exception**.



JDK7

JDBC4 introduces enhanced exception handling. They are of following types:

1. Transient Exceptions
2. Recoverable exceptions
3. Non-transient exceptions

Transient Exceptions

This category of SQLExceptions is normally passing in nature, meaning that a developer is not be able to do anything about this problem by changing the application code. For example, if connectivity with the database server is an issue, the application cannot really do anything about it. If at all possible, the administrator can increase the timeout interval and the same code which failed earlier will now execute. These are grouped under the `java.sql.TransientException`. For example, `SQLTimeoutException`, `SQLTransactionRollbackException`, and `SQLTransientConnectionException`.

Recoverable Exceptions

These are exceptions, which can be overcome or handled by making a provision for it in the application code itself. Normally they are handled in the catch block of the exception and initiate the entire transaction process again. Most of user-defined exceptions fall in this category. For example, InsufficientFundsException, InvalidChequeNumberException, etc.

Non-Transient Exception

There are situations where the problem cannot be dealt by the application, nor can it be solved by running the application once again; because running the application again with different circumstances does not solve the root cause of the SQLException. Such cases of exceptions are called as Non-Transient Exceptions. For example, FeatureNotSupportedException, SQLSyntaxErrorException, SQLDataException.

Depending on the type of exception that is being handled, an application can choose beforehand about ways to deal with the entire category of the said exception.

```
try
{
    //do something
}
catch (SQLTransientException sqltrex)
{
    //do nothing
}
```

The example code will not do anything for all exceptions which belong to the TransientException group, regardless of what specific exception is being thrown.



App Design

Care should be taken that while initiating the transaction, the previous connection object should be discarded and new one obtained, either through connection pool or a new internally created connection object.

try-with-resources – Construct

- A resource is an object that must be closed after it is used.
- try-with-resources statement to automatically close JDBC resources.
 - Consists of a try statement and one or more declared resources.
 - Ensures that each resource is closed at the end of the statement.
- The resources declared must have implemented the java.lang.AutoClosable interface.

JDK 7 introduces a new exception handling mechanism colloquially known as try-with-resources. In this approach of exception handling, the focus is on removing the possibility of an exception occurring rather than handling it.

The code which is going to require a specific resource (currently available for files and JDBC components only), has the responsibility of creating the object of the resource as part of the try keyword syntax as showcased in the example. The scope of the resources is restricted to that specific try block and upon execution of the try-catch-finally block the resource is closed by the runtime implicitly and becomes eligible for garbage collection.

```
public static void viewTable ( Connection con) throws  
SQLException  
{  
String query = "select * from emp";  
try (Statement stmt = con.createStatement()) //resource  
obtained  
{
```

```
ResultSet rs = stmt.executeQuery(query); //resource utilised
while (rs.next())
{
    int empno = rs.getInt("empno");
    String empname = rs.getString("empname");
}
catch(SQLException sqlErr)
{
    //perform exception code
} //resource released
}
```

As the statement object has been created with the try block, its scope is defined for the duration of the try block, as soon as try block finishes, the Statement object becomes a candidate for Garbage Collection.



The try-with-resources is also called as ARM (Automatic Resource Management). A resource is an object that must be closed after the program is finished. ARM or try-with-resources checks that each resource is closed at the end of the statement.

If the object implements interface called `java.lang.AutoCloseable`, which includes all objects which implement `java.io.Closeable`, it can be used as a resource.

Advanced Data Type

- The JDBC API provides default mappings in the form of interface for advanced data types.

BLOB	The BLOB Interface
CLOB	The CLOB Interface
NCLOB	The NCLOB Interface
ARRAY	The Array Interface
XML	The SQLXML Interface
Structured Type	The Struct Interface
REF	The Ref Interface
ROWID	The RowId Interface
DISTINCT	Java.math.BigDecimal (SQL Numeric)
DATALINK	a java.net.URL object

In addition to the standard data types JDBC API provides support to numerous SQL data types. These data types are mapped with interfaces in Java to provide the freedom of implementation to the vendors and to build in a standard mode of execution.

Advanced Data Type

- BLOB ,CLOB and NCLOB
 - An implementation of a BLOB, CLOB or NCLOB object may either be locator based.
 - JDBC driver should implement the BLOB, CLOB and NCLOB interfaces using the appropriate locator type.
 - These objects contains a logical pointer to the SQL BLOB,CLOB or CLOB data rather than the data itself.
 - The Connection interface provides support for the creation of BLOB, CLOB and NCLOB objects using the specific methods like `createBlob()`.

The most frequently used complex data type for JDBC is xLOB, where LOB stands for Large Objects. JDBC provides support to Binary Large Objects (BLOB), Character Large Objects (CLOB) and National Character Large Objects (NCLOB). These data types are usually used to represent character streams representing multimedia files such as images, voice patterns, movie clips, audio sounds etc.

As the name suggests, BLOB objects are a collection of binary data items grouped as a single entity, while CLOB is a collection of binary data items grouped as a single entity. The difference is that CLOB usually has a specific character-encoding associated with it to make sense out of the data being stored as binary streams. NCLOB is a large object used for holding National Character Set and can hold data as big as 4GB in size.

Usually, the actual data of these types of variables is kept in a separate holding location (e.g., data warehouse) and a virtual reference is provided which acts as a link to the data whenever required.

For all three types of large objects, the most crucial aspect of handling them is that the underlying driver should have the capability to understand and implement the functionality associated with these objects.

Following code demonstrates working with large objects:

```
//Creating BLOB,CLOB and NCLOB objects

Connection con = DriverManager.getConenction(url , props);
Blob aBlob = con.createBlob();
int numWritten = aBlob.setBytes(1, val);

//Retrieving BLOB, Clob and NClob Values in a ResultSet

Blob aBlob = rs. getBlob (1);
Clob aClob = rs. getClob(2);

//Accessing Blob, Clob and NClob Object Data
InputStream is = aBlob. getBinaryStream (250, 100);
BufferedReader br = aClob. getCharacterStream (250,
100);

//Storing Blob, Clob and NClob Objects
PreparedStatement pstmt = con.prepareStatement(
    "INSERT INTO bio (image, text) VALUES (?, ?)");

pstmt.setBlob (1, authorImage);
pstmt.setClob (2, authorBio);
```

Important methods associated with the large objects:

Methods	Description
void free()	This method frees the Blob object and releases the resources that it holds.
InputStream getBinaryStream()	Retrieves the BLOB value designated by this Blob instance as a stream.
InputStream	Returns an InputStream object that contains

<code>getBinaryStream(long pos, long length)</code>	a partial Blob value, starting with the byte specified by pos, which is length bytes in length.
<code>byte[]getBytes(long pos, int length)</code>	Retrieves all or part of the BLOB value that this Blob object represents, as an array of bytes.
<code>long length()</code>	Returns the number of bytes in the BLOB value designated by this Blob object.
<code>long position(Blob pattern, long start)</code>	Retrieves the byte position in the BLOB value designated by this Blob object at which pattern begins.
<code>long position(byte[] pattern, long start)</code>	Retrieves the byte position at which the specified byte array pattern begins within the BLOB value that this Blob object represents.
<code>OutputStream setBinaryStream(long pos)</code>	Retrieves a stream that can be used to write to the BLOB value that this Blob object represents.
<code>int setBytes(long pos, byte[] bytes)</code>	Writes the given array of bytes to the BLOB value that this Blob object represents, starting at position pos, and returns the number of bytes written.
<code>int setBytes(long pos, byte[] bytes, int offset, int len)</code>	Writes all or part of the given bytes array to the BLOB value that this Blob object represents and returns the number of bytes written.
<code>void truncate(long len)</code>	Truncates the BLOB value that this Blob object represents to be len bytes in length.

Alternate mechanism to database connectivity

- ResultSet requires a live connection to the database to be available at all points during its lifecycle.
 - This is called as connected architecture for database connectivity
- In situations where execution resources may not support continuous network connectivity, this approach may fail
- Disconnected Architecture for database connectivity allows the client code to manipulate the data without having to maintain a live connection to the database.
- The main drawback for this kind of approach is the possibility of data inconsistency.

ResultSet is a cursor which does not hold data but pointers to the actual data values. Due to this, it becomes mandatory to hold a connection object alive even when not actively using. This enables accessing it when any ResultSet operation demands connectivity with the database. It may not be a problem under typical scenarios. Situations where resources are scarce in terms of network bandwidth or where data access instances are few and far between it is recommended that the application should not control the network.

Database connectivity can be done either with a connected architecture or with a disconnected architecture. ResultSet is part of the connected approach, meaning if Client has a reference to a ResultSet object live during the execution, a live connection to database will have to be maintained till the scope of the ResultSet is valid. To reduce this drain on the resources, JDBC provides components to manipulate the data using the disconnected architecture.



App Design

As disconnected components do not require live connection, possibility of data inconsistency cannot be discounted, hence care needs to be taken about using disconnected architecture.

Disconnected architecture in Java – RowSet

- Java supports disconnected architecture by way of RowSet.
- A RowSet object holds tabular data
 - in a way that makes it more flexible
 - and easier to use than a resultset
- All RowSet Objects are derived from the ResultSet interface only.
- All RowSet Objects are JavaBeans Components.
- Two types of RowSet objects
 - Connected
 - Disconnected

A Rowset is a component that encapsulates the queried data into a serializable and navigable component. A Rowset can be created directly by supplying the query to the Rowset and triggering the execution using RowSet object itself.

```
//Create an object of rowset
JdbcRowSetImpl jrs = new JdbcRowSetImpl();

//specify the query to be executed
jrs.setCommand("SELECT * FROM EMP");

//supply the URL for connectivity
jrs.setURL("<url>");

//supply credential if required
jrs.setUsername("SEED");
jrs.setPassword("java");

//fire the query
jrs.execute();
```

Types of RowSets

- Connected RowSet object:
 - A Connected RowSet object uses JDBC Driver, makes the connection with database and maintains that connection throughout its life span.
 - e.g. JDBCRowSet
- Disconnected RowSet Object :
 - A disconnected RowSet object makes a connection to a data source only to read in data from a resultset object or to write data back to the data source. After reading data from and writing data to its data source, the RowSet object disconnects from it.
 - e.g. CachedRowSet

A Rowset can be either connected or disconnected. If database connectivity is required for performing the operations of a Rowset, it is called as a connected Rowset. If database connectivity is not mandatory for Rowset execution, it is called as a disconnected Rowset.

JdbcRowSet

A JdbcRowSet object is basically an enhanced ResultSet object because it maintains a connection to its data source, just like a ResultSet object.

```
Statement statement = connection.createStatement();
JdbcRowSet jdbcRowSet;
jdbcRowSet = new JdbcRowSetImpl(connection);
jdbcRowSet.setType(ResultSet.TYPE_SCROLL_INSENSITIVE);
String queryString = "SELECT * FROM emp";
jdbcRowSet.setCommand(queryString);
jdbcRowSet.execute();
jdbcRowSet.next();
jdbcRowSet.getString(1);
```

CachedRowSet

A CachedRowSet object is a special object because can operate without connection to its data source, called disconnected RowSet object. CachedRowSet maintains its data in cache, so maximum operations with cache only rather than database. CachedRowSet is super interface for all other disconnected RowSet objects.

Creating a CachedRowSet

```
ResultSet rs= st.executeQuery("select * from emp");  
CachedRowSet cachedRowSet=new CachedRowSetImpl();  
cachedRowSet.populate(rs);
```

or

```
CachedRowSet rs=new CachedRowSetImpl();  
rs.setUrl("jdbc:oracle:thin:@oracleserver:1521:orcl");  
rs.setUsername("mydb");  
rs.setPassword("password123");  
rs.setCommand("select * from emp");  
rs.execute();
```



App Design

A disconnected Rowset is particularly useful in certain situations where the data usage is disconnected from data storage for a considerable amount of time. For example, Medical Representatives visiting doctors in hinterland where connectivity is an issue. In such cases, disconnected rowset provide localized data to the user, which can be synced with the database server whenever connectivity is established.

Chapter - 3

Introduction to Web Technologies



Web applications are **simple, intuitive and responsive** that let their users get work done online with less effort and time. Today, web applications are gaining popularity since they are a gateway to the online market. This chapter covers exhaustive information about web technologies.

Objectives

At the end of this chapter you will be able to:

- Define a web site is and list types of web sites.
- List basic terminologies of web applications.
- List various components of web application architecture.
- Define a browser and list popular browsers.
- List scripting languages used for creating interactive web pages.

Distributed Architecture

- Distributed computing refers to application design paradigm in which the programs, the data they process and actual computations are spread over a network.
- Typical distributed applications are:
 - Two-tier Architecture
 - Three-tier Architecture (n-tier)

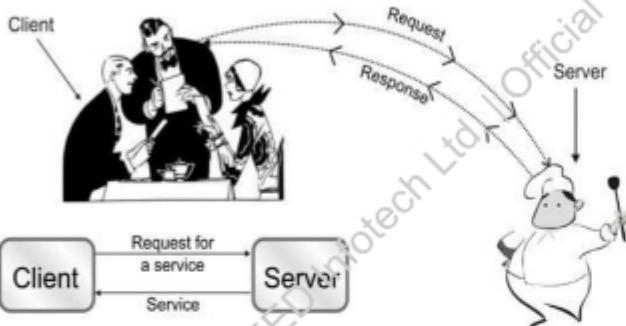
An application that needs to cater to larger client-base may need to be available to multiple clients at the same time. This requirement can be fulfilled by supplying an individual copy of the application to each client. This leads to a problem of data inconsistency since each client will be working with his/her own copy of the data in which the changes made by other users may not be reflecting.

To overcome this problem, the entire application structure is split into two parts. One takes care of the computation, which can be distributed to all the users and the second part being the actual data, is shared with all the users from a single shared location.

Similarly, modification of computation rules forces revision of the code available with each individual user. This may not be practically feasible; hence, similar to earlier solution, the computation part is also shared with every user from a single shared location. The user is provided with only the presentation part.

When a single monolithic application is split into multiple parts and each of these parts is available through a separate computing machine, it is called as distributed computation architecture. These applications are further categorized on the basis of number of splits the application has.

Two-tier Architecture



A distributed application having two components is called as a two tier application.

- The component asking for services / data is a client.
- The component providing services / data is a server.
- A client and server communicate using a pre-defined common language (Protocol)

The most common form of distributed computation is two-tier application. It is also known as client-server architecture. As the name suggests, the application is split into two parts where the user has access to the interface that helps to compute the data. The data is stored on a database server usually away from the user's location.

The two components usually communicate with each other using TCP/IP sockets, but it can also incorporate any other means of communications.

Limitations of Two-tier Architecture

- Platform dependence for data transfer
 - Data transferred from a Java application can be utilized by a Java application only
- Client Installations
 - For linking with back-end, application component on client machine mandatory

There are few limitations of two-tier architecture.

- As the users (client) and the providers (server) normally communicate using the Socket mechanism, it becomes mandatory for the server to understand the request sent by the client and for the client to understand the response sent by the client. This will happen only and only if both components share the same communication language. In simpler words, the client and the server code have to be developed using the same programming language.

For example, a server created using Java can understand requests coming from Java clients only. It cannot handle requests coming from a C# client. This severely restricts the reach and usability of the application.

- Because the server needs special mode of communication, it also becomes mandatory for the client to speak the same language. To ensure that the client and the server are talking the same language and signals, it becomes mandatory to install a piece of the application on the client-side. The lack of

client installation may mean inability of the client to communicate with the server.

As each of the client communicates with the same server (service provider), server may become over-burdened and have performance issues. Due to limited bandwidths, the number of clients that can connect with the server is also limited.

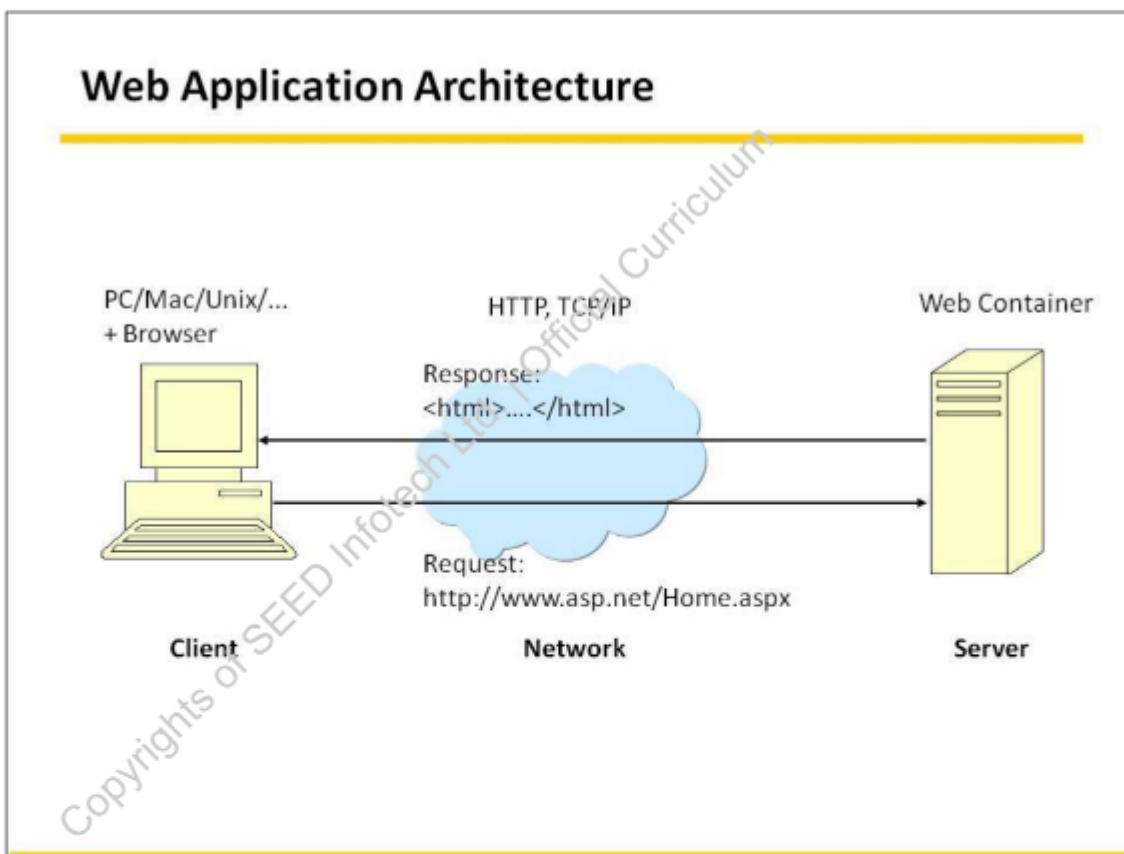
Client-Server To Web

- Use technology independent data transfer mechanism
 - Hypertext Mark-up Language (HTML).
- Use a technology independent thin client
 - Use a web browser on client side.

Client-Server (two-tier) architecture suffers from a few limitations which do not allow it to be used as a solution for an enterprise application. The solutions to the mentioned limitations are:

- Hyper Text Mark-up Language (HTML) provides a mechanism where the data is structured using strings only. Even the instruction set that governs the behaviour of the data is structured using strings. If HTML strings are used for data transfers, technology at either ends of communication does not remain a problem area.
- Usage of string to transfer data also means that the client-side installation mandatory in 2-tier architecture is no longer required. An application which can understand and translate HTML is required to be available at the client-side. This application need not be dependent on any vendor or even a technology. The only expectation from this application is that it should understand HTML and render it for the user. It is also known as a thin client, as its execution does not hog resources. Such an application is also known by a popular name – browser.

Web Application Architecture



Web applications have to be available on the internet; hence they have to be available to concurrent users at all times. To provide this feature the web applications are hosted on a dedicated machine colloquially named as a web server. The web server can be some local machine within an organization as a part of intranet or ISP (Internet Service Provider). Apache Tomcat is one of the most popular server hosts.

Web Application Architecture

Every web application is executed on request-response model. The client requests information by typing the URL (Uniform Resource Locator) in the browser. This request is sent to a web server which is present on a system in a network. Web server locates the resource requested, if available either sends it to the client for processing (client-side page) or processes it and sends the response back to the client. Request travels over the Internet using protocols like HTTP (Hypertext Transfer Protocol), known as HTTP request. Response generated by web server is also in the form of HTTP response.

The components of the web application architecture are as described below:

Web Server

The keyword Web Server can be applied to either of two components it is associated with. The physical machine on which the application resides is also known as a web server. The software which is used to manage the application on the physical machine is also known as a Web Server. From the developer perspective a Web Server is the software which is used to manage the access to the web application. It does not have the capability to host applications. At most it can serve static (HTML) pages.

Apache HTTP Server is an example of a web server.

Web Container

Web Containers are sometimes erroneously clubbed as Web Server. The responsibility of the web container is to provide execution environment for the web application, execute the applications lifecycle and manage the ecosystem in which the application runs. Usually a web container comes bundled with a web server, though it is not mandatory for a container to carry a server.

Apache Tomcat is an example of a web container.

Web Browser

As discussed earlier, text based data is the only way by which dissimilar systems can share data with each other. The data being transferred from one end to another needs to be read, understood and presented to the user in a usable format. Web Browser is a software application that understands and translates HTML content. It allows the user to interact with a remote system by converting his/her requirements into text based content and using internal mechanisms to transfer this content to a processing component.

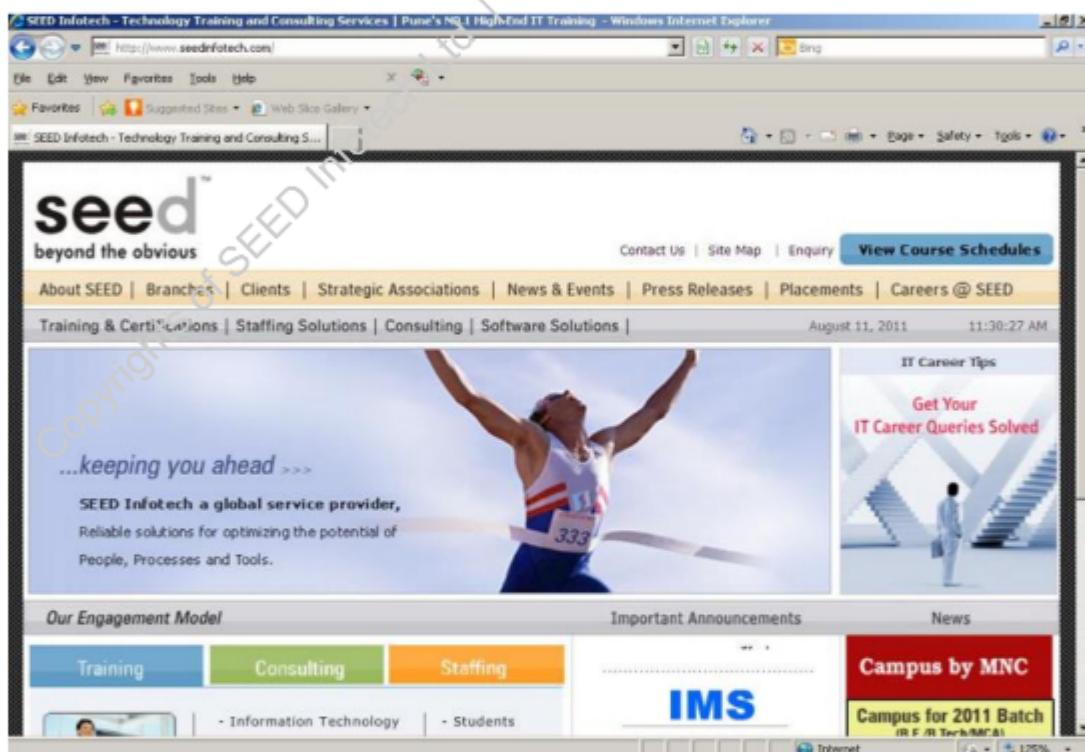
The component that provides services to the user is known as a resource and it is identified using a Uniform Resource Identifier (URI) or a Uniform Resource Locator (URL). Simply put, a browser allows a user to connect with the internet and retrieve or interact with a resource that can follow the text based data transfer rules (HTTP) and is made up of HTML. A web browser (or client) is often referred to as a user agent (UA).

The major web browsers in use are:

- Internet Explorer (IE) – Developed by Microsoft Corporation.

- Firefox – A free and open source web browser managed by Mozilla Corporation.
- Opera – A web browser developed by Opera Software
- Chrome – A web browser developed by Google Inc.

A web browser usually has a Graphical User Interface, which helps the user to understand the content better and faster.



There is an emerging discipline called usability engineering which focuses on science of how to make the user interfaces more user friendly. Software Testing also contains a specialized testing called usability testing. For more information on this discipline refer

http://en.wikipedia.org/wiki/Usability_testing



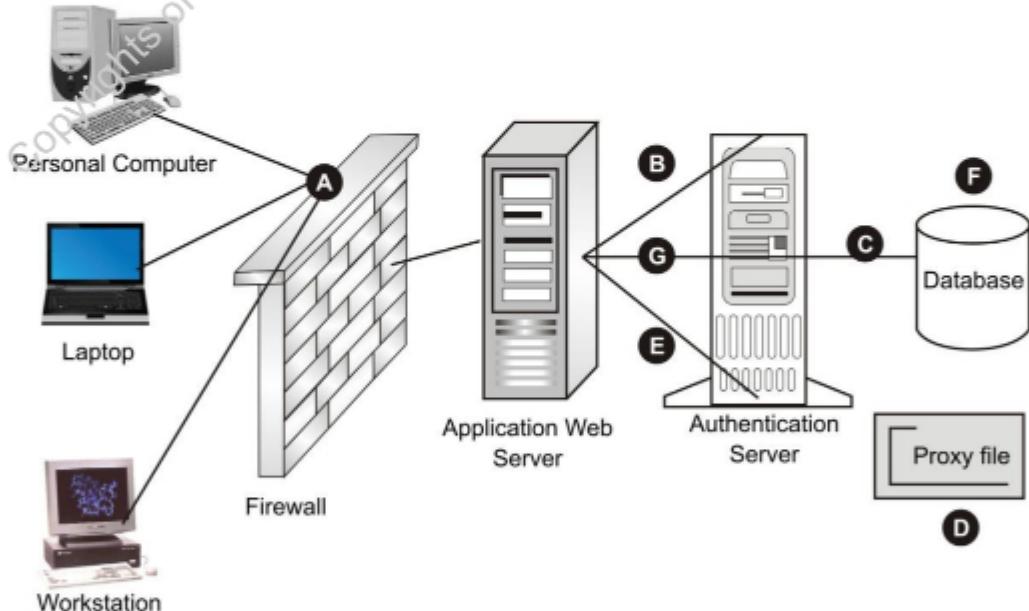
Additional Reading

Study the features of various browsers like IE, Opera, Safari, Chrome etc. available in the market.



Tech App

Every browser handles the rendering of HTML and CSS and the processing of JavaScript slightly different, requiring tests to be run on multiple browsers to ensure web application works in all environments.



Web Application Architecture – Web Container

- Software application used to host web application.
- Provides infrastructural services
 - freeing up the business logic component to focus on business rules execution.
- Provides support to application by extending contact point in form of interfaces.
 - These interfaces have to be implemented by the component that wish to use the service provided.
- Works exclusively on HTTP based request – response architecture.

Web Container

A web container is a container which contains web applications.

In three (or more) tier architecture, the user interacts with the system using the presentation layer, which deals only with the user input and output related activities. The business logic is hidden from the user. The component for business logic is accessed by all the clients at the same time and is normally kept at a central public place known to all existing and potential users (usually by way of a public URL). As the URL is public there is a requirement to cater to multiple user requirements at the same time. To provide concurrent access the business logic needs to be enveloped into threading mechanism, whereby, each client can interact with the processing component by individual threads.

Since the business logic component focuses only on the business rules and should not be catering to application execution related activities, a separate component which provides such services is required. This component provides services to the

application for making them operational. This component is called as a web container.

The primary role of a web server is to deliver web pages on the request to clients. This means delivery of HTML documents and any additional content that may be included by a document, such as images, style sheets and scripts. Though the most common use of web servers is to host web sites, they could also be used for data storage or for running enterprise applications.



A web container is different from a web server. Web server provides redirection services, whereas web container provides services like concurrency, sockets, security.

The most popular Web Servers in the use are

1. Apache Tomcat developed by the Apache Software Foundation.

The most famous and popular of all web servers is Apache Tomcat developed by the Apache Software Foundation. Not only is Apache free but it is also available for several operating systems including Windows, Macintosh and Linux/Unix.

2. Internet Information Services (IIS) –

It was formerly called as Internet Information Server. It is an integral part of Windows Server family of products. It is a web server application and includes a set of modules also referred to as extensions. These modules are individual features that the server uses to process HTTP requests.

Web Browser

- An application software or program designed to enable users to access, retrieve and view documents and other resources on the Internet.
 - A information resources could be a web page, image, multimedia, Adobe files etc.
 - It is identified by a Uniform Resource Identifier (URI) .
 - The most popular browser in the market are :
 - Internet Explorer (IE) - Developed by Microsoft Corporation.
 - Firefox - A free and open source web browser managed by Mozilla Corporation.
 - Opera - A web browser developed by Opera Software.



A web browser is a software application for retrieving, presenting and traversing information resources on the World Wide Web (WWW). The primary purpose of a web browser is to display the web resource requested by the user. This process begins when the user inputs a Uniform Resource Locator (URL), for example <http://www.seedinfotech.com/....>, into the browser. The prefix of the URL determines how the URL is interpreted. The most commonly used kind of URL starts with http: and identifies a resource to be retrieved over the Hypertext Transfer Protocol (HTTP).

All major web browsers allow the user to open multiple web pages at the same time, either in different browser windows or in different tabs of the same window. The most widely used web browsers are Internet Explorer, Google Chrome, Safari, and Opera and many more.



Study the features of various browsers available in the market.

Additional Reading

Hypertext Transfer Protocol - HTTP

- The top-level protocol used to request and return data
 - E.g. HTML pages, GIFs, JPEGs, Microsoft Word documents, Adobe PDF documents, etc.
- The protocol which is used to access web application.
- It works on port 80 and runs on top of TCP/IP protocol.
- Stateless protocol
- Methods: GET, POST, HEAD, ...

HTTP functions as a request-response protocol in the client-server computing model. While translating the data received, the browser confirms that the data has been transferred as a text. HTTP helps identify the data as text. Whenever the user needs to access a web based resource, he/she has to specify that the request is to be made using HTTP protocol. That is the reason normally user has to type in `http://` in front of the URL that is required. Specifying `http://` in front of the URL also signals the browser to transfer the request using port 80, the standard port for HTTP. Any content that the user can possibly require like images, documents, and executables can be transferred using the HTTP.

When HTTP was devised, most of the internet content was made up of static content. The need to maintain user identification was not felt. Hence HTTP was created to be a stateless protocol. HTTP does not hold any user identification or user related data in the request generated. Every request generated by the browser using HTTP is always a new request.

HTTP specification provides with certain methods which are indicative of the process that needs to be performed at the server side when the request reaches there.

The methods are as described below:

GET

It is used to retrieve whatever data is identified by the Uniform Resource Locator (URL), where the URL refers to a data-producing process, or a script which can be run by such a process. The required data is rendered as HTML and returned to client browser

POST

Submits data to be processed (e.g., from an HTML form) to the identified resource. The data is included in the body of the request. It is the responsibility of the component existing on the server side to retrieve the data and process it.



HEAD	The HEAD method is identical to GET except that the server must not return a message-body in response. This method is often used for testing hypertext links for validity, accessibility, and recent modification. This is useful for retrieving meta-information written in response headers, without having to transport the entire content.
PUT	It uploads a representation of the specified resource.
DELETE	It deletes the specified resource.
TRACE	It allows the client to see what is being received at the other end of the request chain and use that data for testing or diagnostic information.
OPTIONS	It returns the HTTP methods that the server supports for specified URL. This can be used to check the functionality of a web server by requesting '*' instead of a specific resource.



Interview Tip

It is recommended that Get method be used for retrieving resources from the server side and Post method be used for submitting data to the server for processing. The methods can be used interchangeably but not recommended.



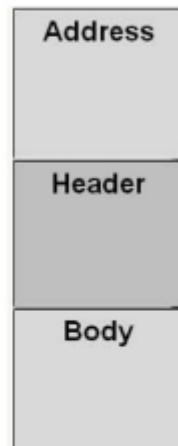
Additional Reading

For additional reading on HTTP specification visit the link below

<http://www.w3.org/Protocols/>

HTTP Message Format

- Request-Method Request-URI Protocol
 - e.g. GET/index.html HTTP/1.1
- Response-Protocol Status-Code Description
 - e.g. HTTP/1.1 200 OK
 - Every response contains a status code.
 - 100 level :Informational :(Continue)
 - 200 level :Successful : (OK)
 - 300 level :Redirection
 - 400 level :Client Error (401 == BAD REQUEST)
 - 404 level: Not found
 - 500 level :Server Error (503 == Service Unavailable)



Any resource on the web is accessed as:

`http://<sitename>/resourcename`

`http://www.seedinfotech.com/index.html`

The resource to execute or retrieve is provided as either a URI or a URL

URI – Uniform Resource Identifier

URL – Uniform Resource Locator

The response code returned by the servers are as follows:

Response Code	Meaning
100 Continue	This means that the server has received the request headers, and that the client should proceed to send the request body.
200 OK	The request succeeded, and the resulting

	resource (e.g. file or script output) is returned in the message body.
300 Multiple Choices	It indicates that further action needs to be taken by the user agent in order to fulfil the request. It indicates multiple options for the resource that the client may need to follow.
404 Not Found	The requested resource does not exist.
500 Server Error	An unexpected server error. The most common cause is a server-side script that has bad syntax, fails, or otherwise cannot run correctly.

Message Headers

- Format : [name] : [value]

Client examples

- Connection: keep-Alive | close
- Accept-Charset: utf-8 |iso-8859-1,*
- Accept-Encoding:<compress|gzip|deflate|identity>
- Cache-Control: no Cache
- From: scott@gmail.com
- Host: www.gmail.com
- Accept: image/gif, image/x-xbitmap, image/jpeg
- Accept-language: en-US

Server examples

- Accept-Ranges: bytes
- Allow: GET,HEAD
- Cache-Control: max-age=3600
- Connection: close
- Content-Encoding: gzip
- Content-language: da
- Content-Length: 450
- Content-Location: /index.html
- Content-Type: text/html
- Date: Wed, 13 Jul 2011 11:57:35 GMT
- Last-Modified: Tue, 15 Nov 2010 12:45:26 GMT
- Server: Apache/1.3.27 (Unix) (Red-Hat/Linux)
- CRLF-(Carriage Return Line Feed)
- [Entity Body]

A browser generates a request and sends it to the resource identified in the request address. Along with the data the browser also generates a set of meta-data which is embedded in the request object as its header. This header information is useful while processing the request data on the server side. Some of the data embedded in the header identifies the data types being passed through the request such as the character set of the request data, the date and time of request generation.

Typical HTTP Session

- Client opens a connection to the server
- Client makes a request to the server:

```
GET/index.html HTTP/1.0
```

- The Server responds to the request

```
HTTP/1.0 200 OK  
[more header stuff]
```

```
CRLF
<html>
[html code from index.html]
</html>
```

- The connection is closed by the server (or client)

Web Application Programming

- Web based applications are developed using
 - Client side programming
 - The class of computer programs on the web that are executed client-side, by the user's web browser
 - HTML,DHTML,JavaScript,VBScript
 - Server side programming
 - Software that runs on the server, using technologies like
 - Active Server Pages (ASP) developed by Microsoft,
 - Java Server Pages (JSP) developed by Sun Micro Systems,
 - Hypertext PreProcessor (PHP) originally created by Rasmus Lerdorf
 - ASP.NET – A successor to ASP ,developed by Microsoft

Among computer programming languages, there is no single application that does all the different things, in all the different ways that programmers need. Because of the great number and diversity of programming tasks, choosing a web application programming language is a critical step.

Database-driven websites can now be built with varied scripting languages as PHP, ASP.NET, JSP, Perl and Cold Fusion, which fall into two main groups – proprietary and open-source. All the examples are open-source except the proprietary Cold Fusion and ASP.NET.

Web application programming languages fall into two categories: Client-side web languages and server-side web languages.

Client-side Scripting

In client-side scripting, web applications are executed by web browsers at client side. They are created using languages like HTML, DHTML (Dynamic HTML) VBScript, and JavaScript. Client-side scripts are written using scripting languages. These scripts are usually a part of HTML file or contained in a separate file, which

is referenced by the Web page(s) that use it. When clients send a request, the required files are sent by web server to the browser. The browser executes the script and displays the output on the client machine.

Advantages of client-side scripting

- Better scalability as less work done on server.
- Reduces traffic network as there is no round trip to server.
- One can create UI constructs not inherent in HTML like:
 - Drop-down and pull-out menus
 - Tabbed dialogs
- Cool effects such as animation can be given.
- Data validation is performed at client side to ensure that it is acceptable by server.
- Opening or popping up a new window with programmatic control over the size, position, and attributes of the new window (e.g. menus, toolbars, etc., are visible).
- Changing images as the mouse cursor moves over them; this effect is often used to draw the user's attention to important links displayed as graphical elements.

Popular scripting languages are: JavaScript and VBScript.

JavaScript is an interpreted programming or script language from Netscape. It is similar in capability to Microsoft's Visual Basic. In general, script languages are easier and faster to code than the more structured and compiled languages such as C and C++. Script languages generally take longer to process than compiled languages, but are very useful for shorter programs.

JavaScript is used in web site development to do following things:

- Perform client side validations
- Automatically change a formatted date on a web page
- Cause a linked-to page to appear in a popup window
- Cause text or a graphic image to change during a mouse rollover
- Display pull down or popup menu.

JavaScript

JavaScript code can be imbedded in HTML pages and interpreted by the web browser (or client). It can also be run at the server-side as in Microsoft's Active Server Pages before the page is sent to the requestor. Both Microsoft and Netscape browsers support JavaScript, but sometimes in slightly different ways.

VBScript

VBScript is an interpreted script language from Microsoft that is a subset of its Visual Basic programming language designed for interpretation by Web browsers. VBScript can be compared to other script languages that can be used on the Web, including: JavaScript by Netscape, Tcl by Sun Microsystem, and UNIX-derived Perl etc.

VBScript is Microsoft's answer to Netscape's popular JavaScript. Both are designed to work with an interpreter that comes with a web browser - that is, at the user or client end of the web client/server session. VBScript is designed for use with Microsoft's Internet Explorer browser together with other programs that can be run at the client, including ActiveX controls, automation servers, and Java applets. Although Microsoft does support Netscape's JavaScript (it converts it into its own JScript), Netscape does not support VBScript. For this reason, VBScript is best used for intranet web sites that use only the Internet Explorer browser.

Server-side Scripting

In server-side scripting, web applications are executed at server-side. They are created using languages like Perl, PHP, and server-side VBScript, and server-side JavaScript. The output is displayed in the web browser in a format understood by the browser. The server side scripting code is not visible to the client.

The server-side program can receive input from URL parameters, HTML form data, cookies, and HTTP headers. It can access server-side databases, e-mail servers, files, mainframes, and so on.

Following are the advantages of server-side scripting:

- Distribution of application code not required.
- Never exposes the server-side script code.
- Emits the HTML markup based on the user's input data.

- It is usually used to provide interactive web sites that interface to databases or other data stores.
- Fundamentally, server-side scripting has the ability to highly customize the response based on the user's inputs, access rights, or queries into data stores.
- Security is another advantage of server-side processing because only the results of code processing are sent to the web browser keeping confidential information such as passwords, credit card numbers, and even the code, on the server.

In addition to the server-side scripts, there are other programming constructs available which help in building a truly responsive web application.

Technology	Description
Servlet	Java based server side component used to build a web application
JSP	Java based component for Rapid Application Development
ASP.NET	A server side technology from Microsoft for creating enterprise class web applications.
PHP	PHP is a general-purpose server-side scripting language originally designed for web development to produce dynamic web pages.
Perl	Perl is a general-purpose Unix scripting language. It is used for graphics programming, system administration, network programming, finance, and other applications.



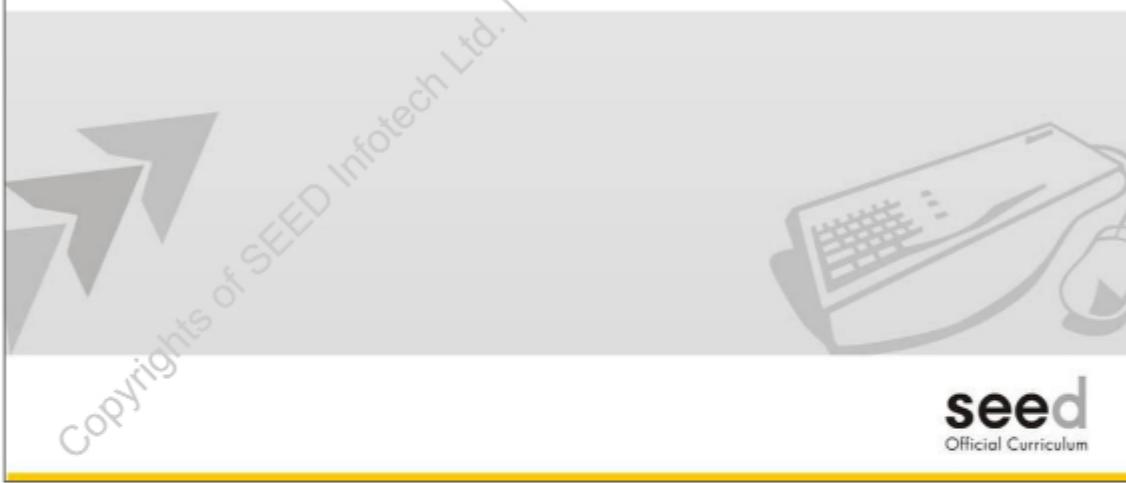
Focus on knowledge about various web programming languages or technologies with their usage.



Identify the list of browsers that support VBScript and JavaScript.

Chapter - 4

HTML and JavaScript



HTML is the basic building block for creating a web application. HTML also ensures that the application reaches multitudes rather than being dependent on the technology being used. A user's first look at the web application is always through the HTML content. It is mandatory to generate HTML content that is pleasing, user-friendly and usable. This can be possible only if the developer also has knowledge about the HTML.

This chapter covers basics of HTML, its various tags, forms, embedded JavaScript and event handling with JavaScript.

Objectives

At the end of this chapter you will be able to:

- Explain HTML syntax.
- Construct an HTML page using basic tags like text field, radio button.
- Construct an HTML form for submitting data.
- Construct an HTML page with JavaScript embedded.
- Construct an HTML page with event handling mechanism enabled with JavaScript.

What is HTML?

- HTML is a language for describing web pages.
- HTML stands for Hyper Text Markup Language.
- HTML is not a programming language, it is a markup language.
- A markup language is a set of markup tags.
- HTML uses markup tags to describe web pages.

Due to cross-platform data transfer issues, text based data transfer is preferred over object based data transfer. The language made up of textual data and control sequence is called as HTML. Hyper Text Markup Language (HTML) is a language of the internet. As HTML is defined to be a standard language all the browsers understand the individual markups and the actions associated with them.

HTML Tags

HTML markup tags are usually called HTML tags. They are keywords surrounded by angle brackets like <html>. HTML tags normally come in pairs like and . The first tag in a pair is the start tag; the second tag is the end tag. Start and end tags are also called opening tags and closing tags.

HTML documents are also known as web pages. HTML documents contain HTML tags and plain text. The purpose of a web browser (like Internet Explorer or Firefox) is to read HTML documents and display them as web pages. The browser does not display the HTML tags, but uses the tags to interpret the content of the page.

```
<html>
<body>
<h1>My First Heading</h1>
<p>My first paragraph.</p>
</body>
</html>
```

- The text between `<html>` and `</html>` describes the web page.
- The text between `<body>` and `</body>` is the visible page content.
- The text between `<h1>` and `</h1>` is displayed as a heading.
- The text between `<p>` and `</p>` is displayed as a paragraph.

HTML can be written and edited using many different editors like Dreamweaver and Visual Studio.



App Design

Extension of an HTML file can be either .htm or .html. Either way, does not make a difference.

HTML Elements

- An HTML element
 - Starts with a start or opening tag
 - Ends with an end or closing tag
- The element content is everything between the start and the end tag
 - Some HTML elements have empty content.
 - Empty elements are closed in the start tag
- Most HTML elements can have attributes

HTML Document

```
<html>
<body>
<p>This is my first paragraph.</p>
</body>
</html>
```

The example above contains three HTML elements.

The `<p>` element:

```
<p>This is my first paragraph.</p>
```

The `<p>` element defines a paragraph in the HTML document. The element has a start tag `<p>` and an end tag `</p>`.

The `<body>` element:

```
<body>
<p>This is my first paragraph.</p>
</body>
```

The `<body>` element defines the body of the HTML document. The element has a start tag `<body>` and an end tag `</body>`. The element content is another HTML element (a `p` element).

```
<html>  
  
<body>  
<p>This is my first paragraph.</p>  
</body>  
  
</html>
```

The `<html>` element defines the whole HTML document. The element has a start tag `<html>` and an end tag `</html>`. The element content is another HTML element (the `body` element), the contained element is known as a nested HTML element.

Most HTML elements can be nested (can contain other HTML elements). HTML documents consist of nested HTML elements.



Do not forget the end tag.

Some HTML elements might display correctly even if you forget the end tag. Many HTML elements will produce unexpected results and/or errors if you forget the end tag.

Empty HTML Elements

HTML elements with no content are called empty elements. `
` is an empty element without a closing tag.



In XHTML, all elements must be closed. Adding a slash inside the start tag, like `
`, is the proper way of closing empty elements in XHTML (and XML). Always use lowercase tags.

HTML Attributes

HTML elements can have attributes. Attributes provide additional information about an element. Attributes must always be specified in the start tag. Attributes

have to be defined in name/value pairs like: name="value". Attribute Values should always be enclosed in a quotation marks. Double style quotes are the most common, but single style quotes are also allowed.



Best Practice

Use Lowercase Attributes. Attribute names and attribute values are case-insensitive. In some rare situations, when the attribute value itself contains quotes, it is necessary to use single quotes.

However, the World Wide Web Consortium (W3C) recommends lowercase attributes / attribute values in their HTML 4 recommendation. Newer versions of (X)HTML will demand lowercase attributes.

Standard attributes

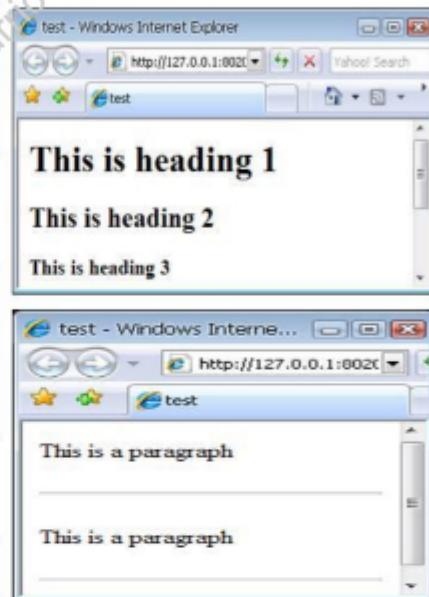
Attribute	Description
classname	Specifies a class name for an element
Id	Specifies a unique id for an element
style	Specifies an inline style for an element
title	Specifies extra information about an element (displayed as a tool tip)

Customizing Look and Feel

```
<h1>This is heading 1</h1>
<h2>This is heading 2</h2>
<h3>This is heading 3</h3>
```

```
<p>This is a paragraph</p>
<hr />
<p>This is a paragraph</p>
<hr />
```

```
<!-- This is a comment -->
```



Look and feel of web pages can be obtained using certain tags.

Heading

HTML headings are used for displaying text as headings. Headings should not be used to make text BIG or bold. Search engines use headings to index the structure and content of web pages. Since users may skim pages by its headings, it is important to use headings to show the document structure.

- H1 headings should be used as main headings, followed by H2 headings, then the less important H3 headings, and so on.
- The `<hr/>` tag creates a horizontal line in an HTML page. The `<hr>` element can be used to separate content.

Paragraphs

Paragraphs are defined with the `<p>` tag. Most browsers display HTML correctly even if there is no end tag.

```
<p>This is a paragraph</p>
<p>This is another paragraph</p>
```

```
<p>This is <br />a para<br/> graph with line breaks</p>
<p>This is a paragraph
<p>This is another paragraph
```

The example above works in most browsers, but it is not reliable. Forgetting the end tag can produce unexpected results or errors.



Tech App

Future version of HTML will not allow you to skip end tags. Use the `
` tag if you want a line break (a new line) without starting a new paragraph.

In XHTML, XML, elements with no end tag (closing tag) are not allowed.

Even if `
` works in all browsers, writing `
` instead works better in XHTML and XML applications.

The `
` element is an empty HTML element. It has no end tag. Large or small screens and resized windows will create different results. With HTML, output cannot be changed by adding extra spaces or extra lines in the HTML code. The browser removes extra spaces and extra lines when the page is displayed. Any number of lines counts as one line, and any number of spaces count as one space. Comments can be inserted into the HTML code to make it more readable and understandable. Comments are ignored by the browser and are not displayed.

Customizing Text

```
<html>
<body>
<p><b>This text is bold</b></p>
<p><strong>This text is strong</strong></p>
<p><big>This text is big</big></p>
<p><em>This text is emphasized</em></p>
<p><i>This text is italic</i></p>
<p><small>This text is small</small></p>
<p>This is<sub> subscript</sub> and
    <sup>superscript</sup></p>
</body>
</html>
```



HTML uses tags like `` and `<i>` for formatting output, like bold or italic text. These HTML tags are called formatting tags. Often `` renders as ``, and `` renders as `<i>`. However, there is a difference in the meaning of these tags. `` or `<i>` defines bold or italic text only. `` or `` means that the text needs to be rendered in a way that the user understands as "important".

All major browsers render `strong` as bold and `em` as italics. However, if a browser wants to make a text highlighted with the strong feature, it might be cursive and not bold.

The `` tag is deprecated in HTML 4, and removed from HTML 5. The World Wide Web Consortium (W3C) has removed the `` tag from its recommendations. Instead of `` tag the style attribute for the component should be used.

```
<p>
<font size="5" face="arial" color="red">
This paragraph is in Arial, size 5, and in red text
color.
</font>
</p>
<p>
<font size="3" face="verdana" color="blue">
This paragraph is in Verdana, size 3, and in blue text
color.
</font>
</p>
```

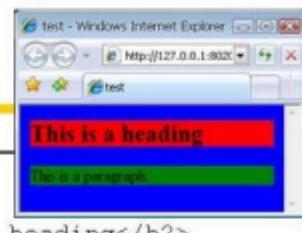


Tech App

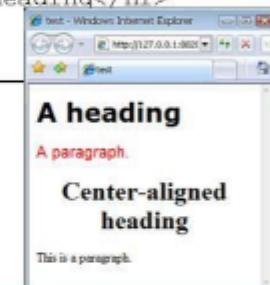
In HTML 4, style sheets (CSS) should be used to define the layout and display properties for the HTML elements.

Styling the Content

```
<html>
  <body style="background-color:blue;">
    <h2 style="background-color:red;">This is a heading</h2>
    <p style="background-color:green;">This is a paragraph.</p>
  </body>
</html>
```



```
<body>
  <h1 style="font-family:verdana;">A heading</h1>
  <p style="font-family:arial;color:red;font-size:20px;">A
  paragraph.</p>
  <h1 style="text-align:center;">Center-aligned heading</h1>
  <p>This is a paragraph.</p>
</body>
```



CSS was introduced together with HTML 4, to provide a better way to style HTML elements. CSS can be added to HTML in:

4. Cascading Style Sheet files (CSS files)
5. The `<style>` element in the HTML head section
6. The `style` attribute in single HTML elements

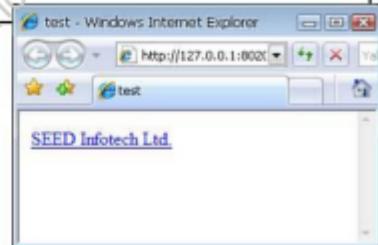
The `text-align` property makes the old `<center>` tag obsolete.



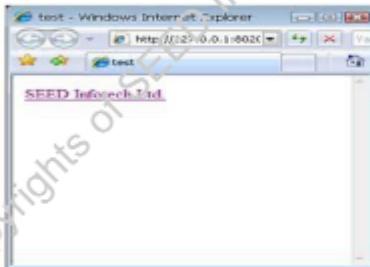
The preferred way to add CSS to HTML is to put CSS syntax in separate CSS files, as it is time consuming and not very practical to style HTML elements using the `style` attribute.

Links to Other Resources

```
<a href="http://www.seedinfotech.com/">SEED Infotech Ltd.</a>
```



```
<a href="http://www.w3schools.com/" target="_blank">Visit W3Schools!</a>
```



A hyperlink (or link) is a word, group of words, or image that can be clicked to jump to a new document or a new section within the current document. When the cursor hovers over a link in a web page, the arrow turns into a little hand.

Links are specified in HTML using the `<a>` tag.

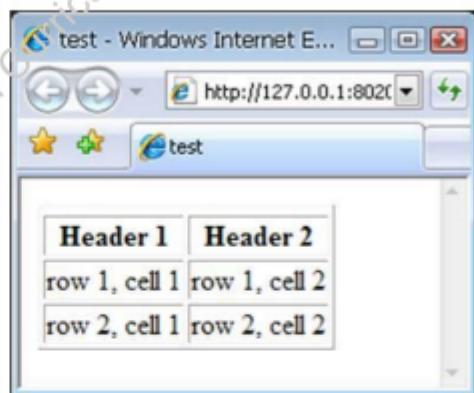
The `<a>` tag can be used in two ways:

- To create a link to another document, by using the `href` attribute
- To create a bookmark inside a document, by using the `name` attribute

The `target` attribute specifies where to open the linked document.

HTML – Tables

```
<table border="1">
<tr>
<th>Header 1</th>
<th>Header 2</th>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>row 2, cell 1</td>
<td>row 2, cell 2</td>
</tr>
</table>
```



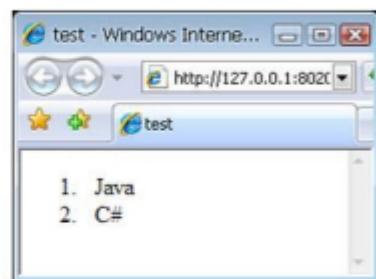
Tables are defined with the `<table>` tag. A table is divided into rows (with the `<tr>` tag), and each row is divided into data cells (with the `<td>` tag). `td` stands for "table data," and holds the content of a data cell. A `<td>` tag can contain text, links, images, lists, forms, other tables, etc. Header information in a table is defined with the `<th>` tag. All major browsers display the text in the `<th>` element as bold and centered.

Creating Lists

```
<ul>  
<li>Java</li>  
<li>C#</li>  
</ul>
```



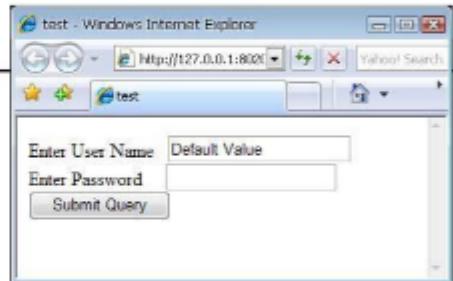
```
<ol>  
<li>Java</li>  
<li>C#</li>  
</ol>
```



An unordered list starts with the `` tag. Each list item starts with the `` tag. The list items are marked with bullets. An ordered list starts with the `` tag. Each list item starts with the `` tag. The list items are marked with numbers.

Creating Data Forms

```
<form action="URL of some other component">  
Enter User Name &nbsp; <input type="text"  
name="username" value="Default Value"/>  
<br/>  
Enter Password &nbsp;&nbsp; <input type="password"  
name="pwd" />  
<br/>  
<input type="submit" name="Submit Data" />  
</form>
```



HTML forms are used to pass data to a server. A form can contain input elements like text fields, checkboxes, and radio-buttons, submit buttons and more. A form can also contain select lists, textarea, fieldset, legend, and label elements. The `<form>` tag is used to create an HTML form.

Data Inputs

```
<form action="URL of some other component">  
Enter User Name &nbsp; <input type="text" name="username"  
value="Default Value"/>  
<br/>  
Enter Password &nbsp; &nbsp; <input type="password" name="pwd" />  
<br/>  
Gender <input type="radio" name="sex" value="male" /> Male  
<input type="radio" name="sex" value="female" /> Female  
<br/>  
Mode of transport <input type="checkbox" name="vehicle"  
value="Bike" /> I have a bike  
<input type="checkbox" name="vehicle" value="Car" /> I have a car  
<br/>  
<input type="submit" name="Submit Data" value="Submit Query" />  
</form>
```



A submit button is used to send form data to a server. The data is sent to the page specified in the form's action attribute. The file defined in the action attribute usually does something with the received input.

Following are the input types that are used on a form:

Input type	Description
Text	Defines a one-line field that accepts text input
Password	Defines a password field for accepts hidden text input
Radio	Defines a set of options as radio buttons
Checkbox	Defines a set of options as check boxes
Submit	Defines a button for submitting data



Tech App

The default width of a text field is 20 characters.

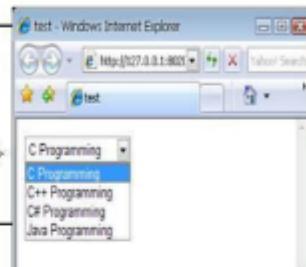
The characters in a password field are masked (shown as asterisks).

Radio buttons let a user select only one of a limited number of choices.

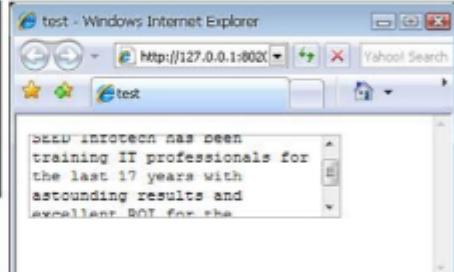
Checkboxes let a user select one or more options of a limited number of choices.

More Data Inputs

```
<select>
    <option value="C">C Programming</option>
    <option value="C++">C++ Programming</option>
    <option value="C#">C# Programming</option>
    <option value="Java">Java Programming</option>
</select>
```



```
<textarea rows="4" cols="30">
SEED Infotech has been training IT
professionals for the last 17 years
with astounding results and excellent
ROI for the students.
</textarea>
```



The `<select>` tag is used to create a drop-down list. The `<option>` tags inside the `<select>` element define the available options in the list.



The `<select>` element is a form control and is used to collect user input; it should not be used without the enclosing `<form>` element.

What is JavaScript?

- A scripting language is a lightweight programming language.
 - JavaScript, VBScript
- JavaScript was designed to add interactivity to HTML pages.
 - Usually embedded directly into HTML pages
 - An interpreted language (means that scripts execute without preliminary compilation)
 - Can be used JavaScript without purchasing a license

Normally, HTML authors are not programmers. JavaScript gives them a programming tool. It is a scripting language with a very simple syntax. Small "snippets" of code can be embedded into an HTML page very easily. JavaScript can react to events – it follows the event driven model, like when a page has finished loading or when a user clicks on an HTML element. JavaScript code can read and change the content of an HTML element using Document Object Model (DOM) objects.

JavaScript can be used to

- Validate data on a form before it is submitted to a server. This saves the server from extra processing.
- Detect the visitor's browser and depending on the browser load another page specifically designed for that browser.
- Store and retrieve information on the visitor's computer.



Interview Tip

Java and JavaScript are two completely different languages in both concept and design. They do not have anything in common other than the name.

Java (developed by Sun Microsystems) is a powerful and much more complex programming language used for building enterprise applications.

Adding JavaScript to HTML page

```
<body>
<h1>My First Web Page</h1>
<script type="text/javascript">
    document.write ("<p>" + Date() + "</p>");
</script>
</body>
```



To insert JavaScript into an HTML page, the `<script>` tag is used. Inside the `<script>` tag the `type` attribute is used to define the scripting language. The `<script>` and `</script>` tags tell where the JavaScript starts and ends. Unlimited number of scripts can be put in the document, and scripts can exist in the body as well as in the head section at the same time. It is a common practice to put all functions in the head section, or at the bottom of the page. This way they are all in one place and do not interfere with page content.

JavaScript is case sensitive; therefore care should be taken to use the correct cases when writing JavaScript statements, create or call variables, objects and functions.

A JavaScript statement is a command to a browser. The purpose of the command is to tell the browser what to do. The semicolon is optional, and the browser interprets the end of a line as the end of the statement.

JavaScript can also be placed in external files. External JavaScript files often contain code to be used on several different web pages. External JavaScript files have the file extension `.js`.



Tech App

External script cannot contain the `<script> </script>` tags. Using semicolons makes it possible to write multiple statements on one line. To use an external script, point to the .js file in the "src" attribute of the `<script>` tag.

JavaScript statements can be grouped together in blocks. Blocks start with a left curly bracket `{}`, and end with a right curly bracket `}`. The purpose of a block is to make the sequence of statements execute together. Browsers that do not support JavaScript display JavaScript as page content. To prevent them from doing this, and as a part of the JavaScript standard, the HTML comment tag should be used to "hide" the JavaScript.



Tech App

To avoid JavaScript code appearing on HTML page, add an HTML comment tag `<!--` before the first JavaScript statement, and a `-->` (end of comment) after the last JavaScript statement. The browser ignores the commented code and executes the JavaScript if it supports, and ignores the commented code if it does not support.

Using Variables in JavaScript

```
<html>
<head>
<title> NEW DOCUMENT </title>
</head>
<body>
<script language="javascript">
    var sample="SEED";
    document.write(SAMPLE);
    sample="HELLO SEED";
    document.write("<BR>");
    document.write(SAMPLE);
</script>
</body>
</html>
```



JavaScript closely follows the 'C' coding structure and naming conventions. It allows the manipulation of the data using variables and variable operations.

Some of the operations available through JavaScript are

Operator	Description
+	Addition of two expressions
-	Subtraction of two expressions
*	Multiplication of two expressions
/	Division of two expressions
%	Modulus of two expressions
++	Increment of an expression
--	Decrement of an expression
+=	Addition of an expression
-=	Subtraction of an expression
*=	Multiplication of an expression
/=	Division of an expression

<code>%=</code>	Modulus of an expression
<code>==</code>	Is equal to
<code>===</code>	Is exactly equal to
<code>!=</code>	Not equal
<code>></code>	Greater than
<code><</code>	Less than
<code>>=</code>	Greater than or equal to
<code><=</code>	Less than or equal to
<code>&&</code>	And condition
<code> </code>	Or condition
<code>!</code>	Not condition

JavaScript Programming Construct

- JavaScript variable can store different types of values.
- JavaScript provides 7 basic data types.
- JavaScript supports Implicit and explicit type conversions.
- JavaScript programming constructs are
 - if ..else
 - switch ..Case
 - while
 - do ..while
 - for
- JavaScript supports three types of Pop up boxes
 - Alert Box
 - Prompt Box
 - Confirm Box

In JavaScript following conditional statements are available

- if statement is used to execute some code only if a specified condition is true
- if...else statement is used to execute some code if the condition is true and another code if the condition is false
- if...else if....else statement is used to select one of many blocks of code to be executed
- switch statement is used to select one of many blocks of code to be executed

if statement example:

```
<script type="text/javascript">
//Write a "Good morning" greeting if
//the time is less than 10

var d=new Date();
var time=d.getHours();
```

```
if (time<10)
{
    document.write("<b>Good morning</b>");
}
</script>
```

The if....else statement should be used when a piece of code is to be executed if a condition is true, and there is an alternate code to be executed when the condition is false.

```
<script type="text/javascript">
//If the time is less than 10, you will get a "Good
morning" greeting.
//Otherwise you will get a "Good day" greeting.
. . .
if (time < 10)
{
    document.write("Good morning!");
}
else
{
    document.write("Good day!");
}
</script>
```

Use if....else if....else statement to select one of several blocks of code to be executed.

```
. . .
if (time<10)
{
    document.write("<b>Good morning</b>");
}
else if (time>=10 && time<16)
{
    document.write("<b>Good day</b>");
}
else
{
    document.write("<b>Hello World!</b>");
```

```
 }  
 . . .
```

In JavaScript, there are two different loops available.

- **for** - loops through a block of code a specified number of times
- **while** - loops through a block of code while a specified condition is true

The **for** Loop

for loop is used when you know in advance how many times the script should run.

```
. . .  
for (i=0;i<=5;i++)  
{  
    document.write("The number is " + i);  
    document.write("<br />");  
}  
. . .
```

The **while** Loop

The **while** loop loops through a block of code while a specified condition is true.

```
. . .  
var i=0;  
while (i<=5)  
{  
    document.write("The number is " + i);  
    document.write("<br />");  
    i++;  
}  
. . .
```

Notifying the User

- Alert box
 - To confirm the data user has entered.
 - User has to press ok to proceed.
 - Alert ("sometext");
- Alert box with line breaks.
 - Alert generated with line break.
- Confirm box
 - Used to verify user input.
 - Confirm ("sometext");
 - User have to click "ok" or "cancel".

In certain situations the application may want to bring the user's attention to a specific event, data or action. This needs to be achieved in such a way that the user has no option but to take a look at result generated by the application. This action is called as notification. JavaScript allows notifying the user in three different ways.

Alert Box

An alert box is often used if information is to be passed on to the user and ensure that user notices it. When an alert box pops up, the user will have to click OK to proceed.

```
<script type="text/javascript">
function show_alert()
{
    alert("I am an alert box!");
}
</script>
</head>
```

```
<body>
<input type="button" onclick="show_alert()" value="Show
alert box" />
</body>
```

A confirm box is often used if the user needs to verify or accept something. When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed. If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

```
. . .
<function show_confirm()
{
var r=confirm("Press a button");
if (r==true)
{
    alert("You pressed OK!");
}
else
{
    alert("You pressed Cancel!");
}
}
<input type="button" onclick="show_confirm()"
value="Show confirm box" />
. . .
```

Prompt Box

A prompt box is often used if the user needs to input a value before entering a page. When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value. If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

```
<html>
<head>
<script type="text/javascript">
function show_prompt()
{
    var name=prompt("Please enter your name", "Harry
```

```
Potter");
if (name!=null && name!="")
{
    document.write("Hello " + name + "! How are you
today?");
}
</script>
</head>
<body>
<input type="button" onclick="show_prompt()" 
value="Show prompt box" />
</body>
</html>
```

JavaScript Functions

- Functions are named blocks of statements to perform a single task or a series of tasks.
 - Reusable code block executed on the occurrence of an event or when called.
 - Can be defined both in the `<head>` section and the `<body>` section.
 - To ensure that functions are read before they are called they are defined in the `<head>` section.

To keep the browser from executing a script when the page loads, the script should be written inside a function. A function contains code that will be executed by an event or by a call to the function. Developer may call a function from anywhere within a page (or even from other pages if the function is embedded in an external .js file).

Functions can be defined both in the `<head>` and in the `<body>` section of a document. However, to ensure that a function is read or loaded by the browser before it is called, it is recommended to put functions in the `<head>` section.

A JavaScript function has syntax similar to 'C' programming language. The keyword `function` identifies start of the function and the last closing bracket `()` defines the end.

If a variable is declared using "var", within a function, the variable can only be accessed within that function. When the function completes, the variable is destroyed. These variables are called local variables. If a variable is declared

outside a function, all the functions on the page can access it. The lifetime of these variables starts when they are declared, and ends when the page is closed.

Syntax of writing function in JavaScript is as follows:

```
function functionName ()  
{  
    // function code  
}
```

Syntax of writing function with arguments is as follows:

```
// var1,var2 are values passed into the function.  
function functionName(var1, var2, ..., varn)  
{  
    // function code  
}
```

Event Driven Model in JavaScript

- Events - describe actions that occur as the result of user interaction with a web page or other browser related activities.
- For Example:
 - Clicking a hyperlink
 - Selecting input box on HTML
 - Submitting HTML form
 - A mouse click
 - A button click

By using JavaScript, dynamic web pages can be created. Events are actions that are detected by JavaScript through the browser.

Every element on a web page has certain events which can trigger a JavaScript. For example, when a user clicks a button on the HTML page, the button click event is said to be generated and developer can write code to handle (perform against) this event.

Examples of events:

- A mouse click
- A web page or an image loading
- Mouse hovering over a hot spot on the web page
- Selecting an input field in an HTML form
- Submitting an HTML form
- A keystroke

Event Handling in JavaScript

- Event Handling
 - The process that is performed in response to the occurrence of an event is known as Event Handling.
- Event handler
 - The code which is executed on occurrence of an event to generate the response is called as Event handler.

Event handling is a process which allows a developer to obtain a handle on the execution cycle of the HTML web page. The event delegation model supplies certain hooks (discussed earlier as events), which facilitate writing of code to be executed when an event occurs. Normally, the code block is written in a callback method identifying the event. For example, mouse click is associated with the `onclick()` method. This method is an event handler.

Implementing event handling in JavaScript requires two-step execution:

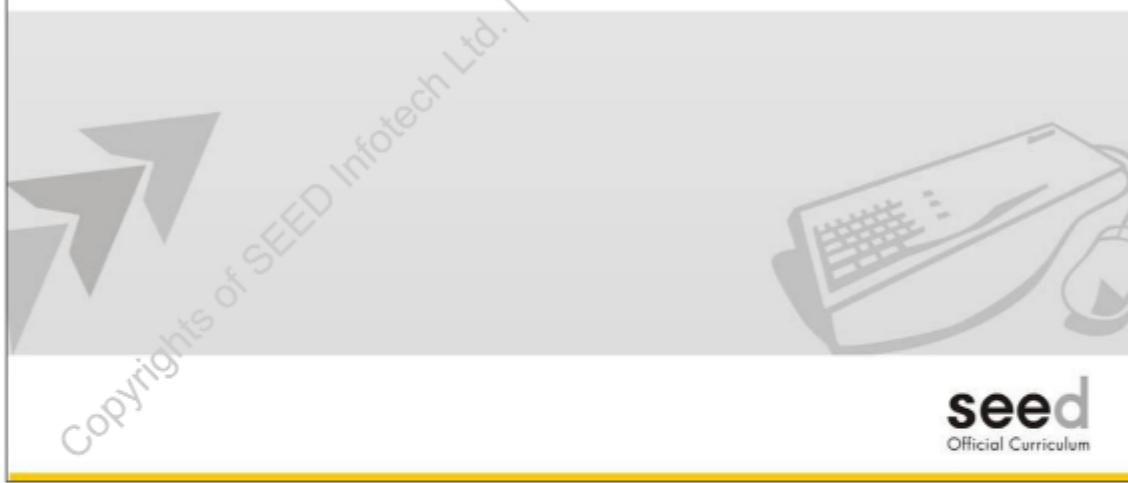
1. The developer has to identify the event which is best-fit for the given problem statement.
2. After identification of the event, the developer has to provide the necessary implementation in the standard method associated with the specific event. For Example, `onClick()` method should be written to handle the button clicks.

Similarly, there are other methods and events which a JavaScript developer can use.

Method	Description
onClick	On button click
onChange	On state change of a component. For example, data entry in a text field.
OnLoad	On page load
onFocus	When a component gains focus for data entry
onBlur	When a component loses focus
onMouseOver	When a mouse cursor hovers over the component.

Chapter - 5

XML in Java



XML is eXtensible Markup Language. Unlike HTML tags, which are primarily used to control the display and appearance of data, XML tags are used to define the structure and data types of the data itself. Before using the XML file for any kind of data processing, the files have to be verified for correctness and validity. XML Parsers (SAX and DOM) provide the functionality for parsing XML files.

This chapter covers XML, its uses, XML components and their use, and XML parsers (SAX and DOM).

Objectives

At the end of this chapter you will be able to:

- Define XML and list its various usages.
- List different components of XML and their usage.
- Describe Simple API for XML (SAX) and Document Object Model (DOM).
- Demonstrate the use of SAX and DOM parsers.

What is XML?

- XML is Extensible Markup Language
 - A tag-based meta language
 - Designed for structured data representation
 - Represents data hierarchically (in a tree)
 - Provides context to data (makes it meaningful)
 - An open W3C standard
 - A subset of Standard Generalized markup Language (SGML).

XML is eXtensible Markup Language. XML and HTML have different goals: HTML is designed to display data and is focused on how data looks, while XML is designed to describe data and focuses on what data is. Like HTML, XML is also a passive component. While XML tags can be used to describe the structure of an item such as a purchase order, it does not contain any code which can be used to send that purchase order, process it, or ensure that it is filled. Other component must have the code to do these operations using the XML data files.

XML uses a set of tags to define elements of data. Each element encapsulates a piece of data that may be very simple or very complex. A developer can define an unlimited set of XML tags. For example, XML tags can be defined to declare data from a purchase order, such as the price, tax, shipping address, billing address, and so on.

The XML specification describes the XML data format and grammar and also specifies two-tier client architecture for handling XML data. The first tier is the XML Processor (also known as the XML parser). The parser ensures that the presumed XML data is well-formed (has the correct structure and syntax), and may

be used to check the validity of the user's data structure. The parser must comply with the XML specification, and pass the content and structure of the XML data to a second tier application (the XML Application) in a prescribed manner.

As the XML file is made up of tags and sub-tags that are interlinked, the parsers may create an in-memory tree structure representing the XML contents. This tree structure helps faster data searches and manipulations.

XML rules are governed by an open source body World Wide Web Consortium (W3C), which regulates the structure and direction for XML technology development.

XML is a relatively a new markup language, but it is a subset of, and is based upon a mature markup language called Standard Generalized Markup Language (SGML) that provides a superset for almost all the markup languages.

Characteristics of XML

Following are the characteristics of XML:

Extensibility

XML is a highly extensible and open ended markup language. XML provides a basic syntax but does not define the actual tags; the tag set can be extended by anyone for their own purpose.

Simplicity

XML is a very simple, hierarchical data representation.

Separation of semantics and representation

XML is all about the description of data and does not say anything about its presentation.

XML works with internet

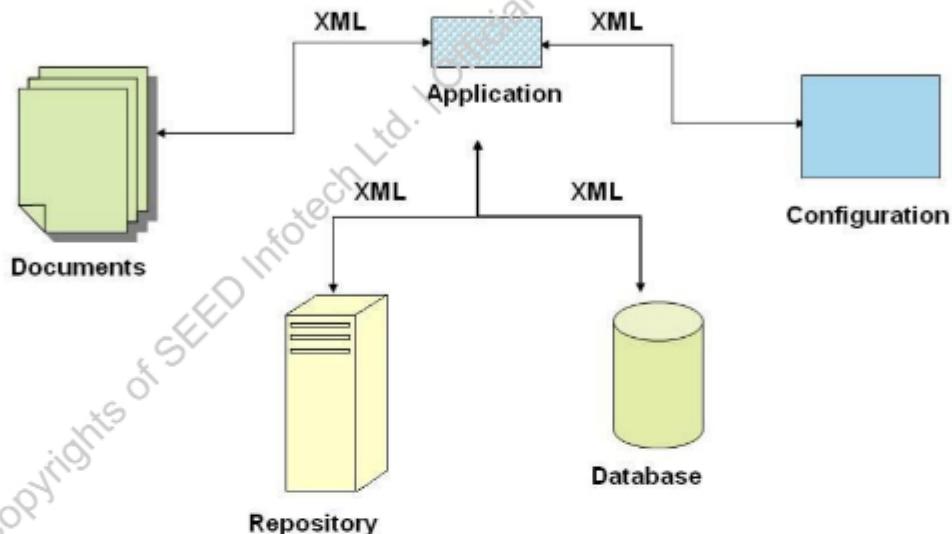
XML is based upon a simple text format. XML document is divided into two parts as tags and the data. XML also uses existing Internet protocols, software, and specifications wherever possible, for easier data processing and transmission.



Why is XML used? Give a real life application of XML.

Benefits of XML

- XML is a “Open standard” data specification



XML is a platform independent and vendor independent way of data representation, making it possible to be used anywhere.

XML Usage

XML is an extremely flexible way to transfer data. The following are examples where XML can be used:

- An ordinary document
- A structured record, such as an appointment record or purchase order
- Internet/intranet web applications that move data
- An object with data, such as the persistent format of an object or ActiveX control
- A data record, such as the result set of a query
- Graphical presentation, such as an application's user interface
- Links between information and people on the web
- Code comments, which can be documented with XML
- Discovery documents used to locate available XML Web services

Advantages of XML

- XML formats are text-based, making them more readable, easier to document, and sometimes easier to debug.
- XML documents can use much of the infrastructure already built for HTML, including the HTTP protocol and some browsers. HTTP allows XML to be transferred across firewalls.
- XML parsing is well defined and widely implemented, making it possible to retrieve information from XML documents in a variety of environments.
- Applications can rely on XML parsers to do some structural validation, as well as data type checking (when schemas are used).
- XML is built on a Unicode foundation, making it easier to create internationalized documents.
- It allows the data representation in a heterogeneous environment, enabling cross platform data integration and interoperability.

XML and Structured data

- Traditional representation of Purchase Order Record.

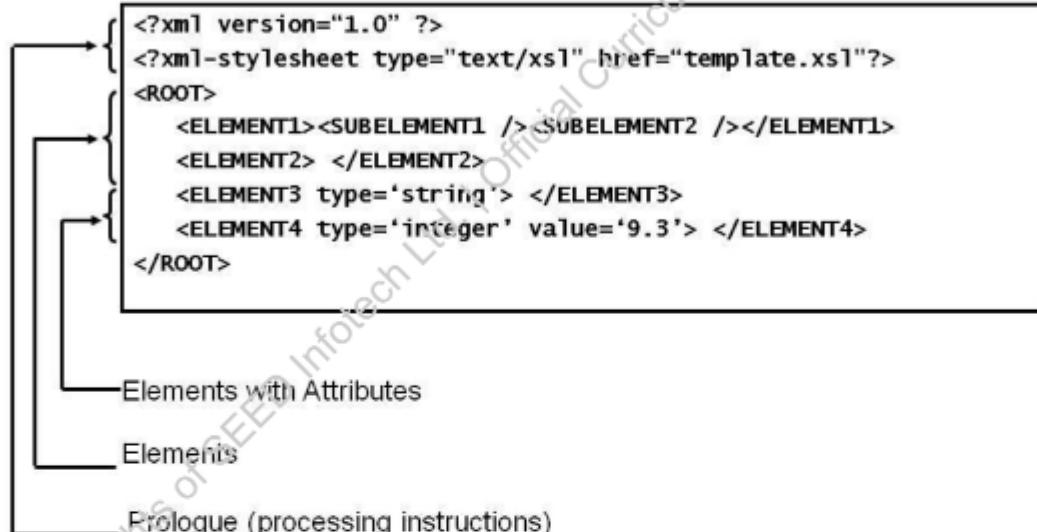
```
"PO-1234","CUST001","X9876","5","14.98"
```

- XML representation

```
<PURCHASE_ORDER>
    <PO_NUM> PO-1234 </PO_NUM>
    <CUST_ID> CUST001 </CUST_ID>
    <ITEM_NUM> X9876 </ITEM_NUM>
    <QUANTITY> 5 </QUANTITY>
    <PRICE> 14.98 </PRICE>
</PURCHASE_ORDER>
```

By now, it is known that XML is a specification to represent structured data. The data can be represented by using user-defined tags. Consider purchase order data consisting of Purchase order number, Customer id, Item no, Quantity, and Price. This data can be represented with user-defined tags. It is shown in the example above.

Components of an XML Document



XML documents are broken into several distinct components, each of which can be created and modified separately, giving XML its strength of focused and independent document processes. XML documents are modular and can be extremely distributed, referencing components on various file systems, but retaining the logic. The main parts of an XML document metadata are: the XML SGML declaration, which is fixed and generally understood by XML-compliant tools; the document type definition, which is the foundation of and used in valid XML documents, and the document instance, in which the content of the document is described by markup.

The XML document content has following components:

Elements

Elements are a discrete unit of content within a marked up document. An element begins and ends with the mark-up tag that delimits and defines it. They can contain sub-elements.

Attributes

Attributes are special values that can be assigned to an element. Attributes provide a means for specifying additional information about the element being marked up. They occur as name-value pairs (`name="value"`) .

The value of an attribute should be always quoted. Writing attributes in user defined tags should be avoided. A "Well Formed" XML document is a document that conforms to the XML syntax. The rules are defined in Document Type Definition (DTD).

XML Prologue

All XML documents should have an XML prologue that states that the document is an XML document. The XML prologue must be the first thing in the document.

Document Type Definition (DTD)

- A "Valid" XML document is a "Well Formed" XML document, which also conforms to the rules of a Document Type Definition (DTD).
- DTD defines the structural specifications of an XML file.
 - All elements and their attributes
 - Relationship between elements
- DTD can be defined
 - Inline in an XML document –Internal DTD.
 - In an external file with .dtd extension and then linked with XML file-External DTD.

XML is a language defined by the developer for performing specific operations. To ensure that the document conforms to the rules there has to be a dictionary containing the rules for constructing an XML document. This dictionary is called as the Document Type Definition (DTD). Any XML which follows the generic markup language rules is called as a well-formed XML. The XML file that follows the grammatical rules as well as data structure rules is called as a valid XML file.

A DTD can be defined either internally or externally. XML files which contain the grammatical rules within itself are said to be having internal DTD. When the rules are encapsulated in a separate file at a central location it is known as an external DTD.

External DOCTYPE declaration

The external DOCTYPE declaration consists of the usual keywords and root element name, followed by another keyword denoting the source of the external DTD, and then followed by the location of that DTD.

The keyword for source can be either SYSTEM or PUBLIC.

For the keyword SYSTEM, the parser has to find the DTD given in the URL alone - a URL (Uniform Resource Locator) directly and explicitly locates the DTD. In that case, what follows "SYSTEM" is a URL naming the DTD file.

```
<!DOCTYPE web-app SYSTEM "../path/to/dtd/file">
```

The PUBLIC keyword allows a non-specific reference to a DTD via a URI, even via a secondary (specific) URI.

The PUBLIC keyword is provided for well-known vocabularies.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Rules for Writing a Well-Formed XML

- With XML, it is illegal to omit the closing tag.
- In XML all elements must have a closing tag.
- XML tags are case sensitive. Any tag which describes the same data should be available in the same case with same spellings. Opening and closing tags must therefore be written with the same case.
- Attribute values in XML always have to be enclosed within a pair of quote marks. Attribute and its value cannot exist without each other.
- An XML document can have one and only one root element.
- Nested elements have to be closed before closing the parent tag.
- A "Valid" XML document is a "Well Formed" XML document, which also conforms to the rules of a Document Type Definition (DTD).

Accessing XML Documents

- Component required to check the correctness of XML file is XML Parser.
 - SAX (Simple API for XML)
 - DOM (Document Object Model)

API's	SAX	DOM
Type of access	Sequential Access	Random Access
Based on	Event driven mechanism	Principle of Tree data Structure
Memory	Less	More

The primary use of an XML file in any application is to define the configurations or to store the data. Configuration related information (data) or the data required for applications need to be validated first, before it can be used within the application. The application in question is created for business logic, and so it should not be burdened with the task of XML checking. Therefore a component is needed whose task will be to check the correctness of the XML file.

Java provides two approaches for checking the correctness, namely SAX (Simple API for XML) and DOM (Document Object Model). These are also known as XML Parsing approaches.

Simple API for XML (SAX)

SAX is a set of API which helps read the XML file sequentially. In this approach, the document is read tag-by-tag and compared with the definitions (DTD, Schema) if required. If there are any discrepancies in the construct of the XML structure, these are caught at the time of accessing the documents. SAX APIs work on event driven mechanism and call back methods. Each event (tag related) is trapped and associated with one call-back method, which contains the code for executions.

Since, the files are read sequentially, the execution of the code is faster as the callback methods are triggered as soon as the tags are encountered.

Document Object Model (DOM)

DOM works on the principles of tree structures. Here the whole file is read (using SAX) and converted into a tree structure, which is then used to traverse for accessing different tags within the file. The main advantage of this approach is that the tags can be accessed randomly because all the tags are available in the memory at any given point of time (after the files has been read into memory), unlike SAX, where the access is always sequential. There are no rules for deciding when to use SAX or DOM, but the conventionally, SAX is used when file sizes are huge, as converting the entire file into memory would require more memory resources.



What is the difference between SAX and DOM?

What is Schema

- XML Schema is an XML-based alternative to DTD.
- An XML schema describes the structure of an XML document.
- The XML Schema language is also referred to as XML Schema Definition (XSD).
- Schema specifies the structure of XML document.
- A Schema is useful for validating XML document content.

XML Schema is an XML-based alternative to DTD. An XML schema describes the structure of an XML document in a XML Schema Definition (.xsd) file.

The purpose of an XML Schema is to define the legal building blocks of an XML document, just like a DTD.

An XML schema defines the following:

- elements that can appear in a document
- attributes that can appear in a document
- which elements are child elements
- order of child elements
- number of child elements
- whether an element is empty or can include text
- data types for elements and attributes
- default and fixed values for elements and attributes

Same Job done by DTD, than why not DTD?

- It's Not XML

XML document is written in one language and the grammar of that document is written using another language (DTD) which is bad, inconsistent.

DTD support a very limited capability for specifying data types.

XML Schemas are used in most web applications as a replacement for DTDs. Here are some reasons:

- They are extensible to future additions
- They are richer and more powerful than DTDs
- They are written in XML
- They support data types
- They support namespaces

One of the greatest strength of XML Schemas is the support for data types. It is also easier to do the following:

- describe allowable document content
- validate the correctness of data
- work with data from a database
- define data facets (restrictions on data)
- define data patterns (data formats)
- convert data between different data types

XML Schema benefits

XML schema is created using XML, which allows the developer to create complex data types to define the data inside the XML. Some advantages of using XML Schema for application configuration are:

- No need to learn a new language
- A simple text-based XML editor can be used to edit Schema files
- A simple text-based XML parser can be used to parse Schema files
- The Schema can be manipulated with the XML Document Object Model (DOM)
- The Schema can be transformed with eXtensible Stylesheet Language Transformation (XSLT).

Secure Data Communication

When sending data from a sender to a receiver, it is mandatory for both to be in sync for understanding the data. With XML Schemas, the sender has to describe the data in a way that the receiver understands.

A date like "03-10-2011" will, in some countries, be interpreted as 3 October and in other countries as 10 March. However, an XML element with a data type like this:

```
<date type="date">2011-10-03</date>
```

ensures a mutual understanding of the content, because the XML data type "date" requires the format "YYYY-MM-DD".

Well-Formed is not enough

Even if documents are well-formed they can still contain errors, and those errors can have serious consequences, as the DTD cannot help match the data with data-types. Hence, XML Schemas are used.

XML Schema

- Following elements make XML Schema
 - Schema
 - ElementType
 - Element
 - AttributeType
 - Attribute
 - Datatype
 - Group

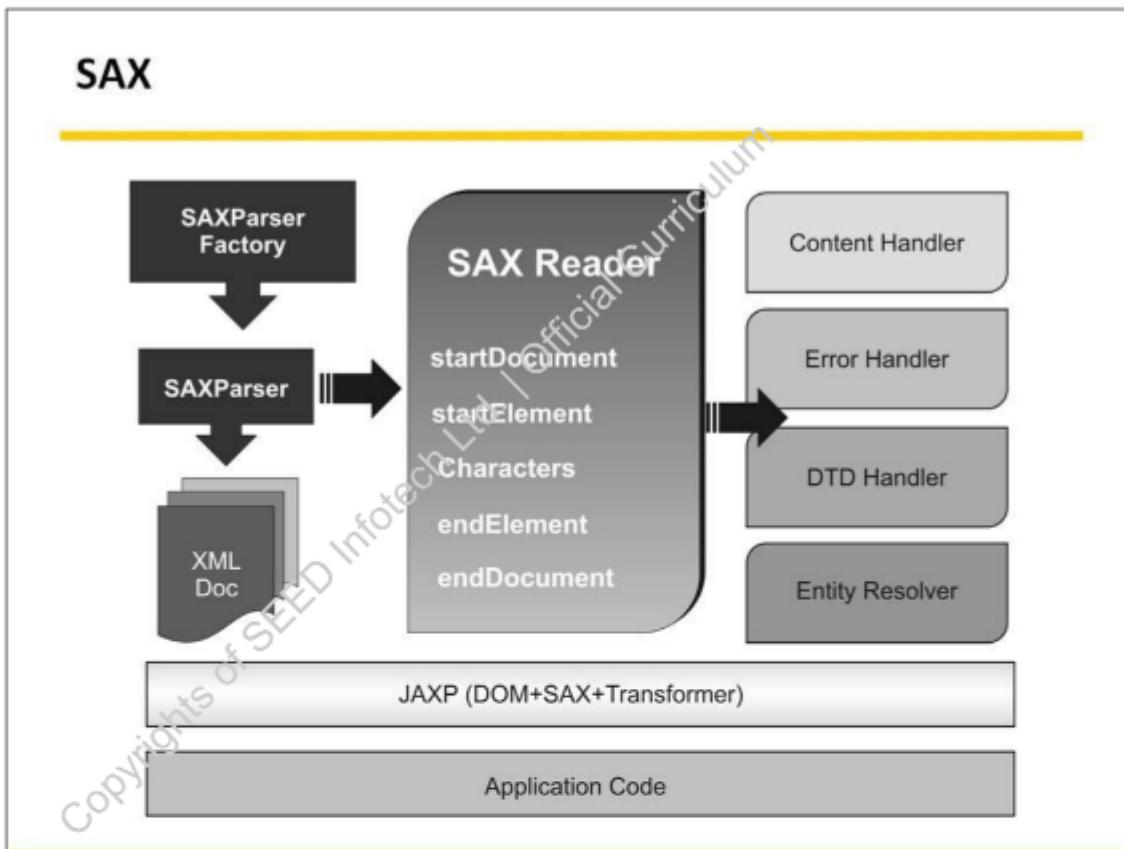
An XML Schema file (XML Schema Definition – XSD) is made up of components listed above. Some examples of XML Schema definitions are:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">
<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
</xs:element>
</xs:schema>
```

XML schema reference

```
<?xml version="1.0"?>
<note
  xmlns="http://www.w3schools.com"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3schools.com note.xsd">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```



A parser can be thought of as a pluggable device with built-in or pre-defined basic functionalities provided and having a mechanism for attaching different implementation for different tasks as per requirement. The major advantage of this approach is that a developer need not provide code for tasks which he/she does not want to handle.

The developers need to override only those methods, in which they are interested; the default implementation is used for the rest of the methods.

For example, if customized tag handling mechanism is to be provided, only the content handler will have to be overridden, the other three handlers need not be disturbed, their default implementation will be considered by the parsing component.

XML Parsing using SAX

- An XML file can be parsed using a SAX parser.

```
try
{
    SAXParserFactory factory =
    SAXParserFactory.newInstance();
    factory.setValidating(true);
    SAXParser parser = factory.newSAXParser();
    parser.parse(new File(args[0]), new MyHandler());
}
catch(Exception e)
{
    e.printStackTrace();
}
```

As discussed earlier, an XML file can be parsed using an XMLParser. There are two possibilities available for parsing an XML document. I) Simple API for XML Processing (SAX) II) Document Object Model (DOM).

The example illustrates the code for building a SAX Parser.

```
public class SAXRead {
    static public void main(String[] arg) {
        String filename = null;

        if (arg.length == 1) {
            filename = arg[0];
        } else {
            usage();
        }
        // Create a new factory that will create the parser

        SAXParserFactory spf = SAXParserFactory.newInstance
```

```
() ;  
        // Create the XMLReader to be used to parse the doc  
        // ument.  
        XMLReader reader = null;  
        try {  
            SAXParser parser = spf.newSAXParser();  
            reader = parser.getXMLReader();  
        } catch (Exception e) {  
            System.err.println(e);  
            System.exit(1);  
        }  
        // Specify the content handler.  
        reader.setContentHandler(new MyContentHandler());  
        // Use the XMLReader to parse the entire file.  
        try {  
            InputSource is = new InputSource(filename);  
            reader.parse(is);  
        } catch (SAXException e) {  
            System.exit(1);  
        } catch (IOException e) {  
            System.err.println(e);  
            System.exit(1);  
        }  
    }  
}  
class MyContentHandler implements ContentHandler {  
    public void startDocument() {  
        System.out.println("-" + "----"  
        Document parse started");  
    }  
  
    public void endDocument() {  
        System.out.println("-" + "----Document parse ended");  
    }  
    public void startElement(String namespaceURI, String  
    localName,  
        String qName, Attributes atts) {
```

```

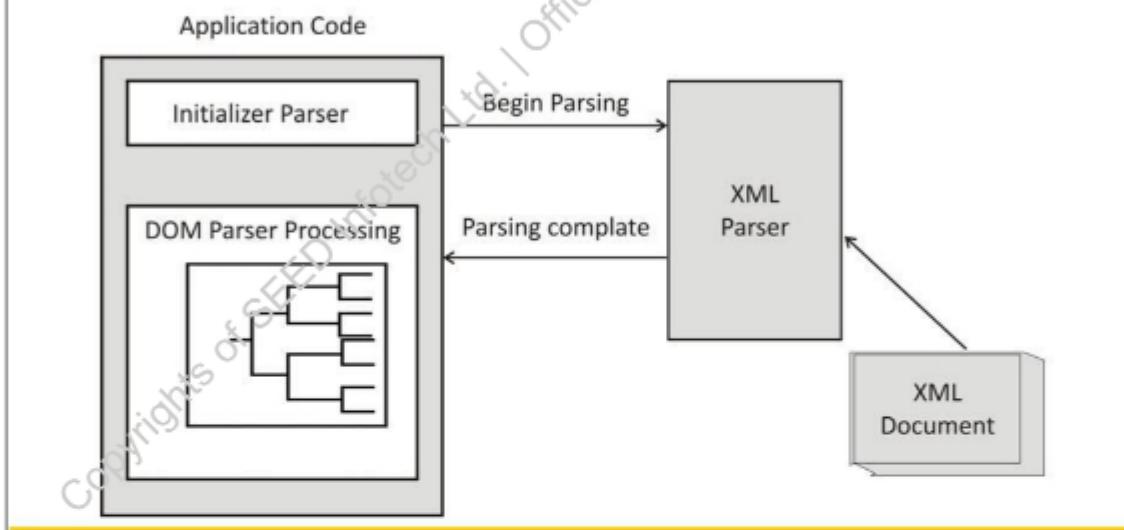
        System.out.println("-" + "----"
Opening tag of an element");
        System.out.println("          Namespace: " + namespace
URI);
        System.out.println("          Local name: " + localName
);
        System.out.println("    Qualified name: " + qName);
for (int i = 0; i < atts.getLength(); i++) {
        System.out.println("          Attribute: " + atts.get
tQName(i) + "=" +
+ atts.getValue(i) + "\"");
    }
}
public void endElement(String namespaceURI, String loc
alName, String qName) {
    System.out.println("-" + "----"
Closing tag of an element");
    System.out.println("          Namespace: " + namespace
URI);
    System.out.println("          Local name: " + localName
);
    System.out.println("    Qualified name: " + qName);
}
public void characters(char[] ch, int start, int leng
th) {
    System.out.println("-" + "----Character data");
    showCharacters(ch, start, length);
}
public void showCharacters(char[] ch, int start, int
length) {
    System.out.print("         \"");
    for (int i = start; i < start + length; i++)
        switch (ch[i]) {
            case '\n':
                System.out.print("\\\\n");
                break;
            case '\r':

```

```
        System.out.print("\r");
        break;
    case '\t':
        System.out.print("\t");
        break;
    default:
        System.out.print(ch[i]);
        break;
    }
    System.out.println("\n");
}
}
```

DOM Parser

- The structure of a DOM parser can be identified using the following diagram.



The responsibility of a DOM parser is to read the XML document specified and convert that into a tree structure suitable for traversal. Internally, a DOM parser takes help from a SAX Parser to read the file and then compares the XML against the DTD or the Schema so that relationships between parent-child tags can be set up and the tag tree is built in the memory.

Parsing the XML using DOM gives access to the root element of the tree, using which the entire tree can be traversed, searched and indexed if required.

XML Parsing using DOM

- XML Document can be verified using the DOM Parser

```
try
{
    DocumentBuilderFactory factory =
    DocumentBuilderFactory.newInstance();
    DocumentBuilder domParser =
    factory.newDocumentBuilder();
    Document doc = domParser.parse(args[0]);
    print(doc);
}
catch(Exception e){
    e.printStackTrace();
```

The code demonstrates usage of a DOM parser to validate an XML file.

```
public class Main {
    public static void main(String[] argv) throws Exception{
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        factory.setValidating(true); // checks for valid
        XML, by default false

        factory.setExpandEntityReferences(false);

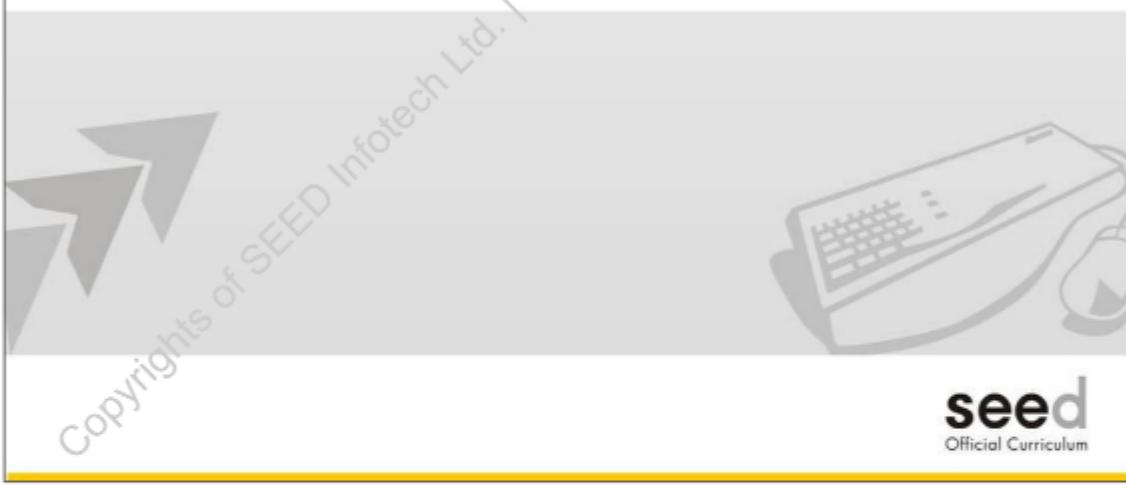
        Document doc = factory.newDocumentBuilder().parse(new File("filename"));

        visit(doc, 0); //recursively iterating over the
        document
    }
}
```

```
public static void visit(Node node, int level) {  
    NodeList list = node.getChildNodes();  
    for (int i = 0; i < list.getLength(); i++) {  
        Node childNode = list.item(i);  
        visit(childNode, level + 1);  
    }  
}
```

Chapter - 6

Servlets



A web application is an application that runs on the web and is responsive to user's requirements. To make an application responsive, there needs to be a component on the server that accepts the user requests and applies business logic to it. The result generated needs to be sent back to the user. This entire chain of events is handled by the web container with the help of specialized Java classes called as Servlets.

This chapter covers servlet life cycle, explains how to create applications using simple Servlets, how to handle initialization parameters, retrieve data from users' requests and use multiple servlet components.

Objectives

At the end of this chapter you will be able to:

- List characteristics of a web application component.
- Describe the life cycle of a servlet.
- Construct a web application using simple servlet.

- Construct an application to handle initialization parameters using `ServletContext` and `ServletConfig` objects.
- Construct an application to handle user's requests and retrieve data from it.
- Construct an application with multiple servlet components.
- Construct an application with response redirection.

Copyrights of SEED Infotech Ltd. | Official Curriculum

Servlet Introduction

- What is a Servlet?
 - Servlets are server-side components that extend the capabilities of the server and provide a powerful mechanism for developing server-side programs.
- Features of Servlets:
 - Efficient
 - Persistent
 - Portable
 - Robust
 - Extensible
 - Secure
 - Scalable

Java Servlets are web components that allow application logic to be embedded in the HTTP request-response process and provide a way by which a web application can be built. Servlets are efficient, portable, robust, extensible, secure, and have a widespread acceptance. Servlets solve many of the common problems faced when using CGI.

Efficient

A servlet's initialization code is executed only the first time the web server loads it. After the server is loaded, handling new requests is only a matter of calling the service method. This is a much more efficient technique than loading a completely new executable with every request.

Persistent

Servlets can maintain state between requests. When a servlet is loaded, it stays resident in memory while serving incoming requests. A simple example of this would be a Vector that holds a list of categories used in an online catalog. When the servlet is initialized, it queries the database table which holds a list of

categories and stores these categories in a vector. As a request for a list of categories comes from the client, instead of querying the database again, advantage of the persistent characteristics of Servlets can be taken.

Portable

Servlets are developed using Java, so they are portable. This enables servlets to be moved to a new operating system without changing the source. Code that was compiled on a Windows NT platform can be moved to a Solaris box without making any changes.

Robust

Because servlets are developed with access to the entire JDK, they are very powerful and robust solutions. Java provides a very well defined exception hierarchy for error handling. It has a garbage collector to prevent problems with memory leaks. In addition, it includes a very large class library that includes network support, file support, database access, distributed object components, security, and many other classes, reducing the possibility of errors in the code.

Extensible

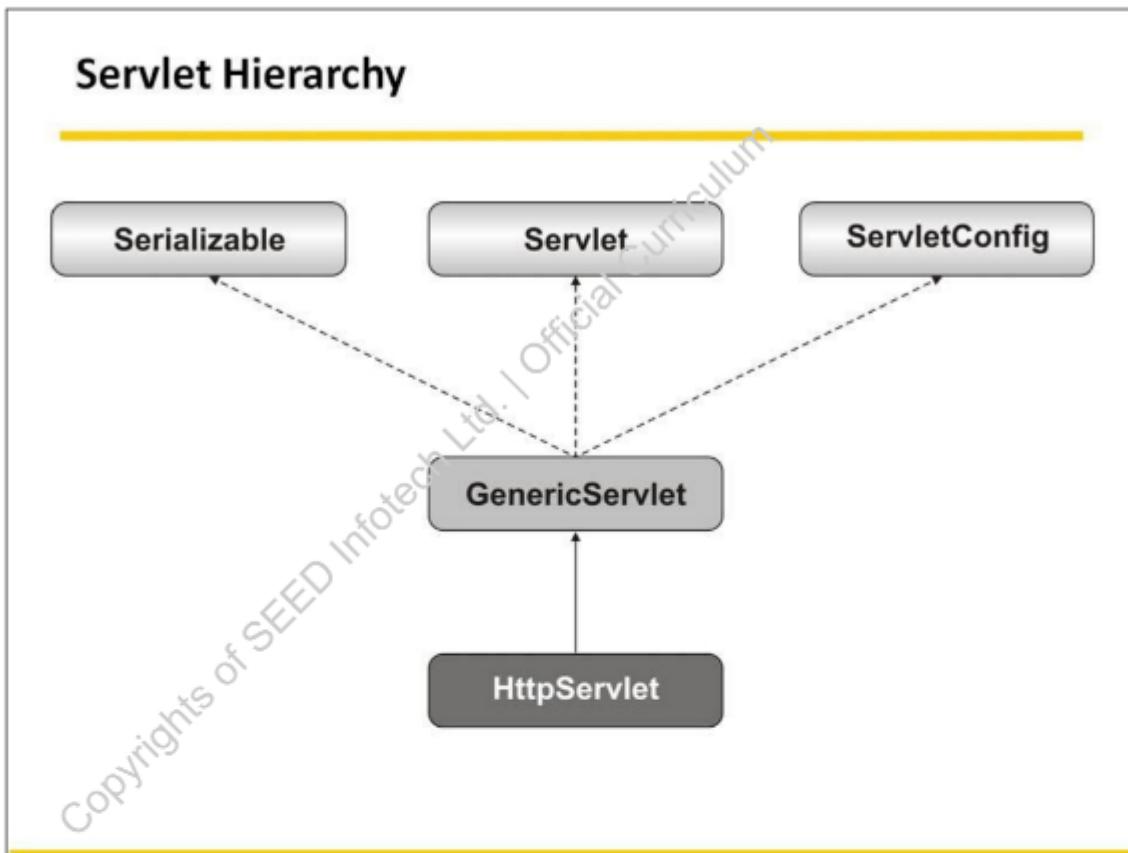
Being developed in an object-oriented language like Java is, servlets can be extended and polymorphed into new objects as per requirement. A good example is an online catalog. Suppose the same catalog search tool is to be displayed at the top of every dynamic page in the web site, the same code need not be added to each of the servlets. So, a base servlet that builds and initializes the search tool can be implemented and then extended to display transaction-specific responses.

Secure

Servlets run on the server-side inheriting the security provided by the web server. Servlets can also take advantage of the Java Security Manager.

Widespread Acceptance

Servlets are widely accepted because of the advantages gained from using Java. Vendors are providing servlet support in two main forms. The first form is servers that have built-in support for servlets, and the second is by using third-party add-on.



Any servlet that needs to utilize services provided by the server has to connect with the hooks supplied by the associated API (Servlet API). These hooks points are methods published through interfaces. The custom component (User defined servlet) has to implements these methods to gain access to services provided by the server.

To ease the application development efforts, Servlet API provides an abstract class `GenericServlet` which has abstract implementation provided for the mandatory methods present in `Servlet` and `ServletConfig` interfaces.

One concrete class `HttpServlet` is also provided which provides default implementations for the mandatory methods. This class is optimized to be worked with the HTTP. Generally, a user defined servlet will be extending from `HttpServlet` class.



App Design

GenericServlet should be used, when a protocol independent component needs to be created.

Servlet interface

This interface specifies the contract between the web container and a servlet. In case of Servlet API, the `javax.servlet.Servlet` is the interface that containers use to reference servlets.

When a servlet is written, this interface must be directly or indirectly implemented. The interface may be implemented indirectly by extending either `javax.servlet.GenericServlet` or `javax.servlet.http.HttpServlet`.

GenericServlet class

This class provides a basic implementation of the `Servlet` interface. This is an abstract class, and all subclasses have to implement the `service()` method.

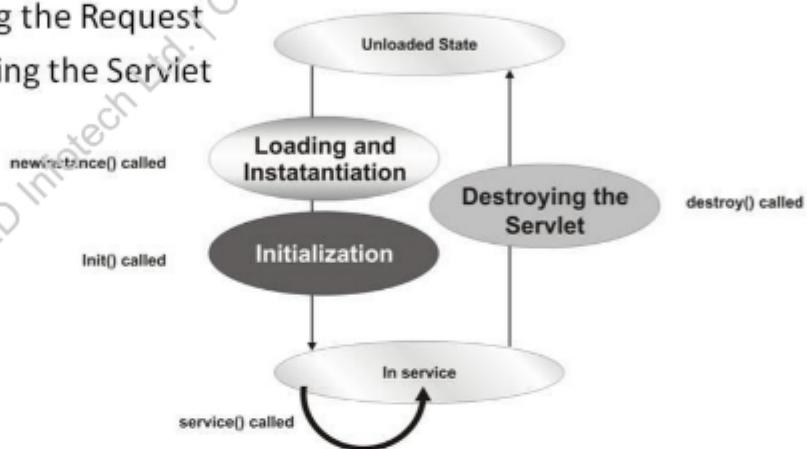
This class also implements the `ServletConfig` interface. This allows the servlet developer to call the `ServletConfig` methods directly without having to obtain the `ServletConfig` object.

HttpServlet class

This class extends `GenericServlet`, and provides an HTTP specific implementation of the `Servlet` interface. Most of the customized servlets extend this class.

Servlet Life Cycle

- Servlet life cycle is divided into 4 parts:
 1. Loading and Instantiation
 2. Initialization
 3. Servicing the Request
 4. Destroying the Servlet



The Servlet life cycle is as shown above. For a request from the client, if the servlet is not already there in the memory, then it will be first initialized and loaded in the memory. The subsequent request to same servlet uses the same instance of the servlet. Once the web application is undeployed or the web-application server is shut down the instance is destroyed. Once the servlet is instantiated in memory, it is always ready to respond to a request.

Loading and Instantiation

The container loads the servlet during startup or when first request is made. Servlet loading depends on `<load-on-startup>` of `web.xml`.

If `<load-on-startup>` value is positive, then the servlet loads along with the container.

- Otherwise it loads on the first request.
- After loading the servlet, the container creates the instances of the servlet by calling `newInstance()` method on it.

Initialization

- After creating an instance, the container calls `init()` method.
- `init()` must be called by the servlet container before the servlet can service any request.
- The `init()` method is called only once throughout the life cycle of the servlet.
- If overridden, must call `super.init(config)` first.

```
public void init( ServletConfig config ){}  
public void init(){}
```

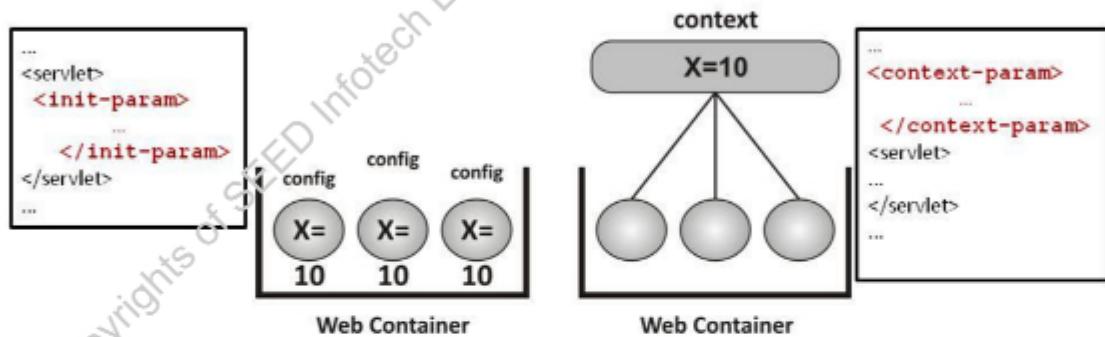
Destroying the Servlet

- `destroy()` is called at servlet unloading.
- Called by the servlet container to indicate that the servlet is being taken out of service.
- Frees the resources and does the cleanup tasks.
- Makes the state persistent.

```
public void destroy()
```

Servlet Initialization

- To initialize parameters in servlet application there are two ways:
 1. ServletConfig Interface
 2. ServletContext Interface



init() Method

Once the servlet has been instantiated, the web container calls the `init()` method. The purpose of this method is to allow a servlet to perform any initialization required before being invoked against HTTP requests.

The servlet specification guarantees that the `init()` method will be called exactly once on any instance of the servlet, and will be allowed to complete before any request is passed to the servlet, provided that it does not throw a `ServletException`.

Some of the typical tasks that can be implemented in the `init` method are:

- Read initialization parameters using `ServletConfig` object.
- Initialize one time activities such as registering a database driver, a connection pool or a logging service using the `ServletContext` object.

```
<web-app>
<servlet>
```

```
...
<init-param>
    <param-name>varX</param-name>
        <param-value>10</param-value>
</init-param>
...
</web-app>
```

```
int var= getServletConfig().getInitParameter("varX");
//retrieve data from the web.xml configuration
```

ServletContext interface

The ServletContext object is contained within the ServletConfig object, which the web server provides the servlet when the servlet is initialized. ServletContext allows servlets in an application to share data. e.g. A Servlet can log events, obtain URL references to resources.

```
<web-app>
    <context-param>
        <param-name>varX</param-name>
            <param-value>10</param-value>
    </context-param>
    ...
</web-app>
```

```
int var=
getServletConfig().getServletContext().getInitParameter
("varX"); //retrieve data from the servlet context
object.
```

Servlet Execution

- Once initialized, the servlet container identifies the method corresponding to the HTTP method (request method)
 - doGet – HTTP get method
 - doPost – HTTP post method
- The request sent by the browser is encapsulated into an HttpServletRequest object and the corresponding response is embedded as HttpServletResponse object.

Once the servlet object is instantiated, the container needs to execute the processing logic available in the service method. Service is a generic method used to encapsulate the servlet execution, but the behavior of the servlet may be different for different HTML requirement.

For HTTP Get method (used for retrieving data from server side) the servlet needs to react differently as compared to the POST (used for transferring data to server side) method. There needs to be a mechanism in the servlet which can differentiate the HTTP method and execute the corresponding piece of code.

- Servlet API provides two methods belonging to HttpServlet doGet() and doPost() which correspond to the HTTP GET and POST methods.
- The container identifies the HTTP method embedded in the HTTP request and invokes the matching method.
- The methods may need user input to start the processing. This user input is provided to these methods in form of a parameter which encapsulates the HTTP request.

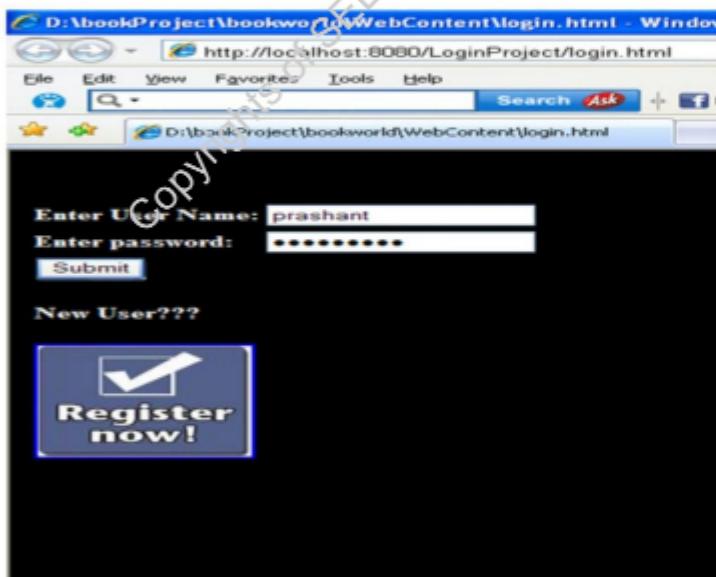
- Similarly, the output generated by the servlet may be required to be posted to either the client or another component. This data needs to be encapsulated into a single object compatible with the HTTP response object. The container provides a corresponding Servlet API object as part of the parameter list to the methods.

The client UI (HTML Form) and the processing logic (servlet) are linked with each other using the <action> tag of the HTML form, the value for this tag matches the URL-pattern defined in the configuration. The developer obtains the user input by using the Servlet API method `getParameter()`, which takes the field name as the parameter and returns the user input as return value.

```
public class GreetingServlet extends HttpServlet
{
    public void init() throws ServletException
    {
        System.out.println("\n\n");
        System.out.println("init() executed");
    }
    public void service(HttpServletRequest req,
    HttpServletResponse res) throws ServletException,
    IOException
    {
        System.out.println("service() method executed");
        If(req.getMethod().equals("post"))
        {
            doPost(req, res);
        }
    }

    public void destroy()
    {
        System.out.println("destroy() executed");
    }
    public void doPost (HttpServletRequest request,
    HttpServletResponse response) throws ServletException,
    IOException
    {
```

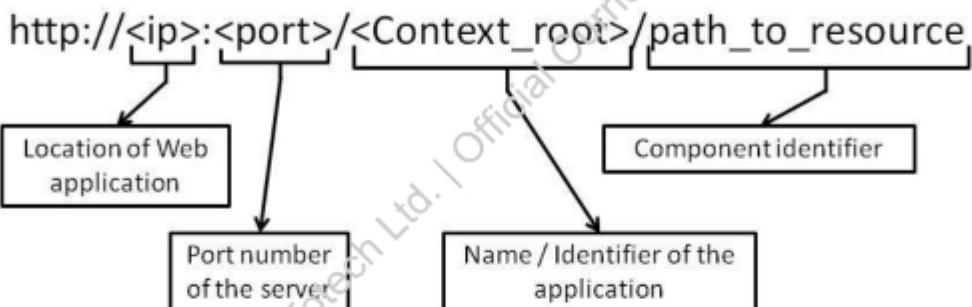
```
System.out.println("In doPost()");
PrintWriter out = response.getWriter();
out.println("<html>");
out.println("<body>");
out.println("<p> Hi, &nbsp; Welcome to Seed </p>");
out.println("<p> Thanks for accessing the
website</p>");
out.println("<p> Thanks and Regards, </p>");
out.println("</body>");
out.println("</html>");
out.close();
}
```



Servlet Code accepts the html values as follows:

```
String name=request.getParameter("username");
String pass=request.getParameter("password");
```

Application Execution



The Servlet container may contain more than one application within it. The URL sent by the browser is used to identify the target application and the individual component within it. The identified component is then executed by the container.

Configuration

- For an application to be identified as a web application, it is mandatory for the configuration to be present.
 - It can be present explicitly (web.xml)
 - It can be present implicitly (annotations). The data required for the component identification needs to be maintained at a central location.
 - Useful for storing multiple component data in a single artifact.
 - This single artifact is known as web.xml

An application which can be contained within a web container is called a web application, but for an application to qualify as a web application it needs to have certain characteristics and structure (refer appendix). Containers normally verify the web application by the existence of a specific configuration file.

A container may have multiple applications embedded within itself. When a request is received from the client-side, the container needs to identify as to which application needs to be executed. This is achieved by parsing the HTTP request identifier, which contains the name/identifier of the application.

The application may also have multiple components within it that take care of diverse requirement. To perform a specific task upon receipt of a specific request, the container needs to be helped with mapping the incoming request with the application component which can handle it.

This mapping is called as the web application configuration and it can be provided to the container in either of the following two ways – explicitly and implicitly.

Explicitly the configuration can be provided by using the web.xml file which contains the mapping of the request (URL-pattern) with the application component (servlet-class) by way of a common servlet name (servlet-name)

```
<web-app>
  <servlet>
    <servlet-name>SimpleServlet</servlet-name>
    <servlet-class>pack.GreetingServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>SimpleServlet</servlet-name>
    <url-pattern>/greeting</url-pattern>
  </servlet-mapping>
</web-app>
```

Writing of this configuration file may become a lengthy process, and in case of a complex enterprise application it may even become confusing. To avoid this and ease the process of creating the web.xml, Java EE provides annotation which can be used to define the configuration using the Java class itself. Usage of annotations ensures that the developer does not have to work with the XML file. Adding @WebServlet before the servlet class definition in the Java class takes care of the servlet configurations.

If the developers want to customize and fine tune the configuration, some more annotations are also provided through the Servlet Annotations available with Servlet 3.0 API.



Though available, best practices suggest that Annotation based configurations should be avoided as the manageability of the application is reduced. Changes in configuration are not centralized and force recompilation of code

@WebServlet – annotation

- In Servlet 3.0, web.xml is optional!
- Configuration part managed by annotation called @WebServlet.
- URL-pattern attribute is mandatory.
- It reduces the complexity of web.xml

Version 3.0 of the web deployment descriptor contains a new attribute called metadata-complete on the web-app element. This attribute defines whether the web descriptor is complete, or whether the class files of the web application should be examined for annotations that specify deployment information.

- If the attribute is set to true, the deployment tool must ignore any servlet annotations present in the class files and use only the configuration details mentioned in the descriptor.
- Otherwise, if the value is not specified or is set to false, the container must scan all class files of the application for annotations. This provides a way to enable or disable scanning of the annotation and its processing during the startup of the application.

```
@WebServlet(urlPatterns = { "/WelcomeServlet"})  
public class WelcomeServlet extends HttpServlet {
```



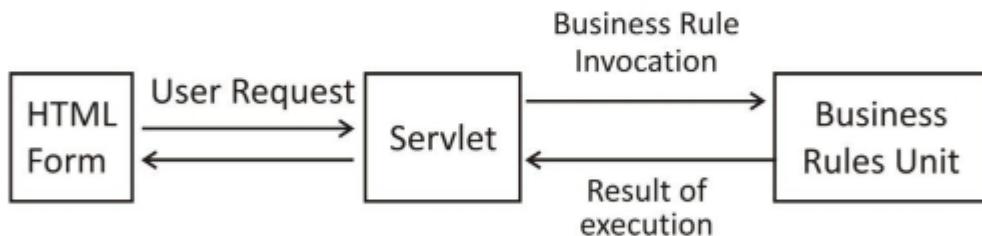
Servlet3

Attributes	Description
name	Name of the servlet - optional
asyncSupported	true/false - Specifies whether the servlet supports asynchronous processing or not.
description	Servlet description - optional
initParams	Array of @WebInitParam, used to pass servlet config parameters - optional
loadOnStartup	Integer value - optional
displayName	Servlet display name - optional
urlPatterns	Array of URL patterns for which the servlet should be invoked - Atleast one URL pattern is required

Servlet Data Retrieval

- Servlet is executed upon receipt of a user request coming through HTTP.
- The incoming request may contain data provided by user.
 - Business rules need to be applied to this data.
- Servlet retrieves the data and passes it to the unit performing business logic.
- Result generated needs to be embedded in the response being sent back to the client.

Standard execution sequence of a web application is shown in the diagram below:



- The user interacts with the system using an HTML form in which he/she may provide some data upon which the business rules are to be applied. For example, salary details on which income tax rules need to be applied.
- This request, through the web server-container path, reaches the servlet where the data embedded in the request needs to be retrieved. The request sent by the user is in form of text. This textual data has to be converted into Java specific data format. This task is managed by the web container when it identifies the component for execution.

- The web container takes the HTTP request sent by the user and encapsulates it into a Java object known as `HttpServletRequest`. This object is provided to the servlet through the parameters of the `doGet` or `doPost` methods. Servlet API provides methods associated with the `HttpServletRequest` to enable the retrieval of the data.
- Similarly, when the result is generated it is generated in Java object format, which is incompatible with textual data formats. To overcome this problem, Servlet API has one more component called as `HttpServletResponse` which holds the data generated by the business rules unit. The web container converts the `HttpServletResponse` object into text based HTML code and pushes it onto the outbound channel which is retrieved by the browser at the client's end.

Request Retrieval and Response Generation

- Web application responds to user's requirement by retrieving the data embedded inside the HTTP request object.
- The data to be shared with the user also needs to be embedded inside the HTTP response object
 - The response could be result of business logic execution or a notification specifying outcome of the process.

```
public void doPost(HttpServletRequest req,  
HttpServletResponse res)  
{  
    PrintWriter out = res.getWriter();  
    String dataElement1 = req.getParameter("<name of  
variable in HTML form>")  
    :  
    out.println("The Result generated is" +  
theResult_or_theException);  
}
```

In the code snippet, the `doPost` method receives the `HttpServletRequest` object generated by the container and makes it available to the developer to manipulate. There is a possibility that more than one data element might be available inside the request object.

The data in request is always in the form of `<key>-<value>` pairs. The name of the component in HTML form becomes the key and data entered by the user is the value. To isolate a specific data item from the collection of data elements, a developer has to provide the name of the HTML form variable to the `getParameter()` method.

Similarly, whatever result the developer wishes to display on the client's browser has to be embedded into the response object by first obtaining the `PrintWriter` based on the response object and then pushing the data into the `out` object. The data that is put in the `PrintWriter` object reaches the client browser.

Some of the methods of `HttpServletRequest` object are as follows:

Method	Description
<code>getAttribute()</code>	Returns value of a named attribute for this request.
<code>getContentLength()</code>	Size of request, if known.
<code>getContentType()</code>	Returns MiME type of the request message body.
<code>getInputStream()</code>	Returns an InputStream for reading binary data from the body of the request message.
<code>getParameterNames()</code>	Returns an array of strings with the names of all parameters.
<code>getParameterValues()</code>	Returns an array of strings for a specific parameter name.
<code>getProtocol()</code>	Returns the protocol and version for the request as a string of the form <protocol>/<major version>.<minor version>.
<code>getReader()</code>	Returns a BufferedReader to get the text from the body of the request message.
<code>getRealPath()</code>	Returns actual path for a specified virtual path.
<code>getRemoteAddr()</code>	IP address of the client machine sending this request.
<code>getRemoteHost()</code>	Host name of the client machine that sent this request.
<code>getServerName()</code>	Name of the host server that received this request.
<code>getServerPort()</code>	Returns the port number used to receive this request.

Servlet Collaboration

- Communication between two servlets is termed as Servlet Collaboration.
- Shared information is passed directly from one servlet to another.
- This process is called 'Request Dispatching'.
- Only request is passed to the next page/servlet, but not values.

It is not possible to perform a complex task using a single programming component. A developer is supposed to break down the problem statement into smaller manageable code structures for better reusability and improved flexibility.

Performing a single atomic unit of a task can be managed by a single servlet component, but in situations where a complex task is made up of multiple execution modules, multiple servlets have to be executed in a sequential order. In other words, the task can be completed only by collaboration between different servlet components.

Usually, servlets that work in collaboration use a common set of data. Hence, when execution control is transferred from one servlet to another, the request object received by the first servlet has to be transferred to the second servlet as well. This means the execution control transfer will have to be achieved using a component that can transfer not only the execution control, but also the request and the response object associated with the invoking servlet.

Summarizing, what a developer needs for achieving collaboration is a component that is capable of dispatching a control sequence along with the request and response objects. The component RequestDispatcher manages this task.

```
RequestDispatcher rd=req.getRequestDispatcher("<path of  
second component");  
rd.forward(req,res);
```

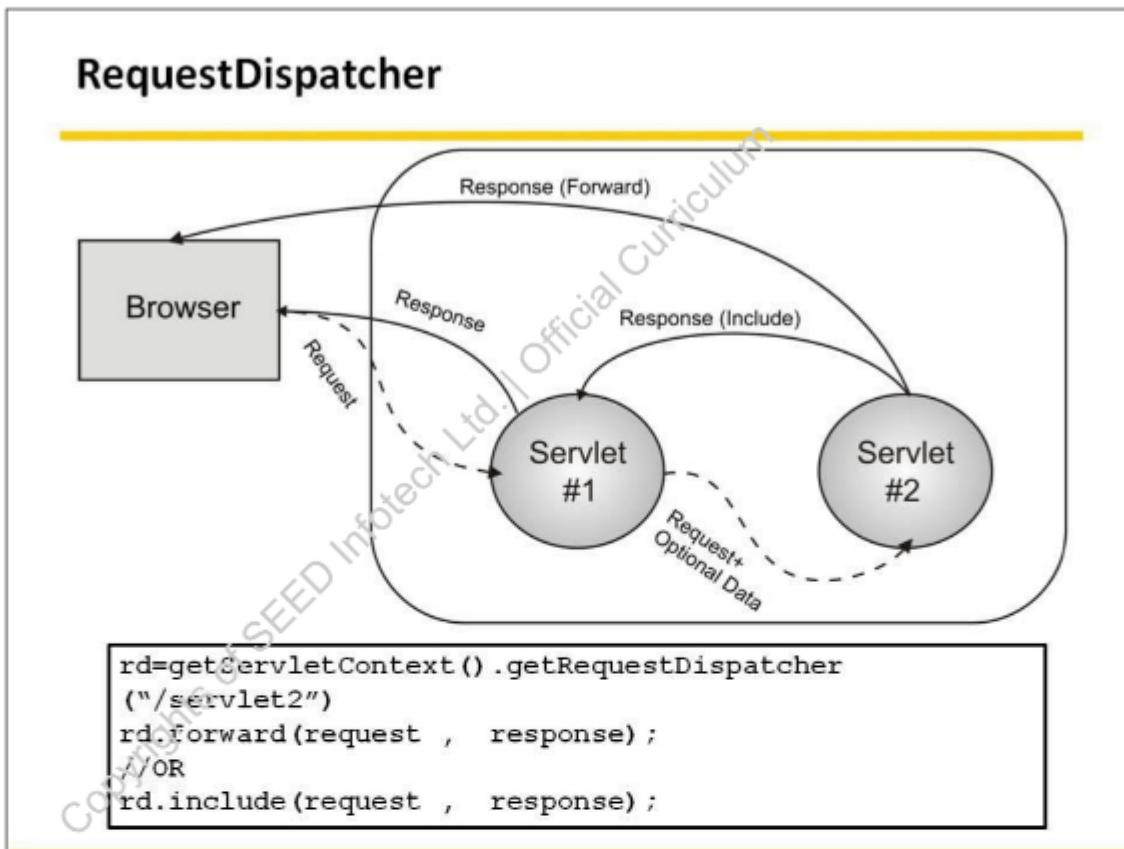
or

```
RequestDispatcher rd=req.getRequestDispatcher("<path of  
second component");  
rd.include(req,res);
```



App Design

It is possible to transfer the control to a static page while performing servlet collaboration, but it strongly recommended not doing so.



Depending upon requirement request dispatching can be done either of two ways:

Forward

In case of forwarding the response, the response generated by the second servlet is directly sent over to the client without going back to the first servlet. In such a case, the response generated by the first servlet is displayed on the client browser, but because the second servlet is also going to use the same browser window to display its response is overwritten.

Include

In case of including the response, the response generated by the second servlet is returned to the first servlet and embedded at the point of invocation within the first servlet's response data.



Best Practice

Generally, when the task is made up of multiple intermediate processing units and a final data collating unit, request forwarding is used.

Generally, when there is a single processing unit obtaining data from different components request inclusion is preferred.



App Design

It is recommended to use forward mechanism of the RequestDispatcher in favour of the include mechanism, as it increases the flexibility and reusability of the application.

Response Redirection

- `HttpServletResponse` contains a method
 - ```
public void sendRedirect(String location)
```
- This method sends a redirect response to the client.
- `sendRedirect()` is used if
  - The browser has to initiate a new request to a different servlet / JSP
  - The servlet / JSP that is to be forwarded is not in the same web application

While executing the application, there might be a situation where the application needs to start further execution with fresh set of data without user intervention (transferring control from one module of application to another), or the user needs to be redirected to another URL (redirection to another site). In such a scenario, transferring the control using the `RequestDispatcher` is not recommended as it also transfers the older request object along with the control.

Response Redirection generates a response programmatically in such a way that the browser upon receiving the response automatically generates a fresh request and tries to connect with the new URL.

`sendRedirect()` is used if

- The browser has to initiate a new request to a different servlet / JSP
- The servlet / JSP that is to be forwarded is not in the same web application

```
res.sendRedirect("http://<path to some other
resource>");
```

The above code redirects the user to the path mentioned as parameter in the `sendRedirect()` method.



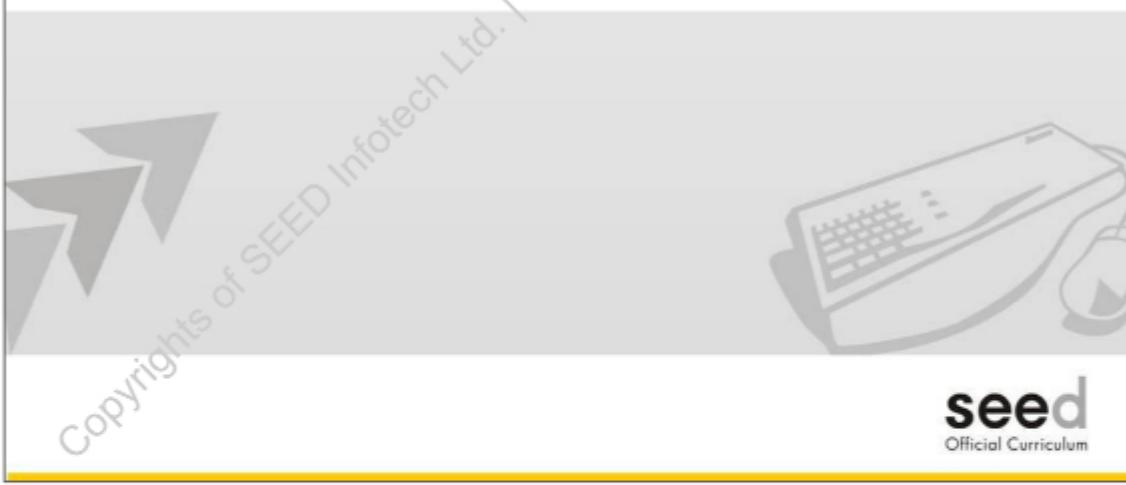
Interview Tip

`response.sendRedirect` is normally used to transfer control to another site or application or to a static page within the same application.

`requestDispatcher.forward` or `include` is used to transfer control to a dynamic component within the same application.

## Chapter - 7

### Java Servlets (Session, Filters)

seed  
Official Curriculum

To make the application truly responsive, the client specific data needs to be maintained on the server. Returning user information needs to be mapped and identified based on the earlier interactions with the application. This is achieved in a web application with a mechanism called as Session Tracking which is the topic of discussion in this chapter.

#### Objectives

**At the end of this chapter you will be able to:**

- Construct an application for Session Tracking using Hidden Form Fields.
- Construct an application for Session Tracking using Cookies.
- Construct an application for Session Tracking using URL Rewriting.
- Construct an application for Session Tracking using HttpSession object.
- Construct an application to filter requests.
- Construct an application to listen to application lifecycle events.

## Session

- To provide implementation for a complex requirement, multiple requests from users are required.
- Entire interaction is called as a session.
  - A session starts from the first user request till the application is closed (implicitly or explicitly).
- The intermediate data generated as part of the request processing needs to be maintained for seamless execution.

An application consists of multiple tasks grouped together to perform a single complex task. In case of web application, these tasks may be represented by corresponding servlets. Right from the moment user starts the interaction with the application by sending in the first request, till the time user is actively involved with application execution, it is a single set of data exchange session between the user and the application. This is known as the User session.

The ‘session start’ always happens when the first request is generated by a user. The ‘session end’ may happen either explicitly (user terminates the application gracefully) or implicitly (user terminates the application by closing the browser). In case of abnormal application termination, the web application maintains the data related to the application for some time, as specified by the configuration file, to allow the user to connect again in a stipulated quantum of time.

It is imperative that the entire session be recorded either in terms of the processing that the user has requested or the intermediate data that has been

generated. This helps make the application more user-specific and gives a sense of personalization for the application.



For every user interaction from login to logout a separate, unique session is created.

## Why Session Tracking?

- Web applications work on HTTP.
- HTTP is a stateless protocol.
  - Client state and execution history is not maintained.
- Dynamic applications are built on the assumption that the same client accesses the application multiple times.
- Client data is lost with every request being considered as a new request.

As mentioned earlier, web applications are accessed using HTTP, which is stateless. HTTP specification does not allow information associated with the senders and receiver's identification to be stored internally.

A responsive application is built with an assumption that the user will be accessing the same application in a single session multiple times. Because HTTP is stateless, it becomes impossible to identify the customer using only the request data being sent through.

Every request generated through HTTP is a fresh request, without having any connection with the earlier request sent from the same physical address to the same application. If the web application is built such that it accepts only the data from the client request, then client's identity is lost in processing and a continuous flow of application execution cannot be simulated. For example,

### Simulation assuming session tracking was absent

| Client                           | Server                                                                                                                              |
|----------------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| Access facebook.com              | Locate the index.html and send it.                                                                                                  |
| Login with username and password | Identify the user and validate, respond with user's home page                                                                       |
| Click on wall                    | Cannot understand which user's wall, as no user information is provided; respond with an error message and ask user to login again. |
| Login with username and password | Identify the user and validate, respond with user's home page                                                                       |
| Click on wall                    | Cannot understand which user's wall, as no user information is provided; respond with an error message and ask user to login again. |

And the same process keeps on happening over and over again.

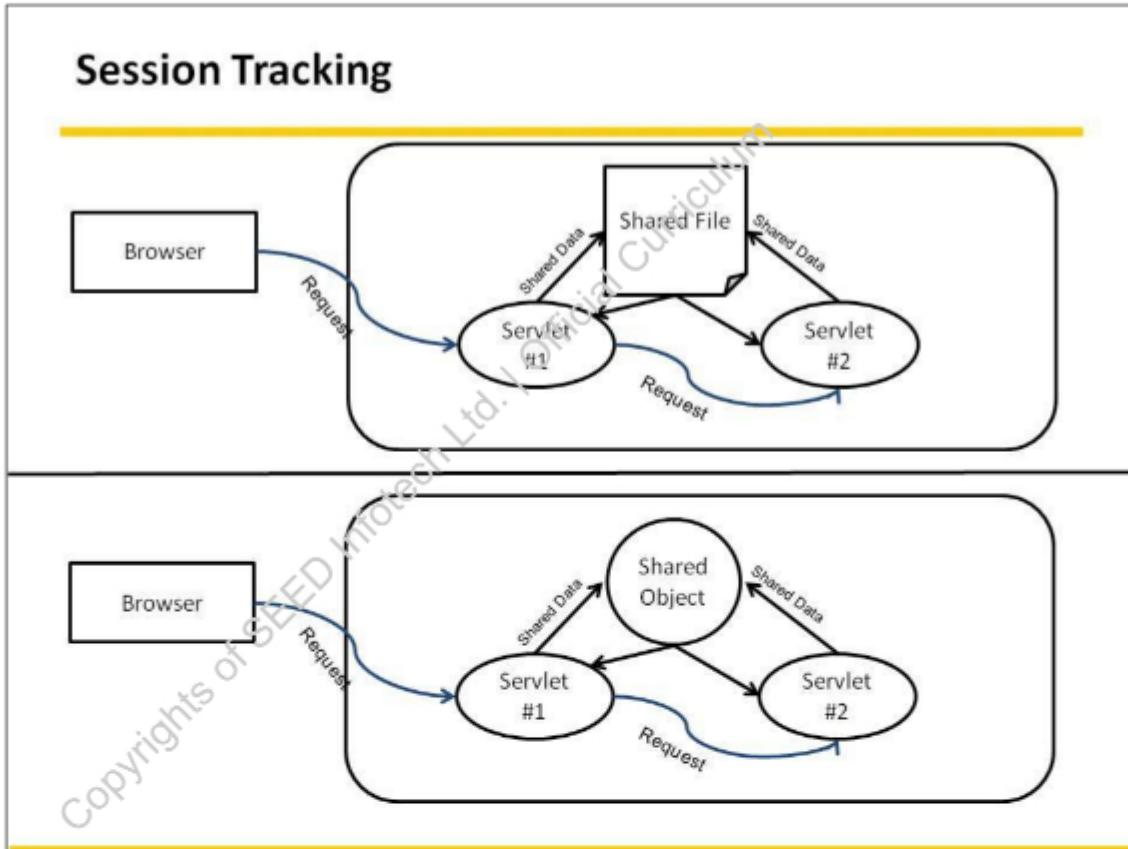
## Why Session Tracking? – Contd.

- For a truly responsive application, the client data needs to be maintained at the server-side for client identification.
- As HTTP does not provide any tools or utilities, programmer has to take care of identifying the client programmatically over the duration of a client's application access (a session).
- There is a need to track the user activity across the entire session.
  - This mechanism is called as session tracking.

As HTTP cannot provide mechanisms to provide user identification data, the developer has to take care of identifying the user interactions with the system and track the user's task execution across multiple request generations. User's interactions with the application from the start (login) to finish (logoff/switch off) are known as a user session.

An application can be considered to be a truly responsive application when it has the capabilities to identify the user and associate users' task execution (by way of data) uniquely. As the protocol or the server cannot provide the capability to identify the user, the developer has to provide this mechanism using code.

The mechanism by which the developer tracks the user actions across the session is called as session tracking.



The basic need for Session tracking can be identified as having to share the data between multiple execution components. For example, when the user logs into an application, he/she will provide their name and credentials. The same data is to be utilized further in the application to customize the look-and-feel based on the user's preferences. To do so, the data provided by the user needs to be shared with the component that generates the look and feel.

One way of doing this could be by using a shared file, which can act as a go-between for multiple components. The data passing component can push the data into a file and the data processing component can pull the data from the file. There is nothing wrong with this mode of data sharing, but it has a huge drawback. The files are stored on secondary devices, and secondary storage manipulations, by rule, are slower as compared to primary memory.

To speed up the data storage and retrieval, the application needs to have a mechanism that can hold the data and user identification in primary memory. This can be achieved in Java by using one of three ways:

7. Hidden Form Field
8. Cookies
9. URL Re-writing

Copyrights of SEED Infotech Ltd. | Official Curriculum

## Hidden form fields in HTML

- Hidden form fields store information about the session. The hidden data can be retrieved later by using the `HttpServletRequest` object
- Form fields can be used only on dynamically generated pages, so their use is limited. And there are security loopholes: people can view the HTML source to see the stored data

In this approach a unique token is embedded within each HTML form. For example, the following HTML specifies an input control of type `Hidden`:

```
<input type= "Hidden" name= "uid" value= "seed">
```

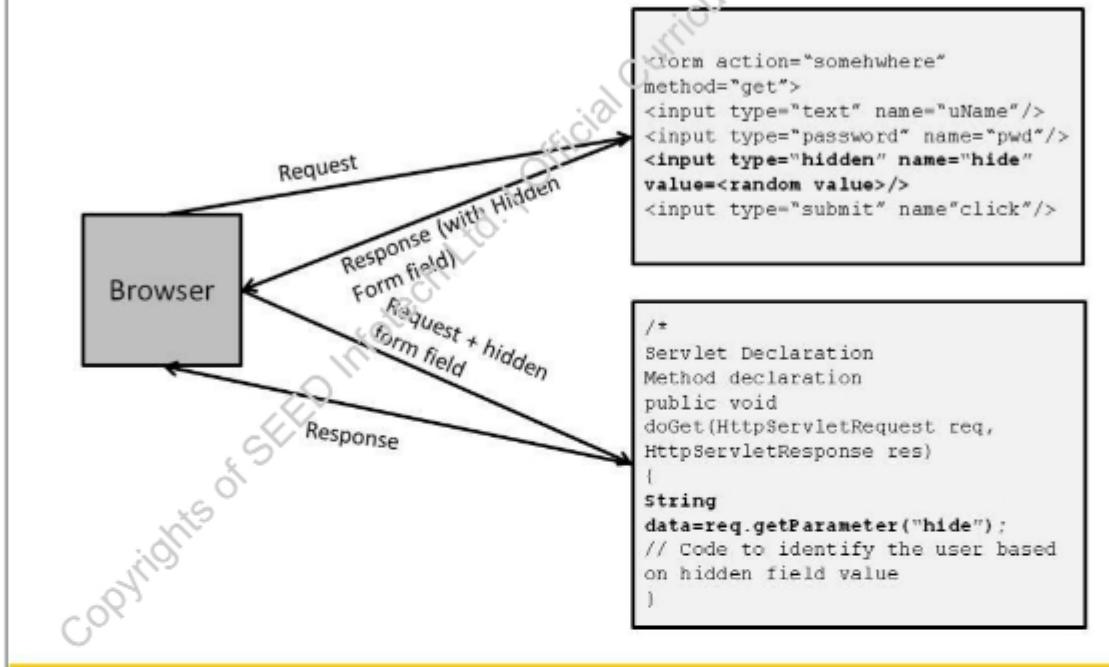
When the request is submitted, the server receives the token as part of the request, which in turn can be used to identify the client by matching the unique token.



App Design

Even though the form field is hidden, it is still available on the client-side in HTML form. Users can use 'view source' feature of the browser to reveal the unique value associated with the specific user. The servlet specification does not recommend this approach.

## Session Tracking Using Hidden Form Field



This approach of session tracking or state management works only if the content is created dynamically. For static content, this methodology is not suitable.

The sequence of implementing this approach is as follows:

1. The user requests a resource from the browser.
2. The resource generates the output dynamically using `PrintWriter`.
3. While generating the dynamic HTML, a hidden form field with a unique value is generated and embedded in the HTML form.
4. The generated form is rendered on the client's browser using `PrintWriter`.
5. User fills up the data and submits the form.
6. The hidden form field, which is part of the HTML form is also packaged inside the request data being sent across.
7. The targeted servlet component receives the HTML request.
8. Hidden form field parameter is retrieved using `getParameter()` method by supplying the hidden form field name as parameter.

9. The retrieved value of hidden form field is compared against the database of unique ids (see step 3)
10. The mapping of retrieved value and the database value identifies the returning user.
11. Steps 2-10 repeat till complete.

### Code Example

Following is a sample code to implement session tracking using Hidden form fields:

The HTML file used to send request.

```
<BODY>
<H3>Included Hidden fields with this Form </H3>
<FORM ACTION="getHiddenformfield.jsp" METHOD="post">

Enter your name :
<input type ="text" name = "name"
value = "">
<input type="hidden" name="hiddenfield" value=<unique
value>">
<input type="submit" value="submit">

</FORM>
</BODY>
```

The Servlet to retrieve the value:

```
public void doPost(HttpServletRequest req,
HttpServletResponse res)
{
 PrintWriter out=res.getWriter();
 String hiddenData=req.getParameter("hiddenfield");
 /* code to compare and identify returning users */
 . . .
}
```

## What is a Cookie?

- A Cookie is a string which is sent to a client at the start of a session.
- If the client wants to continue the session it sends back the Cookie with subsequent requests.
- Most common way to implement session tracking
- Cookies store information about a session in a human-readable file on the client's machine. The server associates a session ID from the cookie with the data from that session
- A cookie cannot grow more than 4K in size, and no domain can have more than 20 cookies
- Cookies pose some privacy concerns for users

Cookies are the most commonly used means of tracking client sessions. Cookies were first introduced by Netscape, and later picked up by virtually all major browser applications.

A cookie is a small piece of textual information sent by the web application to the client, stored on the client, and returned by the client for all requests to the server.

The web application creates a text file on the server side upon receipt of the first request by a specific user. This text file, called a cookie, is attached to the response object being assembled to be sent back to the client. The cookie travels to the client-side along with the response data and the browser extracts this cookie and stores it on local storage medium. The next time when a request is sent to the same application (domain), browser searches in the local storage and identifies the cookie files associated with the application. All the cookies associated with the application are attached to the request and sent along with the actual request data. The application component receiving this request retrieves the cookies separately from the parameter data and the comparison of cookie data is done to identify the specific client sending the specific request.



Web containers send a cookie by sending the Set-Cookie response header in the following format:

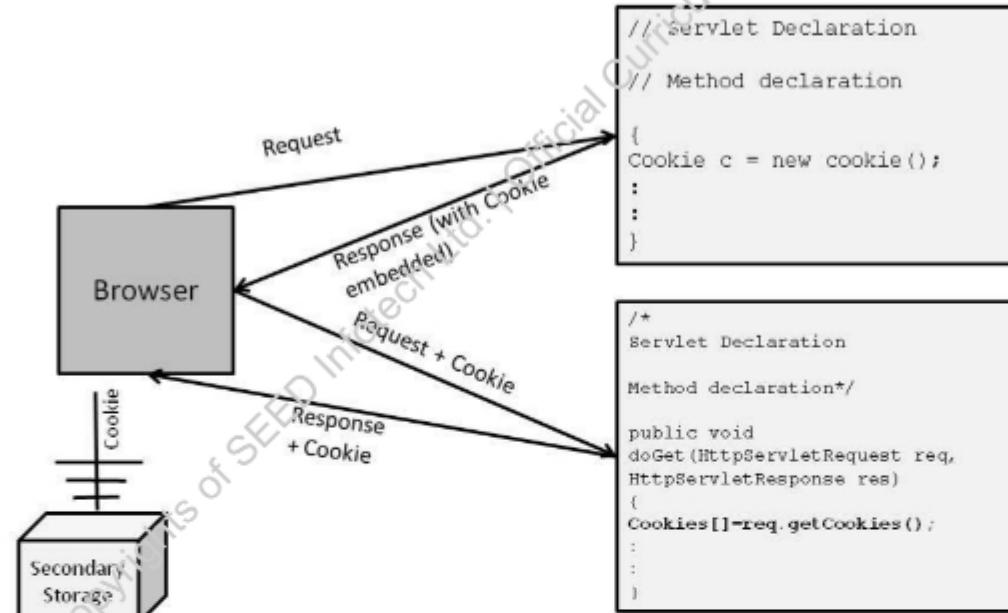
Set-Cookie: Name=VALUE; Comment=COMMENT;  
Domain=DOMAINNAME; Max-age=SECONDS;  
Path=PATH; secure; version=1\*DIGIT



By default, user sessions are maintained using cookies. If cookies are disabled on the client side, some other means of session management is used.

Because cookies containing user specific data are kept on the client-side machines, there are certain security concerns in situations where the client machines are shared.

## Session Tracking Using Cookies



While tracking a session, retrieving cookies becomes a major problem as the browser returns all the cookies associated with a specific domain. To identify a specific cookie of a specific user, the developer has to write some code. An example of such a code is shown below:

```

// Import io package
import java.io.*;

// Import servlet packages
import javax.servlet.*;
import javax.servlet.http.*;

public class CookieCounter extends HttpServlet
{
 // GET request handler
}

```

```
public void doGet (HttpServletRequest request,
HttpServletResponse response) throws IOException
{
 // Define content type
 response.setContentType ("text/html");

 PrintStream pout = new
PrintStream(response.getOutputStream ());

 // Check to see if there are any cookies
Cookie[] cookieArray = request.getCookies ();

 // Default value
int count = 0;

 // Check for cookies
if (cookieArray != null)
{
 for (int i =0; i< cookieArray.length; i++)
 {
 Cookie c = cookieArray[i];

 // Check for the count cookie
if (c.getName ().equals ("count"))
{
 // Parse cookie value and assign to count
try
{
 Integer num = new Integer (c.getValue ());
 count = num.intValue ();
}
 catch (NumberFormatException nfe) { }
}
 }
}

// Increment counter
```

```
count++;

// Send updated cookie
response.addCookie(new Cookie ("count",
String.valueOf(count)));

// Output message
pout.println ("You have visited this page " + count +
" times since your web browser started");
pout.flush();
}

// POST request handler calls GET request handler
public void doPost (HttpServletRequest request,
HttpServletResponse response) throws IOException
{
 doGet(request, response);
}

}
```



The Servlet 3.0 specification allows cookies to be marked as `HttpOnly` cookies. These cookies are not exposed to client-side scripting code. JavaScript code written on the HTML file cannot make use of cookies using the `setHttpOnly` (`boolean isHttpOnly`) method.

To identify whether the cookies is `HttpOnly` or not, use `boolean isHttpOnly()` method.

This mechanism helps to prevent cross-side scripting attacks, meaning client side code cannot use cookies to crack the server side code.

## What is URL Rewriting?

- All links and redirections which are created by a Servlet have to be encoded to include the session ID.
- This is a less elegant solution because the session cannot be maintained by requesting a well-known URL which was created in a different (or no) session.
- does not allow the use of static pages.
- URLs can get quite lengthy.
- You have to be sure to append the information to every URL that references your site.

If cookies are disabled on the client side, then developer can use URL rewriting as a backup. Assuming that developer did his/her part correctly, URL rewriting always works - the client cannot prevent it.

URL rewriting takes the session ID and appends it to every URL that comes into the application, as shown below:

```
URL+;jsessionid=1234567
```

URL is the resource to which the request is being forwarded.

URL rewriting requires that all pages in the application be dynamically generated. URL rewriting cannot be enforced for static HTML pages because the unique URL path parameter (the `jsessionid`) is dynamic.

URL rewriting is used only if cookies fail, and only if developer specifies that the responses should encode the URLs being sent to the client.



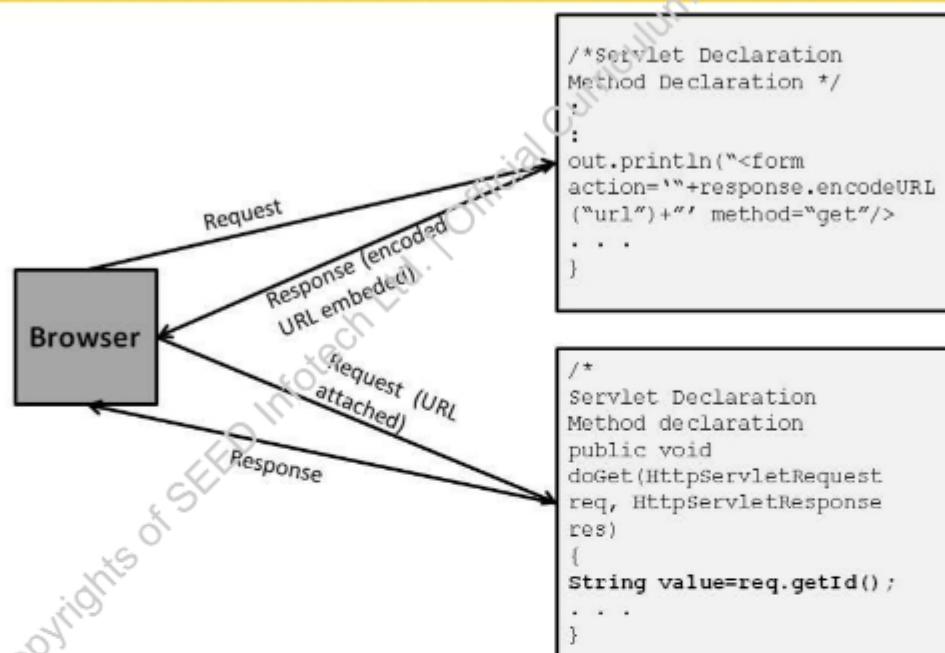
App Design

URL rewriting is dependent on the unique identifier being attached to the URL after being processed through Java code, because of this URL rewriting is unusable in situation where static pages are part of application navigation cycles.

URL rewriting has one more drawback; if the user book-marks the URL and revisits the site using the book-mark, the application can be accessed, which may poses a security threat.

Copyrights of SEED Infotech Ltd.

## Session Tracking Using URL Rewriting



The URL which is being sent back to the client will have to be encoded thus.

```

public void doPost(HttpServletRequest req,
HttpServletResponse res)
{
 PrintWriter out=res.getWriter();
 out.println("Click <a href=" +
res.encodeURL(req.getRequestURL().toString()) + ">this
link");
 out.println(" to access this page again.
");

 .
 .
}

```

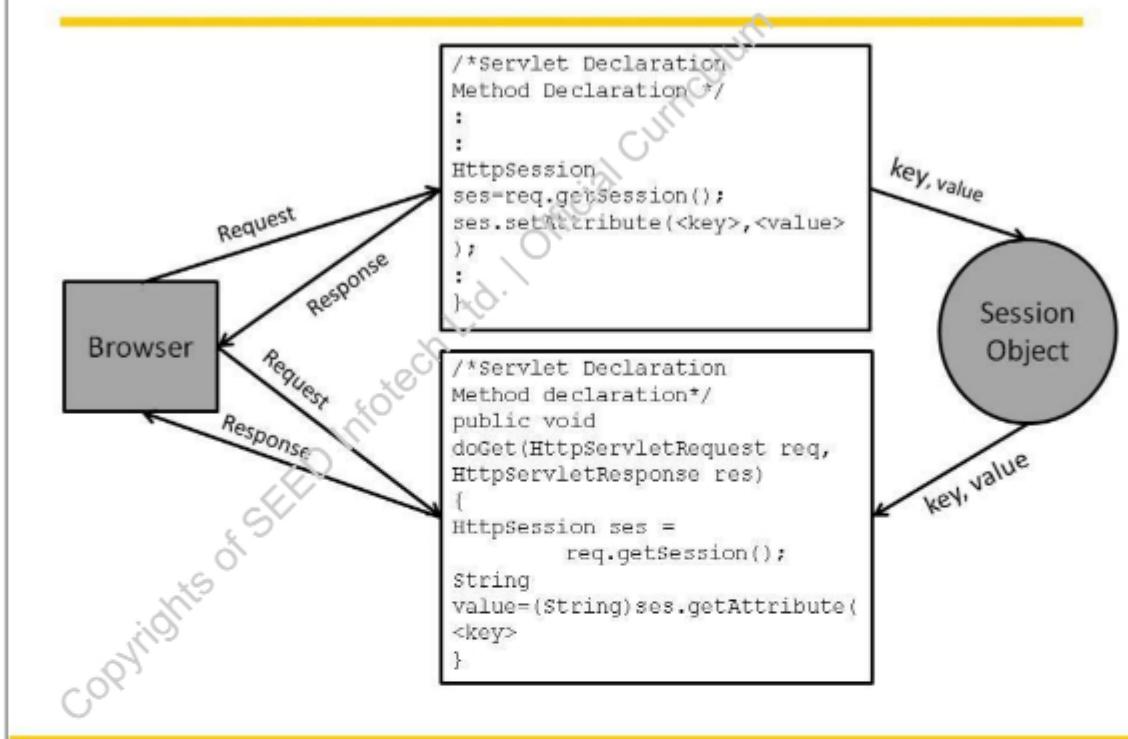
## What is HttpSession?

- HttpSession is an object provided to perform session tracking.
- Provides ease-of-programming to the developer by combining user identification and state management into a single object.
- Other mechanisms force developer to write extensive code for state management.
- HttpSession object is not an original session tracking mechanism.
  - It internally uses cookies to implement session tracking when cookies are disabled it uses URL rewriting.

Over and above the three session tracking mechanisms discussed earlier, Servlet API provides one more way of implementing session tracking. HttpSession is an object internally holding a map implementation. Any application using HttpSession is internally using cookies generated by the container, transparent to the developers. If cookies are disabled then it switches to URL rewriting mode for maintaining the session.

The other session tracking mechanisms provide only session tracking. They help in identifying a returning visitor, but there is no built-in help available in terms of maintaining the user specific data. Cookies provide help in this regard, but coding for maintaining the data is a little complex. To make the state management task easier, HttpSession object exposes certain straight forward methods that the developer has to embed within the code to manage the user state without extra trouble.

## Session Tracking Using HttpSession



The typical code for managing user state through HttpSession usage is as follows:

```

public class AttributeServlet extends HttpServlet
{
 protected void doGet(HttpServletRequest request,
 HttpServletResponse response) throws ServletException,
 IOException
 {
 HttpSession session = request.getSession();
 String name = request.getParameter("attrib_name");
 String value =
 request.getParameter("attrib_value");
 String remove =
 request.getParameter("attrib_remove");
 if (remove != null && remove.equals("on"))
 {
 session.removeAttribute(name);
 }
 }
}

```

```
 }
 else
 {
 if (name != null && name.length() > 0 && (value
!= null) && value.length() > 0)
 {
 session.setAttribute(name, value);
 }
 }
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();

 out.println("<HTML>");
 out.println("<HEAD><TITLE>Session
Attributes</TITLE></HEAD>");
 out.println("<BODY>");
 out.println("<H1>Session Attributes</H1>");
 out.println("Enter name and value of an
attribute");
 String url="/session/servlet/attributes";
 out.println("<FORM ACTION=\"" + url + "\"
METHOD=\"GET\">");
 out.println("Name: ");
 out.println("<INPUT TYPE=\"text\" SIZE=\"10\"
NAME=\"attrib_name\">");
 out.println("Value: ");
 out.println("<INPUT TYPE=\"text\" SIZE=\"10\"
NAME=\"attrib_value\">");
 out.println("
<INPUT TYPE=\"checkbox\"
NAME=\"attrib_remove\">Remove");
 out.println("<INPUT TYPE=\"submit\" NAME=\"update\"
VALUE=\"Update\">");
 out.println("</FORM>");
 out.println("<HR>");
 out.println("Attributes in this Session");
 // Print all session attributes
 Enumeration e = session.getAttributeNames();
```

```
while (e.hasMoreElements())
{
 String att_name = (String) e.nextElement();
 String att_value = (String)
session.getAttribute(att_name);
 out.println("
Name: ");
 out.println(att_name);
 out.println("Value: ");
 out.println(att_value);
}
out.println("</BODY></HTML>");
out.close();
}
```



If no parameter is supplied to `getSession()`, it reverts to a boolean true value. If true is specified to `getSession()`, it creates a new session object if one does not exists. If a session object exists, a reference to the same object is returned.

## Filters

- Filter is a special type of class, implementing the `javax.servlet.Filter` interface.
- It is a ‘pluggable’, ‘reusable’ component which intercepts and mediates all relevant request and responses.
- It is used for performing infrastructural tasks, which are not normally performed within the application.

Filter is a special type of class which implements the `javax.servlet.Filter` interface. They are used for performing filtering effect on the incoming requests, as well as out-going responses, depending on requirement.

There are certain tasks which are essential for building a complete application solution, but are not a part of business logic execution. For example, logging application usage should record user access to the application. As servlets are built for performing operations related to business logic execution, the burden of providing the extra code is split into a separate component.

This type of features are not request specific, that is, these services are required across the entire application regardless of which business rule is executed, hence these components should typically execute based on the application requirement rather than user requirement. Such services are also known as infrastructural services.

The filters can be thought of as a special type of servlet with the same life-cycle and same execution process. One additional feature of filters enables the application of multiple filters to a specific request. Filters dynamically modify the

requests or responses to use the information contained in them. Filters are typically used for:

Security(Authentication Authorization)	and	Logging and auditing
Image conversion		Data compression
Localization		XSL/T transformations of XML content
Encryption		Tokenizing
MIME-type chaining		Caching

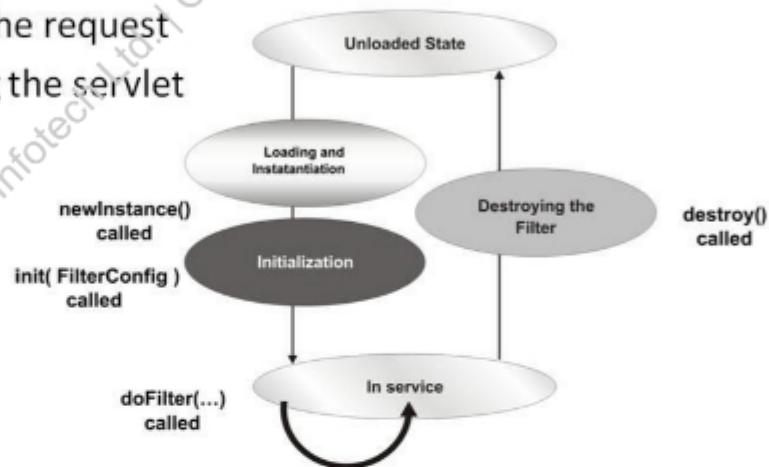


Tech App

A filter can be applied to multiple requests by using the `web.xml` configuration file. `<url-pattern>` tag identifies the URL to which the filter needs to be applied. Creatively providing this value will ensure multiple requests being filtered through a single filter.

## Filter Life Cycle

- Filter life cycle divide into 4 parts:
  - Loading and instantiation
  - Initialization
  - Servicing the request
  - Destroying the servlet



Filters are configured in the deployment descriptor of a web application as follows:

```

<web-app>
<filter>
 <filter-name>name</filter-name>
 <filter-class>package.classname</filter-class>
</filter>
<filter-mapping>
 <filter-name>name</filter-name>
 <url-pattern>URLs to filter</url-pattern>
</filter-mapping>
</web-app>

```

Filter lifecycle closely resembles that of a servlet, with the same states, with only one change. Servlet has a `service()` method to represent the business rule invocation, while Filters have `doFilter()` method for implementing the filtering code.

### Filter Lifecycle

- The filter is instantiated using default or no argument constructor.
- The `init()` method is called on the filter to initialize it.
- The `doFilter()` method is the heart of filter. All requests and responses pass through this method.
- Upon completion of its task `destroy()` is invoked.

`doFilter()` receives `FilterChain` object as one parameter, which encapsulates the filter stack created using the configuration file. If a request is filtered through more than one filter, then each filter is invoked in turn depending on its precedence in the `web.xml` file.

The link between multiple filters are implemented by invoking `doFilter()` on the `FilterChain` object received in the `doFilter()` method. Internally the container manages a stack of filter references which are executed one after the other. The last filter in the stack invokes the `service()` method of the servlet for which the request is intended.



Tech App

A filter is not restricted only to the incoming request. By providing code after the `doFilter()` method, outbound responses can also be filtered through the same filter.



Servlet3

With Servlet 3.0 Filter can be defined using `@WebFilter` annotation.

```
@WebFilter (filterName = "FilterName",
urlPatterns={"/hello"})
```

```
public class TestFilter implements Filter
{
 ...
}
```

Annotation is an easier way of implementing the filter mechanism. The `@WebFilter` annotation is internally processed to generate the corresponding entries in the `web.xml` file.



Annotation based configuration is not recommended, as any change in the configuration forces the code to change and makes it mandatory to recompile the source code.

## Event Listeners in Servlet

- Application lifecycle is controlled by the container
- To map task execution with application component state, listeners are exposed.
- Container listens to the events occurring and checks if a corresponding listener is configured for the application.
  - Subsequently, it also checks for the existence of the specific event method invocation and if it exists executes the method.

There are certain tasks which are related to the application behaviour rather than specific execution events. A customer may wish to perform some activity as soon as the application starts. For example, in an online shopping web application, the daily discount rates should be assigned as soon as the application starts for the day.

To achieve this, servlet specification provides few listeners which listen to the application lifecycle events. The listeners also provide hooks to provide implementation associated with these events. A developer needs to override these hooks to enable the application to start listening to the events.

A listener is configured using the web.xml file.

```
<web-app>
 <listener-class>package.listener</listener-class>
</web-app>
```

The listeners available through the API can be divided into three broad categories:

**1. Servlet context events**

- `ServletContextListener`
- `ServletContextAttributeListener`

**2. Servlet request events**

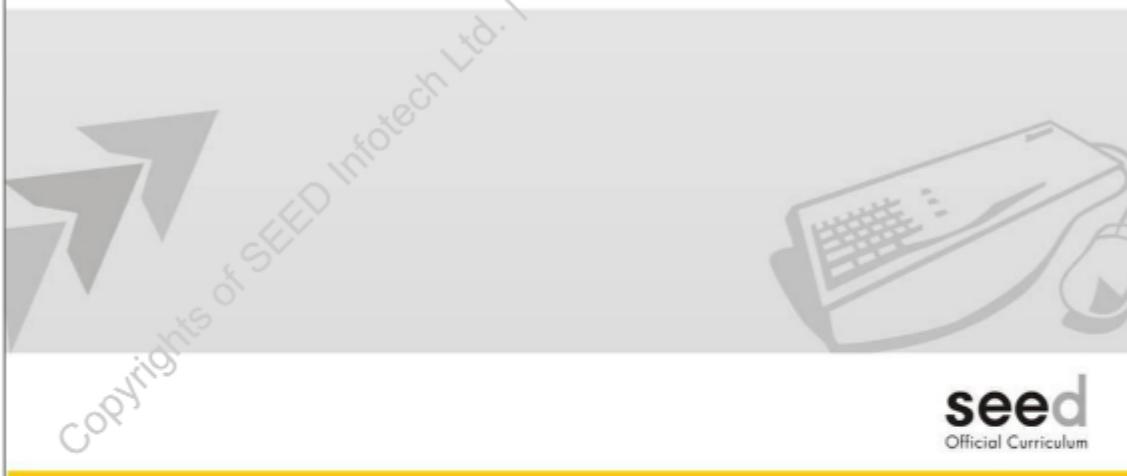
- `ServletRequestListener`
- `ServletRequestAttributeListener`

**3. Http session events**

- `HttpSessionListener`
- `HttpSessionAttributeListener`
- `HttpSessionActivationListener`
- `HttpSessionBindingListener`

## Chapter - 8

### JSP (Basic JSP, MVC and Action Tags)



Web application creation is a complex task involving lots of manpower with lots of skill sets. Two basic skills required for web application building are keen logic for writing the code and strong imagination for building the UI. To ensure a right fit for right task, the web application is divided into two parts; the code part and the UI part. To ensure Rapid Application Development with better readability, Java Server Pages evolved.

This chapter covers the basics of Java Server Pages.

#### Objectives

**At the end of this chapter you will be able to:**

- Construct a simple JSP application.
- Construct a JSP application by customizing the page attributes.
- Construct a JSP application following MVC design pattern.

## Introduction to Java Server Pages (JSP)

- Java Server Pages is a Java technology that generates dynamic web pages based on HTML.
  - Java code embedded in HTML code.
  - More HTML code and less Java code.
  - Simplifies the dynamic presentation layer in a multi-tiered architecture.
  - Separate presentation and business logic.
  - JSP is template based content generation.
  - Servlet is programmatic content generation.

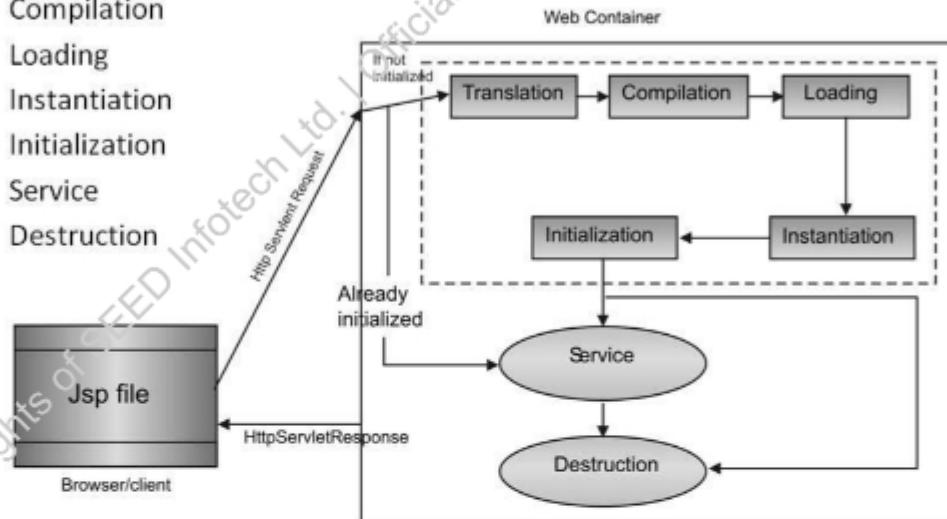
The importance of Java Server Pages Specification is two-fold. Primarily JSP comfortably fits in the MVC design pattern as a Visual or Presentation layer component. The main goal of a Presentation layer component in MVC is to give the user an easy interface and not perform any business logic. Secondarily the JSP contains more of HTML or XML code in form of various tags than Java code, which is unlike Servlets that contains more of Java code and less of HTML and XML.

Since JSP files are supposed to contain HTML and XML they are not stored initially as Java files but as .jsp files. The .jsp files are markup files and hence cannot contain Java code directly. Java code can be embedded in a JSP file using some special construct that will be covered later. JSP file naturally resembles a HTML file. This makes it easy for the designers to work with JSP files. The designers can concentrate on the design aspect of the JSP file without ever concerning themselves about the Java code. A Java programmer can later embed whatever Java code needs to be there in the JSP file for the purpose of integration with rest of the distributed components or code that is going to do some validations at the presentation level.

Since the JSP files are going to work inside a container which is a JVM at the core they have to be presented to the Container as classes. To achieve this, a JSP file is converted to a respective Servlet internally on the first invocation. This is done by a special component of the container called a JSP engine. The Servlets thus generated are in sync with the JSP file. Any modification to the JSP file later is reflected in the Servlet that is generated for that particular JSP.

## JSP Life Cycle

- JSP life cycle is divided into 7 phases
  1. Translation
  2. Compilation
  3. Loading
  4. Instantiation
  5. Initialization
  6. Service
  7. Destruction



JSP Life cycle contains seven steps is as follows

### Translation

The JSP is translated into servlet source code. Here JSP is validated for syntax and tag files.

### Compilation

The servlet source code is compiled into java bytecode i.e. a Servlet class file. Translation and Compilation can occur anytime between deployment and first request by the user.

### Loading

The application loads the servlet class into the memory of container using the application's class Loader.

## Instantiation

The application instantiates an instance of the servlet class for each JSP in the application.

## Initialization

The `jspInit()` method is called on the servlet instance to initialize it. It is called only once during JSP life cycle.

## Service

The `_jspService()` method is invoked for each request received for the JSP. This method cannot be overridden.

## Destruction

The servlet instance is taken out of service `jspDestroy()` method called.

## JSP

- Three types of predefined tags:
  - Directives
  - Scripting elements
  - Actions
- JSP Elements

<!-- - - ->	HTML Comment
<%-- - - %>	JSP Comment
<%!	Declaration
<%=	Expressions
<%	Scriptlets
<%@	Directives

As discussed earlier JSP is a presentation level component, hence it supports direct use of HTML and XML tags within. But there are certain tags that are provided by the JSP specification that are useful for some special tasks. The concerned Servlet is affected by the JSP tags.

Following are the categories of JSP tags mentioned below:

### Predefined Tags

#### Directives

These affect the overall structure of the servlet that results from translation, but produce no output themselves.

#### Scripting elements

These let us embed Java code into the JSP page.

#### Actions

These are the special tags available to affect the runtime behavior of the JSP page. JSP provides some standard actions, such as <jsp:useBean>.

## Custom tags

These facilitate the developer to write their own tags.

Tag	Tag Name
<!-- -->	HTML Comment
<%-- --%>	JSP Comment
<%! %>	Declaration
<%@ %>	Expressions
<%@ %>	Directives

The different types of JSP tags are mentioned in the table above.

Note the following:

- If the Declaration tag in JSP contains variable declaration, it will be converted into an instance variable declaration. If it contains method declaration, it will be converted into a method in the resulting servlet.
- Expressions are used to evaluate an expression or collate a result from method execution and pass it to the `out` object in JSP.
- Scriptlets will be added to the service method of the servlet, as normal Java code to be executed at invocation.
- Directives are special instructions given to the JSP compiler to handle customized requests.
- Normal HTML tags in JSP will be converted into `JSPWriter.println(" ")` code which is sent back to client browser as part of the response object.

## Directives

- Directives provide general information about the JSP page to the JSP engine.
- 3 types of directives:
  1. page
    - Controls properties of the JSP page
  2. include
    - Include the contents of a file into the JSP page at translation time.
  3. taglib
    - Makes a custom tag library available within the including page.
- The general syntax for a directives is

```
<% @ directive [. . .] %>
```

There are basically three types of directives to provide information to jsp page.

### **page directive**

page directive is used to provide pre-compilation instruction to the JSP compiler, while the JSP is getting converted into a servlet. It is used to customize the resultant servlet execution.

### **include directive**

include directive is used to add an existing resource to current JSP at compile time. The external resource contents are added to the JSP and then converted into servlet.

### **taglib directive**

taglib directive is used to provide implementation for custom-tags if they are being used in the JSP.

## Page Directive

Page directive of JSP Page gives the information about the entire JSP page by using different attributes.

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

Attributes of page directive are as follows:

Attribute	Description
language	It defines the scripting language to be used. This attribute exists in case future JSP containers support multiple languages. The default value for this attribute is Java.
extends	The value is a fully qualified class name of the super-class that the generated class, into which this JSP page is translated, must extend.
import	A comma separated list of packages or classes, with the same meaning as import statements in Java class.
session	Session specifies whether the page participates in an HTTP session. When true the implicit object, namely session, which refers to javax.servlet.http.HttpSession is available and can be used to access the current/new session for the page. If false the page does not participate in a session and the implicit session object is unavailable.
buffer	Buffer specifies the buffering for the output stream to the client. If the value is none, no buffering occurs and all output is written directly through to the ServletResponse by a PrintWriter. If a buffer size is specified, output is buffered with a buffer size not less than that value.
autoFlush	If the value is true, the output buffer to the client is flushed automatically when it is full.

isThreadSafe	It defines the level of thread safety implemented in the page. If the value is true, then the JSP engine may send multiple client requests to the page at the same time. If the value is false, then this is same as implementing the javax.servlet.SingleThreadModel interface in the resulting servlet.
info	It defines an informative string that can subsequently be obtained from the page's implementation of the Servlet.getServletInfo() method.
errorPage	errorPage defines a URL to another JSP page within the current Web application. The errorPage is invoked if a checked or unchecked exception is thrown. The page implementation catches the instance of the Throwable object and passes it to the error page processing. This mechanism is useful. It avoids the need for developers to write code to catch unrecoverable exceptions in their JSP pages.
isErrorPage	isErrorPage indicates if the current JSP page is intended to be another JSP page's error page. If true, then the implicit object exception is available and refers to the instance of the java.lang.Throwable thrown at runtime by the JSP that caused the error.
contentType	ContentType defines the character encoding for the JSP and MIME type for the response of the JSP page. The default value for the MIMETYPE is text/html; the default value for the encoding is ISO-8859-1.

Page directive attributes can be configured as:

```
<%@ page import="java.util.* , com.seed.* %>
<%@ page session="true"%>
<%@ page errorPage="errorHandler.jsp" %>
<%@ page isErrorPage="true" %>
```

```
<%@ page language="java" %>
<%@ page buffer="8kb" autoFlush="false"%>
```

## Include Directive

The `include` directive instructs the container to include the content of a resource in the current JSP, inserting it inline, in the JSP page, in place of the directive. The specified file must be accessible and available to the JSP container. The `file` attribute specifies the filename of the file to include. This is a relative path within the current web application, beginning with a forward slash. This is also called as static include as the contents are added at time of conversion only. The consequence is that if the resource changes the change will not be reflected in the JSP.

```
<%@ include file="fileName.jsp/html %>
```

The above code ensures that `filename.jsp` will be embedded in the parent JSP at the specified point.



Tech App

`@include` directive should be coded exactly at the location where the second HTML based files is needed.

This directive is also known as static include, because it adds the second file at the time of translation, hence subsequent changes in second JSP are not reflected.

## Scripting Elements

- These elements actually deal with Java code.
- Scripting elements are divided into 3 parts:

### 1. Declarations

- Declare and define variables and methods

```
<%! int count=0 ; %>
```

### 2. Scriptlets

- Java code fragments that are embedded in the JSP page

```
<%
count++;
out.println("welcome!! You are visitor number: "+ count);
%>
```

### 3. Expressions

- Expressions act as placeholders for Java language expressions

```
<%= count %>
```

JSP scripting elements allow inserting Java code into JSP which in turn is converted into servlet code at the time of JSP translation.

- A JSP declaration lets a developer define methods or fields that are inserted into the main body of the servlet class. Since declarations do not generate any output; they are normally used in conjunction with JSP expressions or scriptlets.
- JSP scriptlets let a developer insert Java code into the JSP which converts into servlet's `_jspService` method (which is called by service).
- JSP 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.

Following is an example of the scripting element:

```
<%! int i=2;%>
<table>
<%
for(int x=1;x<=10;x++)
```

```
{
%>
<tr>
 <td><%=i*x</td>
</tr>
<%
}
%>
```

Even though available, it is strongly recommended to avoid using scriptlets into JSP code. Scriptlets reduce readability and maintainability of the code.



## Actions

- There are three categories of standard actions
  1. Those that control run-time forwarding/including.
  2. Those that prepare HTML for Java plug-in.
  3. Those for using JavaBeans components.
- There are six standard JSP actions
  1. jsp: include
  2. jsp: forward
  3. Jsp:params
  4. jsp: plugin
  5. jsp:setProperty
  6. jsp:getProperty
  7. Jsp:useBean

The JSP action tags enable the programmer to use the built in functionality provided by the servlet container. These are the XML tags those can be used in the jsp page.

### <jsp:include>

Includes the resources at the time of page request. Also known as dynamic include as opposed to static include with @include directive.

```
<jsp:include page="other.jsp" flush="true"/>
```

### <jsp: forward>

Forwards the resources at the time of page request.

```
<jsp:forward page="other.jsp" flush="true"/>
```

### <jsp:param>

Passes the parameters to dynamic pages.

```
<jsp:include page="SomePage.jsp" flush="true">
```

```
<jsp:param name="name1" value="value1"/>
<jsp:param name="name2" value="value2"/>
</jsp: include>
```

**<jsp: plugin>**

Plugs the other components like Applets or beans.

```
<jsp:plugin type="applet" class="MyApplet.class"/>
```

**<jsp:useBean>**

Refer discussion related to <jsp:useBean> later in the chapter.

## Why Layered Application Design?

- A layered application design is necessary for the following reason:
  - Responsibilities can be clearly divided
    - Separate business logic and presentation.
    - Change in business logic layer does not affect the presentation layer and vice-versa.
    - All the 3 tasks can be handled by different components.
      - Manage the user's interaction with the system.
      - Manage actual data.
      - Format data in multiple ways and present it to the user.
    - e.g. Online Stock Trading facilities
  - Solution
    - MVC (Model View Controller)

As mentioned earlier, a web application normally is built using two distinct components: the UI and the business rules. To enable faster development and better maintenance of the application, each of the components is isolated in a separate layer where the responsibilities of the components are also different.

Separating the application in two different layers enables developer team to focus on getting the code out as fast as possible, without having to worry about the look-and-feel of the application. This can be taken care of by a specialized design team, whose responsibility is to ensure the application looks good, without having to worry about its workability.

This also leads to faster testing, debugging processes in the SDLC, as error isolation is localized to individual modules where tracking is easier.

This requirement can be achieved by implementing the MVC design architecture.

## Model

A model represents the business rules and the data associated with the business rules. It can be a Java class, an EJB, a Java Bean, a POJO or even a legacy system.

### Responsibilities of Model

- Performing DB queries.
- Applying the business rules.
- Generating results or exceptions.

## View

View represents the User Interface. Anything to do with user interaction, whether accepting data from the user or sharing data with the user is identified as a view. In specific case of Java web application, a view is any HTML based component like HTML, JSP.

### Responsibilities of View

- Providing Data input screens.
- Providing Data input validations.
- Rendering data generated by the Model.

## Controller

A controller controls the application execution logic. It acts as a linkage between the View on the front-end and the Model at the back end. In Java based web application, Servlets are best-fit for a controller.

### Responsibilities of Controller

- Retrieve data from View and prepare in Model specific format.
- Transfer data to Model.
- Accept result generated by Model.
- Convert Model result into view specific data formats.
- Prepare data for View.
- Identify a View corresponding to the result generated by Model.



Interview Tip

Two words are interchangeably used when discussing MVC: MVC design pattern and MVC architecture. Rather than going into discussion about what is correct, remember that it makes little or no difference to a developer.



Best Practice

Generally, View-Model and Model-View data transfers should be avoided. This will force the View to have data processing code, which is not its responsibility. In case of unavoidable requirement, custom tags should be preferred over usage of scriptlets.



Interview Tip

A controller (Servlet) can have business logic associated with it, but it should NEVER have any business logic implementation.

## Using a Java class with JSP

- Usage of MVC in JSP makes it mandatory to provide the business rule implementations in a Java bean.
- A Java class is used in JSP with the `<jsp:useBean>` predefined tag.
- `<jsp:useBean>` tag instantiates the Java class specified in the attribute list.
- The scope of the object is governed by the scope attribute.
  - Application
  - Session
  - Request
  - Page

MVC can be implemented in JSP by encapsulating the business rules into a Java class (POJO or Java Bean) and using it inside the JSP with a ready-made action tag `<jsp:useBean>`.

### JSP Syntax

```
jsp:useBean id="beanInstanceName" scope="page | request
| session | application"
class="package.class" | type="package.class" | class="package.class" |
type="package.class" beanName="{ package.class }"
<%=expression%> } "type="package.class" } { | other
elements /<jsp:useBean> }
```

This tag undergoes following steps to locate or instantiate the bean:

10. Attempts to locate a Bean with the specified scope and name.
11. Defines an object reference variable with the specified name.
12. If it finds the Bean, stores a reference to it in the variable. If type is specified, gives the Bean that type.

13. If it does not find the Bean, instantiates it from the specified class, storing a reference to it in the new variable. If the class name represents a serialized template, the Bean is instantiated by `java.beans.Beans.instantiate`.
14. If it has instantiated (rather than located) the Bean, and if it has body tags (between `<jsp:useBean>` and `</jsp:useBean>`), executes the body tags.

<b>Attribute</b>	<b>Description</b>	<b>Example</b>
<code>id</code>	Bean is identified in the JSP page	<code>id="address"</code>
<code>scope</code>	Attribute Name	<code>scope="session"</code>
<code>class</code>	The Java class of the bean	<code>class="pl.BusinessAddress"</code>
<code>type</code>	Type of the variable to be used to refer to the bean	<code>type="pl.AddressBean"</code>
<code>beanName</code>	The name of the serialized bean	<code>beanName="AddressBean"</code>

To transfer the HTTP request parameter values in the corresponding model class, `<jsp:setProperty>` tag is used and to retrieve the values from model `<jsp:getProperty>` tag is used. There are four different flavors of `<jsp:setProperty>` tag usage:

15. Set all properties

```
<jsp:setProperty name="person" property="*" />
```

16. Set specific property one by one

```
<jsp:setProperty name="person" property="pname" />
<jsp:setProperty name="person" property="pid"/>
```

17. Set the property using param attribute

```
<jsp:setProperty name="person" property="pname"
param="name" />
```

4. Set the property using value attribute

```
<jsp:setProperty name="person" property="pname"
value="Nik"/>
```

### Code Example

Following code explains the `<jsp:useBean>` action tag which follows MVC design pattern.

```
//View
//personInfo.html
<html>
 <body>
 <form action="\myWorkSpace\personInfo.jsp"
method="get">
 Enter PersonID:<input type="text" name="pid"/>
 Enter PersonName:<input type="text"
name="pname"/>
 </form>
 </body>
</html>
```

```
//Controller
//personInfo.jsp
<jsp:useBean id="person" class="pkg.Person"
scope="session">
 <jsp:setProperty name="person" property="*" />
 <jsp:getProperty name="person" property="pid"/>
 <jsp:getProperty name="person" property="pname"/>
</jsp:useBean>
```

```
//Model
//Person.java
package pkg;
public class Person
{
 private String pname, pid;
 public Person() {}
```

```
public void setPname(String pname) {
 this.pname=pname;
}
public String getPname() {
 return pname;
}
public void setPid(String pid) {
 this.pid=pid;
}
public String getPid(){
 Return pid;
}
}
```



It is a best practice to embed the `<jsp:setProperty>` tag within the body of `<jsp:useBean>` tag. This ensures that if the bean is not instantiated, the `setProperty` tag does not throw an exception.

## JSP Implicit Objects

Object	Class or Interface	Purpose
out	javax.servlet.jsp.JspWriter	The output stream
request	javax.servlet.http.HttpServletRequest	Access the details of request
response	javax.servlet.http.HttpServletResponse	Access the details of response
session	javax.servlet.http.HttpSession	To handle http sessions
application	javax.servlet.ServletContext	Refers web application
page	java.lang.Object	The page it self
pageContext	javax.servlet.jsp.PageContext	Refers page environment
config	javax.servlet.ServletConfig	Servlet configuration information
exception	java.lang.Throwable	Used for error handling

JSP provides certain implicit objects, based on the servlet API. These objects are accessed using standard variables and are automatically available for use in a JSP without having to declare/define them.

### out

This object is the object that writes into the output stream to the client. To make the response object useful, this is a buffered version of the `java.io.PrintWriter` class, and is of type `javax.servlet.jsp.JspWriter`. The buffer size can be adjusted via the `buffer` attribute of the `page` directive.

### request

This object represents the request that triggered the `service()` invocation. This object is a protocol and implementation specific subclass of `javax.servlet.ServletRequest`. It has a request scope.

## **response**

This object is the `HttpServletResponse` instance that represents the server's response to the request. This object is a protocol and implementation specific subclass of `javax.servlet.ServletResponse`. It has a page scope.

## **session**

This object represents the session created for the requesting client. Sessions are created automatically, and a new session is available even when there is no pre-existing session object available. The session object is of type `javax.servlet.http.HttpSession`, and has a session scope.

## **pageContext**

The `pageContext` provides a single point of access to many of the page attributes and is a convenient place to put shared data within the page. It is of type `javax.servlet.jsp.PageContext` and has a page scope.

## **config**

This object is the `ServletConfig` for this JSP page, and has page scope. It is of type `javax.servlet.ServletConfig`.

## **page**

This object is the instance of the page's implementation servlet class that is processing the current request. It is of type `java.lang.Object`, and has page scope. This object can be thought of as a synonym to `this` within the page.

## **Application**

This object represents the servlet context. It is of type `javax.servlet.ServletContext` and has application scope.

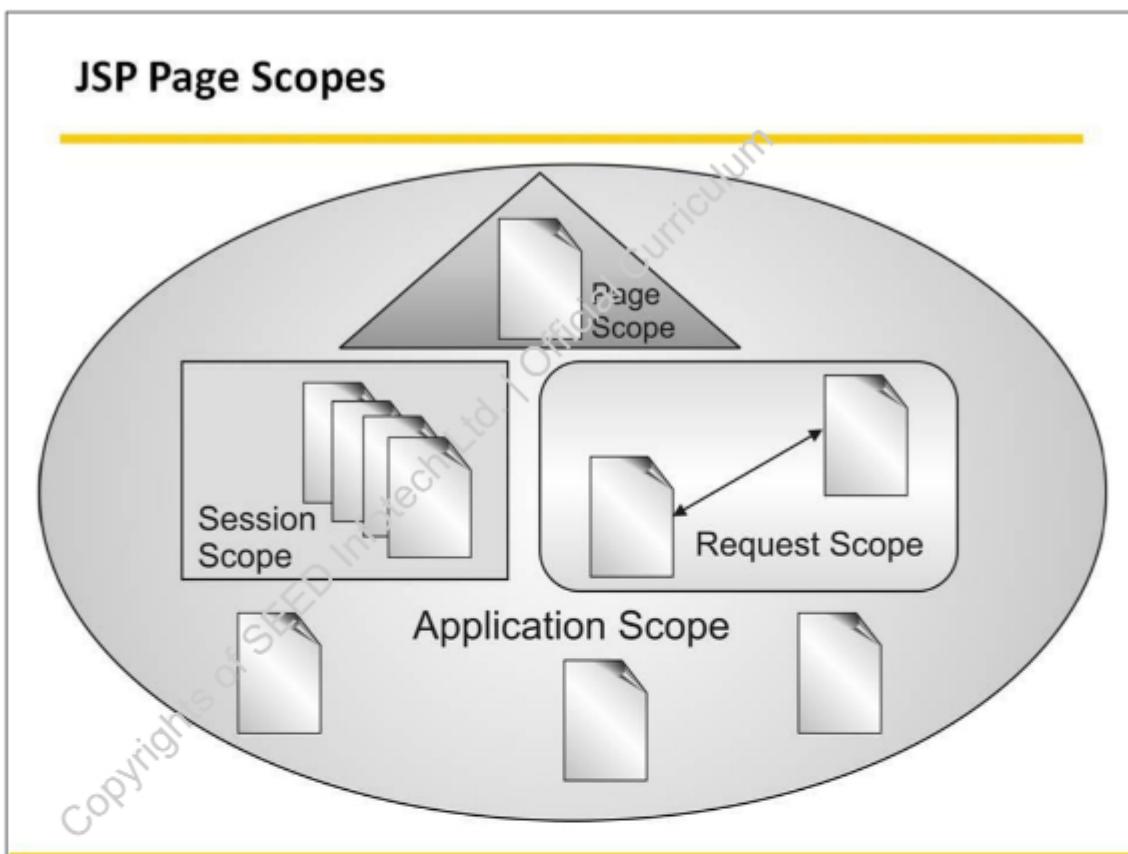
## **Exception**

An error page is essentially the JSP that handles the exception, so the Container gives the page an extra object for the exception. This object is of type `java.lang.Throwable`.

Following code snippet explains how to use implicit objects in an application. It accepts the username and password from login.html page and print username.

```
//Login.html
<html>
 <body>
 <form action="accept.jsp">
 <!-- -->
 </form>
 </body>
</html>
```

```
//accept.jsp
<%
 String name=request.getParameter("username");
 String pass=request.getParameter("password");
 out.println("Hello"+ " "+name);
%>
```



When any object is created it is assigned one of the predefined scopes. The scope of any object defines how widely the object is available and who can access it. Scopes are associated with container objects. JSP have 4 different scopes. All the implicit objects as well as the user-defined objects in JSP page exist in one of the following scopes.

### Application Scope

The objects are shared across all the components of web application and are accessible for the life of the application. Use `setAttribute()` and `getAttribute()` of `ServletContext` interface.

```
<% String
userId=context.setAttribute("userId",userId);%>
<% String
userId=(String) context.getAttribute("userId");%>
```

## Session Scope

Shared across all the requests that belong to single user session and are maintained as attribute-value pairs by HttpSession interface.

```
<% String
userId=session.setAttribute("userId",userId);%>
<% String
userId=(String) session.getAttribute("userId");%>
```

## Request Scope

Shared across all the components that process the same request and the same response and maintained as attribute-value pairs by HttpServletRequest interface.

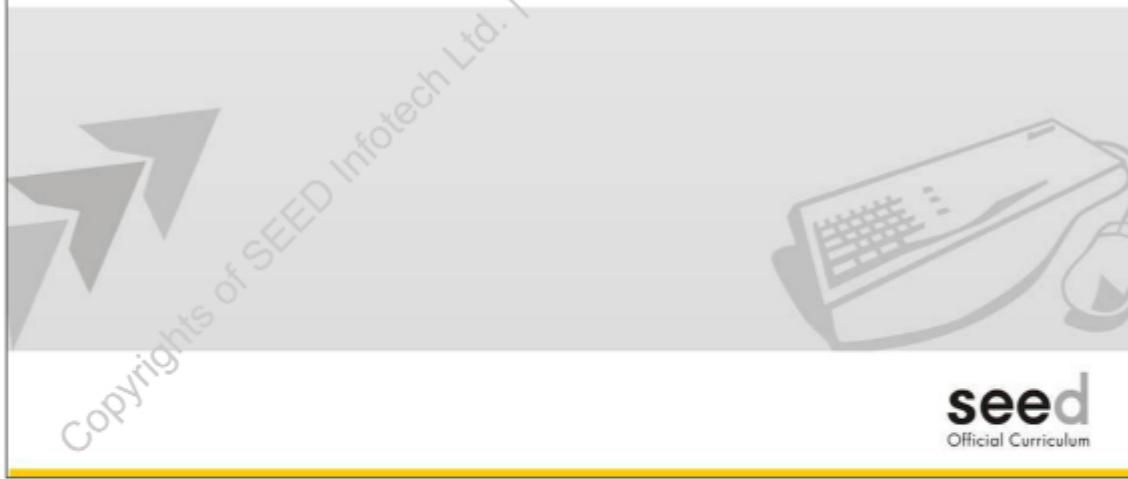
```
<% String
userId=request.setAttribute("userId",userId);%>
<% String
userId=(String) request.getAttribute("userId");%>
```

## Page Scope

Objects in the page scope are accessible only in the translation unit in which they are defined, and maintained as attribute-value pairs by concrete subclass of the abstract class PageContext.

## Chapter - 9

### JSP (Custom Tags, EL and JSTL)



JSPs are great for creating User Interface that is rich, but it has problems in situations where the output on the client's browser needs to be customised depending on requirement and data inputs. Usually, this is implemented using embedded scriptlets in JSP. This leads to reduced readability and maintainability. The Java code required for performing customisation is encapsulated into a Java class and a customised tag for the specific implementation is provided to developers. This is called as custom tags. To assist functioning of custom tags, a special data representation language called Expression Language (EL) and a pre-created set of commonly used custom tags is also provided. JSP Standard Tag Library (JSTL) helps reduce the need of creating commonly required tasks by providing pre-built custom tags.

This chapter covers custom tags, Expression Language and JSP Standard Tag Library.

## Objectives

**At the end of this chapter you will be able to:**

- Create and utilize a custom tag.
- Create an application with Expression Language.
- Create an application with JSTL tags.

## Limitations of Java Scriptlets

- Java Scriptlets are embedded in JSP to implement dynamic code.
- Problem
  - Leads to reduced readability
    - Goes against MVC principles, by mixing responsibilities
  - Development becomes slower
    - Designer and developer are dependent on each other's work.
- Solution
  - Can be avoided by encapsulating the Java code into designer-friends reusable tag.
    - Custom Tag - Tag customised to specific requirements

JSP applications can increase dynamic behavior by embedding Scriptlets the Java code, which responds depending on the data-input. Problem with this approach is that it reduces the readability and leads to larger testing and debugging times, as error isolation becomes difficult.

As per Model-View-Controller, it is imperative that the UI component consist only of programming constructs which deal with generating user interface and there should not be any code that violates this by providing business rules execution, or processing code within it. Putting Java code in JSP breaks the assumption that every component focuses only on its own responsibilities.

In addition to the above problem, there is one more practical problem associated with this approach. Usually the UI is created by the designers who may not be well-verses with the programming language. This forces delayed UI creation as designers have to be dependent on the developer to embed the Java code at specified location. This may lead to either slower development times or higher resource requirements.

The User Interface developers should be creating the UI using only tags and not Java code, but if it is a necessity to embed Java code for responsiveness, a different solution can be thought of. If the Java code is extracted out of the JSP into a separate Java file and a wrapper over the functionality could be provided in terms of a reusable tag, the task becomes easier to handle.

In effect the solution that is required calls for a tag to be created which is suitable for a specific action to be executed. What is required is a custom tag.

## Custom Tags

- Used to completely separate the presentation and business logic.
- Created for specific use
- Advantages
  - Reusable code
  - Extensible JSPs – reduced scriptlets
  - Rapid Application Development
  - Maintainable JSPs
  - Provide HTML friendly layer of abstraction

Custom tags are created with a specific usage in focus; hence these cannot be used for general purpose. The main benefit of using the custom tag is that it helps separate out the presentation logic from the processing and business logic.

### Benefits of using custom tags

#### Reusable code

The functionality is provided in modularized form. The developer creates java code for the features, and a designer uses those in form of tags. This combination can be reused anywhere there is a similar need. The entire custom tag component package can be reused as it does not depend on any external component.

#### Extensible JSPs

Reduction in number of scriptlets embedded in the JSP increases its readability and modularity. This helps in extending JSP functionality in an easier manner. JSP code can now have a focused approach towards providing better and richer presentation rather than processing data.

## Rapid Application Development

As discussed earlier, custom tags are created by the developer and used by the designers, hence the dependence between them is reduced to a greater extent as both these components are modularized and the same custom tag can be reused again in similar requirements.

## Maintainable JSPs

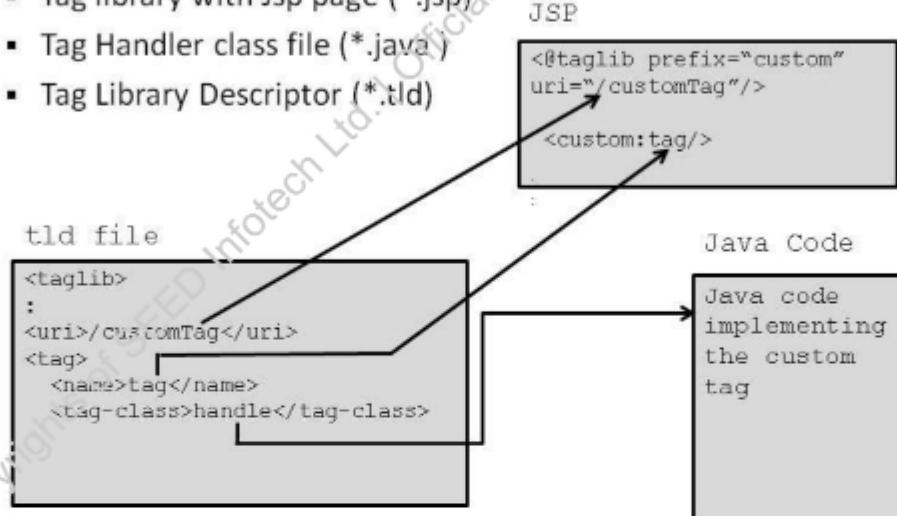
Reduced scriptlets means cleaner code, which in turn leads to better maintenance cycles as error isolations are easier. Maintenance can also have a focus.

## Layer of Abstraction

A designer does not need to understand the Java code running behind the custom tag as it is totally abstracted out of the designer's sphere of influence.

## Custom Tag Construction

- A Custom tag is made up of the following components
  - Tag library with Jsp page (\*.jsp)
  - Tag Handler class file (\*.java)
  - Tag Library Descriptor (\*.tld)



Creation of custom tag requires three components:

18. JSP file - which needs the functionality
19. Java code - which implements the functionality required in the JSP
20. tld file - which acts as a link between the JSP and the Java code.

Custom Tag creation process:

- Create a tag handler class. (.java): Tag handler class is required to provide the implementation of the tag. Perform some task when the tag is encountered and some other task when the tag is complete.
- Create a tag library descriptor (.tld): tld file acts as a medium between the JSP and the Java class. It maps the usage and the implementation by providing specific values.
- Create a JSP (\*.jsp): JSP file utilizes the functionality of the Java code by way of the tag defined in the tld file.

What actually happens internally within the container:

- As per the JSP translation process, the JSP engine tries to convert the JSP into a servlet code.
- It encounters a tag that it does not know how to translate (the custom tag).
- The taglib directive written maps all the tags starting with 'prefix' to a specific uri.
- Container searches for a tld file containing the uri mentioned in the JSP.
- tld file provides a reference of the Java class managing the implementations.
- Container extracts the codes corresponding to the tag start and tag end (and more if it exists).
- This code is embedded with the servlet code that generated at the end of the translation phase.
- The code is embedded as simple object reference<dot>method invocation.

### Types of Tags

There are 3 types of tags possible in custom tag:

Tag Type	Example
Empty Tag	<info: message/>
Tag with attributes	<info: message name="value"> </info: message>
Tags with body	<info: message name="value"> <%=new java.util.Date()%> </info: message>

The type of the tag is defined in the tld file by using the <bodycontent> tag. It can have three possible values.

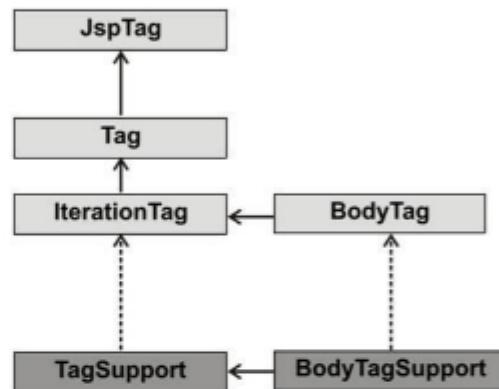
- empty
- JSP
- tagdependent

## TagSupport and BodyTagSupport

- TagSupport
  - A base class for defining new tag handlers implementing Tag.
- BodyTagSupport
  - A base class for defining tag handlers implementing BodyTag.

**Constant Return Values:**

- EVAL\_BODY\_INCLUDE** :Evaluate Body into existing out stream
- EVAL\_PAGE**: Continue evaluating the page
- SKIP\_BODY** : Skip body evaluation
- SKIP\_PAGE** : Skip the rest of the page.
- EVAL\_BODY\_AGAIN**: request the reevaluation of same body



### TagSupport

The **TagSupport** class is a utility class to be used as a base class for handling the tags. The **TagSupport** class implements **Tag** and **IterationTag** interfaces. Many tag handlers extend **TagSupport** class.

#### TagSupport API

Following are the some important methods of **TagSupport**.

Method	Description
<b>TagSupport()</b>	Default constructor, all subclasses are required to define only a public constructor with the same signature, and to call the superclass constructor.
<b>int doAfterBody()</b>	Default processing for a body.
<b>int doEndTag()</b>	Default processing of the end tag

	returning EVAL_PAGE.
int doStartTag()	Default processing of the start tag, returning SKIP_BODY.
Tag getParent()	The Tag instance most closely enclosing this tag instance.
Object getValue(java.lang.String k)	Get the value associated with a key.
void setParent(Tag t)	Set the nesting tag of this tag.
void setValue(java.lang.String k, java.lang.Object o)	Associate a value with a String key.
void setPageContext(PageContext pageContext)	Set the page context.

### **BodyTagSupport**

The BodyTagSupport class implements the BodyTag interface having some additional convenience methods including getter methods for the bodyContent property and methods to get at the out JspWriter. Many tag handlers will extend BodyTagSupport class.

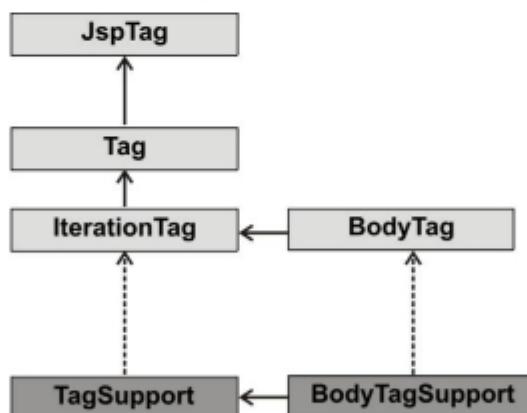
### **BodyTagSupport API**

Following are the some important methods of BodyTagSupport.

Method	Description
BodyTagSupport()	Default constructor, all subclasses are required to only define a public constructor with the same signature, and to call the superclass constructor.
int doAfterBody()	After the body evaluation: do not

	reevaluate and continue with the page.
int doEndTag()	Default processing of the end tag returning EVAL_PAGE.
void doInitBody()	Prepare for evaluation of the body just before the first body evaluation: no action.
int doStartTag()	Default processing of the start tag returning EVAL_BODY_BUFFERED.
BodyContent getBodyContent()	Get current bodyContent.
JspWriter getPreviousOut()	Get surrounding out JspWriter.
void release()	Release state.
void setBodyContent(BodyContent b)	Prepare for evaluation of the body: stash the bodyContent away.

Following diagram gives a complete picture of Tag Handler hierarchy.



These tag handler method returns a constant value which decides the JSP behaviour.

## Constant Return Values

Field	Description
public static final <b>EVAL_BODY_INCLUDE</b>	Evaluate body into existing out stream. Valid return value for <code>doStartTag()</code> .
public static final int <b>EVAL_PAGE</b>	Continue evaluating the page. Valid return value for <code>doEndTag()</code> .
public static final int <b>SKIP_BODY</b>	Skip body evaluation. Valid return value for <code>doStartTag()</code> and <code>doAfterBody()</code> .
public static final int <b>SKIP_PAGE</b>	Skip the rest of the page. Valid return value for <code>doEndTag()</code> .
public static final <b>EVAL_BODY_AGAIN</b>	Request the reevaluation of same body.

Following code prints “Hello World” with current date and time using custom tag.

This file contains HTML design and custom tag declaration and taglib URI .

```
//hello.jsp
<%@ taglib uri="/hello" prefix="examples" %>
<html>
 <head>
 <title>First custom tag</title>
 </head>
 <body>
 <p>This is static output. Tag output is shown in
italics.</p>
 <p><i>
 <examples:hello>
 </examples:hello>
 </i></p>
 <p>Closing the tag without a body will have the
same effect</p>
```

```
<p><i>
<examples:hello />
</i></p>
<p>This is static template data again.</p>
</body>
</html>
```

This is a tag handler class, to handle the tag operation and perform some business logic.

```
//HelloTag.java
package com.seed.tagext;
import java.io.IOException;
import java.util.Date;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.TagSupport;
public class HelloTag extends TagSupport {
 public int doStartTag() throws JspTagException {
 return EVAL_BODY_INCLUDE;
 }
 public int doEndTag() throws JspTagException {
 String dateString = new Date().toString();
 try {
 pageContext.getOut().write("Hello
world.
");
 pageContext.getOut().write("My name is " +
getClass().getName() +
 " and it's " +
dateString + "<p/>");
 }
 catch (IOException ex) {
 throw new JspTagException("Fatal error:
hello tag could not
 write to JSP
out");
 }
 return EVAL_PAGE;
 }
}
```

}

This file contains mapping between tag and tag handler class.

```
//hello.tld
<taglib>
 <tlib-version>1.0</tlib-version>
 <jsp-version>1.2</jsp-version>
 <short-name>simple</short-name>
 <uri>http://tomcat.apache.org/example-taglib</uri>
 <description> A simple tag library for the examples
</description>
 <tag>
 <name>hello</name>
 <tag-class>com.seed.tagext.HelloTag</tag-class>
 <body-content>JSP</body-content>
 <description>
 Simple hello world example.
 Takes no attributes, and simply generates HTML
 </description>
 </tag>
</taglib>
```



Starting from JSP 2.0 onwards, it is now no longer mandatory to provide the tag library information in the web.xml file.

## Limitations of Custom Tags

- Custom tags are used to reduce the scriptlets.
- There are certain tasks that become difficult even with custom tags
  - Data rendering on JSP
  - Response expression
- To perform these operation, scriplets have to be used.
- This problem can be overcome using
  - EL (Expression Language)
  - JSTL (JSP Standard Tag Library)

Custom tags are good for increasing readability and maintainability of the JSP, but there are certain tasks which are difficult to manage with custom tags. For example, if there is a task involving data rendering tasks, then it becomes difficult to manage that using custom tags. For this, scriplets need to be used for faster development. Similarly, it becomes difficult to render expressions using custom tags since the custom tag has to be nested with the JSP expression scriplet.

To overcome this problem the specification provides components like EL (Expression Language) and JSTL (JSP Standard Tag Library).

## Introduction to Expression Language

- Programming language constructs complete with operators, syntax and reserved words.
- Introduced in JSP 2.0 specification.
- It is comparable with traditional JSP script expressions.
  - Output of both is same, but \${} is not java code.
- Goal of EL is to remove java from JSPs.
  - It returns an undefined value.
  - EL expressions cannot use variables declared in script.

```
The outside temperature is <%= temp %> degrees.
```

```
The outside temperature is ${temp} degrees.
```

```
<%! int myVar =10;%>
```

```
The myVar value is ${myVar}
//Not possible
```

Expression Language is a programming language complete with operators, syntax and reserved words. It is a feature introduced in JSP 2.0 specification. EL helps accessing the application data stored in JavaBeans components. It is used with JSP tags to separate the Java code from the tags.

Some of the features of EL are as follows:

- No type casting.
- Type conversions are usually done implicitly.
- Double and single quotation marks are equivalent.



EL expressions cannot use variables declared in script.

<%! int myVar=10;%> this is a declaration in JSP file and hence it is declared as a variable of the resultant Java class.

## EL – Implicit Objects

- Web container is responsible for the life cycle of certain built-in objects such as request or response.
- These objects are already created internally when the request is provided to the JSP.
- EL can access these objects directly to manipulate the data embedded within.
- As developer does not have to create these objects, they are known as Implicit Objects.

To ease the programming efforts, the specification provides certain readymade objects, corresponding to the built-in objects created by the container.

These objects are created automatically by the container and are accessed using standard variables; hence they are called implicit objects.

### EL Implicit Objects

Object	Description	Example
pageContext	Accesses JSP's regular implicit objects	<code> \${pageContext.out.bufferSize}</code>
pageScope	A map containing the page scope attributes	<code> \${pageScope.myVar}</code>
requestScope	A map containing the request scope attributes	<code> \${requestScope.username}</code>
sessionScope	A map containing the session	<code> \${sessionScope.total}</code>

pe		Price}
applicationScope	A map containing the application scope attributes	<code>\$(applicationScope.totalPrice)</code>
param	A map containing a request parameter String	<code>\$(param.name)</code>
paramValues	A map containing a request parameter String []	<code>\$(paramValues.name["0"] )</code>
header	A map containing a request header String	<code>\$(header.accept)</code>
headerValues	A map containing a request header String	<code>\$(headerValues.host["0"] )</code>
cookie	A map matching Cookie fields to a single object	<code>\$(cookie.user)</code>

Following code demonstrates EL implicit objects.

```
<% // To display buffersize of the page's JSPWriter, use
the expression %>
 ${pageContext.out.bufferSize}
<% // To retrieve the request's HTTP method, use this
line of code: %>

 ${pageContext.request.method}

 ${sessionScope.totalPrice}

 ${header.accept}

 ${pageContext.request.requestURI}

 ${header["host"]}

 ${header['host']}

 ${header.host}
```



JSP Implicit objects are not same EL Implicit objects. JSP and EL implicit objects have only one object (`pageContext`) in common. `pageContext` has properties for accessing all of the other eight JSP implicit objects.

## EL – Operators

- EL operators are divided into four categories
  - Operators for property and collection access
  - Arithmetic operators
  - Relational operators
  - Logical operators

EL provides operators for performing its operations.

### Operators for property and collection objects

- Access an object's members
- Retrieve elements of Map, List and Array

### Arithmetic Operators:

The standard arithmetic operators like addition, subtraction, multiplication, division, etc are provided with EL.

### Relational Operators:

- Equality: == and eq
- Non-equality: ! and ne
- Less than: < and lt
- Greater than: > and gt
- Less than or equal: <= and le
- Greater than or equal: >= and ge

**Logical Operators:**

- Logical conjunction: `&&` and `and`
- Logical disjunction: `||` and `or`
- Logical inversion: `!` and `not`

Following code demonstrates using the different operators:

```
<% //EL Arithmetic %>
${2*3.14}

${1.5e6/1000000}

<% // EL relational and logical operator %>
${8.5 gt 4}

${(4 >=9.2) || (1e2 <=63)}

${(5*5) ==25? 1: 0}
```

## EL – Functions

- Custom Tags do not allow invoking functions; it can be done using scriplets.
- EL assists in invoking methods by accessing their corresponding XML tags.
- Only the function names and TLD's URI is required to access method within JSP.

The JSP expression language allows defining a function that can be invoked in an expression. Functions are defined using the same mechanisms as custom tags. Methods are invoked by accessing their corresponding XML tags.

Following code demonstrates accessing EL functions within a JSP:

```
//functionusage.jsp
<%@ taglib uri="http://myFunc/Functions"
prefix="myString" %>
<html>
 <body>
 Enter text:
 <form action="/EL/stringfun.jsp" method="get">
 <input type="text" name="x">
 <p><input type="submit">
 </form>
 <table border="1">
 <tr>
```

```

<td>UpperCase:</td>
<td>${myString:upper(param.x)}</td>
</tr>
<td>String Length:</td>
<td>${myString:length(param.x)}</td>
</tr>
</table>
</body>
</html>

```

The following code shows creating java files with method implementation:

```

//StrMethods.java
package com.seed.myfunc;
public class StrMethods
{
 public static String upper(String x)
 {
 return x.toUpperCase();
 }
 public static int length(String x)
 {
 return x.length();
 }
}

```

Following code shows function definitions and mapping of jsp functions with respective to java class:

```

//functions.tld
<?xml version="1.0" encoding="UTF-8" ?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-
jsptaglibrary_2_0.xsd"
 version="2.0">

```

```
<description>A tag library exercising SimpleTag
handlers.</description>
<tlib-version>1.0</tlib-version>
<short-name>SimpleTagLibrary</short-name>
<function>
 <name>upper</name>
 <function-class>myfunc.StrMethods</function-
class>
 <function-signature>
 java.lang.String upper(java.lang.String)
 </function-signature>
 </function>
 <function>
 <name>length</name>
 <function-class>myfunc.StrMethods</function-
class>
 <function-signature>
 java.lang.int length(java.lang.String)
 </function-signature>
 </function>
</taglib>
```

### Rules when creating methods for EL

- The method must be `public` and `static`.
- The class must be `public`.
- Method's arguments and return value must be valid within EL.

## JSP Standard Tag Library (JSTL)

- There are certain standard tasks which need to be implemented regardless of application domain.
- These standard tasks are encapsulated into a library which provides a custom tag implementation for each.
- This is known as JSP Standard Tag Library (JSTL).
- It greatly reduces the need to provide implementation to oft-repeated tasks, by providing ready-made, reusable custom tags.
- JSP has to provide the prefix provided through the JSTL libraries

```
<%@ taglib uri =http://java.sun.com/jstl/core_rt
prefix="c"%>
```

JSTL provides some simple tags which have the core functionality common to many JSP applications. The tag defined by JSTL works same everywhere. Main advantage of using JSTL lies in its reusable components. Developers do not have to create anything other than the linkage between the JSP and the libraries.

JSTL is further sub-divided based on the tag functionality, and there is a predefined prefix associated with it. Following is a list of the predefined prefixes:

Tag library	Description	Prefix
core	Tags for general purpose processing	c
xml	Tags for parsing, selecting and transforming XML data	xml
formatting	Tags for Formatting data for international use.	fmt
sql	Tags for accessing relational databases.	sql
functions	Tags for accessing relational databases.	fn



Best Practice

It is not mandatory to keep the standard prefixes while using JSTL, but it is recommended to maintain readability and standards.



Tech App

JSTL cannot be used without the standard jar files required. All web containers compliant with JSP 2.0 provide these jar files.

- jstl.jar
- standard.jar

## JSTL – Core

- JSTL core is also divide into 4 parts

JSTL Tag category	JSTL tags	Tag description
General purpose	<c:catch> <c:out>	Catches exceptions within a variable Displays contents within the page
Variable support	<c:set> <c:remove>	Sets the value of an EL variable Removes an EL variable
Flow control	<c:if> <c:choose> <c:forEach> <c:forTokens>	Alters the processing equaling value Alters the processing equaling set of values Repeats processing for each object in a collection Processes processing for each substring in given text field
URL handling	<c:url> <c:import> <c:redirect>	Rewrites URLs and encodes their parameters Accesses content outside the web application Tells the client browser to access a different URL

The core functionality JSTL is further divided into four major parts.

### General purpose

JSTL Tags	Description
<c:catch>	Catch exceptions within a variable.
<c:out>	Displays contents within the page.

Following code snippet shows the use of general purpose tags:

```
<c:catch var="e">
actions that might throw an exception
</c:catch>
<c:out value="\${number}" />
```

## Variable support

JSTL Tags	Description
<c:set>	Sets the value of an EL variable.
<c:remove>	Removes an EL variable.

Following code snippet shows the use of variable support tags:

```
<c:set var="num" value="${4*4}" />
<c:set var="num">
 ${8*2}
</c:set>
<c:set var="num">
 <c:out value="${8+8}" />
</c:set>
<c:set target="emp" property="pincode">
 12351
<c:set>
<c:remove var="num" scope="session"/>
```

## Flow control

JSTL Tags	Description
<c:if>	Alters the processing equaling value.
<c:choose>	Alters the processing equaling set of values.
<c:forEach>	Repeats processing for each object in a collection.
<c:forTokens>	Processes processing for each substring in given text field.

Following code snippet shows the use of flow control tags:

```
<c:if test="${x == '9'}">
 ${x}
</c:if>
<c:choose>
 <c:when test="${color == 'white'}">
 Light !
```

```

</c:when>
<c:otherwise>
 colors!
</c:otherwise>
<c:forEach var="num" items="${numArray}">
 <c:set var="num" value="100"/>
</c:forEach>
<c:forTokens var="num" items="${numList}" delims=",">
 ${num}
</c:forToken>

```

## URL Handling

JSTL Tags	Description
<c:url>	Rewrites URLs and encodes their parameters.
<c:import>	Accesses content outside the web application.
<c:redirect>	Tells the client browser to access a different URL .

Following code snippet shows the use of URL handling tags:

```

<c:url value="/page.html" var="pagename"/>
<c:import url="/content.html" var="newstuff"
scope="session"/>
<c:redirect url="/content.html"/>

```

## JSTL – SQL

- The JSTL SQL actions help to
  - Perform database queries (select)
  - Easily access query results
  - Perform database updates (insert, update, delete)
  - Group several database operations into a transaction

Database manipulation from the JSP is not recommended, but as a feature or facility SQL category helps perform the operations.

Following code shows how to query a database:

```
<sql: query var="customers" dataSource="${dataSource}">
SELECT * FROM customers
WHERE country = 'China'
ORDER BY lastname
</sql:query>
<table>
<c:forEach var="row" items="${customers.rows}">
<tr>
<td><c:out value="${row.lastName}" /></td>
<td><c:out value="${row.firstName}" /></td>
<td><c:out value="${row.address}" /></td>
</tr>
</c:forEach>
```

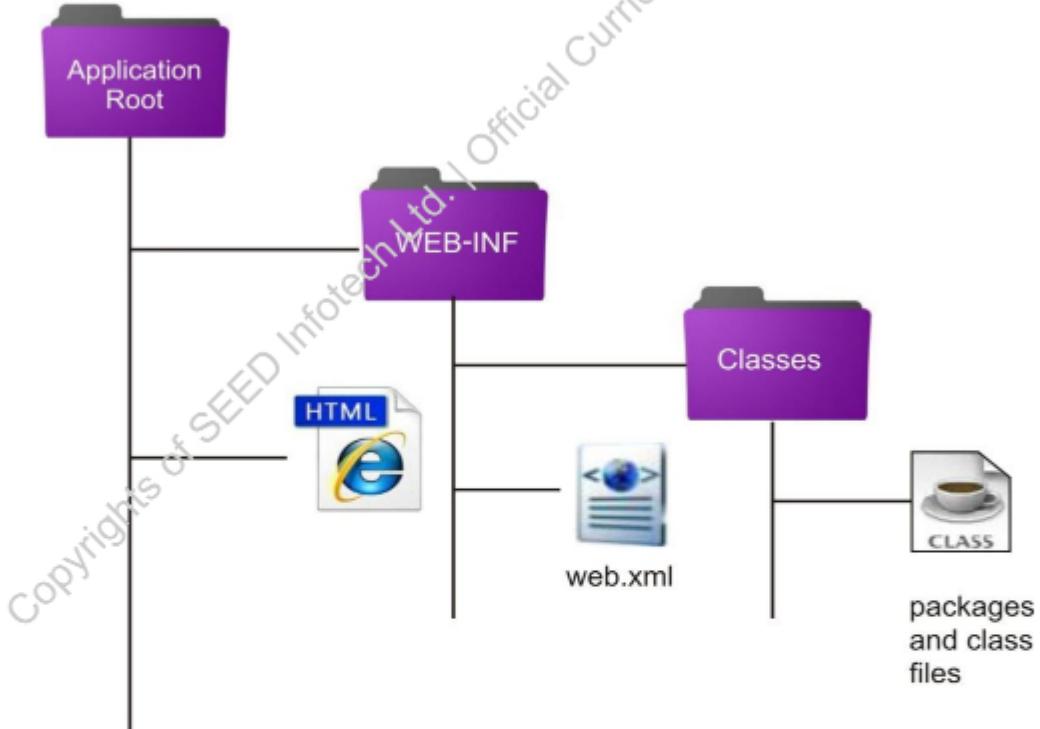
```
</table>
```

Following code shows updating a database:

```
<sql:transaction dataSource="${dataSource}">
<sql: update>
 UPDATE account SET Balance = Balance - ?
 WHERE accountNo = ?
 <sql:param value="${transferAmount}" />
 <sql:param value="${accountFrom}" />
</sql:update>
<sql:transaction [dataSource="dataSource"]
 [isolation= isolationLevel]>
 <sql:query> and <sql:update> statements
</sql:transaction>
```

## Appendix A - Web Application Directory Structure

This appendix covers web application directory structure.



Every web application have a specific directory structure to deploy the web application .The Application Root have WEB-INF folder. WEB-INF folder contain classes folder. Same levels to WEB-INF html, jsp, txt etc. files are maintained. In side WEB-INF and same level to classes folder web.xml is maintained. In side a classes folder \*.class files are available.

### **web.xml**

The web.xml file provides configuration and deployment information for the web applications/components that describe a web application is called “Deployment Descriptor (DD)”.

The structure of web.xml is:

```

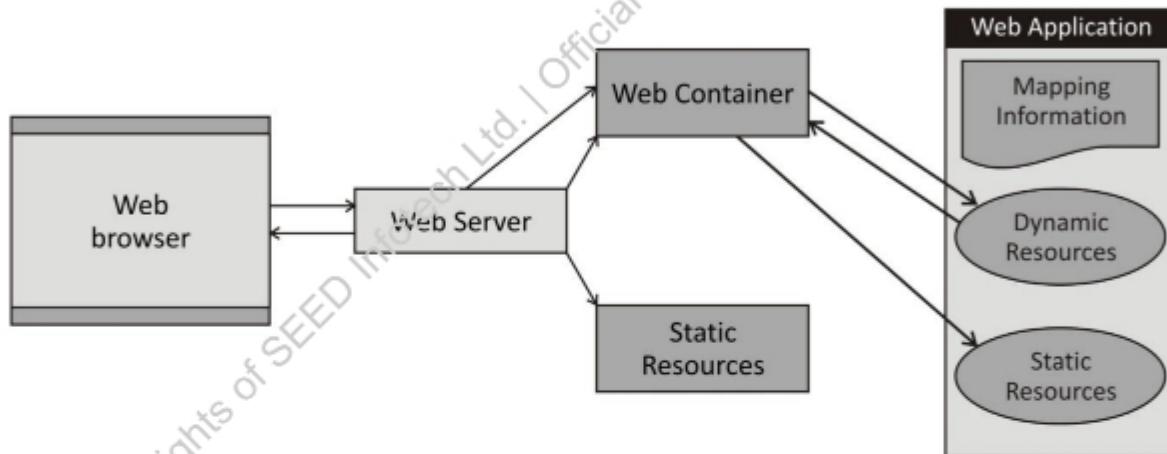
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-
app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
id="WebApp_ID" version="2.5">
 <display-name>LoginProject</display-name>
 <welcome-file-list>
 <welcome-file>index.html</welcome> welcome file list.
 </welcome-file-list>
 <servlet>
 <description></description>
 <display-name>LoginServlet</display-name>
 <servlet-name>LoginServlet</servlet-name>
 <servlet-
class>com.seed.servlet.LoginServlet</servlet-class>
 <init-param>
 <param-name>driver</param-name>
 <param-value>com.mysql.jdbc.Driver</param-value>
 </init-param>
 </servlet>
 <servlet-mapping>
 <servlet-name>LoginServlet</servlet-name>
 <url-pattern>/LoginServlet</url-pattern>
 </servlet-mapping>
</web-app>

```

## Appendix B - Request-Response Flow

This appendix covers request-response flow of web application.

### Request-Response Flow



### A typical Request-Response flow

1. User opens a web browser.
2. User submits a request to perform an operation.
  - a. User may wish to open a website for the first time
  - b. User may wish to perform some operation on the interface provided by the application
3. Browser generates an HTTP request and sends it to the destination.
  - a. Destination could be provided through the address bar or through the HTML components present on the interface
4. The website name is mapped to an IP address with the help of a Domain Name Server.
5. A Web Server receives the request at the specified IP address.
6. If the request is for a static resource (HTML file) maintained at the web server, it is rendered through an HTTP response immediately.
7. If the request is meant for a dynamic resource (JSP, Servlet), the server redirects the request to the associated web container holding the specific application.

8. Web Container receives the request and extracts the web application name from the URL of the request.
  - a. A web container may host more than one application. The name of the application differentiates the requests meant for specific application resources.
9. If the request is for a static resource (HTML file) within the application, it is rendered through an HTTP response immediately.
10. Web Container refers to the mapping information (web.xml) available in the web application for pin-pointing the specific dynamic resource within the web application.
11. If an object of the dynamic resource is not available in memory, it is created by the container.
12. The dynamic resource's lifecycle is executed by the container.
13. The response generated by the resource is rendered on the client's browser.
14. Same cycle runs for each and every request generated by the client's browser.

## **Java Web Component Developer**

---

### **Lab Manual and Appendix**

Copyright of SEED Infotech Ltd. | Official Curriculum

▶ ▶ ▶ **Contents**

Sr. No.	Chapter Name	Page No.
1.	Introduction to Lab Manual	283
2.	Java Database Connectivity(JDBC)	286
3.	Advanced JDBC	289
4.	HTML and JavaScript	292
5.	XML in Java	296
6.	Servlets	297
7.	Java Servlets (Session, Filters)	301
8.	JSP (Basic JSP, MVC and Action Tags)	308
9.	JSP (Custom Tags, EL and JSTL)	311
10.	Appendix C	312
11.	Appendix D	325

## Introduction to Lab Manual

This lab manual aims at giving complete understanding of the core concepts of the topics and applying them in the exercises.

The lab manual consists of a set of lab exercises defined chapter-wise. Each exercise has a definite objective defined. These objectives map with the terminal objectives defined at the beginning of each chapter. The problem statement is defined with clear instructions.

Some exercises have specific configuration or pre-condition mentioned. Advanced lab exercises are also given for some topics.

Appendix C given at the end of lab manual contains stepwise instructions to use eclipse IDE for building web applications.

## Configuration

jdk 1.7 should be installed. Tomcat 6 and above should be installed.

Specific configuration related to a particular exercise is mentioned. If it is not mentioned then the above configuration should be considered.

## Structure of Lab Manual

Lab manual consists of different sections. The explanation of each structure is given below.

### Objective

It states what you will achieve after completing a particular application. At the end of each lab exercise you should keep a track whether the objectives of that session are achieved.

### Configuration

It is an optional section and is present for specific lab exercises. If absent, then the configuration mentioned earlier should be considered.

### Pre-condition

You should have understood the concepts explained in a particular chapter in the courseware thoroughly to solve the exercises mentioned.

But some lab exercises may have pre-condition explicitly mentioned.

## Problem Statement

Problem statement for each lab exercise is given. It gives clear instructions to achieve the defined objective.

## How to use the Lab Manual?

Whenever a programmer has to transform a problem statement into a program which a computer can execute, the activity should be split in the following manner:

- Read the problem statement carefully. Hint is given for some problem statements to help you to solve the problem.
- Preparation
  - Decide the User interface (CUI or GUI) before hand.
  - Write the algorithm or steps to be followed to solve the problem. You can also draw flowcharts if required.
  - The program is made modular by writing functions. So pen down expected function prototypes and arguments on paper.
  - Also decide how one module (function) would communicate with other module (function).
- Give a dry run to the algorithm written.
- Write the code.
- Execute the code.

## Coding practices

The maintenance of code is easy if the code is written using coding practices. You should follow following coding practices while solving the lab exercises in this lab manual:

- Use meaningful names for variables, functions, file.
- Use uniform notation throughout your code.
- Code should be properly indented.
- Code should be commented. While documenting your code, write the purpose of your piece of code.

- What task is assigned to a method, what arguments are passed to it, what it returns should be clearly stated. Write a clear comment if you have added any statements for testing purpose.

Writing a right kind of well-documented software is an art along with your technical skills. Enough necessary documentation should be done. This makes the code easily maintainable.

## Chapter 1 - Java Database Connectivity(JDBC)

### Lab Exercise - 1

#### Objective

- Access data from the tables in the MySQL database and perform operations like insert, update and delete using JDBC.

**Configuration:** MySQL should be installed.

#### Pre-condition

- employee table should be present in the database. The table should have the fields employee\_id, name, salary and deptno.

#### Problem Statement 1

Display all employee information from the employee table at command prompt.

#### Problem Statement 2

Insert the following information in the employee table.

employee\_id=123,           ename=Prashant,           salary=12000,  
deptno=12

#### Problem Statement 3

Update employee salary to ₹15000 where employee\_id is 123.

#### Problem Statement 4

Delete all the records from the employee table.

### Lab Exercise - 2

#### Objective

- Perform transactions using JDBC.

#### Pre-condition

- account table should be present in the database. The table should have the fields accno, acc\_holder\_name, balance.

- account table should contain at least 2 records:

accno	acc_holder_name	balance
1001	Prashant	20000
1099	Anuradha	30000

- For SQL statements to create the structure, refer 'script.txt'.

### Problem Statement

Perform the account transaction between Prashant's account and Anuradha's account.

- a. Transfer the ₹2000 from Prashant's account to Anuradha's account.
- b. If transfer operation fails then rollback the transaction.
- c. If amount is successfully transferred from Prashant's account to Anuradha's account then commit the transaction.

### Hint

The signature of the method should be as follows:

```
public void transfer (Account from, Account to, int
amount)
{ . . . }
```

### Lab Exercise - 3

#### Objective

- Invoke the stored procedure present in the SQL Server database.

#### Pre-condition

- Stored procedure named "account\_rout" to update the record from the table. It should take accno as a parameter.
- For code of the procedure refer 'script.txt' .

### Problem Statement

Create a procedure for account table to update the balance.

**Hint**

Update the balance by ₹2000.

**Lab Exercise - 4****Objective**

- Use of resultset metadata.
- Use of database metadata.

**Configuration** MySQL should be installed.

**Pre-condition**

- employee table should be present in the database. The table should have the fields `employee_id`, `name`, and `salary`.

**Problem Statement 1**

Display all employee information in swing application from the `employee` table with its column names.

**Problem Statement 2**

Display all the database information in swing application like reserved keywords, table names, indexes etc.

## Chapter 2 - Advanced JDBC

### Lab Exercise - 5

#### Objective

- Retrieve data from the table in the MySQL database using scrollable resultset.
- Update data in the table in the MySQL database using updatable resultset

**Configuration** MySQL should be installed.

#### Pre-condition

- employee table should be present in the database. The table should have fields like employee\_id, name, and salary.

#### Problem Statement 1

Display all employees from last record to first record in a swing application from the employee table.

#### Problem Statement 2

Display only first and last record from employee table in swing application.

#### Problem Statement 3

Update the salary of all employees whose salary is less than Rs.15000 by Rs.2000 to the employee table. Display the updated records at the command prompt.

### Lab Exercise - 6

#### Objective

- Perform transactions using batch update.

#### Pre-condition

- account table should be present in the database. The table should have fields like accno, acc\_holder\_name, balance.
- account table should contain 2 records like:

accno	acc_holder_name	balance
1001	Prashant	20000
1099	Anuradha	30000

- SQL statements refer 'script.txt'.

### Problem Statement

Perform the account transaction between Prashant's account and Anuradha's account.

- a. Transfer the 2000/-Rs. from Prashant's account to Anuradha's account.
- b. If transfer operation fails than accounts should be restored to original state.
- c. If amount is successfully transferred from Prashant's account to Anuradha's account then commit the transaction.

### Hint

The signature of the method as below:

```
public void transfer (Account from, Account to, int
amount)
{ . . . }
```

## Lab Exercise -7

### Objective

- Insert and retrieve images using BLOB.

**Configuration** MySQL should be installed.

### Pre-condition

- employee table should be present in the database. The table should have fields like employee\_id, name, salary and emp\_photo.

### Problem Statement1

Insert the following information in employee table.

employee\_id=123, name=Joy, salary=12000,  
emp\_photo=<filename>.

### Problem Statement2

Display the all employee information with image from the employee table using swing UI.

### Lab Exercise - 8

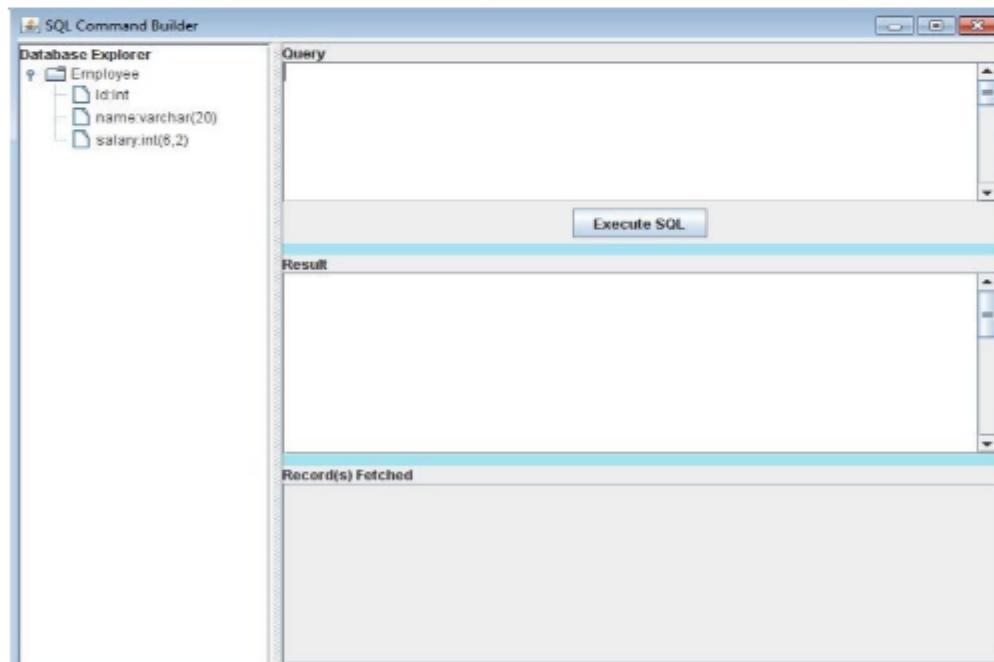
#### Objective

- Grip Assignment

**Configuration** MySQL should be installed.

#### Problem Statement 1

Create a Query builder which performs basic CRUD operations.



## Chapter 4 - HTML and JavaScript

### Lab Exercise - 9

#### Objective

- Construct static web pages using HTML and JavaScript.

**Configuration** IE or Mozilla Firefox should be installed.

#### Pre-condition

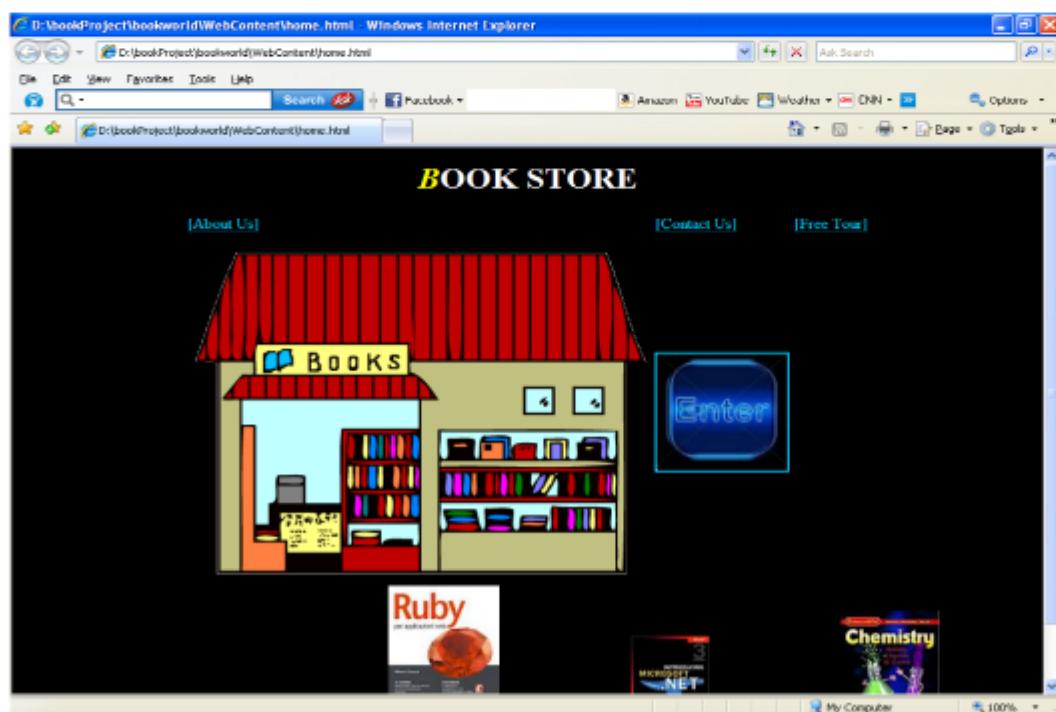
- Required images should be available.

#### Problem Statement

Create a website for bookshop having functionality using HTML.

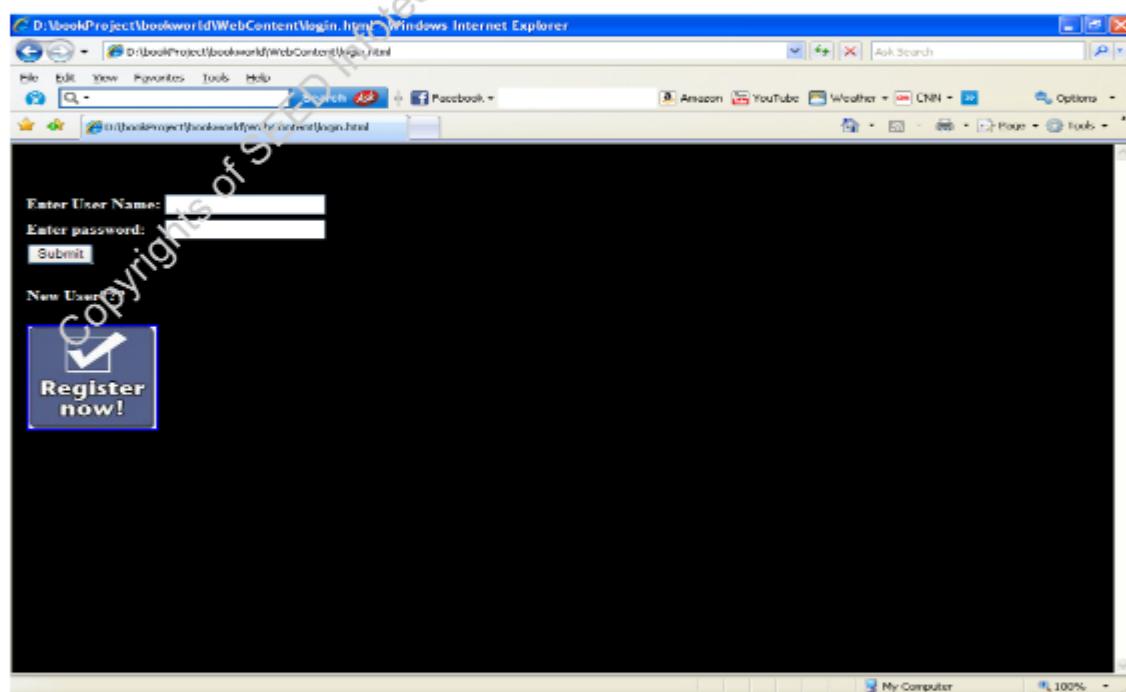
##### 1. home.html

- a. Web site heading on the top.
- b. Provide links About Us, Contact Us, Free Tour.
- c. Provide image hyperlink to enter the next page.
- d. Scroll some book images.



## 2. login.html

- a. Design login page which contains username and password field.
- b. Create an image link or button for new users – “register now!”
- c. Handle JavaScript validation as follows:
  - i. Username and password fields are mandatory.
  - ii. Password must be at least 8 characters long.
  - iii. Atleast one special character.e.g. is(!,#,\$,^,&) should be used in the password.



## 3. register.html

- a. This is registration page and should contain following fields:

fields	input type
username	text
password	password

confirm password	password
firstname	text
lastname	text
address1	textarea
address2	textarea
city	combobox (select and option tag)
state	combobox (select and option tag)
pin code	text
mobile number	text
email address	text

b. Handle JavaScript validation as follows:

- i. Username, password, firstname and lastname are mandatory.
- ii. Password and confirmed password must match with each other.
- iii. Zip code must be in number and must be 6 digits long.
- iv. Email validation check for @ and . characters. These characters must be in proper sequence.

Registration Form

New User Registration

(Field marked \* are necessary)

User Name\*:

Password\*:

Confirm Password\*:

First Name:  Last Name:

Address:   
Address2:

First Name:  Last Name:

Address:   
Address2:

City\*:   
State\*:   
Pin Code:   
Row No:   
Module No:   
Office No:   
Email Address:

Submit Reset

## Chapter 5 - XML in Java

### Lab Exercise - 10

#### Objectives

- Construct XML document.
- Construct DTDs and XSDs.

#### Problem Statement 1

“BookWorld.com” is an online book shop containing number of books belonging to different categories. It is possible to identify a specific book using the book id. Book Id can identify specific book name, author name, price of that book or edition of that book. Some books are available in different formats such as html or pdf. Some books are freely downloadable or downloadable on payment.

Using above details, identify the elements, attributes and create a DTD.

#### Problem Statement 2

Using above xml document create XSD.

#### Problem Statement 3

Using above DTD and XSD information indentify elements and attributes and create XML document.

#### Problem Statement 4

Create an application to parse an XML and display its contents using a SAX parser.

#### Problem Statement 5

Create an application to parse an XML and display its contents using a DOM parser.

## Chapter 6 - Servlets

### Lab Exercise - 11

#### Objective

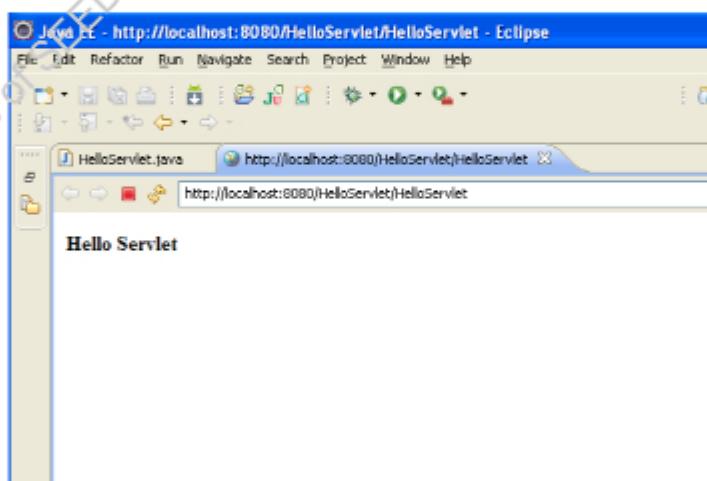
- Construct Servlet life cycle methods.

#### Pre-condition

- login.html should be created.

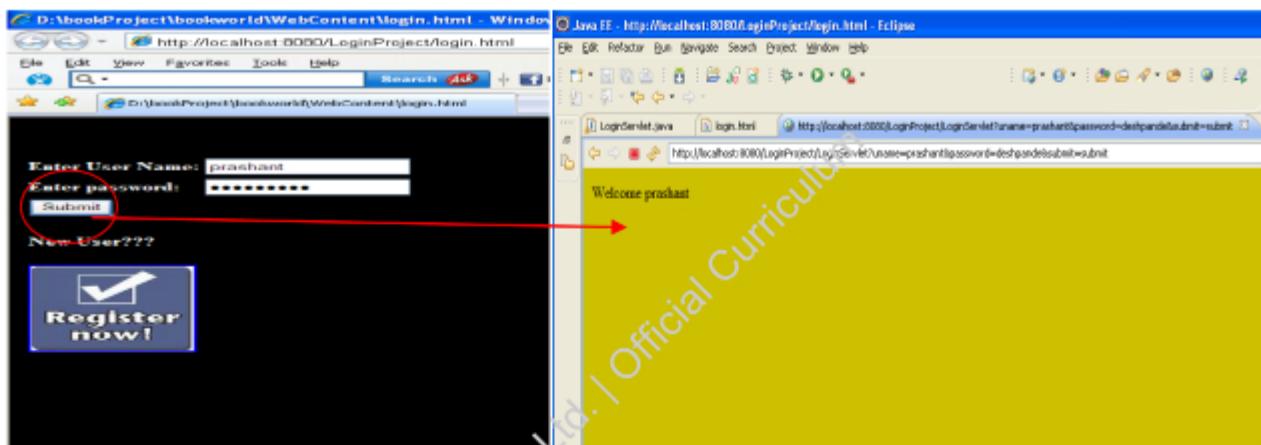
#### Problem Statement 1

Create a Servlet application which will give a response “Hello!!! Servlet”.



#### Problem Statement 2

Enter username and password in login.html and click on submit button. The request should be passed to LoginServlet which will check whether entered username and password are valid or not. If they are valid then print message “Welcome <username>” otherwise print “Incorrect username or password”.



## Lab Exercise - 12

### Objective

- Construct a Servlet application using database.

### Pre-condition

- Lab Exercise-11 should be created.

### Problem Statement1

Establish a connection with MySQL database and check that entered username and password are valid or not.

### Problem Statement2

Establish a connection with MySQL database and check that entered username and password are valid or not.

use Hint to solve this problem statement.

### Hint

Configure the database using either `ServletConfig` interface or `ServletContext` interface.

## Lab Exercise - 13

### Objective

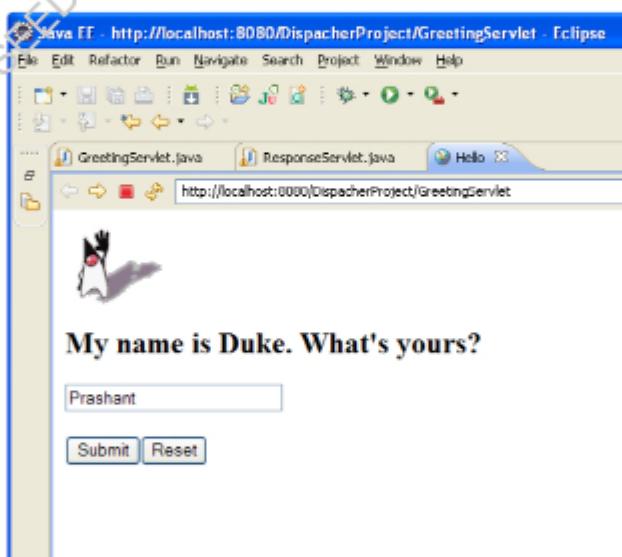
- Construct a Servlet application for Servlet collaboration.

### Pre-condition

- Lab Exercise - 12 should be created.

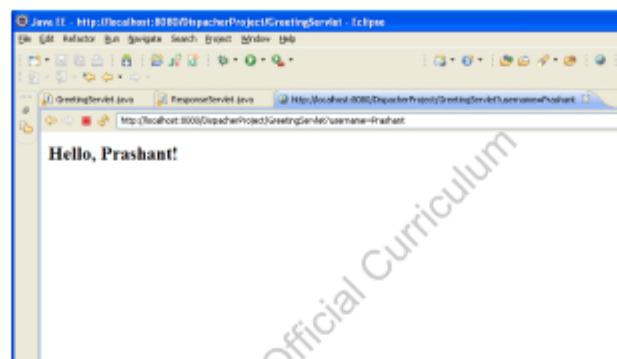
### Problem Statement

Create an html page inside GreetingServlet which accepts the username. On clicking the submit button the request should be passed to the second Servlet called ResponseServlet. This Servlet should generate a response Hello, <username>!

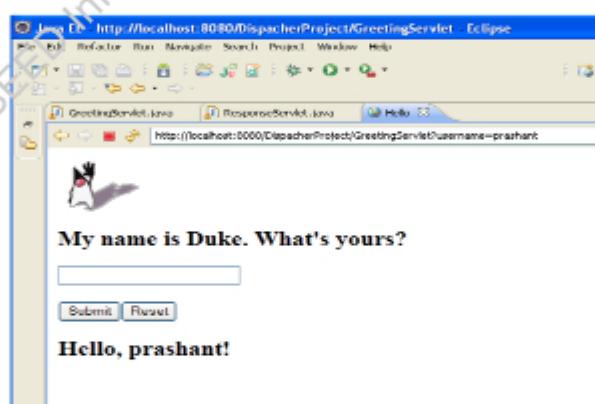


This response should be generated in two ways:

- i. Forward the response to second servlet and show the response in second servlet only.



- ii. Forward the response to second servlet and show the response along with the response of the first servlet.



## Chapter 7 - Java Servlets (Session, Filters)

### Lab Exercise - 14

#### Objectives

- Construct a Servlet application using session tracking mechanisms.

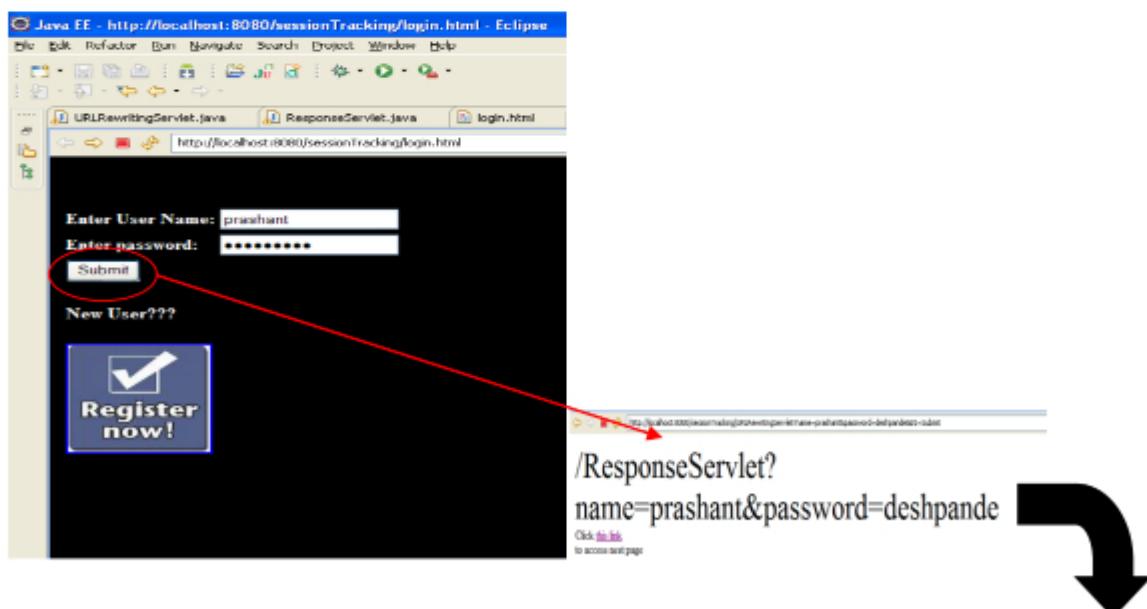
#### Pre-condition

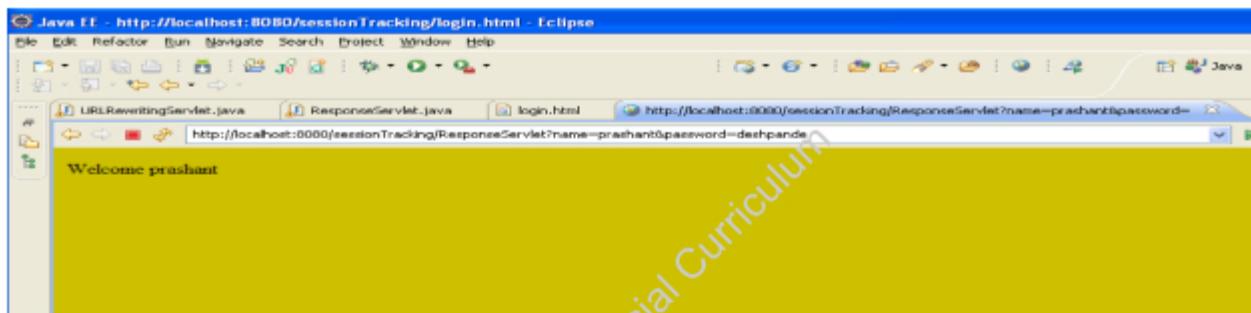
- Lab Exercise-11 should be created.
- 

#### Problem Statement 1

Enter username and password in login.html and click on submit button. The request should be passed to URLWritingServlet which should check whether entered username and password are valid or not. When user clicks on submit button request should be passed to the second Servlet called ResponseServlet. This Servlet should generate a response Hello, <username>! Otherwise print "Sorry <username>, try again".

Use URL rewriting mechanism to handle the session.

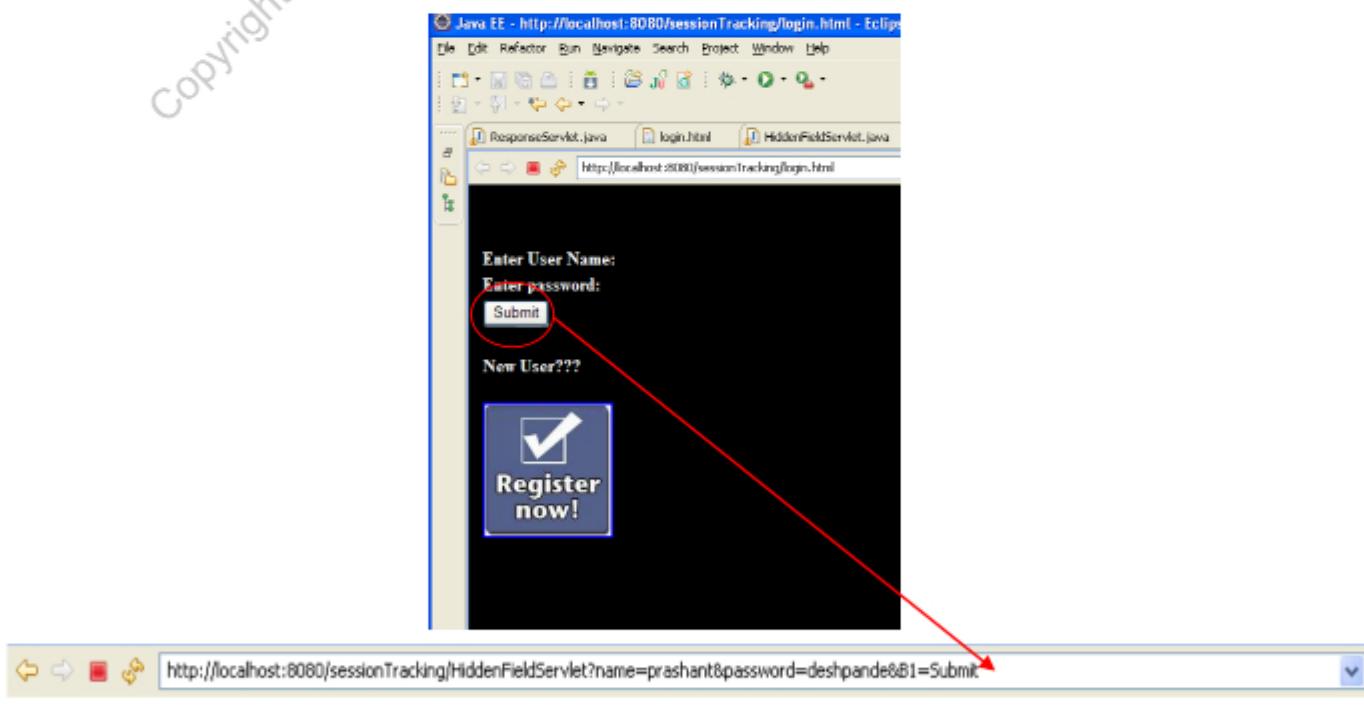


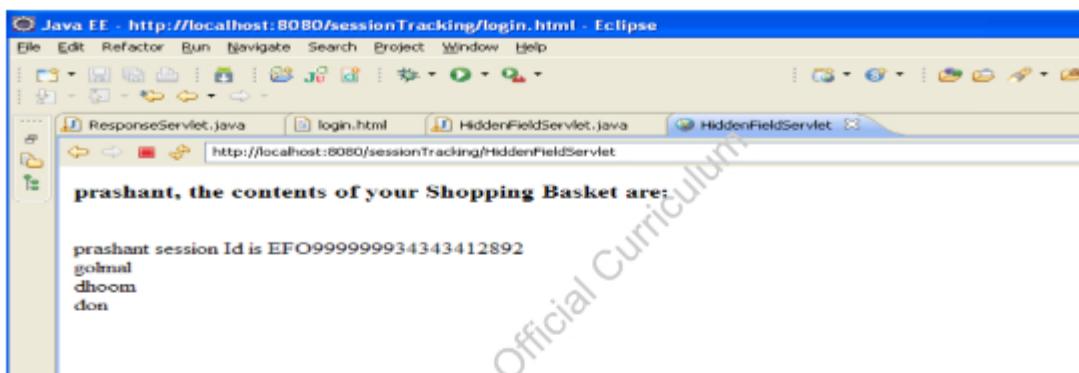


## Problem Statement 2

Enter username and password in login.html and click on submit button. The username and password are passed as a hidden parameter to the HiddenFieldServlet which should accept movie names and show all the movie names with username. The application should also show the sessionId.

Use hidden form field mechanism to handle the session.

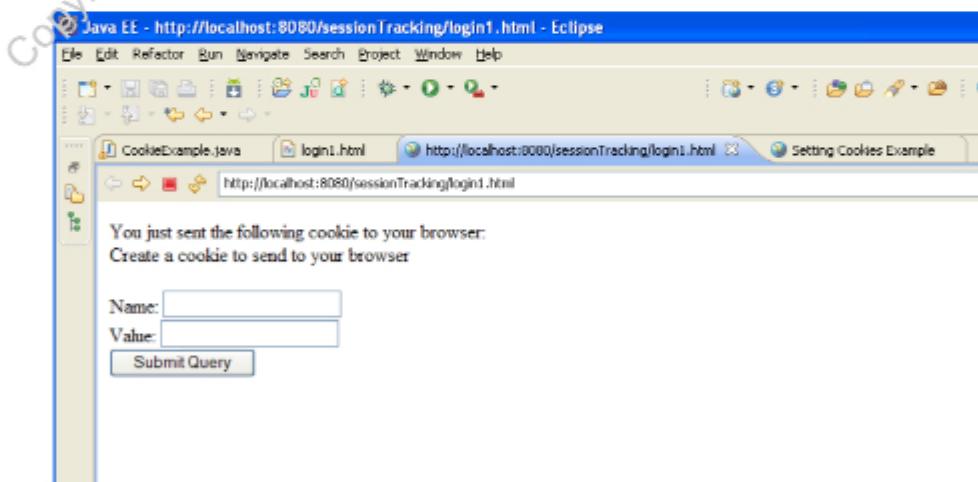




### Problem Statement 3

Enter username and password in a login.html and click on submit button. The username and password should be passed as a cookie parameter to the CookiServlet which should accept name and password and should print the name and password with the sessionId.

Use cookies mechanism to handle the session.





## Problem Statement 4

Create an html page inside an AttributeServlet which should accept the name and value. Clicking on update button, should show the name and value in a cart. If cookies are disabled then code should automatically execute using URL rewriting. Use same application for session tracking using HttpSession. Print sessionId and session creation time.



## Lab Exercise - 15

### Objectives

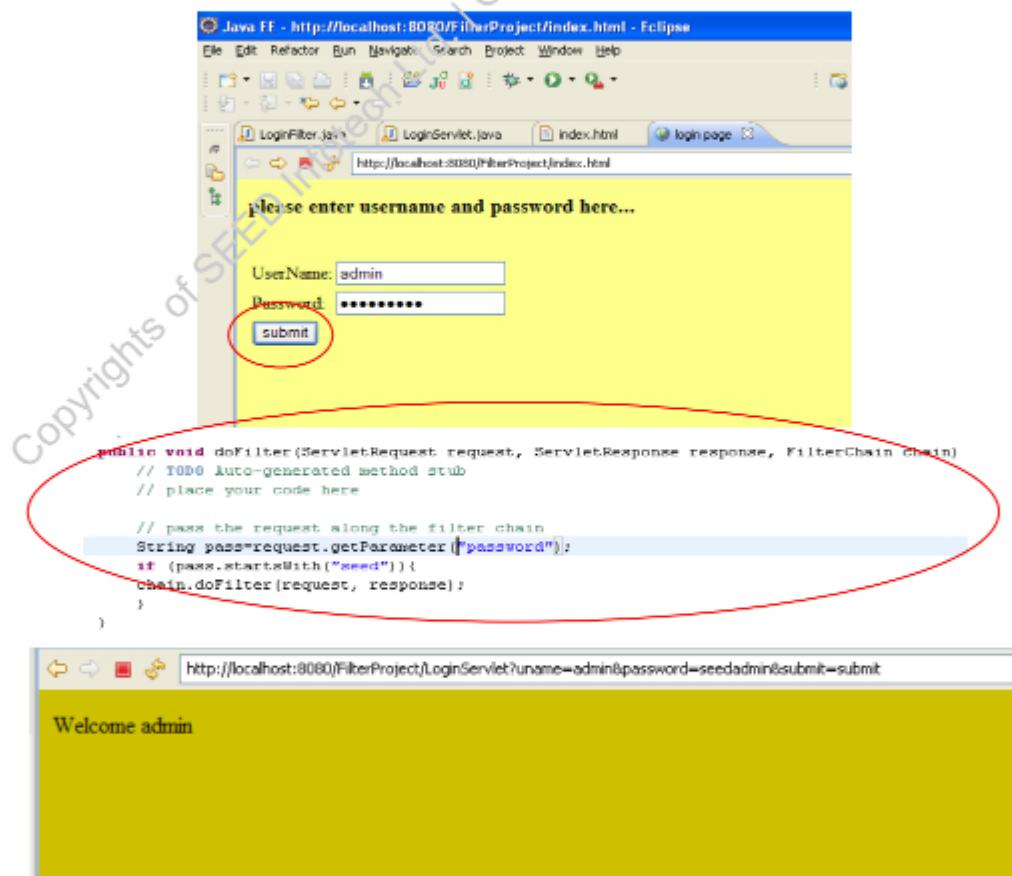
- Construct a Servlet application using filter.

### Pre-condition

- Lab Exercise-11 should be created.

## Problem Statement 1

Enter username and password in login.html and click on submit button. The request should be passed to LoginServlet which should check whether entered username and password are valid or not. Before request is passed to LoginServlet it should be passed through LoginFilter. If password is "seed" only then the request should be accepted otherwise the should be passed again to same html page i.e. login.html.



## Lab Exercise - 16

### Objectives

- Construct a Servlet application using listener.

### Pre-condition

- Lab Exercise-11 should be created.

### Problem Statement 1

Test Servlet life cycle using listeners.

```
context initialized
Jan 16, 2012 9:33:28 AM org.apache.coyote.http11.Http11Protocol start
INFO: Starting Coyote HTTP/1.1 on http-8080
Jan 16, 2012 9:33:28 AM org.apache.jk.common.ChannelSocket init
INFO: JK: ajp13 listening on /0.0.0.0:8009
Jan 16, 2012 9:33:28 AM org.apache.jk.server.JkMain start
INFO: Jk running ID=0 time=0/47 config=null
Jan 16, 2012 9:33:28 AM org.apache.catalina.startup.Catalina start
INFO: Server startup in 2007 ms
request initialized
Request Destroyed
request initialized
Request Destroyed

Jan 16, 2012 9:38:50 AM org.apache.coyote.http11.Http11Protocol pause
INFO: Pausing Coyote HTTP/1.1 on http-8080
Jan 16, 2012 9:38:51 AM org.apache.catalina.core.StandardService stop
INFO: Stopping service Catalina
context destroyed
```

### Pre-condition

- Lab Exercise-14 should be created.

### Problem Statement 2

Test Session tracking management using listeners.

```
cpntext initialized
Jan 16, 2012 9:37:24 AM org.apache.coyote.http11.Http11Protocol start
INFO: Starting Coyote HTTP/1.1 on http-8080
Jan 16, 2012 9:37:24 AM org.apache.jk.common.ChannelSocket init
INFO: JK: ajp13 listening on /0.0.0.0:8009
Jan 16, 2012 9:37:24 AM org.apache.jk.server.JkMain start
INFO: JK running ID=0 time=0/47 config=null
Jan 16, 2012 9:37:24 AM org.apache.catalina.startup.Catalina start
INFO: Server startup in 1232 ms
request initialized
session created http session
Request Destroyed
request initialized
Request Destroyed
request initialized
attribute added in HttpSession
Request Destroyed
Jan 16, 2012 9:38:50 AM org.apache.coyote.http11.Http11Protocol pause
INFO: Pausing Coyote HTTP/1.1 on http-8080
Jan 16, 2012 9:38:51 AM org.apache.catalina.core.StandardService stop
INFO: Stopping service Catalina
context destroyed
INFO: JK: ajp13 listening on /0.0.0.0:8009
Jan 16, 2012 9:55:43 AM org.apache.jk.server.JkMain start
INFO: JK running ID=0 time=0/47 config=null
Jan 16, 2012 9:55:43 AM org.apache.catalina.startup.Catalina start
INFO: Server startup in 1258 ms
http session destroyed
http binding removed
```

## Chapter 8 - JSP (Basic JSP, MVC and Action Tags)

### Lab Exercise - 17

#### Objective

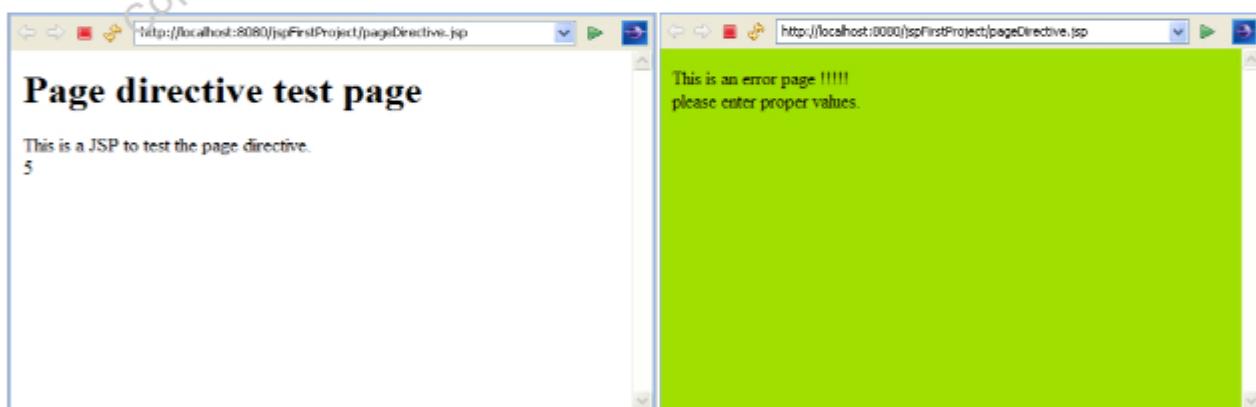
- Construct jsp applications using jsp directives and scriptlet.

#### Problem Statement 1

Write a “pageDirective.jsp” page to check the page directive with its attributes. Check `errorCode` and `isErrorHandler` attribute. If exception occurs then it should automatically pass to `errorPage` and display the error message.

```
<%@ page language="Java" import="java.util.*,java.util.*"
 session="true" buffer="12kb" autoFlush="true"
 info="my page directive jsp" errorPage="error.jsp"
 isErrorPage="false" isThreadSafe="true" %>
```

If some exception occurs then show `<errorPage>` (`error.jsp`) .



#### Problem Statement 2

Include and show the current date and copyright message in `include.jsp`. Generate current date in `incuded.jsp` and copyright message in `copywrite.html`.

`included.jsp`

`copyright.html`

Include these two files in `include.jsp`.



### Problem Statement 3

Check scriptlet tags using any simple 'C' based program such as printing table of 2.

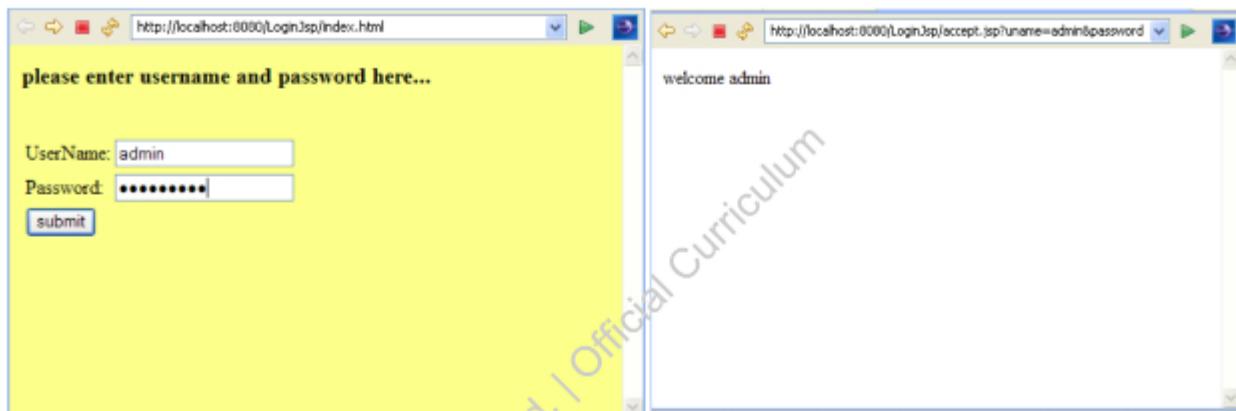
### Lab Exercise - 18

#### Objectives

- Include and forward action.
- Implicit objects.

#### Problem Statement

Enter username and password in a `login.html` and click on submit button. The request should be passed to `accept.jsp` which should check whether entered username and password are valid or not. If they are valid, then "welcome <username>" should be printed otherwise "Incorrect username or password" should be printed.



## Lab Exercise - 19

### Objective

- Using UseBean.

### Problem Statement

Accept a name and favorite programming language from the beans.html page and click on submit information button. This information should be passed to beans.jsp. This page should pass control to LanguageBean.java is a model class to handle the business logic. As a response a comment should be shown on the selected language.

The image shows two adjacent browser windows. The left window, titled 'http://localhost:8080/useBean/beans.html', is titled 'useBean action test page'. It contains a form with three fields: 'Please enter your username: prashant', 'What is your favorite programming language? Perl', and a 'Submit information' button. The right window, titled 'http://localhost:8080/useBean/beans.jsp', is titled 'useBean action test result'. It displays the following text:  
Hello, prashant.  
Your favorite language is Perl.  
My comments on your language:  
OK if you like incomprehensible code.

## Chapter 9 - JSP (Custom Tags, EL and JSTL)

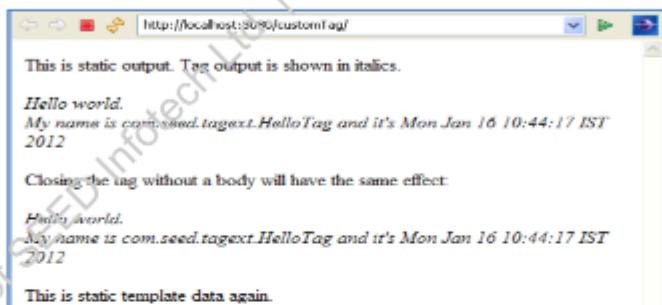
### Lab Exercise - 20

#### Objective

Using Custom Tag.

#### Problem Statement

Develop a “hello world” application which will print current date and time using custom tag.



### Lab Exercise - 21

#### Objective

- Use of JSTL and EL.

#### Problem Statement 1

Develop an EL application to check EL implicit objects and EL operators.



#### Problem Statement 2

Develop a JSTL application to check core tags.

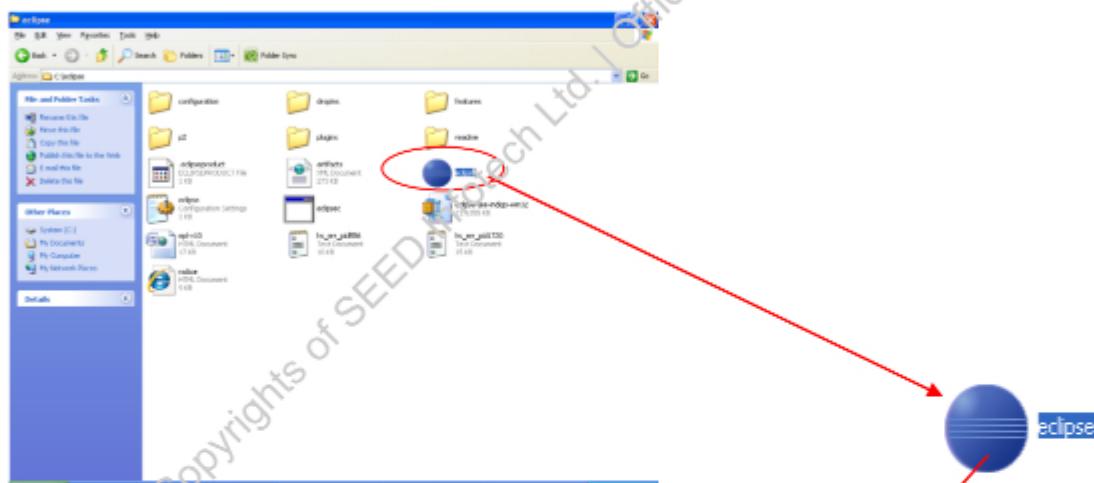


Appendix C

This document is a guideline to use eclipse.

## **1. To open Eclipse.**

Go to the eclipse folder and click on eclipse icon.



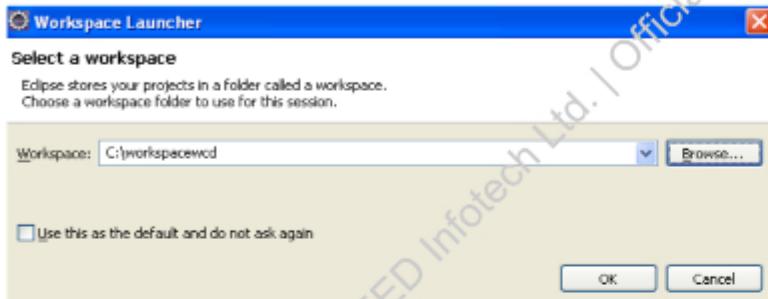
Once we click on eclipse icon, eclipse gets start.



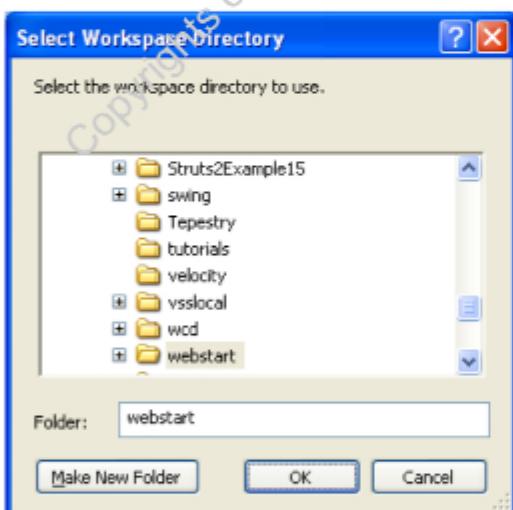
## 2. Create workspace

Once the eclipse gets start it asks for workspace. So, create Workspace on your local drive. Choose a workspace folder to use for this session.

**Workspace:** Eclipse stores your projects in a folder called a workspace.



Click on 'Browse' button to select your workspace.

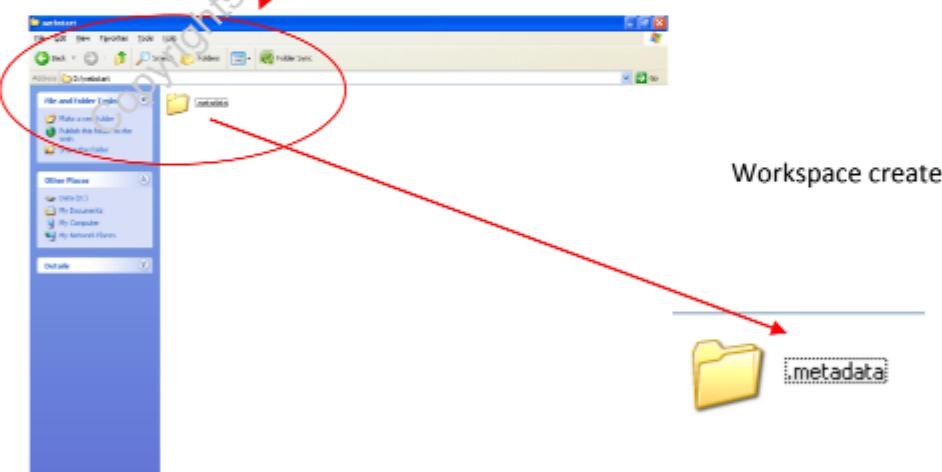


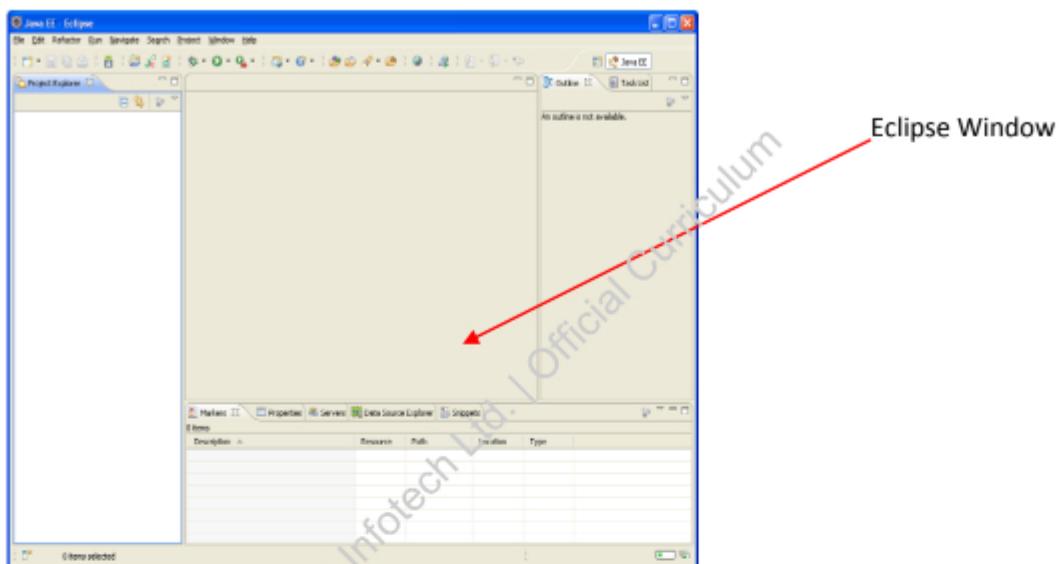
Once you click on the 'Browse' button after that new window appear 'select Workspace Directory'.

Select your folder if already created or create new folder and than select.

To create new folder click on button 'Make New Folder' and finally click on 'OK' button.

Once you click on 'OK' button .It starts to create workspace for you and eclipse create workspace on your current directory with a folder name .metadata. At the same time it opens new window which is something called it as Eclipse window.

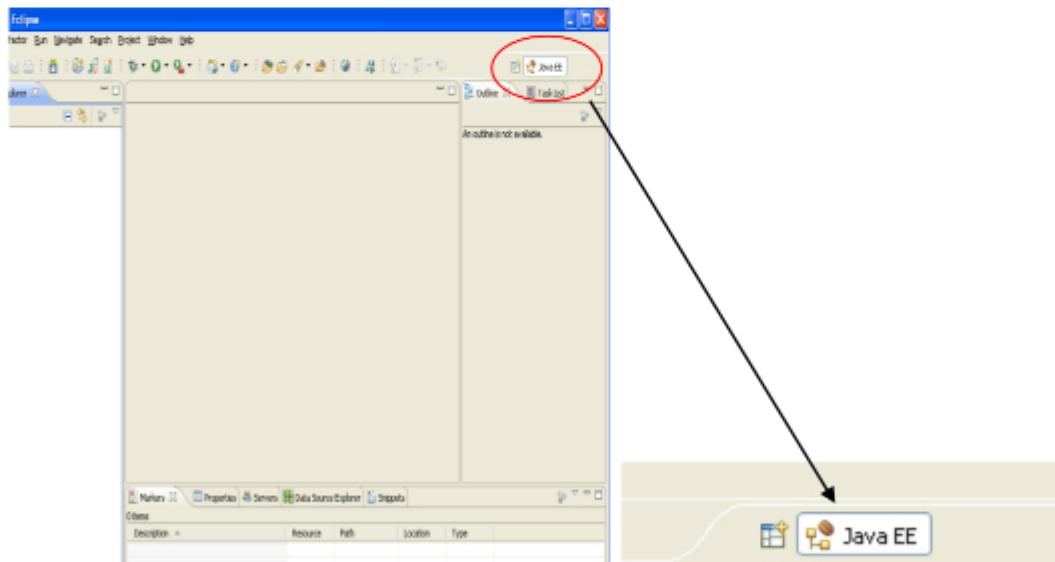




### 3. Select your perspective

Eclipse is an ID for the different types of applications like using this ID we can execute Core Java applications as well as java server based applications and so on. When we want to execute simple core java related application required core java specific tools and properties. When we want to execute server based application we required server environment so its server perspective.

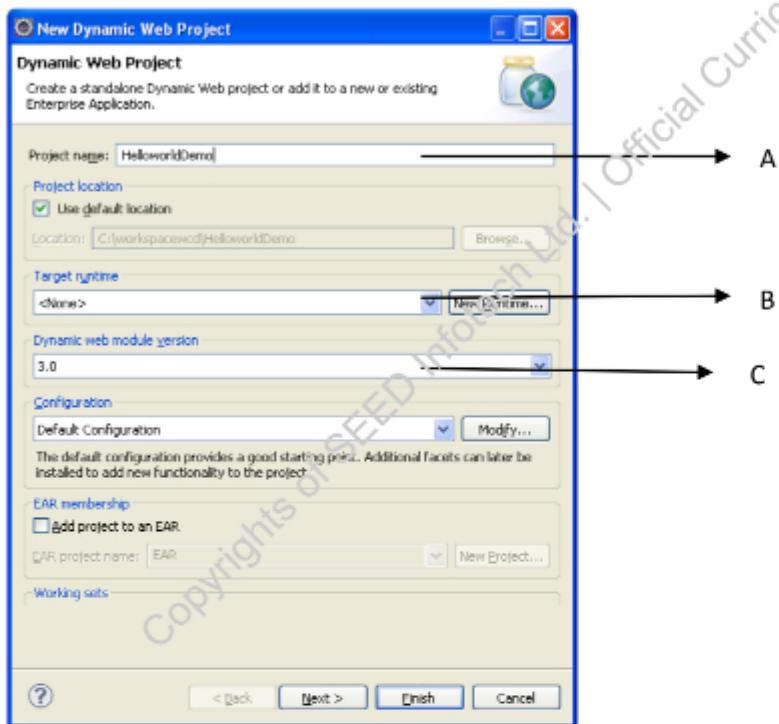
So, According to your requirement select perspective



#### 4. Create new Project.

Click on File menu:

File → New → Dynamic Web Project



new Project window appear. Now enter some information to create new Project like:

a. Project name : enter your Project name e.g. HelloServletDemo

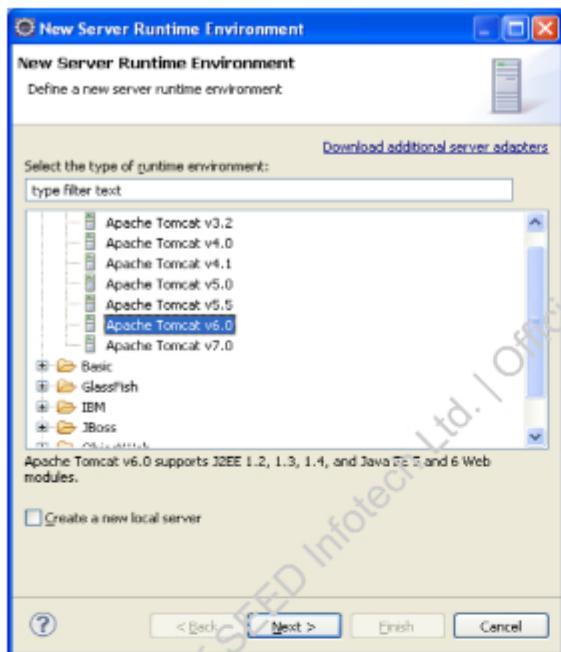
Project name:  A

b. Target runtime like

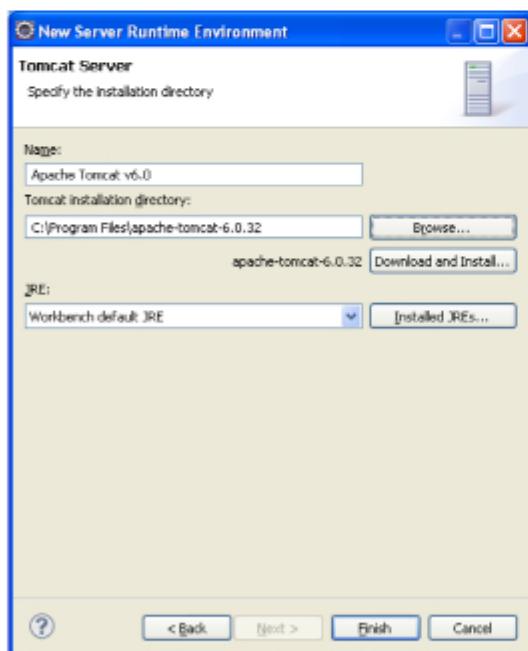
Use an execution target server: select your specific server e.g. Tomcat 6

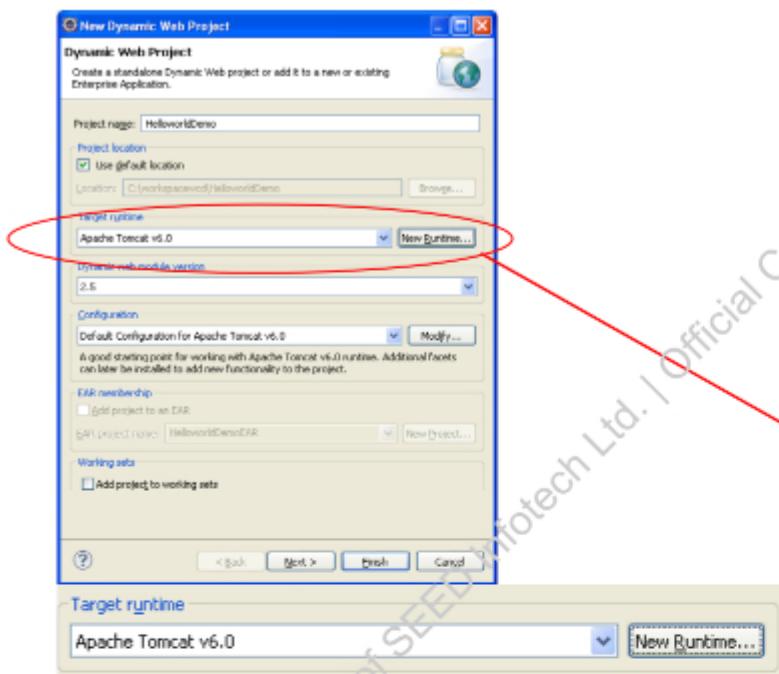
Target runtime  
<None> B

By default Target runtime is <none>, now click on New Runtime button to select specific server.



Click on next button and browse the server installation directory and finally click on finish button.

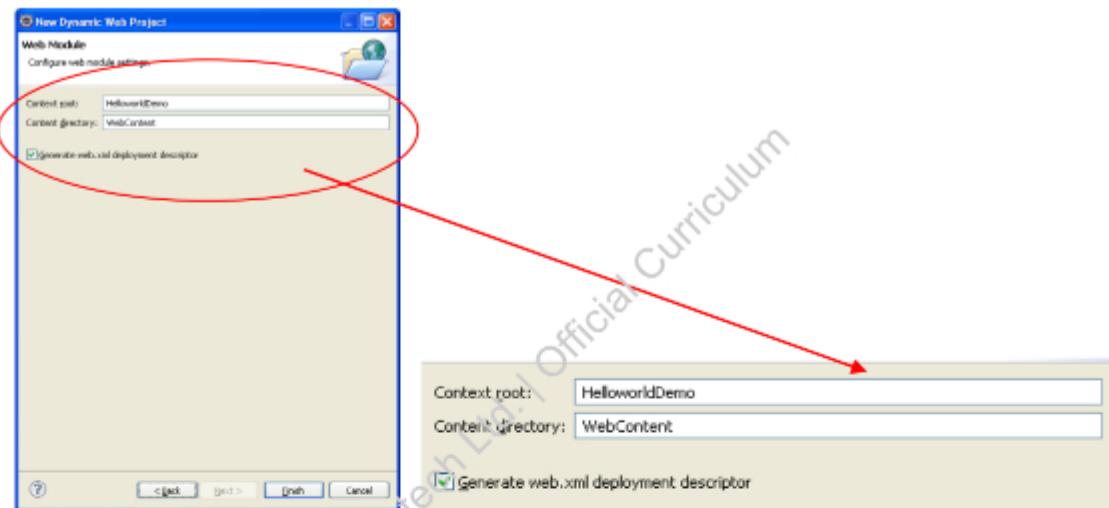




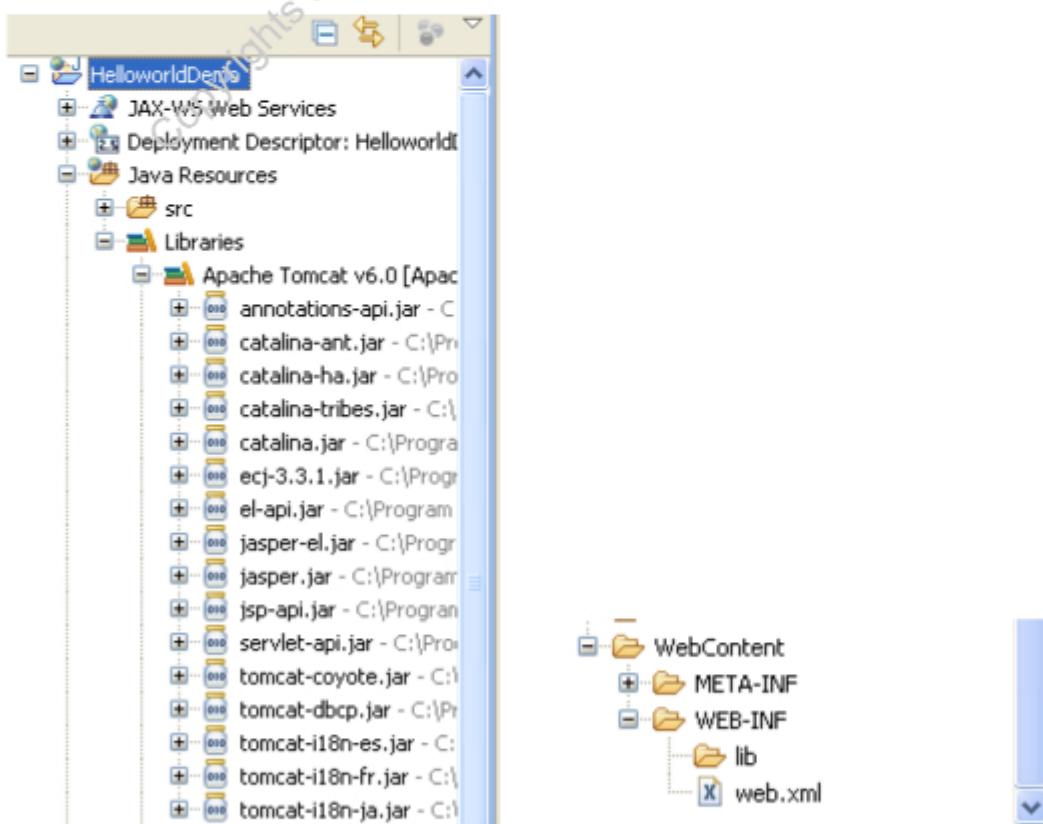
c. Dynamic web module version: This will give information like server support to specific version applications. e.g. Tomcat 6 support up to servlet 2.5



After that click on 'next' button. It will appear New Dynamic Web Project window. This window contain context root and contet directroy and one check mark for generate web.xml deployment descriptor automatically.



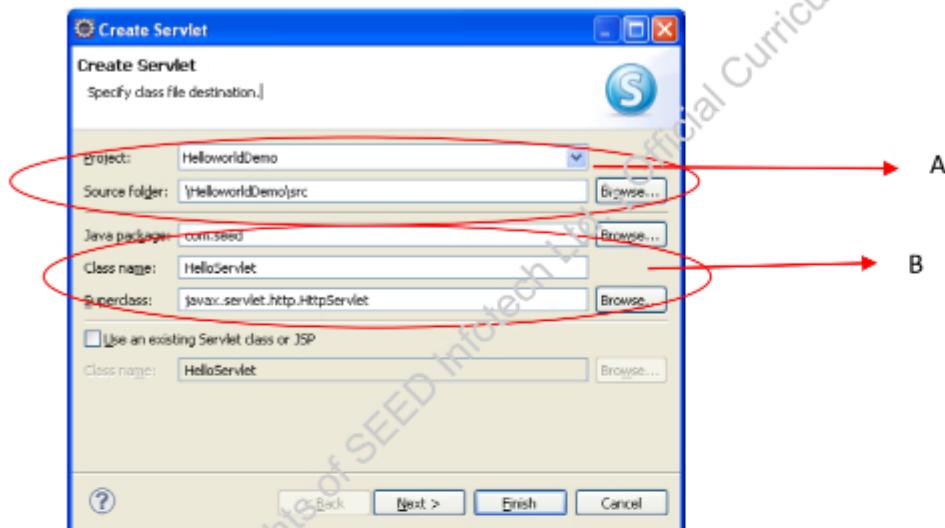
Once we click on finish button web application project create with required lib files i.e with server environment.



## 5. Create Servlet File.

Select your current project and after that click on File menu.

File → New → Servlet



create servlet class window appear.create a new servlet class for that purpose enter some information

like:

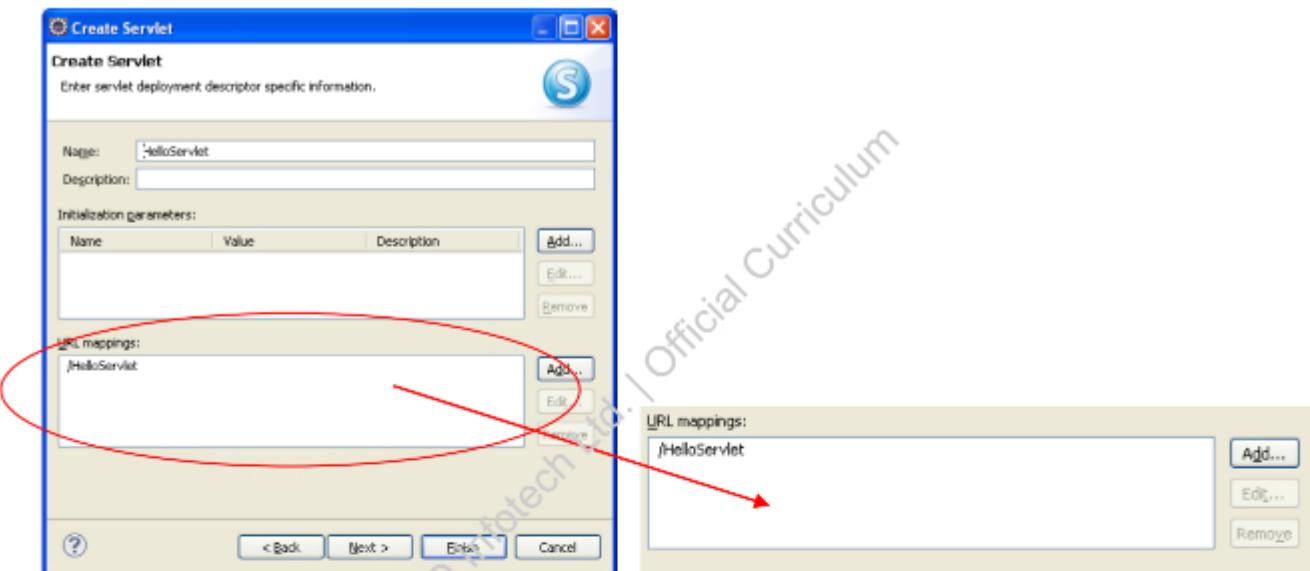
- a. Project and source folder: enter your folder name or source location by default it is your project name src folder. e.g. \HelloworldDemo\src



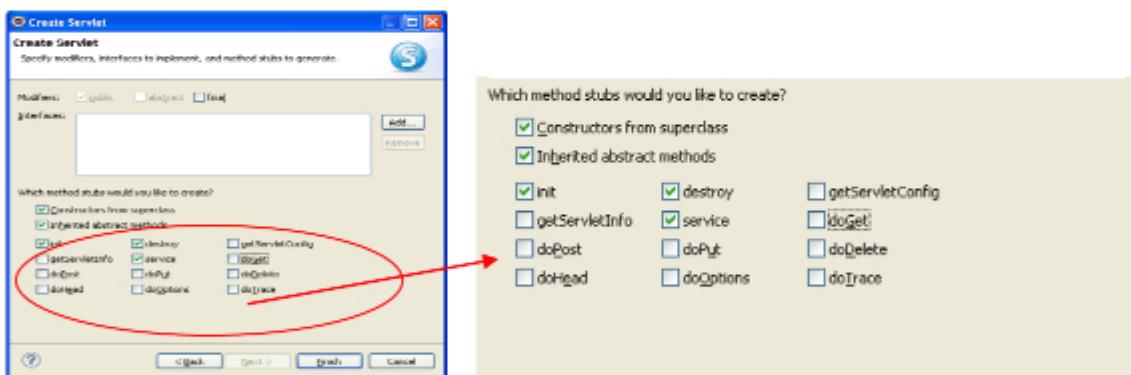
- b. Java Package and class name :enter package name if required.If you not maintain package name it will take default package C for the class name.It is mandatory.Servlet class by default extends to HttpServlet.



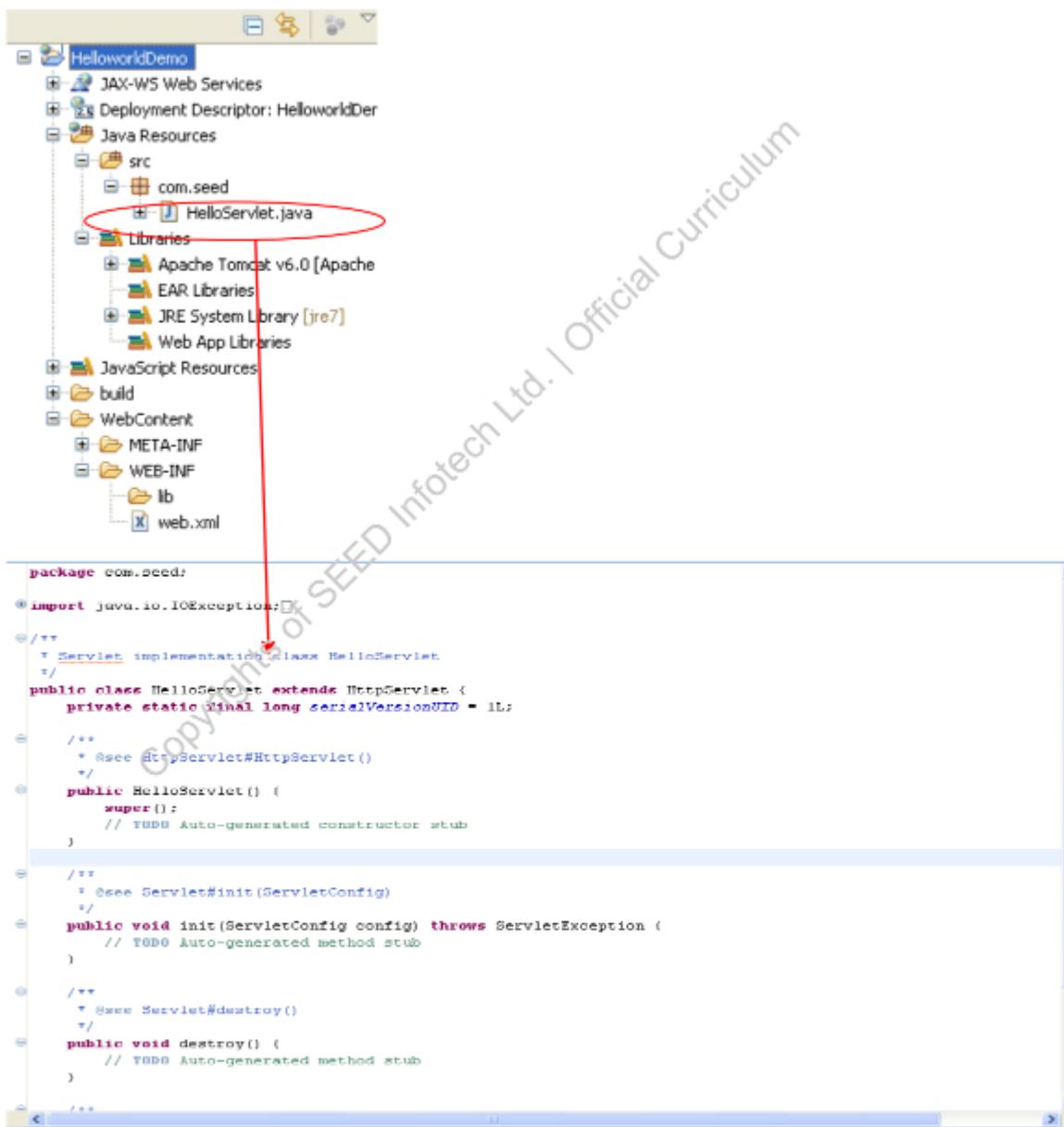
After that click on 'next' button and set url mapping parameter.Keep is as default or edit the url mappings.



After that click on next button to select required methods in servlet application.



Once we click on 'finish' button servlet class create under the web application environment.



## 6. Execution of Web Application

Lets take a example simple 'HelloServlet' program which will print 'Hello Servlet!!!'

```

HelloServlet.java
/*
 * @see HttpServlet#HttpServlet()
 */
public HelloServlet() {
 super();
 // TODO Auto-generated constructor stub
}

/*
 * @see Servlet#init(ServletConfig)
 */
public void init(ServletConfig config) throws ServletException {
 // TODO Auto-generated method stub
}

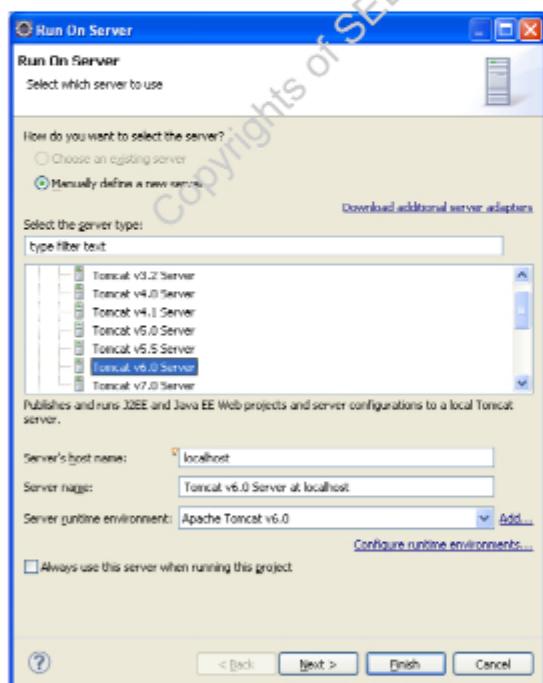
/*
 * @see Servlet#destroy()
 */
public void destroy() {
 // TODO Auto-generated method stub
}

/*
 * @see HttpServlet#doService(HttpServletRequest request, HttpServletResponse response)
 */
protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException,
 // TODO Auto-generated method stub
 PrintWriter out=response.getWriter();
 out.println("Hello Servlet !!!");
}

```

To execute or run the web application

Run → Run As → Run on server

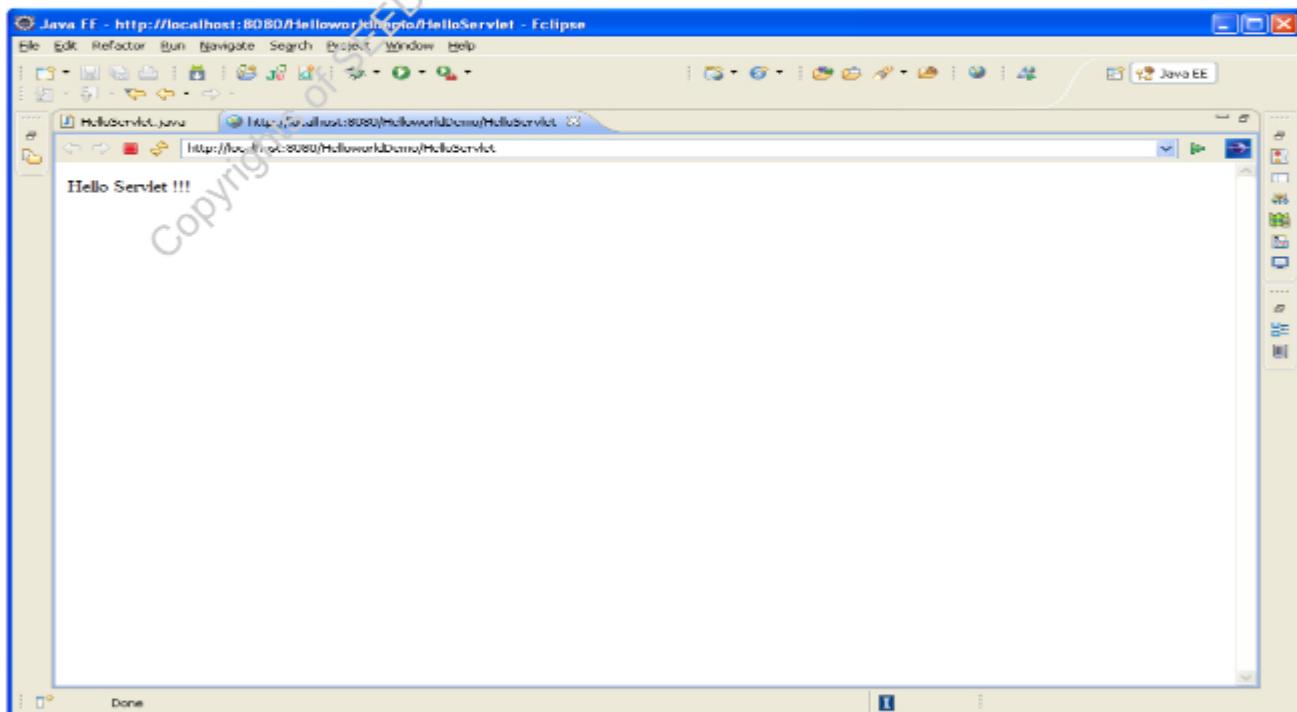


click on next and finish.



```
Tomcat v6.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\jre7\bin\javaw.exe (Jan 16, 2012 1:52:48 PM)
Jan 16, 2012 1:52:49 PM org.apache.catalina.core.AprLifecycleListener init
INFO: The APR based Apache Tomcat Native library which allows optimal performance in produc
Jan 16, 2012 1:52:49 PM org.apache.tomcat.util.digester.SetPropertiesRule begin
WARNING: [SetPropertiesRule]{Server/Service/Engine/Host/Context} Setting property 'source'
Jan 16, 2012 1:52:50 PM org.apache.coyote.http11.Http11Protocol init
INFO: Initializing Coyote HTTP/1.1 on http-8080
Jan 16, 2012 1:52:50 PM org.apache.catalina.startup.Catalina load
INFO: Initialization processed in 1475 ms
Jan 16, 2012 1:52:50 PM org.apache.catalina.core.StandardService start
INFO: Starting service Catalina
Jan 16, 2012 1:52:50 PM org.apache.catalina.core.StandardEngine start
INFO: Starting Servlet Engine: Apache Tomcat/6.0.32
Jan 16, 2012 1:52:51 PM org.apache.coyote.http11.Http11Protocol start
INFO: Starting Coyote HTTP/1.1 on http-8080
```

Program gets execute and it will print output in side a browser.



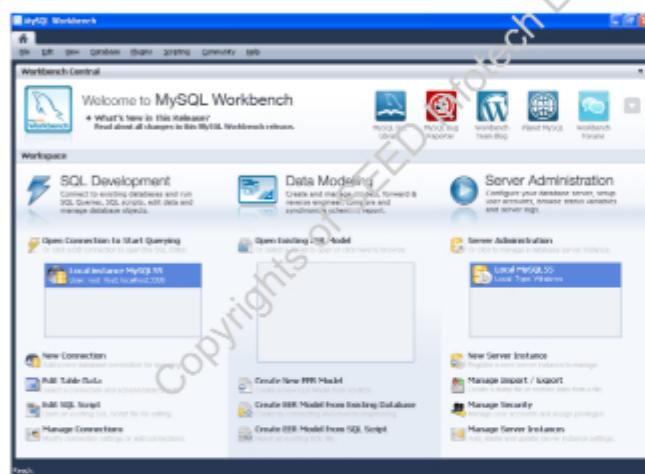
## Appendix-D

This document is a guide to use MySQL database.

### 1. To Open database and establish the connection.

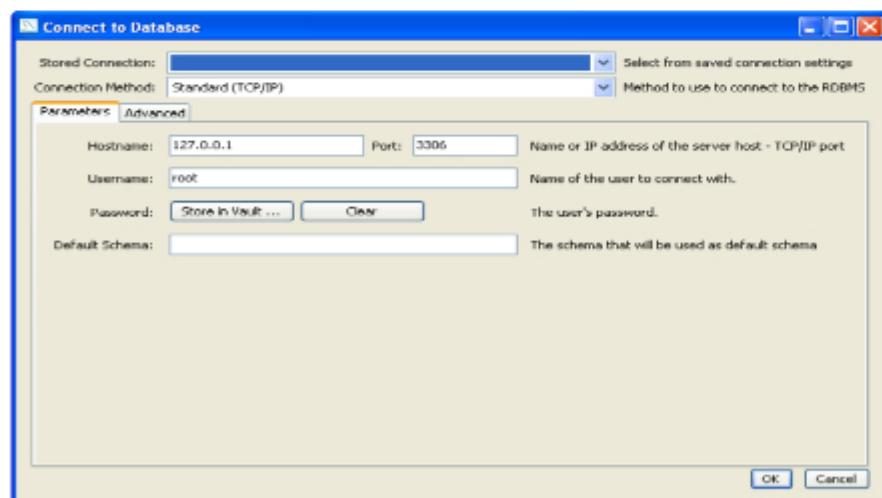
#### Step 1

Open MySQL database and click on 'open connection to Start Querying' for login purpose.



#### Step 2

Connect to Database Window opens.



Connect to Database:

Enter database specific information as follows and click on 'OK'.

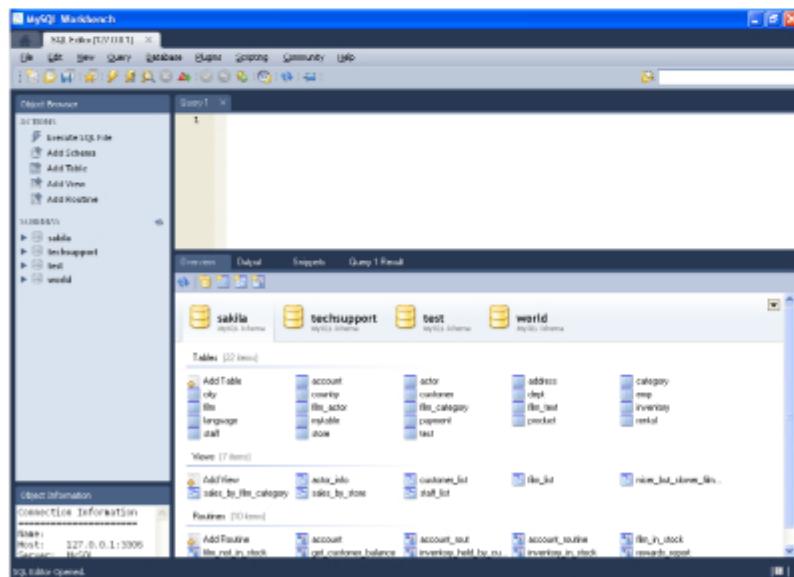
- Hostname: (name or IP address of server host) e.g. 127.0.0.1
- Port: (TCP/IP Port) e.g. 3306
- UserName: (name of the user to connect with) e.g. root
- Password: (the users password) e.g. myroot
- After that click on OK button.
- 

### Step 3

Enter the password to connect the MySQL service and click on OK button.



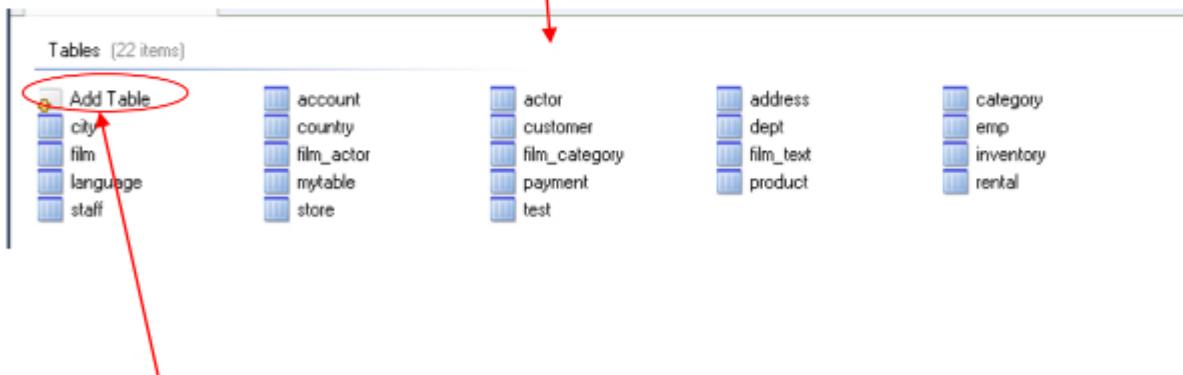
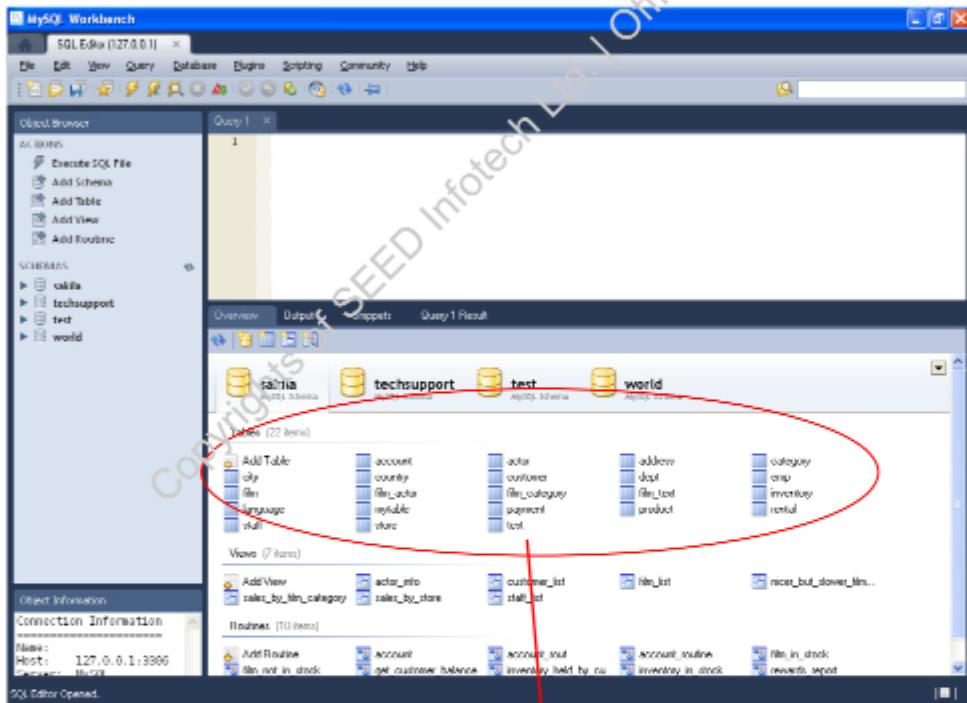
This will give you MySQL Workbench window.



## MySQL Workbench:

MySQL Workbench is a GUI screen that used to write SQL statements .It is SQL editor. Here tables and related operations like select, insert, update and delete using MySQL wizard. Using this wizard tables views SQL stored Procedures can also be created. This Workbench provides some default schemas custom schemas.

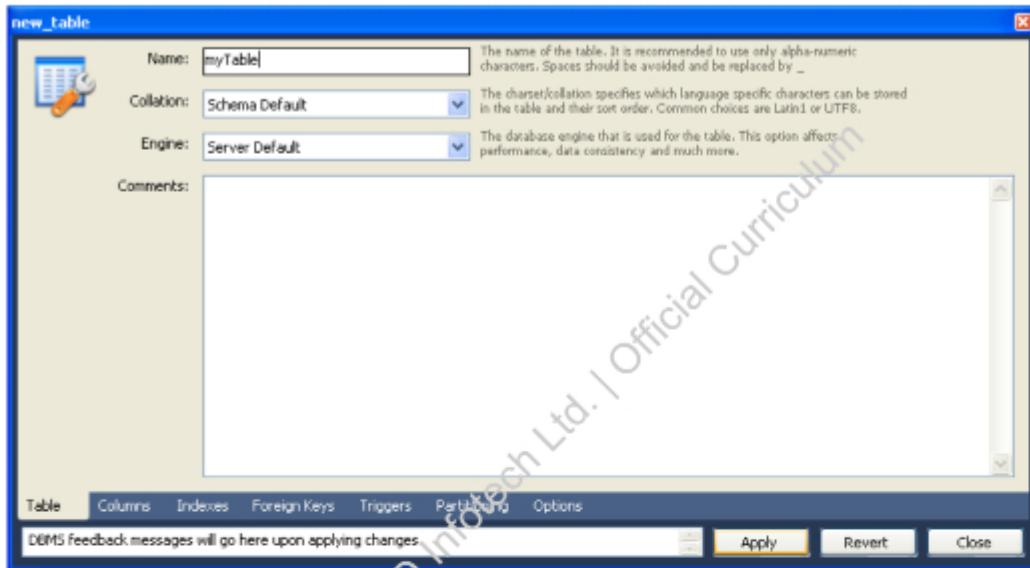
### 2. To create table in MySQL.



#### Step 1

Click on the 'Add Table' to create a new table.

'new table' window come into view.

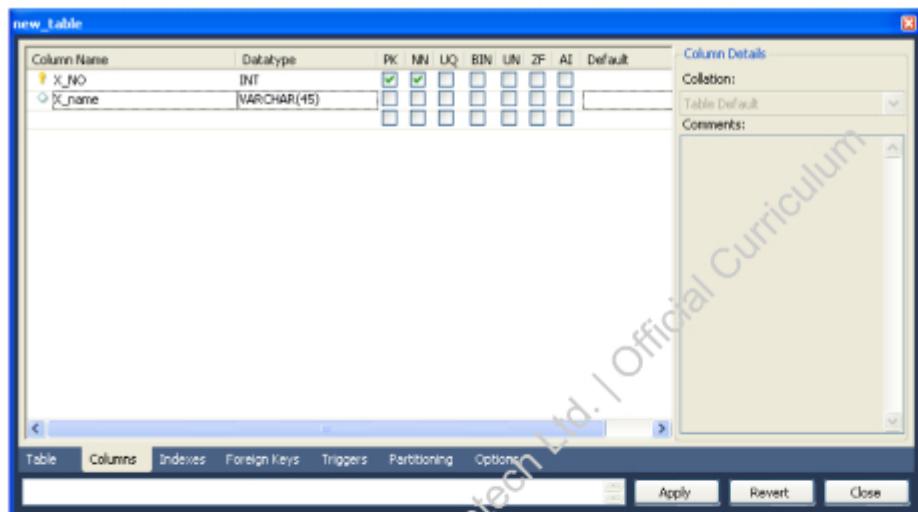


Enter some information like:

- Name: (The name of the table) spaces should be avoided if required use '\_'.recommended use only alpha numeric characters. This information is mandatory.
- Collaction: The charset specifies which language specific characters. Common choices are Latin1 or UTF8.Recommendation is keep default value.
- Engine: The database engine that is use for table. Recommendation is to use keep default values only.
- Comments: optional information.

## Step 2

Select Column tab to enter field information.



Enter some information like:

1. Column name
2. Datatype
3. Constraints
  - a. PK: Primary Key
  - b. NN: Not Null
  - c. UQ: Unique Index
  - d. BIN: Is Binary Column
  - e. UN: Unsigned data type
  - f. ZF: Fill up values to that column with 0's if its is numeric
  - g. AI: Auto Incremental

After that click on 'Apply' button.

### Step 3

Review the SQL script on the database and click on 'Apply' button.



Once you click on 'Apply' button SQL script will be applied to the database.  
Click on 'Finish' button.

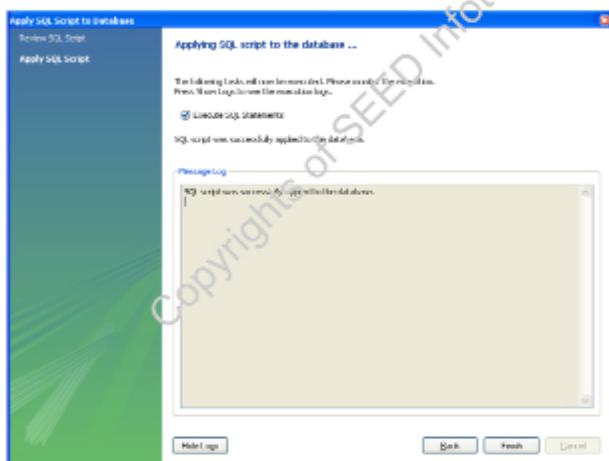


Table will now be created in the database. The snapshot given below.

The screenshot shows the MySQL Workbench interface. On the left, the 'Query 1' editor displays the following SQL code:

```

1 delimiter $$;
2
3 CREATE TABLE `mytable` (
4 `X_NO` int(11) NOT NULL,
5 `X_name` varchar(45) DEFAULT NULL,
6 PRIMARY KEY (`X_NO`)
7) ENGINE=InnoDB DEFAULT CHARSET=utf8$$
8
9

```

An arrow points from the 'mytable' entry in the 'Tables' list below to the 'mytable' entry in the query editor. The 'mytable' entry in the query editor is highlighted with a red circle.

Tables (23 items)	
Add Table	account
city	country
film	film_actor
language	mytable
rental	staff
	mytable
Views (7 items)	

### 3. To insert record in a database table.

The screenshot shows the MySQL Workbench interface. On the left, there's a tree view of 'Tables (23 items)' and 'Views (7 items)'. A red arrow points from the 'myTable' entry in the 'Tables' list to the 'myTable' entry in the 'Views' list. The main area is a query editor titled 'Query 1 Result' with the following content:

```

Overview Output Snippets Query 1 Result* Query 1 Result 2
[SQL] [View] [Edit] [Insert] [Replace] [Drop] [Truncate] [Empty] [Refresh] [New Row] [Run] [Cancel]
+----+----+
| no | name |
+----+----+
| 1 | aaa |
| null | null |
+----+----+

```

Click on the table name so that Query Result window appears.

The screenshot shows the MySQL Workbench interface. On the left, there's a tree view of 'Tables (23 items)' and 'Views (7 items)'. A red arrow points from the 'myTable' entry in the 'Tables' list to the 'myTable' entry in the 'Views' list. The main area is a query editor titled 'Query 1 Result' with the following content:

```

Overview Output Snippets Query 1 Result* Query 1 Result 2
[SQL] [View] [Edit] [Insert] [Replace] [Drop] [Truncate] [Empty] [Refresh] [New Row] [Run] [Cancel]
+----+----+
| no | name |
+----+----+
| 1 | aaa |
| null | null |
+----+----+

```

Enter the values.

OR

Alternative way to insert records in side database table is :

Step 1: Right click on table name

Step 2: Send to SQL editor

Step 3: Insert Statement

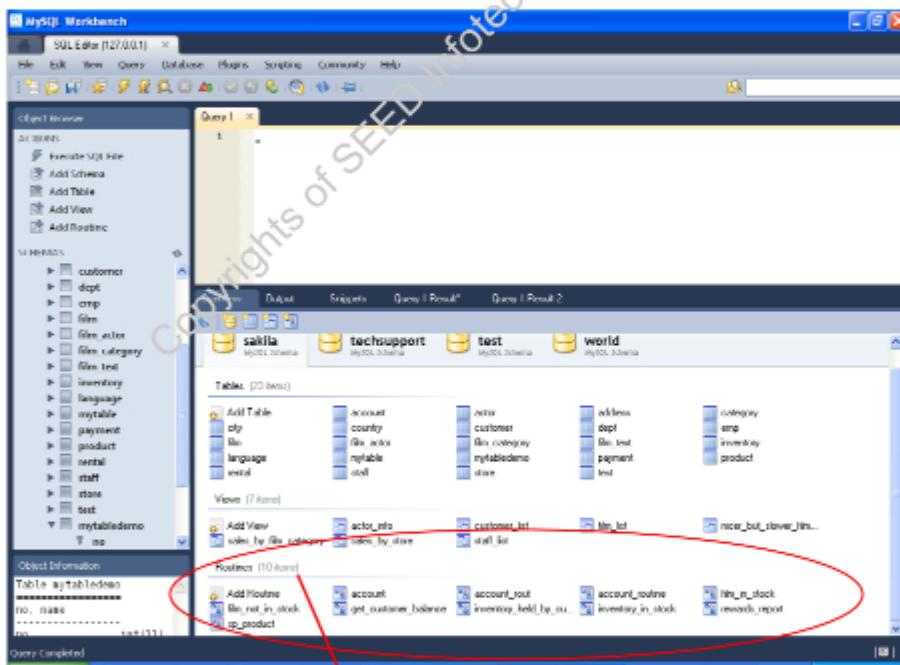
This will give you general syntax of Insert statement where you just enter your values like:

```
Query 1
1 • INSERT INTO sakila . myTable
2 (X_no ,
3 X_name)
4 VALUES
5 (1 , 'Aname') ;
6
```

Step 4: Select all query statement and click on 'Query' menu and then click on 'Execute All'.

Step 5: Record will be created in the database table.

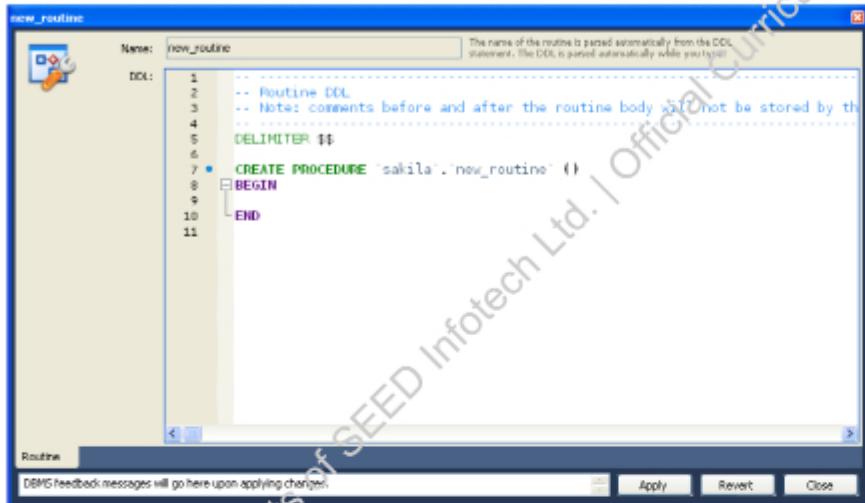
#### 4. To create stored procedure (Routine).



## Step 1

Click on the 'Add Routine' to create a new table.

'new routine' window comes into view.



Enter information like:

Name : Need not required to enter .the name of routine is passed automatically from the DDL statement. The DDL is passed automatically while you type.

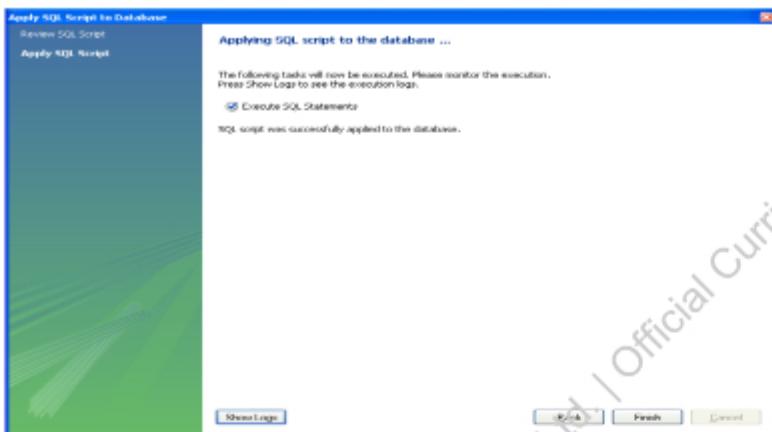
Once you write routine successfully after that click on 'Apply' button.

## Step 2

Review the SQL script on the database. After that click on 'Apply' button.



Once you click on 'Apply' button SQL script applying to the database. Click on 'Finish' button.



Once you click on 'Finish' button routine create in side the database. The snapshot given below.

```

Query 1 X
1 DELIMITER $$*
2
3 • CREATE DEFINER='root@localhost' PROCEDURE `account_rout1` (IN a_acno INT,INOUT a_t
4 BEGIN
5 SELECT balance FROM ACCOUNT WHERE accno=a_acno;
6 UPDATE account SET balance=a_balance WHERE accno=a_acno;
7 END
8

```

Routines (11 items)

- Add Routine
- account
- account\_rout
- account\_rout1
- account\_routine
- film\_in\_stock
- film\_not\_in\_stock
- get\_customer\_balance
- inventory\_held\_by\_cu...
- inventory\_in\_stock
- rewards\_report
- sp\_product

## 5. MySQL Database information.

### Step 1

The JDBC users required some information to connect with MySQL database like:

- Driver name: com.mysql.jdbc.Driver
- URL: jdbc:mysql://localhost:3306: DBName
  - Host: localhost
  - Port: 3306
  - Username: root
  - Password: Any Password for e.g. <myroot>

### Step 2

To execute JDBC application need to set class path so required jar file is:  
Mysql-connector-java-5.1.15-bin.jar

Copyrights of SEED Infotech Ltd. | Official Curriculum



