

Advanced Java

Prerequisites

- Java Installation
- Eclipse : <https://drive.google.com/open?id=1zz4ehVhdrdullEVXXBePclXnKcrhuqKD>
- Tomcat : <https://tomcat.apache.org/>
- Java API Link : <https://docs.oracle.com/javase/7/docs/api/>
- Servlet API Link : <https://tomcat.apache.org/tomcat-5.5-doc/servletapi/>
- Oracle xe : <https://www.oracle.com/database/technologies/xe-downloads.html>

Course Title : Advanced Java.

Course Objective : Developing java based online projects

(Real time dynamic website development)

Course content : Servlets, JDBC, JSP, (HTML, CSS and JavaScript.)

Q : What is Java.. ?

- Object oriented programming language.
- A group of technologies.
- Platform (J2SE, J2EE and frameworks)

Q : Where is Java used mostly or what is the main objective of Java. ?

- To develop a Java based dynamic website.

- To develop web enabled applications(Online projects).

Q: What is a website ..?

- A collection of webpages that mostly belong to a single organization.
- A website is hosted onto a web server and accessed through network
- i.e Internet using a network address known as URL (Uniform Resource Locator)
- A Collection of publicly accessible websites is nothing but www.
- EX: <https://www.facebook.com/>

Q : What is a web page..?

- A fundamental unit of information that can be sent from the website to the end- user is nothing but a web page.
- A web page is an HTML document with plain text and some formatting and instructions of Hypertext markup language.
- A web page contains text, images, video, audio as well.
- A web page is transported from the web server to the browser using http.

Q: How are websites classified..?

- Static and Dynamic
- What is the architecture of a static website..?

Web browser → Http Request →

Http Response ← Html documents / image files, audio,video files.

- What is the architecture of a dynamic website ..?

Web browser → Http Request →

Http Response ← Html documents / image files, audio,video files. → Dbms.

Q: Explain about a static website..?

- If a website is providing only information to the end users but not interaction to the end users, it is a static website.
- A static website doesn't involve database or server side programming.
- It provides general information which is common to every end user of the site.
- It can't provide user-specific information on demand.
- A static website contains only static web pages.

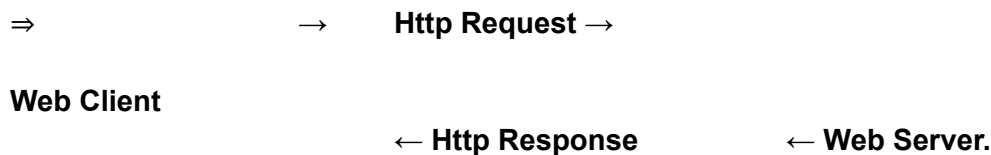
Q: Explain about a dynamic website ..?;

- If a website is providing not only information but also interaction to the end users,
- It is a dynamic website.
- A dynamic website involves database and server side programming.
- It provides general information which is common to every end user of the site +

User specific information on demand.

- A dynamic website contains static and dynamic web pages.

Q: What is the general architecture of a website...?



- A Website involves client-server architecture.

Q: What is a web client ..?

- Resources sake request making → application / program / software / process is nothing but a client.
- The machine in which the client is running is a client machine.

- Web resources sake request making software is a web client.
- Web client is also known as http Client as it uses HTTP to communicate with the web Server.
- Infact, browsers are web clients.

Q: Explain about Web Server ..?

- A web server is server software that provides web resources to web clients.
- A web server has two major duties.
 - 1) Providing Connections to web clients.
 - 2) Providing the requested web resource to web clients.
- Web server uses HTTP to communicate with web clients (browsers).

Therefore it is also known as HTTP Server.

Q: What is the limitation of a web server..?

- It can't communicate with the database. → who..? → JDBC
- It can't process data.--> who..? → Servlet
- It can't produce the response page(According to the user requirement). → who..? JSP
- Web servers on their own can't support dynamic websites.
- It can support only a static web site.

Q: How to overcome the limitations of the web server .. ?

- By performing server side (web server) programming.
- In the context of a website web server side programming is required to overcome the limitations of the web server and thereby making the website dynamic.

Q: What are the general duties of server side programming..?

- Capturing the user input.
- Communicate with the DBMS
- Processing of data.
- Producing the response page.
- Handling over the response page to the web server.(Deployment to the Server)

Q: What is a Java based dynamic website..?

- If a website is made interactive by using the following things, it is known as a java based dynamic website.
 - 1) Programming Language Java.
 - 2) Web technologies, Servlets and JSP.
 - 3) Data access technologies JDBC.

Q: What are the similarities between static and dynamic web pages.?

- Both are developed using HTML.
- Both are sent to the web client by the web server.
- Both are executed and rendered (Display) to the end user by the browser.

Q: How is web programming classified..?

- Client side programming.
- Server side programming.
- Javascript language is used to perform client side programming.
- Client side programming is mostly meant for validating the user input.
- For a website client side validation is performed to enforce complete and current input into the website while the user is expecting an online service.

Q) What are the Diff Between Static and Dynamic Web pages?

Static :

- It is pre-created.(created before any client request comes) page.
- It is stored in the file system of the web server machine with html or htm execution.
- Content of the page remains static i.e with the user interaction page content doesn't change.
- Server side programming is not required to create it.

Dynamic :

- Created only after receiving the client request that too based on the user input.
- Not stored in any file system.
- Changed based on user input.
- A server side program creates it.

Q: How are Java applications classified .. ?

- Windows based (Desktop) application.
- Web based (online / internet based) applications.
- GUI (Screens) of the project is nothing but the front end.
- Database (Collection of tables) is nothing but the back end of the project.
- Data processing and data accessing from the back- end is done by application programs.

Front end → Data processing → Back end ⇒ Java Application

Q: What is the drawback of windows based applications..?

- A windows based application can't provide online business to the customers.
- A customer should physically visit the business organization for the service.
- Staff of that business organization only became the end-users on their desktop only services are available.
- In case of Java based windows application swings and AWT are used as GUI i.e java Acts as front end.
- Almost all Java business applications are web based only.

Q : Explain about web based applications ..?

- If the services of a computer application are accessible through the web it is known as a web application.
- Ex :- Online Banking, online insurance, online reservation sys||, online shopping etc...
- Dynamic website is nothing but a web application.
- Web applications provide services to customers online round the clock around the world.

Servlets → service layer

Q: What are not servlets ..?

- Not a programming language.
- Not software to install a computer system.

Q : What are servlets ..?

- Servlets is a Java based web technology from sun micro system.
- It is an API. (Application programming interface).
- API means collections of classes interfaces together inside a package.
- Servlets is a Specification (From sun micro system) for web container manufacturers.
- It is J2EE technology .

Q: What is the purpose of Servlets..?

- To Develop Java based Dynamic websites. Or Java Web application development.
- Note : Java is mostly used in business applications. To make those business applications online, servlet technology is used.
- Web enabled enterprise Java application development is the main purpose of servlet technology.

Q: What is a business application ..?

- To computerize the business activities of a business organization (enterprise) whatever computer application that we develop is a business application.

- A business application is also known as an enterprise application.
- For ex: Banking application, insurance application, hotel management system etc...

Q : What is a servlet ..?

- Servlet is an interface.
- A servlet is a web server side programming piece of code that enhances the functionality of the web server.
- It is a dynamic web resource.
- A servlet is a web component.
- Servlet container managed.

public java class that implements Javax.servlet.Servlet interface is nothing but a servlet.

XyzServlet → implements → Servlet

Q: What are the general duties of a servlet.. ?

- Capturing the user input.
- Communicate with the database.
- Processing of data.
- Producing the response page.
- Handling over that response page (Dynamic page) to the web server.

Q: What is the content of a servlet (Source code) .. ?

→ A servlet = Java code + HTML Code.

Q: Explain about web resources..?

- In a website (Dynamic) we have 2 kinds of web resources.
 - 1) Static web resources.(For ex: HTML Documents and image files)
 - 2) Dynamic web resources (For ex: Servlets and JSPs)
- Every web resource has a URL.
- It is the presence of dynamic resources that makes the website dynamic.

Q: Explain about a web Container..?

- A web container is a server software written in java according to Servlets and JSP specification.
Ex: Tomcat.
- A web container has 3 modules.

1) Web Server

2) Servlet Container(Servlet Engine)

3) JSP Container.

- User defined servlets are executed by servlet containers.
- Jsps are executed by the JSP container.
- Communication with the web clients is performed by the web server.
- It is also known as HTTP Service / Default handler.
- Web servers alone can't execute servlets and jsps.
-

Without servlets and jsps websites can't be dynamic.

Therefore, a web container is required for a java based dynamic website.

3) Servlet container. → java class implements Servlet -

4) Jsp Container.

Q: Explain about Servlet Specification ..?

- A set of rules developed by sun microsystem for web container manufacturers to standardize the procedure of servlet engine module development is nothing but servlet specification.

Q: Explain about Servlet API..?

- A collection of library methods is nothing but API (application programming interface.)
- Servlet API is collection of library methods of classes and interfaces of

Javax.servlet and

javax.servlet.http packages are nothing but servlet API.

- Using servlet API we can perform server side programming to make the website dynamic.

Q : How to write a sample servlet program in Java.

- Need to Create a Dynamic web project in Eclipse.
- Give Project name → need to check web.xml deployment descriptor → Finish.
- Create a java file i.e select a servlet in the src folder.
- Give the class name and remove super class.
- Need to give URL Mapping(any name we can give that would be for URL in browser)
- By default the servlet interface will be available and click Finish.

Q : How to Deploy or run the program..?

- Right click on the project.
- Run As → Run on Server → Manually Define a New Server.
- Next → Move the resources to the right to configure them on the server.

Q: What is the general structure of a simple servlet..?

```
Import javax.servlet.*;
```

```
public class SomeServlet implements Servlet
```

```
{
```

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

```
        //resources allocation
```

```
    }
```

```
    Public void service(ServletRequest request , ServletResponse response){
```

```
        //dealing with client request
```

```
    }
```

```
    Public void destroy(){
```

```
        // resources deallocation
```

```
    }
```

```
}
```

Q: Explain about the life cycle of a servlet..?

- Servlet Container the life cycle of a servlet.
- Servlet life cycle describes 4 (life cycle)phases and 3 (life cycles) methods.
- Life cycle phases are :

1) Instantiation Phase.

2) Initialization Phase.

3) Servicing Phase.

4) Destruction Phase.

- Life cycle methods are : 1) init 2) service 3) destroy.
- Whenever something happens in the life of a servlet,

Servlet Engine calls these methods implicitly..

ServletEngine loads servlet class.

SE creates the instance of the servlet class. //Instantiation

SE creates a ServletConfig object.

SE calls init(ServletConfig) //Initialization

SE creates ServletRequest and ServletResponse objects.

SE calls service(request, response)

Yes ← One more client request

| no
Servlet waits for client request //Servicing phase

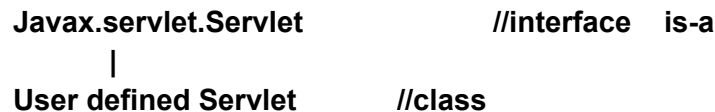
|
Is the servlet unloaded → No → Servlet waits for client request

| Yes
SE calls destroy() on the servlet instance

|
SE makes the servlet instance for garbage collection. //destruction phase

Q: Explain about the instantiation phase of a servlet..?

- Creating the object of a class is nothing but class instantiation.
- A servlet is a specialized user defined java class.



- Servlet engine creating the object of a user defined servlet class is nothing but the instantiation phase of a servlet.
- Whenever a client request comes for a servlet for the first time, the servlet engine creates the servlet instance.
- “only one servlet instance attends any number of client requests for that type of online service i.e A servlet in a java web application is a singleton.”
- In the lifetime of a servlet container, instantiates the servlet only once unless the application is reloaded or container restarted.



Q: Explain about the initialization phase of a servlet.

- During the instantiation phase the servlet is missing 2 pieces of information.
Context information.
Initial Configuration information.

- Therefore the servlet instance can't serve the client request.

- Servlet Container creates `ServletConfig` object encapsulating the context information and initial configuration information into it.

- Servlet engine calls `init()` method of the servlet by supplying `ServletConfig` object reference as argument.

- Once the `init()` method is completely executed, the servlet instance acquires its complete servleness and is ready to serve the client request.
- `init` method is called only once in the lifetime of a servlet.
- As servlet developers we write that code in `init` method which provides some application specific resources to the servlet.

For Ex: database connection creation

Q: Explain about the servicing phase of a servlet.. ?

- The Servlet container creates `ServletRequest` and `ServletResponse` objects.
- Servlet Engine calls the `service()` method of the servlet by supplying request and response object references as arguments .
- within the service method request object is used to capture the user input, response object is used to produce the dynamic page for the client.
- `service()` method execution is over means request- response cycle is complete and the client is served with the response.
- servlet container calls service method for each client request i.e a servlet spends most of its life in servicing phase only.
- If no client request comes, the servlet instance is not destroyed; instead, it waits for the client request.
- Web container is multi threaded. Therefore , without any thread creation code implementation in the servlet, it is able to handle multiple concurrent client requests.

Q: Explain about the destruction phase of a servlet ..?

- When the application is unloaded by the administrator or the container is shutdown gracefully, container calls `destroy` method of the servlet and releases the reference of the servlet instance that it holds and thereby the servlet instance is destroyed i.e garbage collected.
- Servlet developer writes that code in `destroy()` method that releases the resources allocated to the servlet.
- Servlet Engine calls `destroy()` method only once in the lifetime of a servlet.

Q: Develop a java web application in which a servlet displays “Hello Java Web world ” to the user upon client request.

- There are four steps in the development of any java web application.

Step 1: Creating a structured hierarchy of directories.

Step 2: Developing the web resources.

Step 3: Developing the deployment descriptor.

Step 4: Configuring the application files.

Step 1: Directory structure creation.

```
helloworldapplication
|
WEB-INF
|
Classes → .class
lib → jar files
src → .java
```

→ Create a root folder with any name, within that folder we should create ‘WEB-INF’ folder.

→ within this folder we need to create two more folders

1. classes 2. lib

Step 2: Web resources development .

→ In this application we need to create only one web resource i.e. servlet.

```
→ Javax.Servlet.Servlet      → interface
    |
    javax.servlet.GenericServlet → abstract class
    |
    HelloWorldServlet          → our class
```

- If a user-defined servlet class directly inherits from the servlet interface, even though not required, all the abstract methods of that interface must be implemented.

- **GenericServlet** is an abstract class in which only **service()** is not implemented. If a user defined class inherits from this class, only the service method we need to implement. Other methods if required can be implemented.

HelloWorldServlet.java

Step 3: Deployment descriptor development web.xml

- There are too many elements in the web.xml file. Here is the illustration of some elements that are used in the above web.xml file. The elements are as follows:

<web-app> represents the whole application.

<servlet> is sub element of **<web-app>** and represents the servlet.

<servlet-name> is sub element of **<servlet>** represents the name of the servlet.

<servlet-class> is sub element of **<servlet>** represents the class of the servlet.

<servlet-mapping> is sub element of **<web-app>**. It is used to map the servlet.

<url-pattern> is sub element of **<servlet-mapping>**. This pattern is used at client side to invoke the servlet.

Step 4: Configuring the application files

- **Html documents, image files and jsps** are to be placed into the root folder.

Q: How to execute the above application ..?

- Any java web application can't provide its services to web clients unless it is deployed into any web container.

For ex :- Tomcat

- Loading the web application into the web container is nothing but web application deployment.

- Following steps are involved in deploying a java web application into tomcat.

Step : 1

Copy the application(directory structure with all files). Into “webapps” folder of tomcat instantiation directory.

Step : 2

Start Tomcat

Tomcat 6.0/ bin → tomcat run (double click)

- To invoke the servlet we need to open the browser and specify the following URL.

<http://localhost:8081/helloworldapplication/hello>

Http → how

localhost:8081 → where

helloworldapplication/hello → what

Q: What are the system requirements to develop and run java web applications..?

- 1) Install J2SE platform (JDK 6 or higher)
- 2) Install any web container preferably Tomcat6.0
- 3) Place servlet-api.jar file (in case of tomcat) into classpath.

For ex:

Set CLASSPATH =c:\program Files\ApacheSoftwarefoundation\Tomcat6.0\lib\servlet-api.jar

Q: Explain about the following method call in the service method of HelloWorldServlet.

`response.setContentType(“text/html”);` → MIME Type

- To specify MIME type (Multipurpose Internet mail extensions) this method call is made.
- Some MIME types are text/plain text/html, image/gif etc..

Q: Explain about the following method call in service method.?

- `PrintWriter out = response.getWriter();`
- `java.io.printwriter` is a character oriented I/O stream class.
- “out” is the reference of the written into this stream is sent to the client and hence the name

Q : Develop a java web application in which the end-user should be able to enter two numbers into the web page and get their sum as the response.

Step 1: Directory Structure creation

additionapplication

WEB-INF

classes/

lib

src

Note : Container doesn't recognise src folder.

It is meant for developers to place all the java files of the application.

Step 2: Web resources development.

- In this application we need a static web resource to create the input screen and dynamic web resource to provide the functionality.

a) numbers.html

b)AdditionServlet.java

Step 3: Deployment descriptor development

web.xml

<web-app>

<servlet>

<servlet-name> two</servlet-name>

<servlet-class>AdditionServlet</servlet-class>

</servlet>

<Servlet-mapping>

<servlet-name>two</servlet-name>

<url-pattern>/add</url-pattern>

</servlet-mapping>

</web-app>

Step 4:

additionapplication

-numbers.html

WEB-INF

classes

AdditionServlet.class

lib

src

AdditionServlet.java

<http://localhost:8081/additionapplication/numbers.html>

numbers.html:

<HTML>

```
<BODY>

<CENTER>

    <H1>Numbers Entry Screen</H1>

    <FORM ACTION=".add">

        Number1 <INPUTTYPE ="text" NAME="t1"> <BR><BR>

        NUMBER2<INPUT TYPE="text" NAME="t2"><BR><BR>

        <INPUT TYPE="Submit" VALUE="ADD">

    </FORM>

</CENTER>

</BODY>

</HTML>
```

AdditionServlet.java

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

public class AdditionServlet extends GenericServlet
{
    public void doPost(ServletRequest r1, ServletResponse r2) throws IOEXCEPTIONS
    {
        String a= r1.getParameter("t1");
        String b=r1.getParameter("t2");

        int n1=Integer.parseInt(a);
        int n2=Integer.parseInt(b);
```

```
int sum=n1+n2;

r2.setContentType("text/html");

PrintWriter out = r2.getWriter();

out.println("<HTML>");

out.println("<BODY BGCOLOR =cyan>");

out.println("the sum is : " +sum);

out.println("</BODY>");

out.println("</HTML>");

out.close();

}

}
```

Q: How can a java program communicate with the database.?

- **Using JDBC.**

Q: What is not JDBC..?

- **It's not a programming language like java.**
- **It's not a software product to be installed into a computer system in order to use it.**
- **Not an acronym (abbreviation) for JavaDataBaseConnectivity.**

Q: What is JDBC..?

- JDBC is a JavaBased DataAccess Technology from sun micro system.
- As part of JDBC, the sun micro system released two documents.

1) API

2)Specification

- JDBC API for Java Developers. → JRE → java.sql
- JDBC Specification for JDBC Driver manufacturers.

Q: What is the purpose of JDBC..?

To enable any kind of java program perform CRUD operations with any DBMS in a standard manner.

C= Creation of Data.

R= Retrieval

U=Updation

D=Deletion

Q: Explain about JDBC architecture ..?

- JDBC architecture is a client server architecture.
- There are 5 elements in this architecture.
 - 1) JDBC Client → Servlet Programming or any Java Program is connecting to DB
 - 2)JDBC API → java.sql
 - 3) Driver Manager →
 - 4)JDBC Driver → ojdbc.jar → OracleDriver
 - 5)Database Server → oracle XE

Q: What is a JDBC Client ..?

- Any Java program that communicates with a Database server performing CRUD operations by using JDBC as data access technology is known as a JDBC Client.

- JDBC client has the following responsibilities..

- 1) Requesting for database connection.
- 2) Submitting appropriate SQL Statements.
- 3) Processing the response of the database server.(DBMS)
- 4) Dealing with Exceptions.
- 5) Dealing with transactions whenever necessary.
- 6) Closing the database connection.

Q: Explain about the JDBC API..?

- A collection of library methods of library interfaces and classes of **java.sql** package using which JDBC clients perform CRUD operations with the DBMS is nothing but JDBC API.

Q: Explain about DriverManager..?

- DriverManager is the connection provider to JDBC Clients.
It is not the connection creator.
- DriverManager has no role in CRUD Operations.

Q: Explain about JDBC Driver..?

- JDBC driver is a translation software written according to JDBC Specification.

(set of rules and specification)

- JDBC driver performs the following duties..

- 1) Establishing the connection b/w JDBC Client and database Server.
- 2) Receiving the JDBC method call intended for CRUD operation from the JDBC client.
- 3) Translating the JDBC API call into DBMS understandable format and passing on the same.
- 4) Translating DBMS given response into java format and passing on the same to the JDBC client.

Q: Explain about a JDBC Client getting connected to a database server.

Loading the JDBC driver.

|

Preparing the connection String

|

Establishing the Connection

Loading the driver.

→ In the Java.lang package there is a class by name "Class" .

→ In "Class" there is a static method "forName"

→ forName() method is used to load JDBC driver class dynamically from secondary memory into the primary memory.

For Ex:

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

- When the method is successfully executed three things happen in the background.
 - 1) OracleDriver class is loaded into memory
 - 2) OracleDriver class object is created.
 - 3) OracleDriver class object is registered with the DriverManager.
- If the specified driver class is not available forName() throws java.lang.ClassNotFoundException

Q: What is Connection String..?

- Database is nothing but Connection String.

For Eg.

"jdbc:oracle:thin:@localhost:1521:xe"

jdbc=main protocol

oracle=sub protocol

thin=type of driver(TYPE4)

localhost=IP address of the computer where database server is running. **localhost** indicates that JDBC client is running in which machine in that machine only database server also running.

1521 = port number of oracle database server

xe= database service name.

Note : Port number and database service name has to be checked in "tnsnames.ora" file in the oracle installation folder.

JDBC Driver

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

1. JDBC-ODBC bridge driver.

The JDBC-ODBC bridge driver uses an ODBC driver to connect to the database.

The JDBC-ODBC bridge driver converts JDBC method calls into ODBC function calls.

This is now discouraged because of thin drivers.

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

Advantages:

- easy to use.
- can be easily connected to any database.

Disadvantages:

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

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

2 Native-API driver (partially java driver)

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

Advantage:

- Performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

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

3 Network Protocol driver (fully java driver)

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

Advantage:

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

Disadvantages:

- Network support is required on client machines.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol drivers becomes costly because it requires database-specific coding to be done in the middle tier.

4 Thin driver (fully java driver)

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

Q: How to establish the database Connection.?

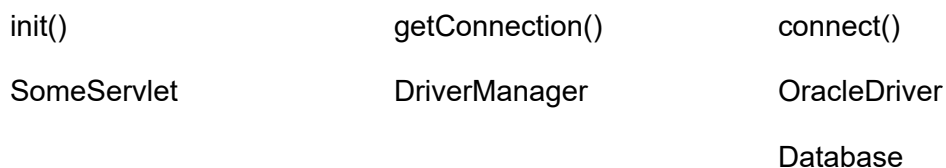
- By calling **getConnection()** method of **DriverManager**

for Ex:

```
Connection con = DriverManager.getConnection("Connection String",
"Username","Password" );
```

- This method throws **java.sql.SQLException** if anything goes abnormal.

Q: Explain how a JDBC Client gets connected to the database server..?



JDBC Client process.

Q: How to perform CRUD operations from a JDBC client with the database server (DBMS)..?

Step 1: Establish the connection.

Step 2: Create **java.sql.Statement** Object.

Statement st= con.createStatement();

Step 3 : Submit the query by calling any one of the following methods of the Statement object.

a) **executeUpdate**(String dml) :- This method is used to send any DML SQL statement to the DBMS.

This method returns a **number** that indicates the number of records affected in the database as a result of that SQL Statement. If anything goes abnormal this method throws SQLException.

b) **executeQuery**(String dml) :- This method is used to submit SELECT SQL Statement to the DBMS.

Step 4 : Close the Statement object to release the JDBC resources of the JDBC client held at database server.

Step 5 : Close the Connection to release the N/W resources of the JDBC client held at database server.

Q: Develop a piece of JDBC code to insert a record into a database table.

```
try{  
  
Class.forName("oracle.jdbc.driver.oracleDriver");  
  
Connection con=  
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");  
  
Statement st = con.createStatement();  
  
int n = st.executeUpdate("INSERT INTO ACCOUNT VALUES(1001,'SRI',50000)");  
  
System.out.println(n+ "account stored");  
  
st.close();  
  
con.close();
```

```
}  
try{  
}  
catch(ClassNotFoundException e);  
{  
Sys.out.println(e);  
}  
catch(SQLException e){  
sys.out.println(e);  
}  
}
```

Q: Explain a piece of JDBC code to increase the balance of an account by Rs500. If that number account does not exist, display the same.

```
try{  
    Class.forName("oracle.jdbc.driver.OracleDriver");  
    Connection con =  
    DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:server","scott", "tiger");  
    Statement st = con.createStatement();  
    Int n = st.executeUpdate("UPDATE ACCOUNT SET BALANCE= BALANCE +500 WHERE  
    ACCNO =1002");  
    if(n==0)  
        System.out.println("Account does not exist");  
    else  
        System.out.println("Update Successfully");  
}
```

```
st.close();
con.close();
}
catch(ClassNotFoundException e){
    System.out.println(e);
}
catch(SQLException e){
    System.out.println(e);
}
```

Develop a piece of JDBC code to delete the record from the database.

```
try{
    Class.forName("oracle.jdbc.driver.OracleDriver");
    Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:server","scott",
    "tiger");
    Statement st = con.createStatement();
    int n = st.executeUpdate("Delete from ACCOUNT WHERE ACCNO =1002");

    if(n==0)
        System.out.println("Account does not deleted");
    else
        System.out.println("Deleted Successfully");
}
```

```
st.close();  
con.close();  
  
}  
catch(ClassNotFoundException e){  
    System.out.println(e);  
}  
catch(SQLException e){  
    System.out.println(e);  
}
```

How to retrieve data from the Database ..?

Establishing the Connection

|

Creating the statement object

|

Submitting the SELECT statement

|

Does record exist..? → No

|

Yes

|

Process the record (yes)

|
One or more record

|
No
Closing the ResultSet

|
Closing the Statement

|
Closing the Connection

Explain about executeQuery() method..?

- This method is used to submit a SELECT Statement to the DBMS.

This method takes SELECT SQL statements as String arguments.

→ Return type of this method is **java.sql.ResultSet**.

→ If anything goes abnormal this method throws **SQLException**.

→ Prototype of this method is

Public ResultSet **executeQuery**(String dql) throws SQLException.

→ **Ex: ResultSet rs = st.executeQuery("SELECT * from ACCOUNT WHERE ACCNO=1001");**

Explain about ResultSet.

→ Object oriented representation of table of records retrieved from the database is nothing but ResultSet Object.

→ **ResultSet** object is logically represented in memory as follows.

Zero record Area

|

Records holding Area

|

No records Area

→ As soon as the **ResultSet** is open i.e resultSet object is created, a logical pointer (similar to file pointer) known as cursor points to ZERO RECORD AREA of the ResultSet.

→ Processing the row(record) of the **ResultSet** means reading each column value of the row of the ResultSet.

→ When the cursor is pointing to the ZERO RECORD AREA or NORECORD AREA, we should not process the row. Otherwise SQLException is raised I.e when the cursor is pointing to the records area only we need to process the records.

→ ResultSet Object has 15 methods to deal with the cursor most widely used is "next()".

→ **next()** method does 2 things

1. Moves the cursor in the forward direction by one record area.
2. After moving the cursor it returns true if record is existing in that area otherwise false.

→ **ResultSet** object has getXXX() method to read the column values.

→ These methods take column number of the ResultSet row as argument and return column value.

→ XXX stands for column data type EX:- getInt(), getString(), getChar().

⇒ Develop a piece of JDBC code to retrieve account details from the database. If with that A/C No. no account exists, display the same.

```
Int accno = Integer.parseInt(request.getParameter("t1"));

ResultSet rs = St.executeQuery ("SELECT FROM ACCOUNT WHERE ACCNO = "+accno);

if(rs.next())
{
{
Int ano=rs.getInt(1);

String nm=rs.getString(2)

Float bal=rs.getFloat(3)

out.println("A/C no: "+accno);

out.println("A/C holder name :"+nm);

out.println("Balance Rs."+bal);

}

Else

out.println("Account doesn't exist");

}

rs.close();

St.close();

con.close();
```

⇒ Develop a piece of JDBC code that displays to the end-user all the A/c holders name and balance whose balance is more than 10,000/- (10k).

```
ResultSet rs=St.executeQuery("SELECT NAME,BALANCE FROM ACCOUNT WHERE BALANCE>10,000");
```

```
System.out.println("Account Balance");
```

```
While (rs.next())
```

```
{
```

```
System.out.println(rs.getString(1)+"\t"+rs.getFloat(2));
```

```
}
```

```
rs.close();
```

```
st.close();
```

```
con.close();
```

⇒ Modify the above piece of code to display accounts with that balance range that do not exist.

Example 1:

```
import java.sql.*;
class OracleCon{
public static void main(String args[]){
try{
//step1 load the driver class
Class.forName("oracle.jdbc.driver.OracleDriver");

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

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

//step4 execute query
```

```
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));

//step5 close the connection object
con.close();

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

}
}
```

Request Dispatcher in Servlet

RequestDispatcher is an interface, implementation of which defines an object which can dispatch requests to any resources(such as HTML, Image, JSP, Servlet) on the server.

Servlet: Methods of RequestDispatcher

RequestDispatcher interface provides two important methods

Methods	Description
void forward(ServletRequest request, ServletResponse response)	forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server

```
void include(ServletRequest request, ServletResponse response)
```

includes the
content of a
resource
(servlet, JSP
page, HTML
file) in the
response

How to get an Object of RequestDispatcher

`getRequestDispatcher()` method of `ServletRequest` returns the object of `RequestDispatcher`.

```
RequestDispatcher rs = request.getRequestDispatcher("hello.html");  
rs.forward(request,response);
```

```
RequestDispatcher rs = request.getRequestDispatcher("hello.html");  
  
rs.forward(request,response);
```

ServletRequest object

resource name

forward the request and response to
"hello.html" page

OR

```
RequestDispatcher rs = request.getRequestDispatcher("hello.html");  
rs.include(request,response);
```

```
ServletRequest object      Resource name
RequestDispatcher rs = request.getRequestDispatcher("first.html");

rs.include(request, response);
include the response of "first.html" page in current
servlet response
```

Example demonstrating usage of RequestDispatcher

In this example, we will show you how RequestDispatcher is used to forward or include response of a resource in a Servlet.

Here we are using index.html to get username and password from the user, Validate Servlet will validate the password entered by the user, if the user has entered "srikanth" as password, then he will be forwarded to Welcome Servlet else the user will stay on the index.html page and an error message will be displayed.

Files to be created :

- index.html will have form fields to get user information.
- Validate.java will validate the data entered by the user.
- Welcome.java will be the welcome page.
- web.xml , the deployment descriptor.

index.html

```
<form method="post" action="Validate">
Name:<input type="text" name="user" /><br/>
Password:<input type="password" name="pass" ><br/>
<input type="submit" value="submit">
```

</form>

Validate.java

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

public class Validate extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        try {
            String name = request.getParameter("user");
            String password = request.getParameter("pass");

            if(password.equals("srikanth"))
            {
                RequestDispatcher rd = request.getRequestDispatcher("Welcome");
                rd.forward(request, response);
            }
            else
            {
                out.println("<font color='red'><b>You have entered incorrect password</b></font>");
                RequestDispatcher rd = request.getRequestDispatcher("index.html");
                rd.include(request, response);
            }
        }
        finally {
            out.close();
        }
    }
}
```

Welcome.java

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

public class Welcome extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

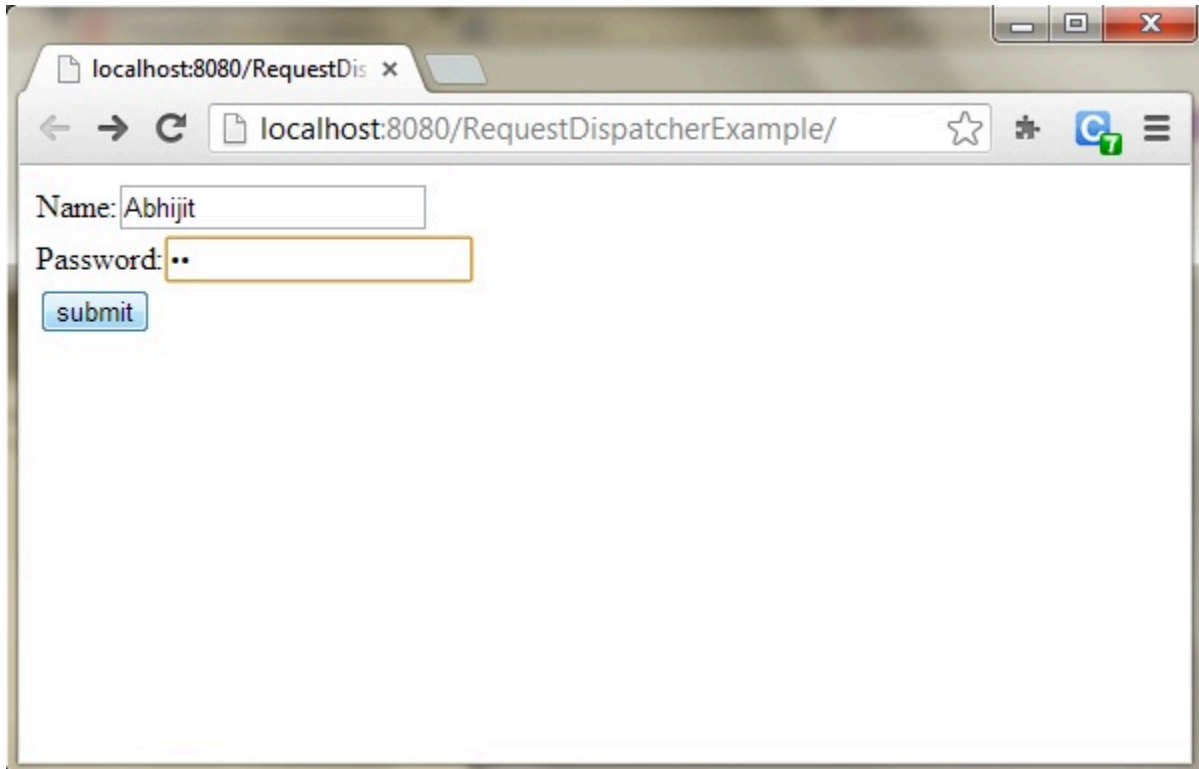
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            out.println("<h2>Welcome user</h2>");
        }
        finally {
            out.close();
        }
    }
}
```

web.xml

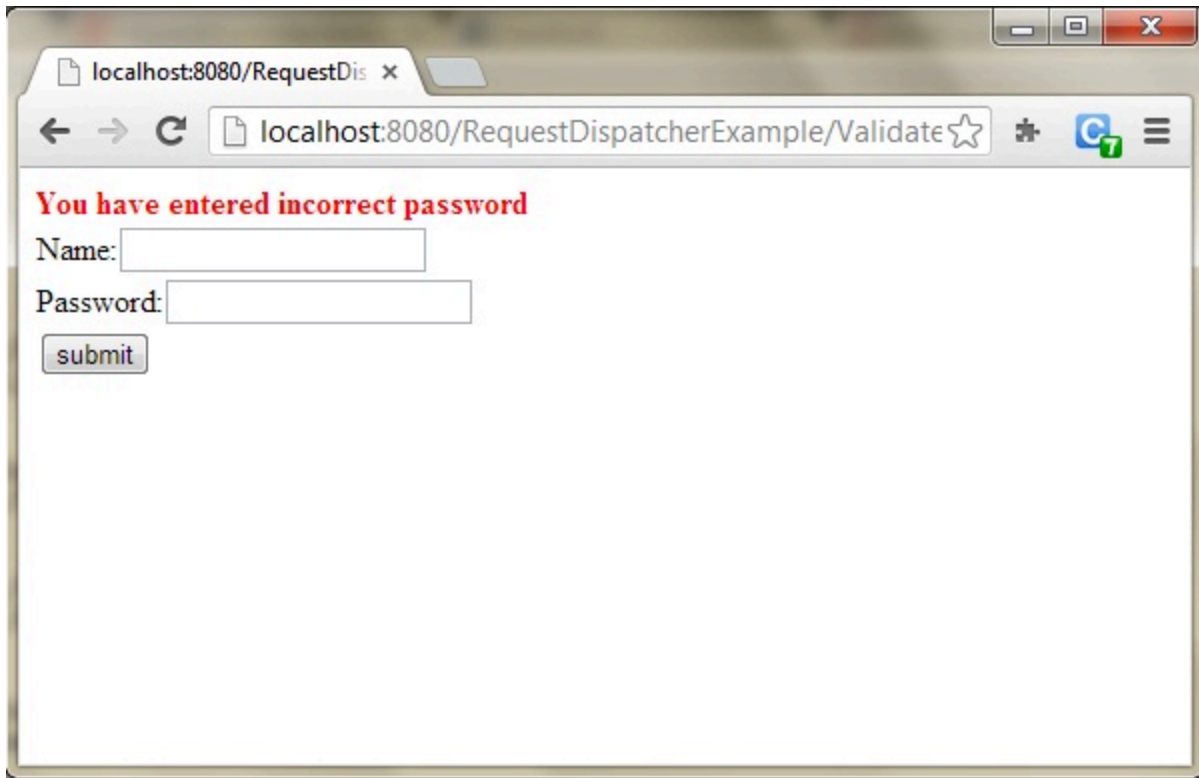
```
<web-app>
  <servlet>
    <servlet-name>Validate</servlet-name>
    <servlet-class>Validate</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>Welcome</servlet-name>
    <servlet-class>Welcome</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Validate</servlet-name>
    <url-pattern>/Validate</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>Welcome</servlet-name>
    <url-pattern>/Welcome</url-pattern>
  </servlet-mapping>
</web-app>
```

```
</servlet-mapping>
<welcome-file-list>
  <welcome-file>index.html</welcome-file>
</welcome-file-list>
</web-app>
```

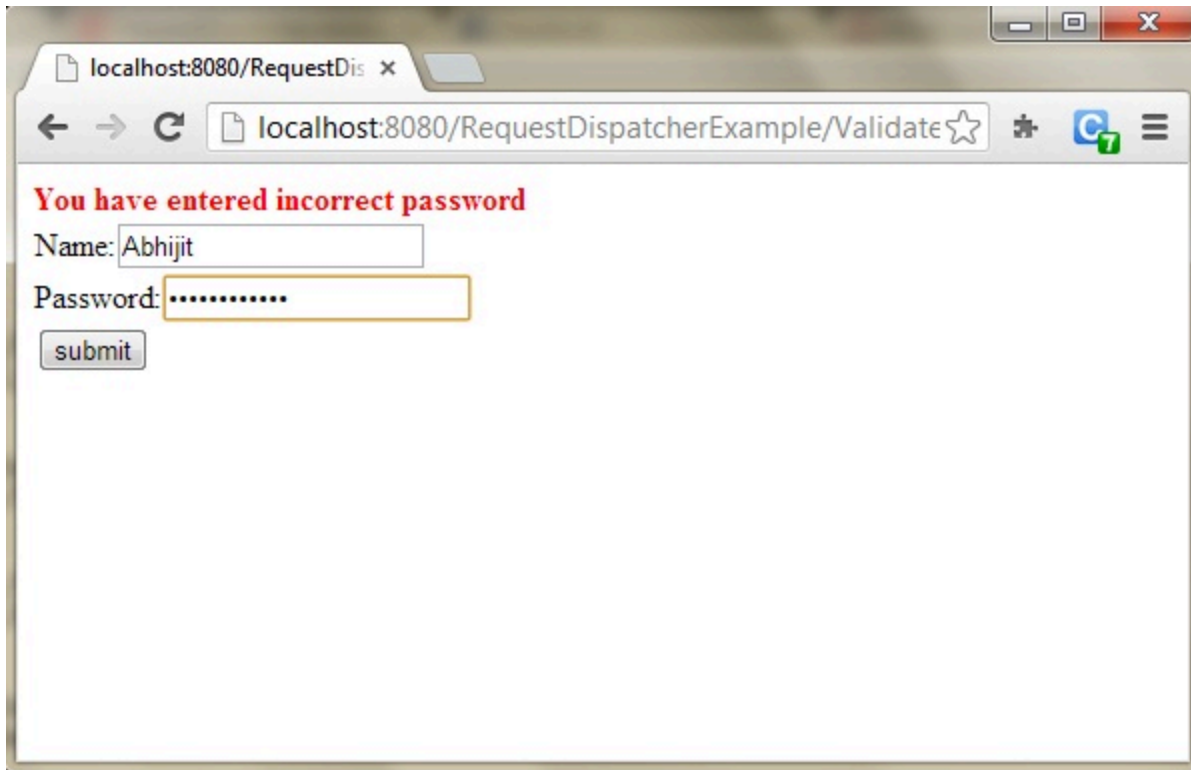
This will be the first screen. You can enter your Username and Password here.

A screenshot of a web browser window. The address bar shows 'localhost:8080/RequestDispatcherExample/'. The page content includes a form with two input fields: 'Name:' with the value 'Abhijit' and 'Password:' with two dots. Below the password field is a blue 'submit' button. The browser window has standard OS controls (minimize, maximize, close) in the top right corner.

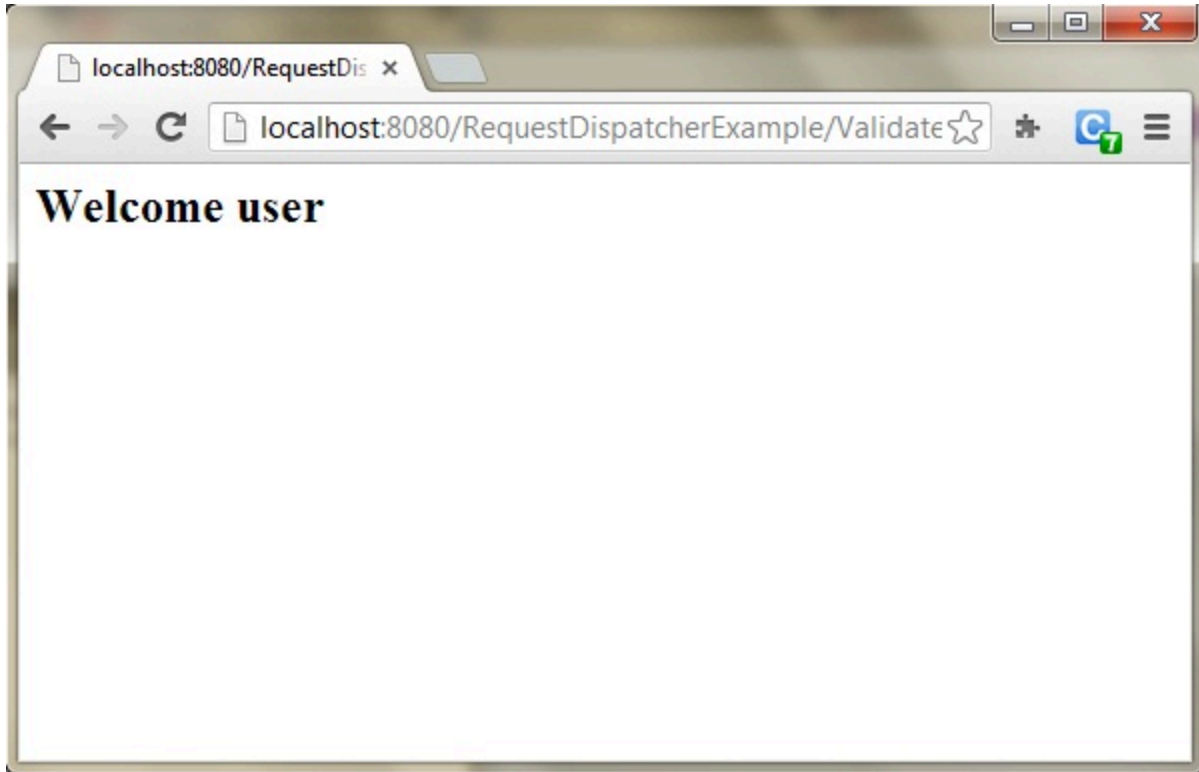
When you click on Submit, Password will be validated, if it is not 'srikanth', an error message will be displayed.



Enter any Username, but enter 'srikanth' as password.



Password will be successfully validated and you will be directed to the Welcome Servlet.



Session Management :

Managing Session in Servlets

- We all know that HTTP is a stateless protocol.
- All requests and responses are independent.
- But sometimes you need to **keep track of client's activity** across multiple requests.
- **For eg.** When a User logs into your website, not matter on which web page he visits after logging in, his credentials will be with the server, until he logs out. So this is managed by creating a session.
- Session Management is a mechanism used by the Web container to store session information for a particular user.
- There are four different techniques used by Servlet applications for session management.

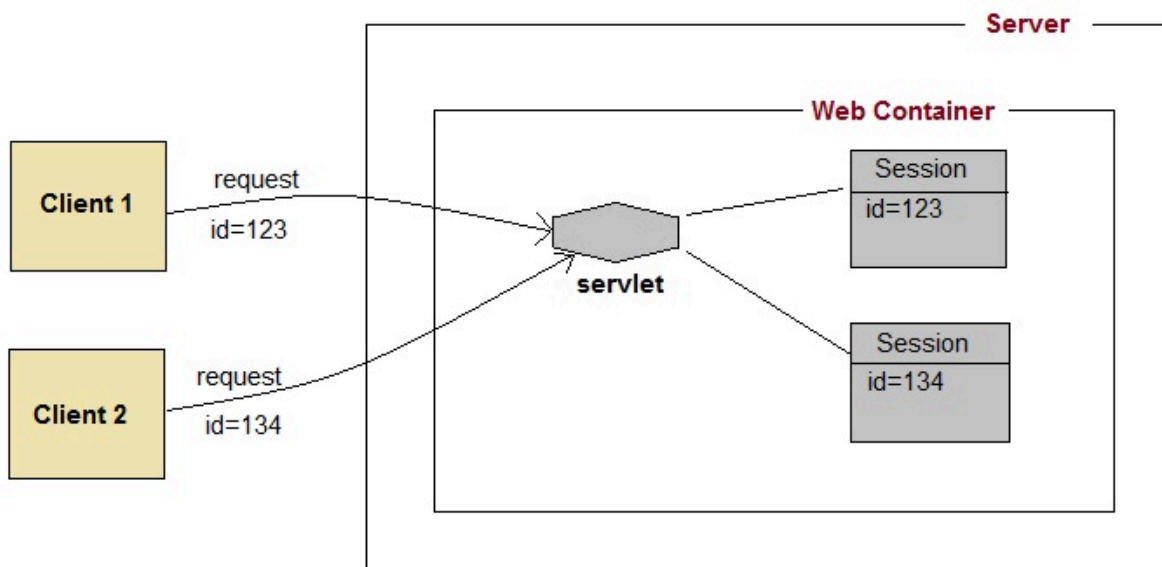
They are as follows:

Cookies
Hidden form field
URL Rewriting
HttpSession

Session is used to store everything that we can get from the client from all the requests the client makes.

How Session Works

session management in java



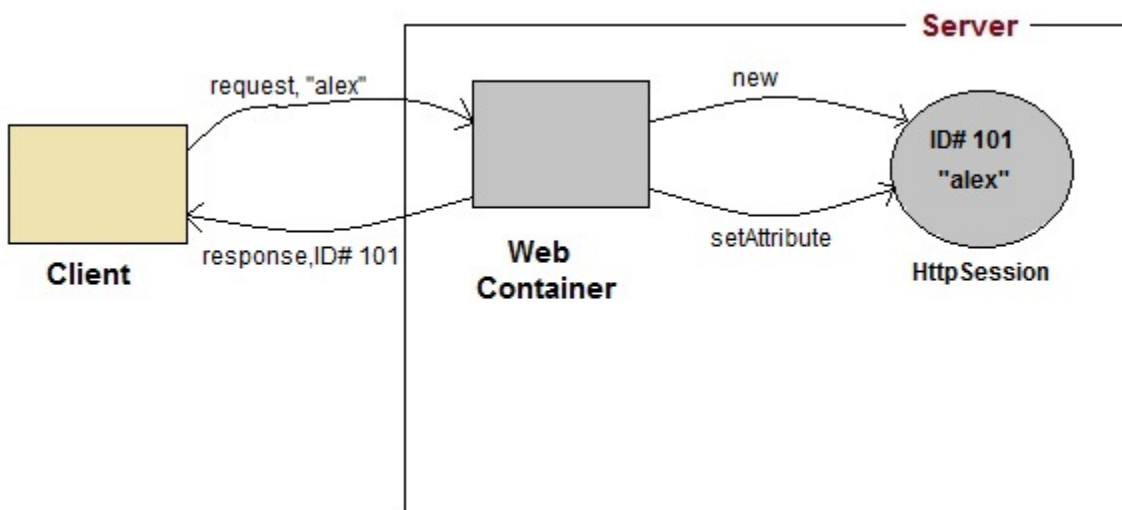
- The basic concept behind session is, whenever a user starts using our application, we can save a unique identification information about him, in an object which is available throughout the application, until its destroyed.
- So wherever the user goes, we will always have his information and we can always manage which user is doing what.
- Whenever a user wants to exit from your application, destroy the object with his information.

Servlet: What is HttpSession?

HttpSession object is used **to store an entire session** with a specific client.

We can store, retrieve and remove attributes from HttpSession objects.

Any servlet can have access to HttpSession objects throughout the getSession() method of the HttpServletRequest object.



1. On the client's first request, the Web Container generates a unique session ID and gives it back to the client with response.

This is a temporary session created by a web container.

2. The client sends back the session ID with each request.
3. Making it easier for the web container to identify where the request is coming from.
4. The Web Container uses this ID, finds the matching session with the ID and associates the session with the request.

Creating a new session

```
HttpSession session = request.getSession();
```

*getSession() method returns a session.
If the session already exist, it return the
existing session else create a new
session*

```
HttpSession session = request.getSession(true);
```

*getSession(true) always return
a new session*

Getting a pre-existing session

```
HttpSession session = request.getSession(false);
```

*return a pre-existing
session*

Destroying a session

```
session.invalidate();
```

destroy a session

Complete Example demonstrating usage of HttpSession

All the files mentioned below are required for the example.

Index.html

```
<form method="post" action="Validate">  
  User: <input type="text" name="user" /><br/>  
  Password: <input type="text" name="pass" ><br/>  
  <input type="submit" value="submit">  
</form>
```

web.xml

```
<web-app.>  
  
  <servlet>  
    <servlet-name>Validate</servlet-name>  
    <servlet-class>Validate</servlet-class>  
  </servlet>  
  
  <servlet>  
    <servlet-name>Welcome</servlet-name>  
    <servlet-class>Welcome</servlet-class>  
  </servlet>  
  
  <servlet-mapping>  
    <servlet-name>Validate</servlet-name>  
    <url-pattern>/Validate</url-pattern>
```

```
</servlet-mapping>

<servlet-mapping>
    <servlet-name>Welcome</servlet-name>
    <url-pattern>/Welcome</url-pattern>
</servlet-mapping>

<welcome-file-list>
    <welcome-file>index.html</welcome-file>
</welcome-file-list>

</web-app>
```

Validate.java

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

public class Validate extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");

        String name = request.getParameter("user");
        String pass = request.getParameter("pass");

        if(pass.equals("1234"))
```



```
{  
    //creating a session  
    HttpSession session = request.getSession();  
    session.setAttribute("user", name);  
    response.sendRedirect("Welcome");  
}  
}  
}
```

Welcome.java

```
import java.io.*;  
import javax.servlet.*;  
import javax.servlet.http.*;  
  
public class Welcome extends HttpServlet {  
  
    protected void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        response.setContentType("text/html;charset=UTF-8");  
        PrintWriter out = response.getWriter();  
        HttpSession session = request.getSession();  
        String user = (String)session.getAttribute("user");  
        out.println("Hello "+user);  
    }  
}
```

Cookies

Cookies are small pieces of information that are sent in response from the web server to the client.

Cookies are the simplest technique used for storing client state.

Cookies are stored on the client's computer.

They have a lifespan and are destroyed by the client browser at the end of that lifespan.

Using Cookies for storing client state has one shortcoming though, if the client has turned off Cookie saving settings in his browser then, client state can never be saved because the browser will not allow the application to store cookies.

Servlet: Cookies API

Cookies are created using the **Cookie** class present in Servlet API.

Cookies are added to the response object using the **addCookie()** method.

This method sends cookie information over the HTTP response stream.

`getCookies()` method is used to access the cookies that are added to the response object.

Creating a new Cookie

```
Cookie ck = new Cookie("username", name);
```

← creating a new cookie object

Setting up lifespan for a cookie

```
ck.setMaxAge(30*60);
```

← setting maximum age of cookie

Sending the cookie to the client

```
response.addCookie(ck);
```

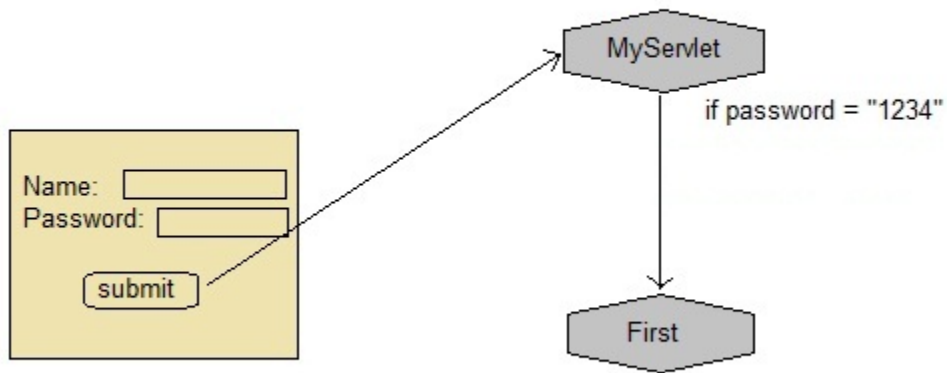
← adding cookie to **response** object

Getting cookies from client request

```
Cookie[] cks = request.getCookies();
```

← getting the cookie for **request** object

Example demonstrating usage of Cookies



Below mentioned files are required for the example:

index.html

```
<form method="post" action="validate">
  Name:<input type="text" name="user" /><br/>
  Password:<input type="text" name="pass" ><br/>
  <input type="submit" value="submit">
</form>
```

web.xml

```
<web-app...>

  <servlet>
    <servlet-name>validate</servlet-name>
    <servlet-class>MyServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>validate</servlet-name>
    <url-pattern>/validate</url-pattern>
  </servlet-mapping>
```

```
<servlet>
  <servlet-name>First</servlet-name>
  <servlet-class>First</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>First</servlet-name>
  <url-pattern>/First</url-pattern>
</servlet-mapping>

<welcome-file-list>
  <welcome-file>index.html</welcome-file>
</welcome-file-list>

</web-app>
```

MyServlet.java

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

public class MyServlet extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        String name = request.getParameter("user");
        String pass = request.getParameter("pass");

        if(pass.equals("1234"))
        {
            Cookie ck = new Cookie("user", name);
            response.addCookie(ck);
            response.sendRedirect("First");
        }
    }
}
```

First.java

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

public class First extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        Cookie[] cks = request.getCookies();
        out.println("Welcome "+ cks[0].getValue());
    }
}
```

URL Rewriting

Using URL Rewriting for Session Management in Servlet

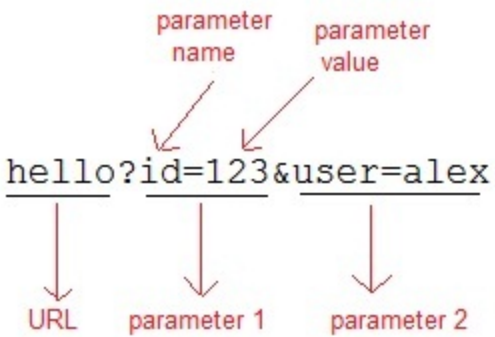
If the client has disabled cookies in the browser then session management using cookies wont work. In that case **URL Rewriting** can be used as a backup.

URL rewriting will always work.

In URL rewriting, a token(parameter) is added at the end of the URL.

The token consists of a name/value pair separated by an equal(=) sign.

For Example:



When the User clicks on the URL having parameters, the request goes to the **Web Container** with an extra bit of information at the end of the URL.

The **Web Container** will fetch the extra part of the requested URL and use it for session management.

The `getParameter()` method is used to get the parameter value at the server side.

Example demonstrating usage of URL rewriting

Below mentioned files are required for the example:

index.html

```
<form method="post" action="validate">
  Name:<input type="text" name="user" /><br/>
  Password:<input type="text" name="pass" ><br/>
  <input type="submit" value="submit">
</form>
```

web.xml

```
<web-app...>
  <servlet>
    <servlet-name>validate</servlet-name>
    <servlet-class>MyServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>validate</servlet-name>
    <url-pattern>/validate</url-pattern>
  </servlet-mapping>

  <servlet>
    <servlet-name>First</servlet-name>
    <servlet-class>First</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>First</servlet-name>
    <url-pattern>/First</url-pattern>
  </servlet-mapping>

  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>

</web-app>
```

MyServlet.java

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



```
public class MyServlet extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        String name = request.getParameter("user");
        String pass = request.getParameter("pass");

        if(pass.equals("1234"))
        {
            response.sendRedirect("First?user_name="+ name);
        }
    }
}
```

First.java

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

public class First extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        String user = request.getParameter("user_name");
        out.println("Welcome "+user);
    }
}
```

Hidden Form Field

Using Hidden Form Field for Session Management in Servlet

Hidden form fields can also be used to store session information for a particular client.

In the case of a hidden form field a hidden field is used to store client state.

In this case user information is stored in hidden field value and retrieved from another servlet.

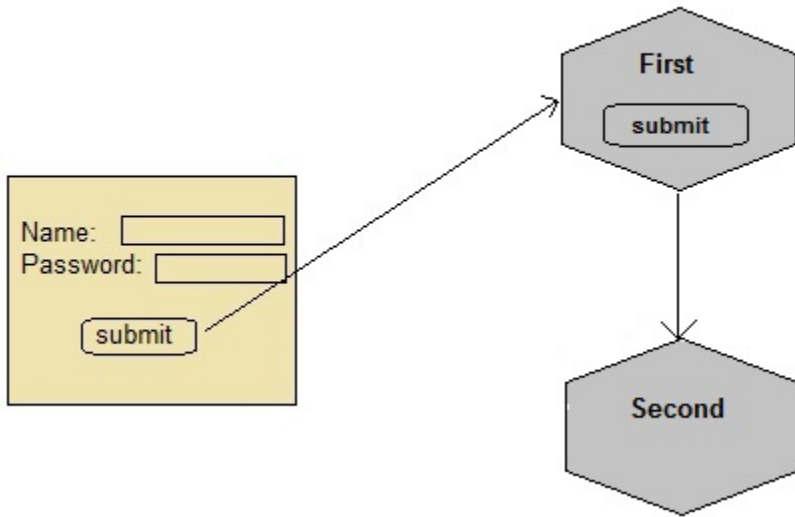
Advantages of Using Hidden Form Field for Session Management

- Does not have to depend on the browser whether the cookie is disabled or not.
 - Inserting a simple HTML Input field of type hidden is required. Hence, its easier to implement.
-

Disadvantage of Using Hidden Form Field for Session Management

- Extra form submission is required on every page. This is a big overhead.
-

Example demonstrating usage of Hidden Form Field for Session



Below mentioned files are required for the example:

index.html

```
<form method="post" action="validate">
  Name:<input type="text" name="user" /><br/>
  Password:<input type="text" name="pass" ><br/>
  <input type="submit" value="submit">
</form>
```

web.xml

```
<web-app...>

  <servlet>
    <servlet-name>First</servlet-name>
    <servlet-class>First</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>First</servlet-name>
    <url-pattern>/First</url-pattern>
```

```

</servlet-mapping>

<servlet>
    <servlet-name>Second</servlet-name>
    <servlet-class>Second</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>Second</servlet-name>
    <url-pattern>/Second</url-pattern>
</servlet-mapping>

<welcome-file-list>
    <welcome-file>index.html</welcome-file>
</welcome-file-list>

</web-app>

```

First.java

```

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

public class First extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        //getting value submitted in form from HTML file
        String user = request.getParameter("user");

        //creating a new hidden form field
        out.println("<form action='Second'>");
        out.println("<input type='hidden' name='user' value='"+user+"'>");
        out.println("<input type='submit' value='submit' >");
        out.println("</form>");
    }
}

```

Second.java

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

public class Second extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        //getting parameter from the hidden field
        String user = request.getParameter("user");
        out.println("Welcome "+user);
    }
}
```

Like we created a hidden field in the First Servlet, populated the value of the user, and sent it to the **Second** Servlet, now the Second Servlet also has the user information. Similarly we will have to keep sending this information, wherever we need this, using hidden fields.

===== Servlets End =====

JSP

⇒ What is JSP ?

→ JSP is a Java based web technology used to implement a **Java based dynamic website**.

⇒ What is a JSP container..?

- JSP Container or engine is a specialized software written in java according to the JSP specification.
- JSP Container executes jsps. I.e, JSP container acts as a platform (execution environment) for JSPs.
- JSP container implements JSP API.

Note : As part of JSP Technology creation, sun microsystems developed 2 things.

- 1) JSP API
- 2) JSP Specification.

⇒ What is a jsp ..?

→ A jsp is a **web server side piece of user defined code** that enhances the functionality of the web server.

⇒ What can a jsp do in a java web application.?

- A jsp can perform the following task to assist the web server and thereby make the website dynamic.
 - Capturing the user input.
 - Communicating with the database.
 - Processing of data.
 - producing the response page.
 - Handing over the response page to the web server.

⇒ What are the similarities between servlets and jsps.?

- Both are user defined.
- Both are web components.)
- Both are dynamic web resources.
- both contribute to making the website interactive.
- Both need the J2EE platform to get executed as they are J2EE components.

⇒ Explain about the life cycle of a jsp.

- JSP container controls the life cycle of a jsp.
- A jsp life cycle is described by 3 life cycle methods and 6 life cycle phases.
- **life cycle methods.**
 - jspInit().
 - _jspService().
 - jspDestroy().
- **Life cycle phases are**
 - Translation phase.
 - Compilation Phase.
 - Instantiation Phase.
 - Initialization phase.
 - Servicing phase.
 - Destroy phase.

⇒ Translation Phase :-

- A JSP can contain both java code and HTML Code.
- But it is neither a java program nor a html document.
- A jsp is a text file with any name and (but) with “.jsp” extension.

Ex: Helloworld.jsp.

- **Helloworld.jsp → JSP Engine → Helloworld_jsp.java**

→ When a JSP container receives that client requests for the first time, it makes a

full read of the JSP,

Verifies the syntax of the jsp elements of the jsp and converts it into a .java file.

→ JSP engine converts jsp into a servlet during translation phase, this servlet is known as **container generated servlet** or page implementation class

→ jsp development is **another style of servlet development only**. instead of developer developing, the servlet asking the container to develop the servlet.

⇒ compilation Phase:-

→ JSP Engine compiling the page implementation class to the **.class** file is nothing but the compilation phase of a jsp.

helloworld_jsp.java → JSP Engine + Java Compiler → helloworld_jsp.class

Note :

→ Translation and compilation happens only once in the lifetime of a jsp.

→ JSP engine uses "time stamp" to decide whether to translate and compile the JSP or Not.

⇒ Explain about Instantiation, Initialization, Servicing and destruction phases of a JSP.

→ Container creating an instance of a page implementation class is nothing but the instantiation phase of a JSP.

→ Container calls **jspInit()** method. It is the initialization phase of the jsp.

→ Both instantiation and initialization happens only once in the life of the jsp.

→ Container calling of **_jspService()** method is nothing but the servicing phase of the JSP.

→ For each client request to the jsp, the container calls this method once.

→ When the jsp is unloaded, the container marks the page implementation class object for garbage collection and calls the **jspDestroy()** method. This is nothing but the destruction phase of the jsp.

It happens only once in the lifetime of the Jsp.

⇒ What are the components in jsp..?

- HTML Code.
- Java Code
- Jsp elements.

⇒ Explain JSP elements.

→ There are 3 kinds of JSP elements.

(SDA)

- Scripting elements. 3
- Directives.
- Actions (tags)

→ JSP technology provides jsp elements that are used in JSP's **to instruct the JSP engine** as how to build the servlet.

⇒ JSP scripting elements.

→ Scripting elements are those JSP elements using which **java code is directly** embedded into a jsp.(DES)

→ There are 3 scripting elements.

- Declaration.
- Expression.
- scriptlet.

⇒ Declaration :-

→ It starts with “<%!” and ends with “%>”

→ **Class Scoped variables and methods** are declared or defined using a JSP declaration in a jsp.

Ex:

```
<%!  
    int a;
```

```
float b;  
void method1()  
%>
```

→ During translation phase, variables and methods of the JSP declaration become the members of the container generated servlet.

→ We can have any number of JSP declarations in a JSP.

⇒ Expression :-

→ It starts with “<%=” and ends with “%>”.

→ only one java expression is allowed in a JSP expression.

Ex:-

- 1) <%= a+b %>
- 2) <%= rs.getFloat(3) %>

→ During the translation phase of jsp, java expression of jsp expression is placed into **_jspService()**.

→ We can have any number of jsp expressions in a jsp.

In the same order java code falls into **_jspService()**.

→ when a JSP Expression is executed 2 things happen in the background.

- Java expression is evaluated.
- The result is printed to the client.

⇒ Scriptlet.

→ JSP scriptlet is meant for embedding java code directly into the JSP.

→ It starts with “<%” and ends with “%>”.

→ whatever java code we write in the **service** method of a servlet can be placed in a **scriptlet**.

→ During translation phase all java code placed in the scriptlet falls into **_jspService()** method.

→ We can have any number of scriptlets in a jsp. In the same order java code is placed into **_jspService** method.

⇒ Develop a web application in which a jsp displays “Helloworld” to the client.

→ Helloworld.jsp

→ web.xml

URL : <http://localhost:8081/jspapplicaion/helloworld.jsp>

Helloworld.jsp:

```
<HTML>
<BODY BGCOLOR ="CYAN">
<H1>Hello World</H1>
</HTML>
```

WEB.xml

```
<web-app>
</web-app>
```

Note : A JSP need not be registered with the container, if required can be registered.

Q: develop a java web application to implement the user case of addition of two numbers.

URL: <http://localhost:8081/jspadditionapplication/numbers.html>

Numbers.html:-

```
<FORM ACTION ="addition.jsp">
```

Addition.jsp:

```
<%
int n1=Integer.parseInt(request.getParameter(t1));
Int n2=Integer.parseInt(request.getParameter(t2));
%>
<HTML>
<BODY BGCOLOR="Yellow" >
<H1>
The sum of <% n1%> and <% n2%> is <%= n1+n2%>
</H1>
```

```
</body>
```

```
</html>
```

⇒ How many java packages are implicitly available in a JSP?

4 Packages.

- 1) **Java.lang**
- 2) **javax.servlet**
- 3) **javax.servlet.http.**
- 4) **javax.servlet.jsp**

⇒ Explain about JSP Directives.

→ A translation time instruction to the JSP Engine is nothing but a JSP Directive.

→ There are 3 kinds of JSP Directives.

- 1) Page Directive
- 2) Include Directive
- 3) Taglib directive

=> Explain about the include directive.

→ Include directive is used to include other page content into the current page inline.

→ Using include directive, include mechanism is implemented in jSPS.

→ Include mechanism facilitates the reusability of output generation code across the multiple jsps of the web application.

→ Include directive has the following syntax.

<%@ include file = "Include file name"%>

⇒ Explain about Page directive.

→ Page directive specifies the structure of the page implementation class to the JSP container.

- 1) import
- 2) errorPage
- 3) isErrorPage
- 4) Language
- 5) contentType
- 6) session
- 7) extends
- 8) isELIgnored
- 9) autoFlush
- 10) info
- 11) pageEncoding
- 12) buffer
- 13) isThreadSafe

⇒ Explain the import attribute of the page directive.

→ It is used to import a java package into a jsp.

Ex:

```
<%@ page import ="java.sql.*" %>  
<%@ page import ="java.util.*" %> or <%@ page import ="java.sql.* , java.util.*" %>
```

⇒ **What are hidden Comments?**

→ JSP comments are known as hidden comments unlike HTML comments in a JSP.

Ex:

```
<% --- Some COmments ---- %>
```

⇒ Explain about the “taglib” directive.

→ In a jsp we can have not only html tags but also JSP **tags**,

→ JSP Tags are of two types.

- 1) Standard tags.
- 2) Custom Tags.

- Into a jsp standard tags are implicitly available like that html tags into a jsp so as to use them.
 - JSP custom tags must be explicitly imported into a JSP in order to use them, “Taglib” directive is meant for that purpose.
 - “Taglib” directive has the following syntax.
- ```
<%@ taglib prefix="alias name of the custom tag library"
 uri="actual name of the custom tag library"%>
```

## ⇒ What are the JSP Implicit Objects.?

- JSP Technology facilitates implicit availability of 9\* objects into JSP known as JSP implicit objects.

| Object Name          | Type                                            |
|----------------------|-------------------------------------------------|
| → <b>config</b>      | → <b>javax.servlet.ServletConfig</b>            |
| → <b>session</b>     | → <b>javax.servlet.http.HttpSession</b>         |
| → <b>request</b>     | → <b>javax.servlet.http.HttpServletRequest</b>  |
| → <b>response</b>    | → <b>javax.servlet.http.HttpServletResponse</b> |
| → <b>application</b> | → <b>javax.servlet.ServletContext</b>           |
| → <b>out</b>         | → <b>javax.servlet.jsp.JSPWriter</b>            |
| → <b>exception</b>   | → <b>java.lang.Throwable</b>                    |
| → <b>pageContext</b> | → <b>javax.servlet.jsp.PageContext</b>          |
| → <b>page</b>        | → <b>javax.lang.Object</b>                      |

- During translation phase of a jsp container declares 9\* specific user defined **local variables** in `_jspService()` method and are implicitly made available to JSPS.
- These are in fact implicit object references.

## ⇒ Explain about “out” implicit object ..?

- “out ” is a reference to the instance of **javax.servlet.jsp.JSPWriter** class.
- It is a browser stream similar to that printwriter in user defined servlets.
- Whatever is written into the “out” stream is sent to the client and hence known as the browser stream.
- “out” implicit object is the **invisible work horse** in jsps.
- Total output generated by the jsp is sent to the client through this stream only.

**Note** :JSPwriter has more buffering capabilities than that of Printwriter.

## ⇒ Explain about the implicit object “Page”

- It is of **java.lang.Object** type.
- It holds the reference of a Page implementation class instance.
- It is **seldom** used in jsps.

## ⇒ Explain about isErrorPage & errorPage attributes of page directive .

→ By default a jsp is not an ErrorPage (exception handling page) as default value of “isErrorPage” attribute of page directive is **false**.

→ To designate a jsp as an error page we use the “isErrorPage” attribute.

**Ex:-** <%@ Page isErrorPage="true" %>

## exception

implicit object is not available in processing pages.

→ It is available only in error Pages.

→ ErrorPage attribute is not meant for converting a processing page into an error page.

It is meant for attaching an error page to a processing page.

→ this attribute is used in the processing page so that if any exception is raised in the processing page. The error message placed in the error page is displayed to the client.

## ⇒ What is a JSP action tag?

- Request processing time (runtime) instruction to the JSP container is nothing but a JSP Tag.
- There are two types of JSP tags.
  - 1) Standard tags.
  - 2) Custom tags.

## ⇒ Explain about standard tags.

- pre-defined tags shipped with the container and whose meaning is already known to the JSP engine are standard tags.
- Every standard tag has the following syntax.  
**<JSP:tagname attribute ="value" attribute ="value"> </JSP:tagname>**
- JSP Technology given the following standard tags.
  - 1) include
  - 2) forward
  - 3) param
  - 4) useBean
  - 5) setProperty
  - 6) getProperty

## ⇒ Explain about forward standard tag.

- This tag is used to implicit inter-jsp communication with forward mechanism of request dispatching.
- For ex:

**Source.jsp**

**<jsp:forward page="target.jsp">**

**Client → Req → Source.jsp → Target.jsp**

|  
←      ← **Response** ←



## ⇒ Explain about the param standard tag.

→ This tag is used to pass request parameters from one jsp to another jsp during inter jsp communication.

```
<jsp:forward page="target.jsp">
<jsp:param name="user" value="scott">
<jsp:param name="pwd" value="tiger"> </jsp:forward>
```

Target.jsp:

```
<%
String username =request.getParameter("user");
String password =request.getParameter("pwd");
%>
```

## ⇒ Explain about the standard tag.?

→ this tag used to implement the include mechanism in jsps.

→ include standard tag overcomes the limitation of include directive, but it is relatively less performing.

→ Limitation of include directive is if the include page is modified at a later stage it will not be reflected in the response of including page.

→ Syntax of include action is

```
<jsp:include page="includefile"/>
```

# JSP Implicit Objects – Syntax and Examples

This article discusses the implicit objects and all the methods related to them.

It discusses in detail about their functioning, usage etc. So let's begin!!!

## JSP Implicit Objects

Web containers create these implicit objects when they translate JSP pages to servlets.

They are available to all JSP pages.

They are written in scriptlet tags except declaration.

It is due to the fact that everything that is in the scriptlet tag goes to `_jspService()` method where processing of requests occurs, whereas declaration goes to source file generated outside `_jspService()` method.

They get called directly and need not to be declared. They are also called in-built objects.

There are 9 implicit objects available in the web container. Out of these 9, 7 are objects while 2 are passed as parameters to the `_jspService()`. All the objects are:

1. out
2. request
3. response
4. config
5. session
6. application
7. page

8. `pageContext`

9. `exception`

## 1. `out`

It is the most used implicit object that comes from **`java.servlet.jsp.JspWriter`**.

Its work is to write data into a buffer which would be sent to the client as output.

It possesses the access to servlet output stream.

Servlet uses **`PrintWriter`** but `JspWriter` has some additional features to it.

`JspWriter` has options for buffer and I/O Exceptions which are not available in `PrintWriter`.

Syntax: **`out`**.`methodName()`;

`Out` uses various methods such as:

a. void **`print()`**: It prints the data to the output. One can specify the data type too.

```
out.print(datatype d);
```

b. void **`println()`**: It prints the data type and terminates the line.

```
out.println(datatype d);
```

c. void **`flush()`**: The contents in the buffer(if present) gets written to the output and the stream is flushed.

```
out.flush();
```

d. void **`clear()`**: It clears the buffer. Even if something is present in the buffer, it doesn't get written to output. It will throw an exception, if this method is called on a buffer that is already flushed.

```
out.clear();
```

e. void clear **buffer**(): It is similar to clear but it doesn't give an exception if called on a flushed buffer.

```
out.clear buffer();
```

f. boolean **isAutoFlush**(): This method checks that the buffer is set to flush automatically or not.

```
out.isAutoFlush();
```

g. int **getBufferSize**(): This method returns the size of the buffer in bytes.

```
out.getBufferSize();
```

h. int **getRemaining**(): This method returns the remaining bytes of the buffer that are empty before the buffer overflows.

```
out.getRemaining();
```

#### Out2.jsp

```
<html>
<head>
<title>IMPLICIT OBJECT</title>
</head>
<body>
<%
out.println("first statement");
out.println("second");
out.println("third");
%>
</body>
</html>
```

Explanation: Using the method out.println(), this example shows the use of out implicit object. Output gets shown on the screen.

## 2. request

This is an in-built implicit object of **javax.servlet.http.HttpServletRequest**.

It is passed as a parameter to the **\_jspService** method.

This instance is created each time a request is generated.

Using this object we can request for a parameter session, cookies, header information, server name, server port, contentType, character encoding etc. It has following methods:

Syntax: request.methodname("value");

Methods under request parameter are:

a. **getParameter()**: This method is used so that the value of request can be got.

String obj=request.getParameter("name");

b. **getParameterNames()**: It gives the Enumeration of requested values.

c. **getParameterValues**(String name): It returns a whole array of parameters.

d. **getAttribute**(String name): This method gets the value of the attributes present.

request.getAttribute("value");

e. **getAttributeNames()**: This method is used to generate all the attributes that are present in that session.

f. **setAttribute**(String, Object): This method sets the value of an attribute according to user requirements. For example, If we want to set password value as admin then,

str=admin

```
String str=request.setAttribute("Password", str)
```

This will set the admin as a password.

g. **removeAttribute**(str): This method will remove the mentioned attribute from that JSP page.

```
request.removeAttribute("value");
```

h. **getCookies**(): This method of request will give an array of cookie objects used under cookie handling.

i. **getHeader**(String name): This method will get the information regarding the header of the request.

j. **getHeaderNames**(): This method will return an enumeration of all the header names available.

k. **getRequestURI**(): This method will return the Current JSP page's URL to the user.

l. **getMethod**(): This method will return the method used for the request. For example, GET for `getMethod()`, POST for `postMethod()`.

m. **getQueryString**(): This method will return the string in the url after the question mark for the associated page.

Example:

#### **form.html**

```
<html>
<head><title>My login</title>
</head>
<body>
<form action="user.jsp">
<a>Username:<input type="text" name="username">

<a>Password:<input type="text" name="password">
<input type="submit" value="go">

</form>
</body>
</html>
```

#### user.jsp

```
<html>
<head><title>login</title></head><body>
<%
String name=request.getParameter("username");
String pass=request.getParameter("password");
out.print("Welcome user, you entered your name as "+name);
%>
</body>
</html>
```

**Explanation:** This example gets the Parameter value from the user in the form of username and password using `getParameter()` and shows a welcome message to the user.

## 3. response

This is an object from `HttpServletResponse`. This object rather gets passed as a parameter to `jspService()` like `request`. It is the response given to the user. One can redirect, get header information, add cookies, send error messages using this object.

Syntax: `response.nethodName();`

Methods used with response object are:

a. void **setContentType**(String type): This method sets the content type of browser and browser interprets according to it.

```
response.setContentType("text/html");
```

```
response.setContentType("image/gif");
```

b. void **sendRedirect**(String address): This method will redirect users to other destinations.

```
response.sendRedirect("http://www.google.com");
```

c. void **addHeader**(String name, String value): This method adds header to the response

```
response.addHeader("Address", "Google.com");
```

d. void **setHeader**(String name, String value): This method will set the header value

```
response.setHeader("Address", "G.com");
```

e. boolean **containsHeader**(String name): It checks if header is present or not. It can have 2 values either true/false

```
response.containsHeader("Address");
```

f. void **addCookie**(Cookie value): Response will contain cookie using this method.

```
response.addCookie(Cookie kk);
```

g. void **sendError**(int status\_code, String message): Error response is sent using this method.

```
response.sendError(404, "error: Page was not found ");
```

For Example:

#### **redirect.html**

```
<html>
<head><title>My login</title>
</head>
<body>
<form action="user2.jsp">
<a>Username:<input type="text" name="username">

<a>Password:<input type="text" name="password">
<input type="submit" value="go">

</form>
</body>
</html>
```

#### **user2.jsp**

```
<html>
<head><title>login</title></head>
```



```
<body>
<%
response.sendRedirect("http://www.google.com");
%>
</body>
</html>
```

## 4. config

It is an object of **javax.servlet.ServletConfig**.

It is used to get the information regarding configuration of JSP pages.

For example get Servlet Name etc. It gives out the parameters related to servlet mapping initialization.

Syntax: config.methodName();

Methods used with config object are:

1. String **getInitParameter**(String name) – This method gets the parameter name.
2. Enumeration **getInitParameterNames**( ) – This method will return an enumeration of initialization params.
3. **getServletContext**( ) – It will return reference to Servlet context.
4. String **getServletName**() – It returns the name of the servlet which was defined in the web.xml file used for mapping.

```
String s = config.getServletName();
```

## For Example:

```
<html>
<head><title>Config</title>
</head>
<body>
<form action=.jsp">
<a>Username:<input type="text" name="username">

<input type="submit" value="go">

</form>
</body>
</html>
```

### web.xml

```
<web-app>
<servlet>
<servlet-name>kajalmoryani</servlet-name>

<jsp-file>/one.jsp</jsp-file>
<init-param>
<param-name>drivername</param-name>
<param-value>abc.JdbcOdbcDriver</param-value>
</init-param>
</servlet>
<servlet-mapping>
<servlet-name>kajalmoryani</servlet-name>
<url-pattern>/one</url-pattern>
</servlet-mapping>
</web-app>
one.jsp
<html>
<head><title>Config2</title></head>
<body>
<%
out.print("Welcome "+request.getParameter("username"));
String driver=config.getInitParameter("drivername");
out.print("driver name is="+driver);
%>
</body>
```

```
</html>
```

## 5. session

session object is the implicit object under **javax.servlet.http.HttpSession** . It is used to get session information as it tracks sessions between client requests.

It can be used to get, set as well as remove under scope of session.

As session creation and management is a cumbersome project so if you don't want a session to be created we can set the session in page directive as false.

Syntax: `session.getAttribute()`

Methods used with session object are:

- a. **setAttribute**(String, object) – This method will assign a string to the object and save it in session.
- b. **getAttribute**(String name) – This method gets the object that is set by `setAttribute`.
- c. **removeAttribute**(String name) – Objects that are in the session can be removed using this method.
- d. **getAttributeNames** – It returns enumeration of the objects in session.
- e. `getCreationTime` – This method returns time when the session was in active mode.
- f. **getId** – This method gives out the unique id of the session.
- g. **isNew** – This method checks if a session is new or not. It would return true if cookies are not there.

h. **getMaxInactiveInterval** – Returns time span, the session was inactive.

i. **getLastAccessedTime** – This method gives the last accessed time of a session.

For Example:

```
<html>

<head><title>Config</title></head>

<body>

<form action="first.jsp">

<input type="text" name="username">

<input type="submit" value="go">

</form>

</body>

</html>
```

#### **First.jsp**

```
<html>

<head><title>Config</head></title>

<body>

<%

String myname=request.getParameter("username");

out.print("Welcome "+myname);
```

```
session.setAttribute("user",myname); %>
```

```
second jsp page
```

```
</body>
```

```
</html>
```

### **second.jsp**

```
<html>
```

```
<body>
```

```
<%
```

```
String name=(String)session.getAttribute("user");
```

```
out.print("Hello "+myname); %>
```

```
</body>
```

```
</html>
```

Explanation: This code will generate a session. The form takes in the username and is reverted to first.jsp which is a session page on clicking the second jsp page link, we get to the final page i.e. second.jsp that says Hello message.

## 6. application

It is an implicit object of ServletContext from **javax.servlet.ServletContext** implementation.

This object is created by the web container only when an application is deployed.

It is generated one per application. As the name suggests, it interacts with servlet.

It can get set or even remove the attributes under application scope.

It can exclusively be used to get the object Request Dispatcher. Using Request Dispatcher, it forwards the request to other pages.

Syntax: `application.methodName("value");`

Methods available under application scope are:

a. Object **getAttribute**(String attributeName): This method gets the object available in the attribute and returns it.

`String a = (String)application.getAttribute("Name");`

b. void **setAttribute**(String attributeName, Object object): This method will set the value of attribute.

`application.setAttribute("Address", "Value of Address")`

c. void **removeAttribute**(String objectName): This method will remove an attribute from application.

`application.removeAttribute("Name");`

d. String **getRealPath**(String value): This method will return the absolute path of the file.

`String abspath = application.getRealPath("/form.html");`

e. void **log**(String message): This method stores logs to default log file of JSP Engine

`application.log(" error 404 page not found");`

f. String **getServerInfo**(): Returns name and version of web container

`application.getServerInfo();`

For Example:

```
<html>

<head>

<title>Application object</title>

</head>

<body>

<a>This is an example for application implicit object

<% application.getRealPath("form.html"); %>

</body>

</html>
```

Explanation: The object used in the example is application along with the method getContextPath method. It gives the Context Path of the related JSP page.

## 7. PageContext

It is the implicit object from **javax.servlet.jsp.PageContext** implementation where PageContext is an abstract implementation.

This object has the capability to get, set and remove attributes that fall under 4 scopes—page, request, session and application scopes.

This object has references to other implicit objects.

It contains information about directives, buffer information, error page URL. It holds reference to request and response as well. This can support over 40 methods that are inherited from ContextClass. It is even used to get info of a complete JSP page.

Syntax: `pageContext.methodName("parameter");`

Some useful methods used with pageContext:

a. Object `getAttribute(String AttributeName, int Scope)`: This method finds/gets the attribute from the suggested scope

- `Object obj1 = pageContext.getAttribute("name", PageContext.SESSION_CONTEXT);`
- `Object obj2 = pageContext.getAttribute("address", PageContext.REQUEST_CONTEXT);`
- `Object obj3 = pageContext.getAttribute("number", PageContext.PAGE_CONTEXT);`
- `Object obj4 = pageContext.getAttribute("data", PageContext.APPLICATION_CONTEXT);`

b. void `setAttribute(String AttributeName, Object AttributeValue, int Scope)`: This method will set the attribute in the scope along with its value.

`Object obj1 = pageContext.setAttribute("name", "Kajal", PageContext.SESSION_CONTEXT);`

c. void `removeAttribute(String AttributeName, int Scope)`: This method will remove the attribute mentioned from referred scope.



```
Object obj1 = pageContext.removeAttribute("name", PageContext.SESSION_CONTEXT);
```

d. Object findAttribute (String AttributeName): This method will find the attribute in all the four scopes and when found none, return null value. The difference between get and find is that scope is defined in the get method.

```
Object obj2 = pageContext.findAttribute("address");
```

For Example:

#### **myindex.html**

```
<html>

<head><title>PageContext</title>

</head>

<body>

<form action="PageContext1.jsp">

<input type="text" name="username">

<input type="submit" value="go">

</form>

</body>

</html>
```

#### **PageContext1.jsp**

```
<html>

<head><title>PageContext</title></head>

<body>
```

```
<%

String myname=request.getParameter("username");

out.print("Welcome "+myname);

pageContext.setAttribute("client",myname,PageContext.SESSION_SCOPE); %>

 go to second jsp page

</body>

</html>
```

### **PageContext2.jsp**

```
<html>

<body>

<%

String myname=(String)pageContext.getAttribute("client",PageContext.SESSION_SCOPE);

out.print("Here you will see your name "+myname);

%>

</body>

</html>
```

**Explanation:** In this example pageContext1 sets attribute client in Session scope and forwards to PageContext2 where it gets the attribute and shows the result.

## 8. page

This implicit object comes under **java.lang.Object** class.

Its work is to provide reference to the servlet class generated.

Type casting is thus required to access this object as shown below.

```
<% (HttpServletRequest)page.log("message"); %>
```

We use this rarely. Instead, we use it as:

Syntax:

Object page=this;

```
<% this.log("message"); %>
```

For Example:

```
<html>

<head><title>Object page</title>

</head>

<body>

<%

this.log("message");

%>

</body>

</html>
```

**Explanation:** A reference page is generated over successful compilation.

## 9)exception :

Exception is an implicit built-in object of java.lang.Throwable. This is useful for handling exceptions in JSP. As these pages handle errors, they can only be used for JSP error pages. This implies that if `<%@ page isErrorPage="true"%>`, then only you can use this object.

Syntax is as follows:

```
<%=exception%>
```

For Example:

form2.html

```
<html>
```

```
<head>
```

```
<title>Enter two Integers to divide</title>
```

```
</head>
```

```
<body>
```

```
<form action="divide.jsp">
```

```
Enter First Integer:<input type="text" name="number1" />

```

```
Enter Second Integer:<input type="text" name="number2" />
```

```
<input type="submit" value="Result"/>
```

```
</form>
```

```
</body>
```

```
</html>
```

divide.jsp

```
<%@ page errorPage="exception.jsp" %>
```

```
<html>
```

```
<body>
```

```
<%
```

```
String num1=request.getParameter("number1");
```

```
String num2=request.getParameter("number2");
```

```
int n1= Integer.parseInt(num1);
```

```
int n2= Integer.parseInt(num2);
```

```
int result= n1/n2;
```

```
out.print("Output is: "+ result);
```

```
%>
```

```
</body>
```

```
</html>
```

05/01/2024 :

```
<%@ page isErrorPage="true" %>
```

```
<html>
```

```
<head>

<title>Exception</title></head>

<body>

<a href: Hey,Got this Exception: <%= exception %>

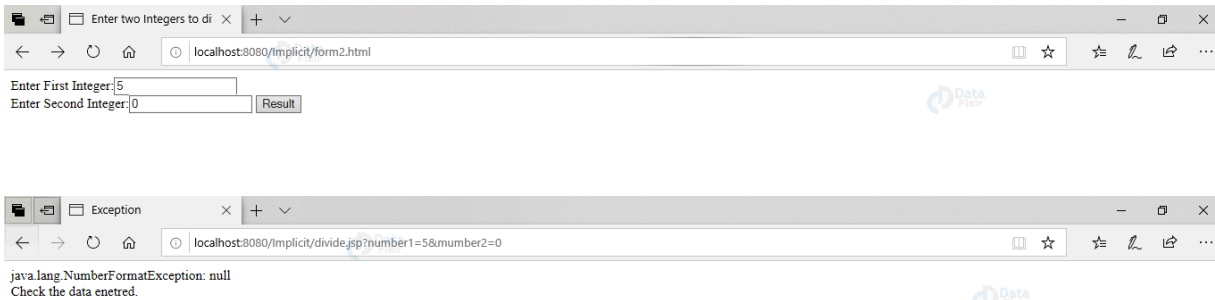
<a>Check the data entered.

</body>

</html>
```

Explanation: In this example html page opens and asks for input. These entered numbers are forwarded to divide.jsp that does calculation and if the page creates an exception, then exception.jsp is called and it throws java.lang.NumberFormatException and Check the data entered.

Output:



=====JSP end =====