



For More Tutorials Visit

ameerpetmaterials.blogspot.in

Advance Java

(Suresh Sir)

13/07/2009

RS:- 140/-

Rs: 140/-

ADVANCED JAVA

Suresh

9849867104

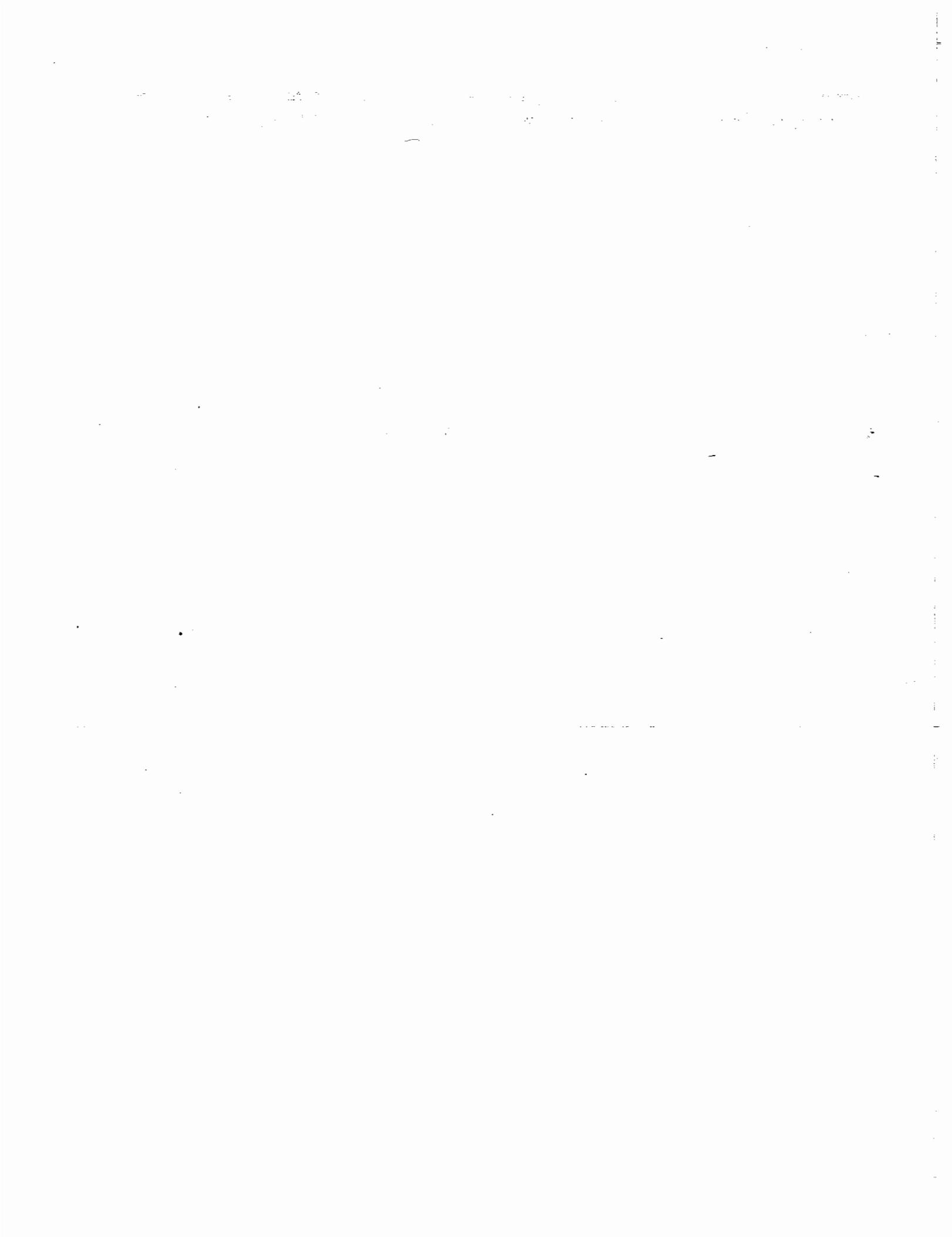
Contents :-

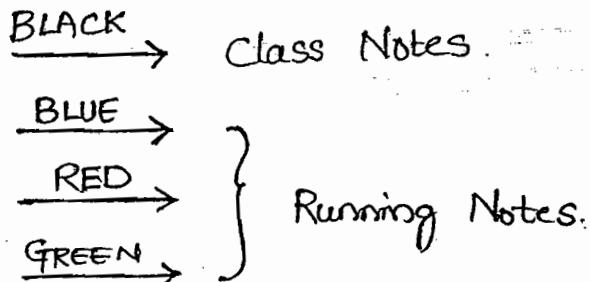
- 1) JDBC.
- 2) Servlets.
- 3) JSP.

Objective :-

Developing Web Based Projects i.e, On-line Applications using Java. On-line Project Development Such as On-line Banking, Hotel Management, Hospital Management etc.

request.getParameter
request





→ Where is Java used?

Java is mostly* used in Business Application Development.

→ What is a Business Application?

Any Computer Application is known as a Business Application if it is computerizing (automating) the manual Business process for Example:

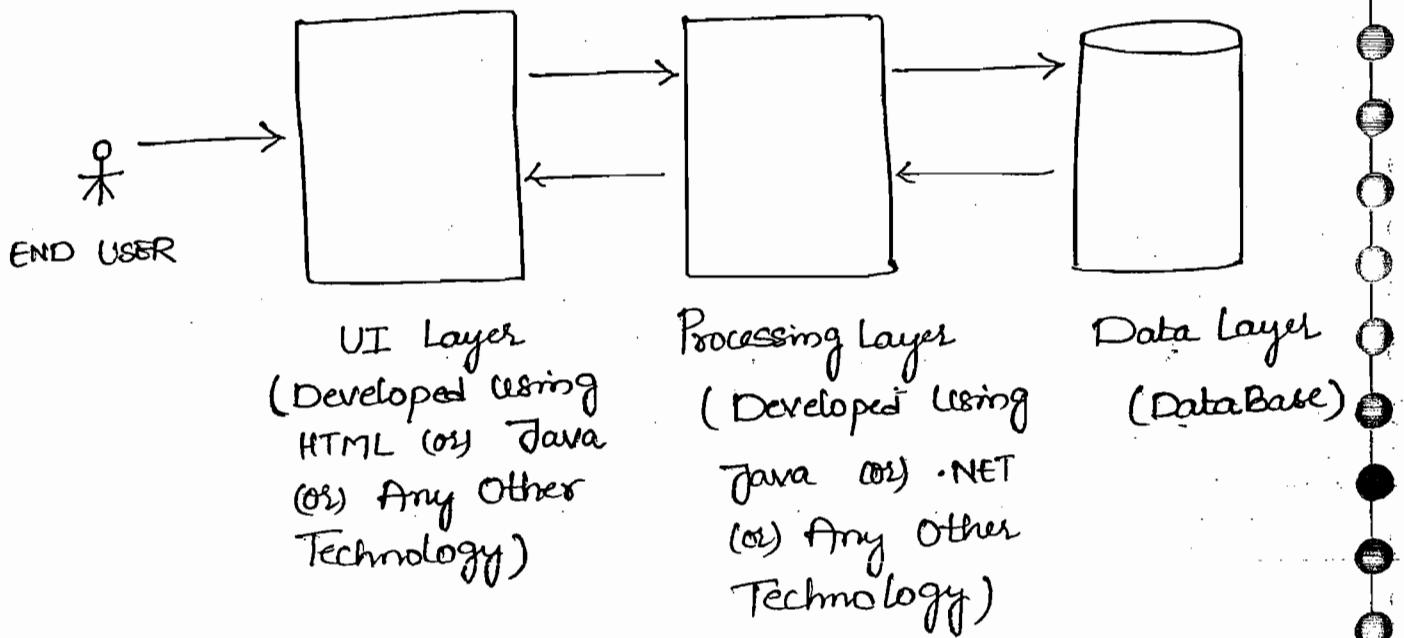
Banking Application, Insurance Application, Stock Exchange Application, Hospital Management, Hotel Management Application etc, are few Business Applications.

GENERALIZED ARCHITECTURE OF A BUSINESS APPLICATION:

Any Business Application has a minimum of three layers.

1) User Interaction (UI) layer. It is also known as the Front END.

3) Processing Layer. It is the Business Layer also known as the Business Tier.



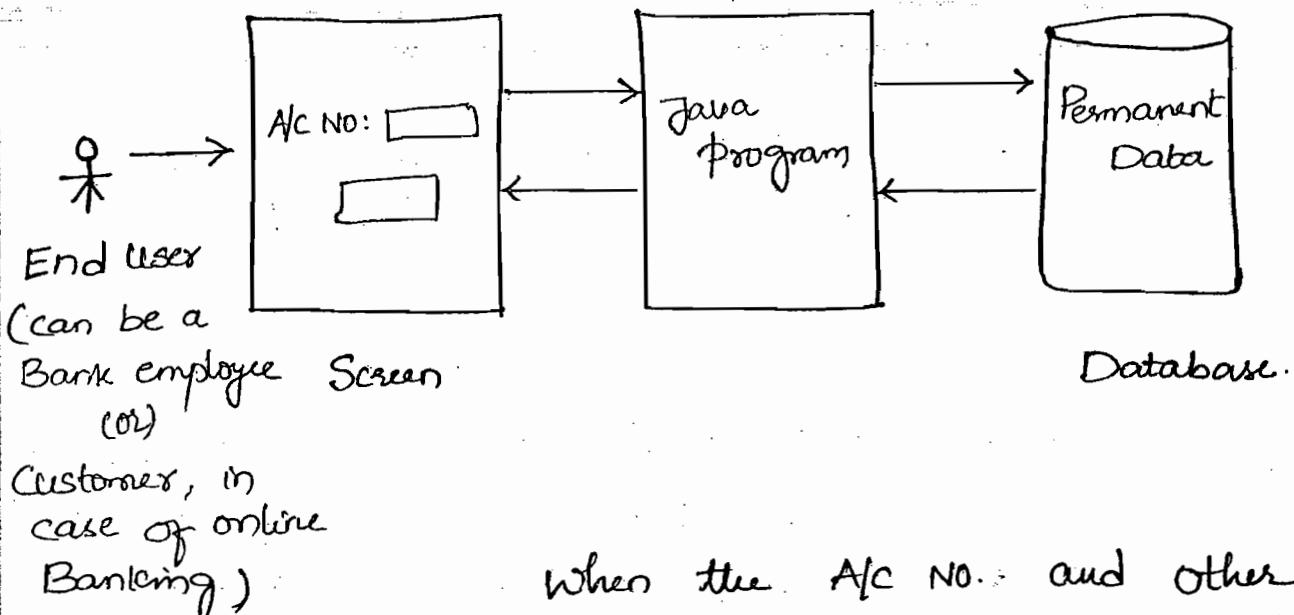
14 | 07 | 09

JDBC

- Java is used in Business Applications.
- Business data is stored in all Business Applications permanently in database (almost all). This is called data permanency or persistent Data. Example for Such a business process is a Banking Application where previous data is almost everytime necessary for future transactions.

Using Java, data cannot be stored permanently as data in Variables is volatile. Java can store data permanently on the hard disk as files, using I/O Streams. But it is not a wise procedure as files lack Security which is very much needed in a Business process.

A Database is excellent in storing the data and providing Security to it. But a database is weak in processing and presenting the data. Presenting the data implies, representing the data in an understandable manner to the end user. This can be achieved using Java like Technologies.



When the A/c No. and other details are entered into the Screen, the front end cannot directly contact the database to get the data. It is the duty of Java program to take these inputs, contact the Database and fetch the required data. It also presents the data in meaningful manner and gives the output to UI layer i.e. the front end.

Hence we can conclude that:

- Databases are excellent in data storage but weaker in processing and presenting the data to the end user.
- Java is excellent in data processing and presenting the results, but is poor in data storage.

- Business Applications are meant for providing Business Services to the customers. To provide Business Services to customers, data storage, data processing and data presentation in user understandable manner is mandatory.
- As long as Java is used in Business Application it has to communicate with the databases, since the data is always present in the database.

A DBMS cannot understand a Java program and vice-versa.

DBMS only understands SQL and Java only understands its method calls

(Java's power is in its method calls, C-language power is in its function calls etc.)

This is a heterogeneous environment i.e. Java and database have no commonness among them. To make this communication possible, we have to go for JDBC.

(Even C,C++, .NET and other application development languages can connect to the database. Then it is the responsibility of these languages or technologies to communicate with Database but the reverse is not true.)

8.

* (Java is a language, but is also a Technology. Majority of Java is Technology and language is minor part. Core Java is to learn the Java Language. Using only core Java, Business Applications cannot be developed) *

→ Java Environment and Database Environment is different. Inspite of such heterogeneity, Java and Database communication should happen in a Business Application.

→ What is JDBC?

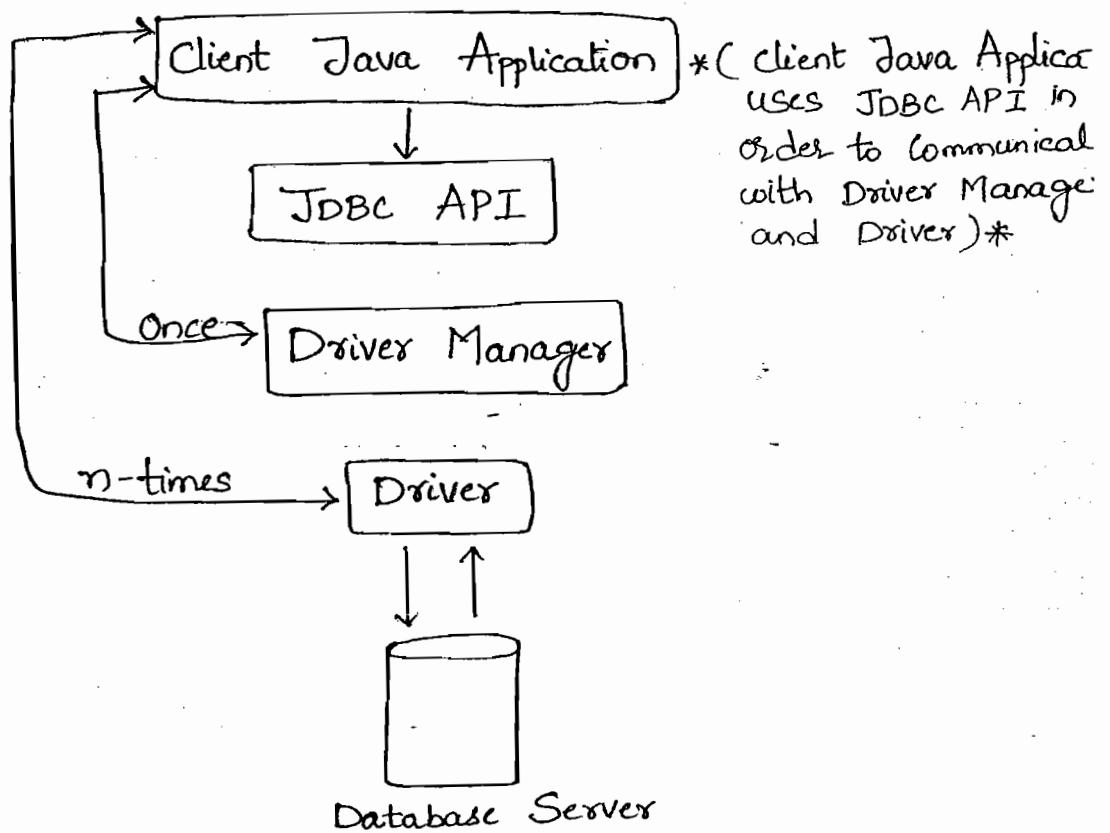
⇒ JDBC is a Service Technology* from Sun Microsystems that enables any kind of Java program to communicate with any kind of Database in a Standard Manner.

Any kind of Java Program ⇒ public static void main programs, Servlet program, EJB program, Applet program etc.

Any kind of Database ⇒ Oracle, MS Access, MySQL, SyBase etc.

⇒ JDBC is an API.

⇒ JDBC is a Specification.

JDBC Architecture

(Client Java Appa uses JDBC API in order to Communical with Driver Manage and Driver)

Without JDBC, no Java program can communicate with Database. JDBC is not a Software nor a Language, it is a technology.

In a Java program, method call is done and an SQL Statement is passed as a String Argument. This method call is not directly passed to database. It is passed to the driver. This driver is a tool which handles method call and hands over the SQL Statement to database. The data returned by the database is again taken by driver and is handed over to Java program. This is in general

Client Java Application

→ What is Client ?

- Person ---> End user
- Computer ---> Client Machine
- Program ✓

In programming terminology i.e., Software terminology, any Software or program that makes a request either for resources or data is a client. Similarly, any Software or program that grants its Services to a client program is Server. The machine on which Client program resides is called a Client Machine and the machine on which Server program resides is called a Server Machine. (Same is the case for browsers. It is a Client program that requests the web Server program for data). Hence, Client & Server does not imply Hardware.

→ Any Computer Application (Software) that makes requests to other application for the sake of resources / data is nothing but a client.

→ In the Context of Java and Database Communication, Java program Should be requesting database Server for the sake of data (Database Operations).

→ Client Java Application has the following responsibilities to get database Services from database Server

- 1) Establishing the Connection with the Database Server (This is a process to process Connect i.e, Java program to Database program connection. This is taken care by another method call in Java)
- 2) Submitting the Appropriate SQL Statement (Appropriate implies, whether to INSERT, SELECT, UPDATE or DELETE the Data according to Business Requirement) to the Database Server.
- 3) Processing (and Presenting) the Results.
- 4) Error (Exception) Handling.
- 5) Closing the Connection with Database.

These are the responsibilities of Java program i.e, indirectly they are the responsibilities of the Java programmer.

All these responsibilities or jobs are performed by methods in java. These are special library methods already developed. The Set of all these library methods is JDBC API.

JDBC API (Application Programming Interface)

By importing these packages, we are making API of Java available to the Java program.

Similarly, By including header files in C-program, we are making C-API available to the C-program.

- A set of library methods is nothing but API (as these methods are acting as an interface to connect to database Server, hence called application programming Interface)
- JDBC API is nothing but some special library methods that enable Java programs to perform database operations with the databases.
- JDBC API is made available to Client Java Program through java.SQL package.

16/07/09

Driver

- It is a translation Software
- A Driver is said to be a JDBC driver if it is written in Java according to JDBC Specification.

JDBC Specification implies a set of rules given by Sun Microsyst-

→ Driver implements JDBC API

→ Driver performs the following Duties :-

- 1) Establishing Database Connection for the Client Java Application. Java program makes a method call whose body is in driver. Hence indirectly Driver makes the Connection possible.
- 2) Receiving the JDBC method calls.
- 3) Translating the method calls into DBMS understandable calls.
- 4) Receiving the Results from the Database.
- 5) Translating the Results into Java format and passing them to the Client.

Note : For Each database operation, Java Application Communicates with JDBC Driver using JDBC API.

println(), thread.start(), ... etc are normal library methods hence come under normal library API. Special library methods used by java to communicate with a DBMS, only come under JDBC API.

Java program, in order to connect to the DBMS, not only requires JDBC API but also the Driver which acts as a Translator.

Call to Submit SQL Statement, one to get Results and process etc. These special methods are library methods in JDBC API.

JDBC API has only library method names. The definitions for those methods are present in the driver. Hence, driver is a java program which contains definitions of almost all the library methods. Hence, driver implements JDBC API. Hence a java program makes use of JDBC API to connect with driver. Connection is with driver but not directly to database. Hence Sun Microsystems gives names for special library methods in JDBC API. The definitions for those methods are written by driver manufacturers. This is the case in advanced Java i.e, Servlets and JSP too.

But in Core Java, special library methods such as println(), thread.start(), ... etc are both named and defined by Sun Microsystems.

Driver Manager :-

→ Java Application requests the Driver Manager for database Connectivity. It initiates Connection process on the Driver. Driver Creates the Connection and gives to Driver Manager which

Driver Manager is the Connection provider to the Java Application but not the Connection Creator.

→ Once Java Program gets connected to Database Server, Driver Manager role ends.

Database Server :-

→ In Industry Strength business (enterprise) applications, almost all the times RDBMS is used as Database Server.

for Example : Oracle,

Sybase,

MS SQL Server, → Microsoft

DB2, → IBM.

Informix →

etc.

→ All these Database Management Systems (Software) understand a common language SQL (Structured Query Language).

MS Access, FoxPro, DBase etc are not used as Databases in industries since they are not RDBMS. They lack in Security.

Java - Database Connectivity Approaches

- JDBC application can connect to database Server using any one of the two Approaches:
- 1) JDBC - ODBC Approach
 - 2) Pure JDBC Approach. (This is Industry Standard).

ODBC ⇒ Open Database Connectivity.

This Technology is given by Microsoft. Only Java programs use JDBC to communicate with Database. Any Other program use only ODBC to communicate with Database. Hence prior Java, languages like C, VB etc used ODBC to connect with the Database Server.

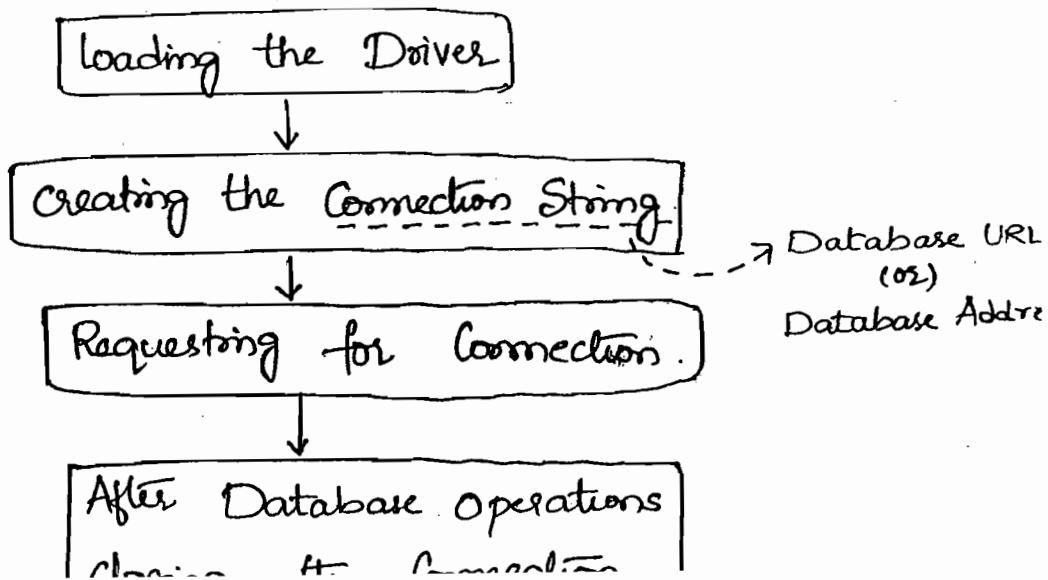
Java program does not use ODBC directly because:

- ODBC is not object oriented and is written in procedure oriented languages like C.
- If ODBC is used, the features of java such as Security, portability etc have to be compromised.

JDBC - ODBC Approach

- In this approach of connectivity, JDBC Application uses JDBC-ODBC bridge driver developed by Sun Microsystems, for database communication.
- Sun Driver (JDBC-ODBC bridge driver) receives JDBC method calls and translates them into ODBC function calls
Hence, one more driver is required in order to convert these function calls into SQL Statements. Hence performance is lost. That is the reason why Industry does not use this Approach.
- To translate these ODBC function calls, (another) ODBC driver is required for the Client (ie, Java Program).

Steps to Connect to Database Service Using JDBC-ODBC Approach



Loading The Driver:

→ JDBC Application needs JDBC driver (Special Software) to communicate with database.

Bringing driver file from Secondary memory into Primary memory is nothing but loading the driver.

Driver implies A Java Program. Hence it is made of a class i.e., it is stored as a class file on hard disk. Therefore, loading is nothing but bringing that class file into RAM.

→ `java.lang.Class` class has a static method "forName()" for this purpose.

Example :

```
Class.forName("sun.jdbc.odbc.JdbcOdbc  
Driver");
```

↑
---, 'Static method.
Hence it is
called using class
name in which
it is available.

Class A {

```
void x() { } ---> instance method (or)  
Static void y() { } Object method (or)  
} Non static method.
```

A a = new A();

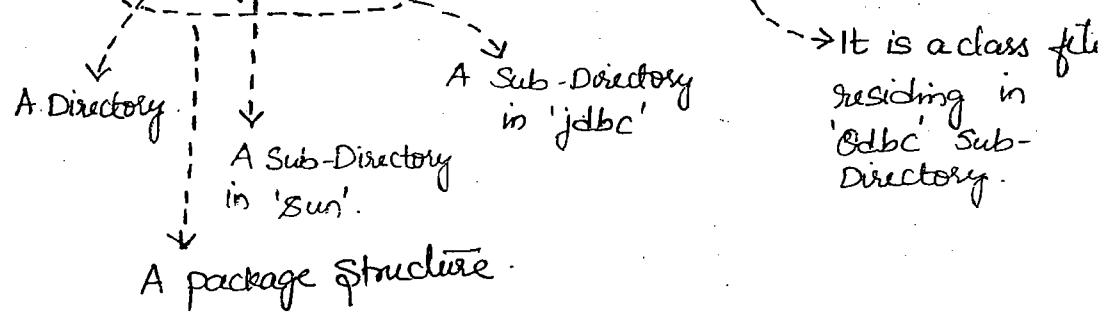
a.x(); ---> method is called on* object
i.e., using a reference, method
is called.

`a.y();` ---> Static method can also be called on the object. It can also be called using Class Name directly, without creating an Object.



`A.y();`

`Class.forName("Sun.jdbc.odbc.JdbcOdbcDriver")`



Hence, we are giving the driver class file as argument to 'forName()' static method and hence dynamical loading the file into RAM.

20/07/09

Creating the Connection String:

→ Database URL is known as Connection String. In JDBC-ODBC Approach of java-database Connectivity, Connection String is as follows:

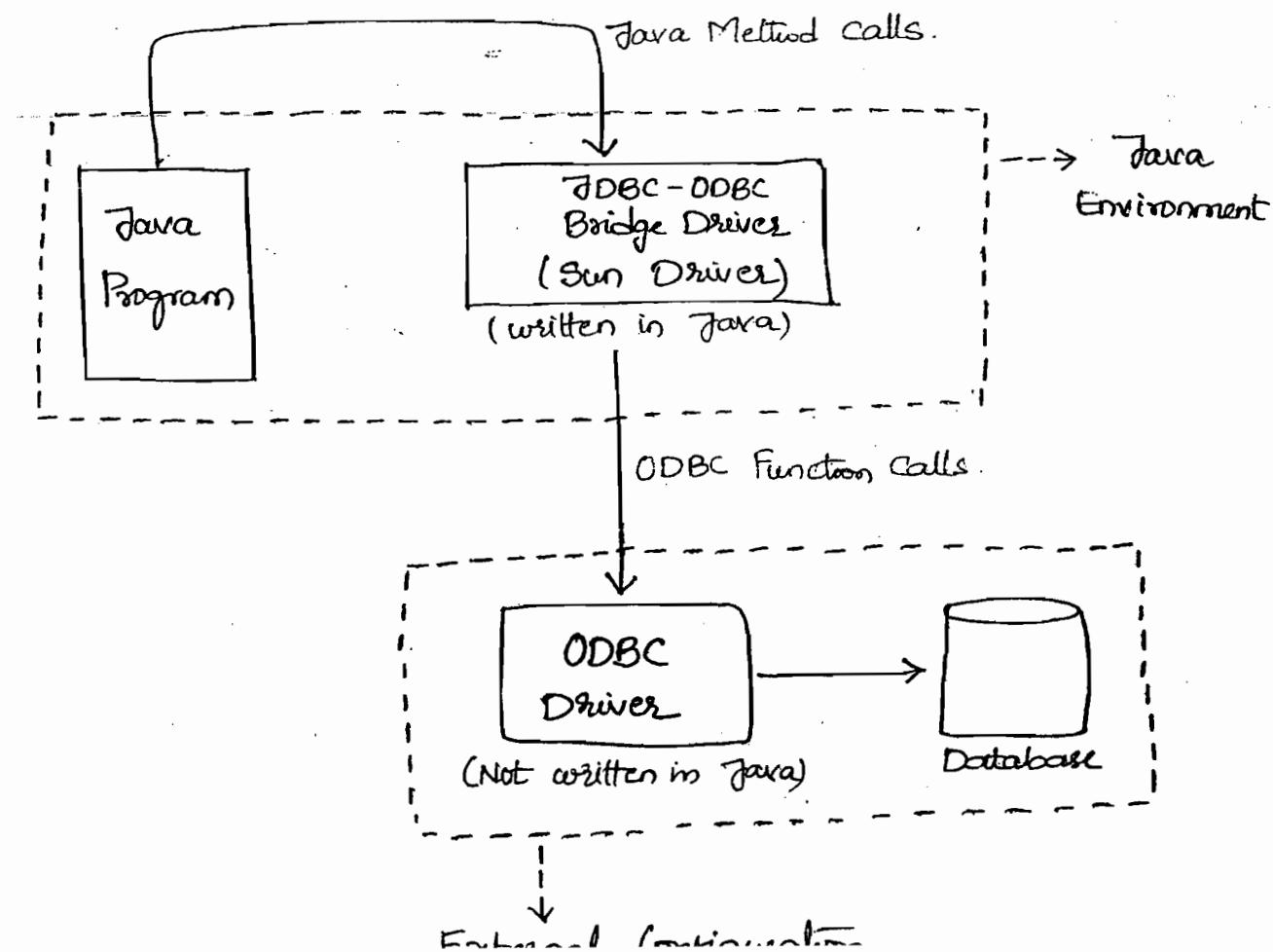
"`jdbc:odbc:dsn`"

Note: In Order to attach ODBC Driver to the JDBC-ODBC Bridge Driver, we create our own DSN.

→ What is DSN?

It is a Special named Configuration in which ODBC Driver and Database are involved.

We Configure Data Source Name External to the Java program and use that Name in the Java program so as to associate ODBC driver with Sun Driver i.e., without directly loading ODBC Driver into Java Environment, we use its Services.



Hence, the External Configuration is Setup outside Java Environment. There is a link between ODBC Driver and Database internally. This Setup is given a Name called Data Source Name (DSN). This DSN is used in Java Program to Connect the Sun Driver with ODBC Driver.

Hence, this Combination of ODBC Driver and Database is configured outside Java Environment and this configuration is given a DSN. Only the Name is used in Java Program, therefore ODBC Driver which is not written in Java is not loaded into Java Environment.

Requesting for Connection:

Driver Manager is also a class.

It has a Static method `getconnection()` for this purpose. This method takes three arguments "database URL", "username", "password". If the database does not require authentication, only one argument is given by using overloading Concept in Java.

- JDBC Application makes the following method call to establish the connection with the database Server process:

The Socket Connection program (a low level Networking program written in Java to connect two processes) is written within the Driver. Driver Manager has a program to manage this Driver. After the Connection object is created and the reference is given to Java Program, Java process to Database process connection is established.

Closing the Connection :

→ After performing Database Operations with the Database Server, JDBC Client Application should close the connection in order to Release the Networking Resources* that are held with the database Server.

Configuration of Database & ODBC i.e., Datasource i.e., Creating DSN :-

Click Start ⇒ Settings ⇒ Control Panel ⇒ Administrative Tools
 ↓
 Data Sources.

Click Add Button ⇒ A list of ODBC Drivers
 Select ⇒ Microsoft ODBC Driver for Oracle.
 Give any Name for the Data Source Name

Hence, we can see that the configuration of DSN

→ Develop a Stand alone JDBC Application that Connect to the Oracle Database Server.

```
import java.sql.*;
```

```
class FirstExample
```

```
{
```

```
public static void main (String a[]) throws Exception
```

```
{
```

```
    Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
```

```
    Connection c = DriverManager.getConnection ("jdbc:
```

There is chance
for class not found ← ←
Exception.

```
        odbc:praveen", "scott", "Tiger");
```

```
    System.out.println ("Connected");
```

There is a chance of
Sql Exception.

```
c.close();
```

```
    System.out.println ("Disconnected");
```

```
}
```

There is a chance
for Sql Exception.

```
}
```

These two types of Exceptions are "Checked Exceptions". Hence they must be Handled. Otherwise Compiler raises three Exceptions.

Checked Exceptions → Compiler will Check whether Exception Handling is done or not.

Unchecked Exceptions → Compiler will not Check whether Exception Handling is done or not.

In the above program, if Exception Handling is not done,

time. The reason for such behavior is as follows: 24.

Consider `forName()` method. It is a called method.

`main()` method is the calling method.

The `forName()` method is a library method and hence would be defined by a Designer. Let it be defined as shown below:

```
public class Class
```

```
{
```

```
    public class forName() throws ClassNotFoundException
```

```
        Exception
```

```
{
```

```
    ===
```

```
}
```

```
}
```

This is just announced,
but Exception handling
is not done.

Hence, we can see that, `forName()` method was just announced to throw `Exception` but `Exception handling` has not been done. Therefore, whenever this method is called, it would become the responsibility of calling method to handle this `Exception`.

Hence, called method is just passing on the duty of `Exception Handling` to the calling method. When the calling method is compiled, as there is a chance for `Exception` to occur (since it is defined in such a manner in the called method), the compiler raises an error in the form of those `Exceptions`, so that the programmer handles the `Exceptions` at

Same Explanation goes for other library methods used in the above program i.e. the methods which raise sql-Exceptions.

In the program, we can see that, instead of handling the Exceptions, we are just bypassing it to the next level by announcing that the main() method 'throws Exception'. Now the Compiler is made to believe that Handling of Exceptions would be done somewhere else.

Stand Alone Application \Rightarrow not internet oriented. A Stand Alone Application has main() method. Servlets, JSP, Beans etc. does not have main() method.

21/07/09

Explanation of the First JDBC Application

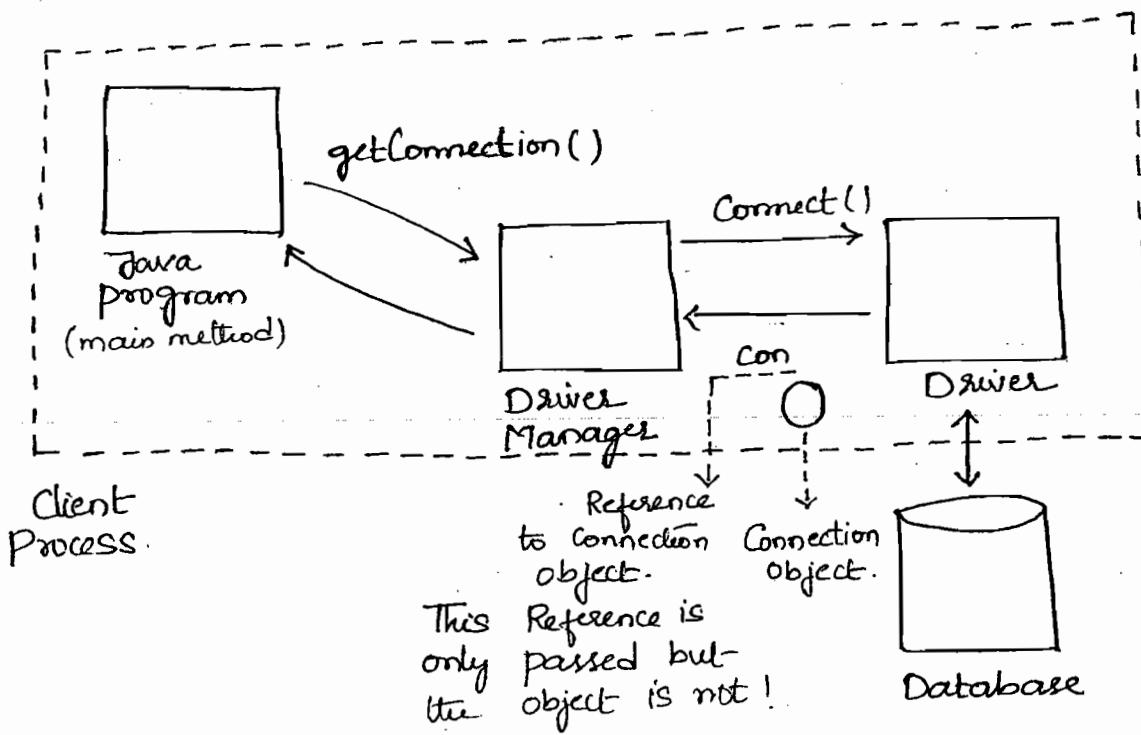
Any Java Program that communicates with Database is a JDBC Application.

- In order to make JDBC API available to the JDBC Application, we import sql-package
- In the main() method body, there is a

the next level. Therefore we have specified 'throws' clause for the main method declaration.

→ When `forName` method is Executed, three things happen in the background:

- 1) Driver class is loaded into memory.
- 2) Driver class Object is created.
- 3) Driver class Object is registered with Driver Manager.



main method calls `getConnection()` method with three arguments, which belongs to `DriverManager`. The `Driver` Object which is already loaded and registered in the above step is supplied with these arguments. `Driver` internally has `connect()` method. Using those arguments and `connect` method `driver` establishes

This socket connection is converted into high level Connection object by Driver. The reference of this object is returned by connect() method of Driver to getConnection() method, which in turn returns this reference to main() method. This is the reference 'c' used in above program.

→ What is a connection in the context of JDBC Application?

Object Oriented Representation of one Database Session of the JDBC Application with the Database Server process is nothing but Connection Object.

→ we close the connection in order to release the Networking Resources at the Database Server side.

22/07/09

The forName() method only loads the Driver Class file into the memory.

Driver Object creation and its registration with Driver Manager is done by a 'static Method' which is defined by the Driver Manufacturers.

```

package sun.jdbc.Odbc;
class JdbcOdbcDriver
{
    This is
    written by
    Driver
    Manufacturers
    according
    to their
    Specifications

    {
        Static
        {
            JdbcOdbcDriver d = new JdbcOdbcDriver();
            JdbcOdbcDriver.registerDriver(d);
        }
        JdbcOdbcDriver()
        {
            System.out.println("Object Created");
        }
    }

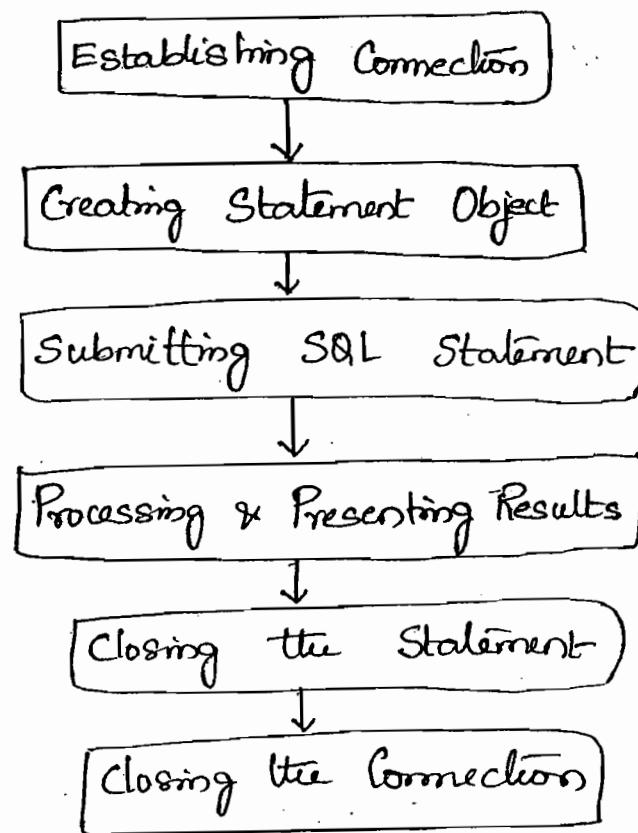
    Class Class
    {
        public static void main(String[] args) throws
        Exception
        {
            Class.forName("sun.jdbc.Odbc.JdbcOdbcDriver");
        }
    }
}

```

only the Driver class file
is loaded into RAM. Object
creation and registration
are done by above Static
Block.

After creating the Connection, the next step would be to perform Database Operations. This is as follows:

Performing Database Operations



Creating Statement Object

→ `java.sql.Statement` Object is the designatory JDBC Object used by JDBC Application to Submit SQL Statement to DBMS.

`java.sql.Statement`

A class in this Package.
The object of this class is used to submit SQL Statement to DBMS.

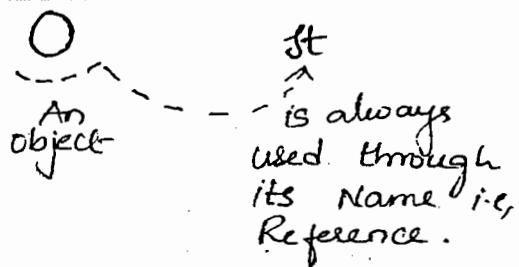
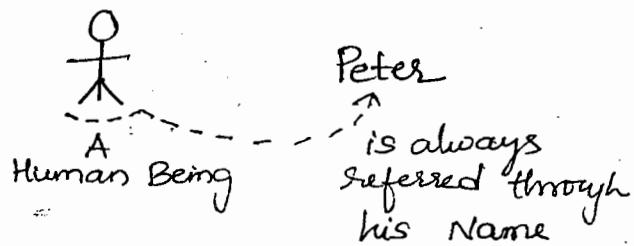
Object, Statement Object is created.

Statement `'st' = con.createStatement();`



Statement Object Name i.e, A Reference to Statement Object.

In Java, Generally, objects are not used directly but their names or References are used. Hence object Services are used through its name. This reference is a pointer to the object i.e, the reference holds address of the object.



Submitting the SQL Statement

→ `java.sql.Statement` has two important methods to submit SQL statements:

- 1) `ExecuteQuery (String dql)`
- 2) `ExecuteUpdate (String dml)`

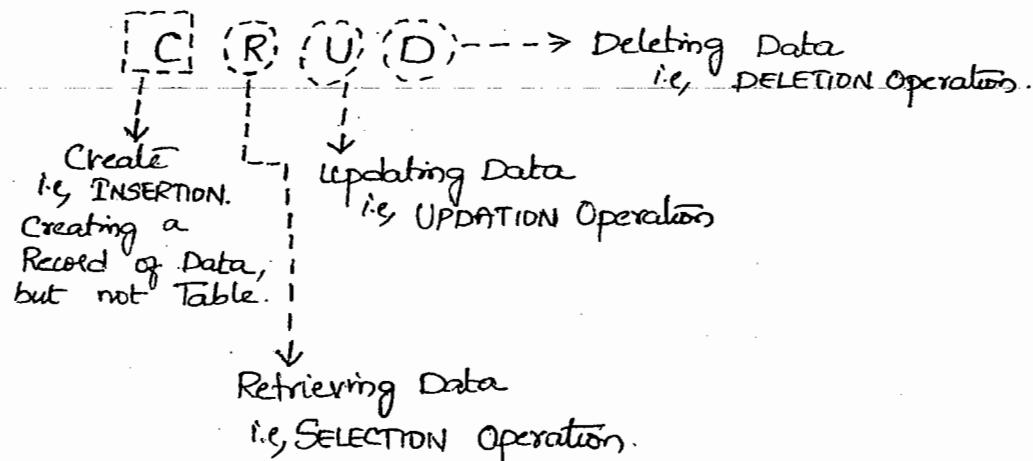
Note: If no arguments are given then dml can be nothing.

→ The first method is used to Submit SELECT Statement to the DBMS and retrieve Data.

→ The Second method is used to Submit INSERT, DELETE and UPDATE SQL Statements to the DBMS. The return type of this method is 'int'.

DDL Operations such as CREATE, ALTER and DELETE Table can also be done using this method. But this is not advisable, as Java programmer cannot involve in such operations. They are performed by Database Administrators.

Therefore, in a Business Process, Generally four operations are performed frequently. They are called CRUD operations.



→ By calling above two methods on Statement object, JDBC Application performs CRUD operations with the Database.

- To Close the Statement \Rightarrow st.close();
- To Close the Connection \Rightarrow con.close();

→ Q) Develop a Stand Alone JDBC Application that stores one Employee Information into the Database.

// Stand Alone Application that Stores one Employee Information

```

import java.lang.*; //optional.
import java.sql.*;

class Storage
{
    int id; → Instance Variable, It has a class scope.

    public static void main(String[] args) throws
        Exception
    {

        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); //optional

        Connection con = DriverManager.getConnection("jdbc:
            odbc:praveen", "scott", "Tiger");

        Statement st = con.createStatement();

        int re = st.executeUpdate("INSERT INTO EMPLOYEE
            VALUES (1001, 'Prasad Rao', 9800)");
            → SQL Statement

        System.out.println(re + " Employee Details stored
            Successfully");

        st.close(); // closing Statement.

        con.close(); // closing Connection.

    } // main

} // class.

```

It is a variable i.e, local Variable as it is defined and used inside a method.

Statement st = con.createStatement();

`(int) re = st.executeUpdate(" ");`

↓
Primitive Variable

Primitive Datatype

This local variable stores a Number, which is nothing but the no. of records affected by executeUpdate() method.

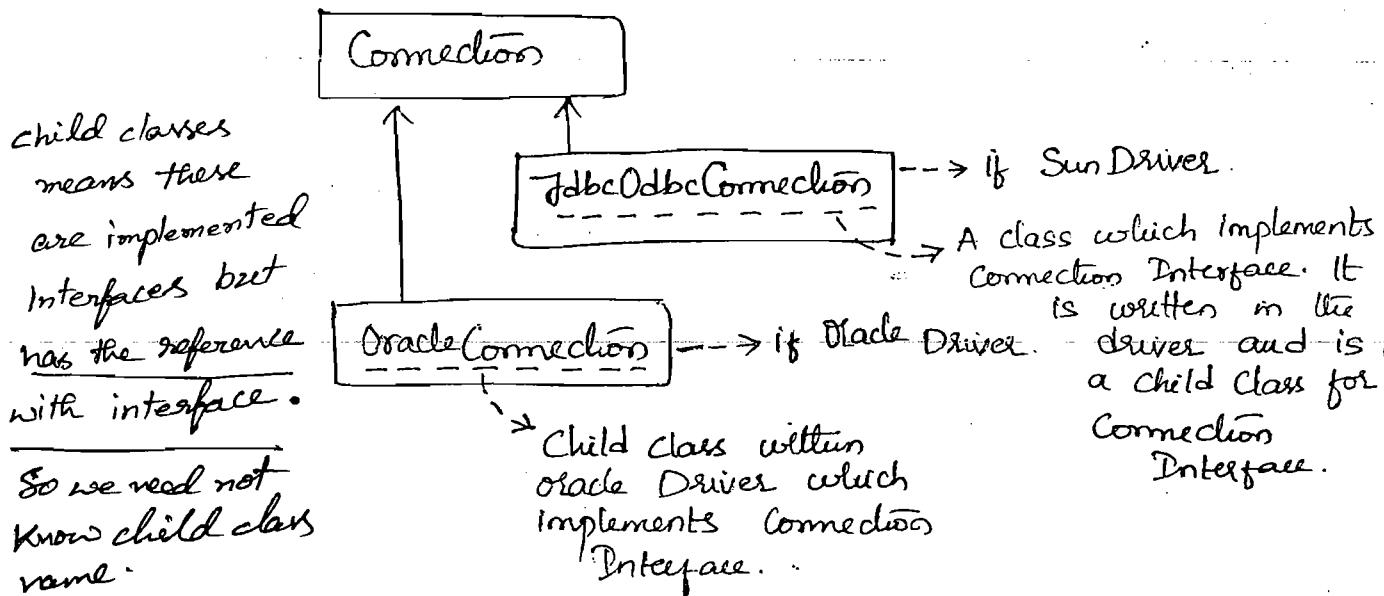
In the above program, loading the Driver class file i.e., Class.forName("sun.jdbc.Odbc.JdbcOdbcDriver"), is mentioned optional because, for 'jdk 1.6' version and higher, the Sun Driver is automatically loaded. This is not the case for lower versions. Other Drivers need their class files to be loaded first, irrespective of the version of 'jdk' used.

executeUpdate() method submits the SQL Statement to the Driver as a method call. Driver translates it to the Database as a pure SQL Statement. (of course, there is ODBC Driver in between i.e., the Sun Driver converts java method calls into ODBC function calls and then ODBC driver submits SQL Statement to Database. Similar procedure in reverse i.e., to get results and give them to java program.) Database, after processing the SQL Statement, performs the required operation (i.e, INSERT, DELETE or UPDATE) and returns a number which is nothing but the number of rows or records affected by that operation. This number is taken by driver and is returned to executeUpdate() method.

method is an integer. This integer value is stored in another local variable and is used in Java Application for processing and presenting purpose.

23/07/09

Connection, Statement are not classes but are Interfaces. Then, con, st are objects of Interfaces would be Blunder. These Objects are created from child classes which implement those Interfaces.

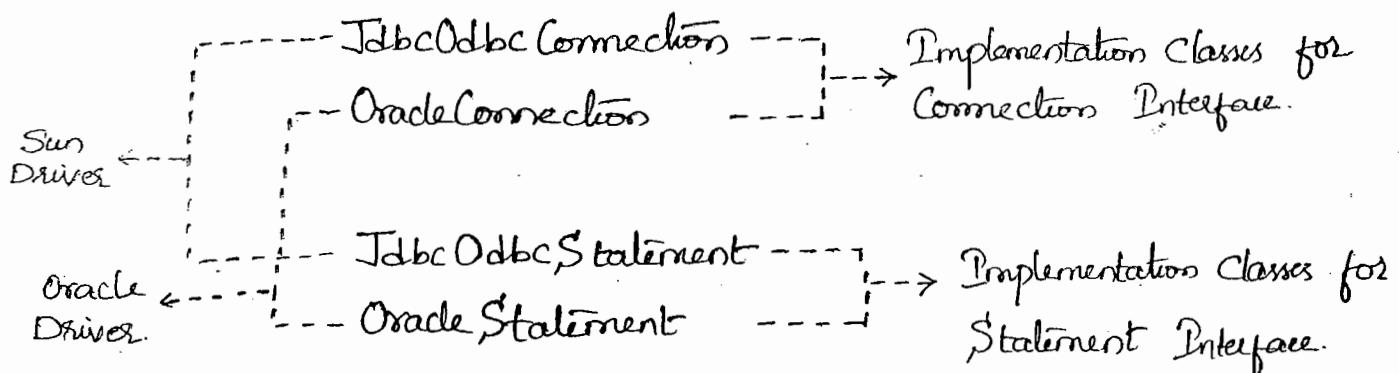


Hence, the objects are created according to the child classes that are present in Driver Software. Therefore, even if the Driver Software changes, the code in Java program need not change, as every Driver Manufacturer defines a Child class which implements the Interfaces. Hence, every Driver should have Child classes to implement all the different Interfaces available like Connection, Statement etc.

System.out.println(con.getClass().getName()); ---
 System.out.println(st.getClass().getName()); ---
They can be
used to know
the actual class
Names of the
Objects created.

Hence, in Advanced Java, J2EE, STRUTS, HIBERNATE etc,
mostly interfaces are used to maintain the flexibility.

Hence Objects are created for Child classes of Different
Drivers which implement the Interfaces.



→ Q) Develop a Stand Alone JDBC Application that prompts the User to enter employee Details and stores that into Database.

/* Stand Alone Application that takes Data from user
 × Stores Employee Details into Database */

```

import java.lang.*; // optional.
import java.sql.*;
import java.util.*;
  
```

```
public static void main( String[] args) throws
Exception
{
```

It is a
class in
util package
to read
data from
Keyboard.

Scanner s = new Scanner(System.in)

constructor takes
Keyboard as
Argument.

```
System.out.print("Enter Employee Number : ");
int eno = s.nextInt();
```

```
System.out.print("Enter Employee Name : ");
```

```
String name = s.next();
```

```
System.out.print("Enter Employee Salary: Rs.");
```

```
float Sal = s.nextFloat();
```

```
Connection con = DriverManager.getConnection ("jdbc:
odbc:praveen", "scott", "tiger");
```

```
Statement st = con.createStatement();
```

```
int re = st.executeUpdate ("INSERT INTO EMPLOYEE
VALUES ('"+eno+"', '"+name+"', "+Sal
+ ")");
```

```
System.out.println (re + " Employee Details stored
Successfully");
```

```
st.close(); // closing Statement
```

```
con.close(); // closing Connection
```

} // main

} // class

→ Q) Develop a JDBC Application that receives Employee Number from the keyboard and Deletes the Employee Details from the Database.

/* Stand Alone Application that Receives Employee Number from user and Deletes that Record from Database */

```
import java.lang.*; // optional.
import java.sql.*;
import java.util.*;
```

Class Deletion

{

```
public static void main(String[] args) throws
Exception
```

{

```
Scanner s = new Scanner(System.in);
```

```
System.out.print("Enter Employee Number for Deletion");
int eno = s.nextInt();
```

```
Connection con = DriverManager.getConnection("jdbc:
odbc:proveen", "scott", "Tiger");
```

```
Statement st = con.createStatement();
```

```
int re = st.executeUpdate("DELETE FROM EMPLOYEE
WHERE Empno = "+eno);
```

```
if (re > 0)
```

```
: System.out.println(re+" Employee Details Deleted")
```

```

System.out.println("Employee Does not Exist");
st.close(); // closing Statement
con.close(); // closing Connection
} // main
} // class

```

- Q) Develop a JDBC Application that increases the Salary of a particular Employee, which Employee , How much Salary to increase, take from Keyboard.
- /* Stand Alone Application that Increases Salary of Particular Employee By Respective Amount as Entered by User */

```
import java.lang.*; // optional.
```

```
import java.sql.*;
```

```
import java.util.*;
```

Class Updation

{

```
public static void main(String[] args) throws
Exception
```

{

```
Scanner s = new Scanner(System.in);
```

```
System.out.print("Enter Employee Number : ");
```

```

System.out.print("Enter Amount to Increase : Rs. ");
int amt = s.nextInt();
Connection con = DriverManager.getConnection("jdbc:odbc:
praveen", "scott", "Tiger");
Statement st = con.createStatement();
int re = st.executeUpdate("UPDATE EMPLOYEE SET SALARY
SALARY + " + amt + " WHERE EMPNO = " + empno);
if (re > 0)
    System.out.println(re + " Employee Salary Details Updated");
else
    System.out.println("Employee Does not Exist");
st.close(); // closing Statement
con.close(); // closing Connection
} // main
} // class.

```

→ How to locate Driver?

jdk1.6\jre\lib\rt.jar;

↓
This contains all
the packages i.e., util,
lang, sql etc

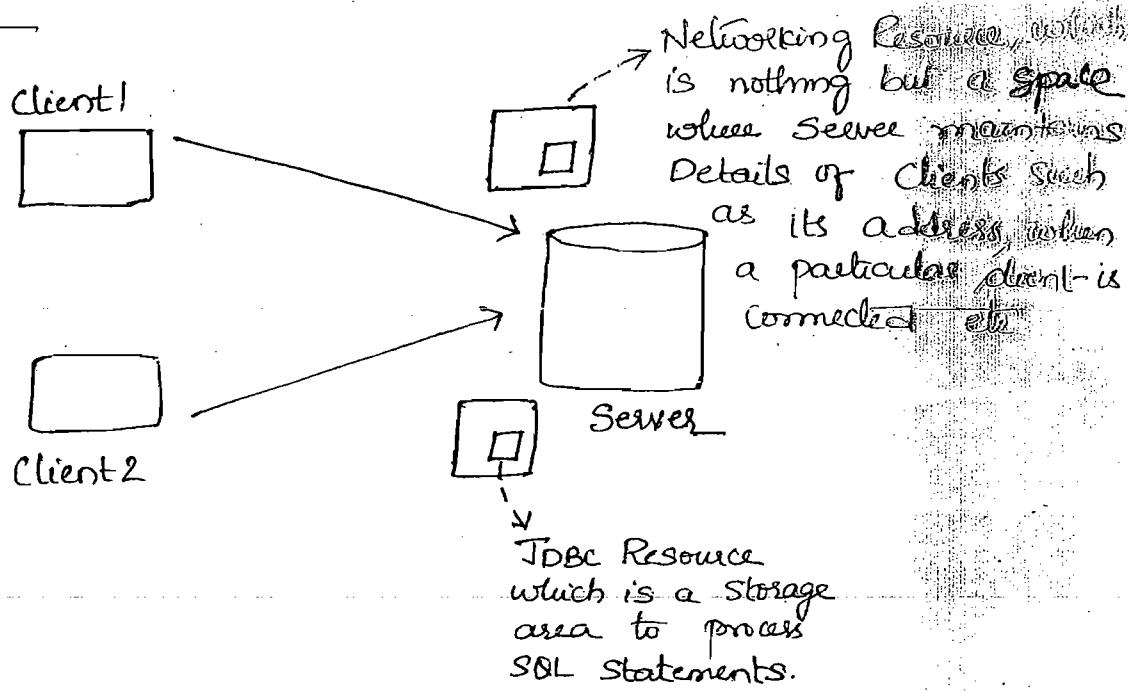
The .jar file can be unzipped. It has to be done in a Separate folder.

Ex:- E:\Folder> jar -xf rt.jar

24/07/09

Connection is closed (i.e. `con.close()`) in order to release the Networking Resources.

Statement must be closed (i.e. `st.close()`) in order to release JDBC Resources. This Resource is nothing but a Storage Space which is allocated to process the SQL Statements.



From the above figure it is evident that, JDBC Resources must be freed first and then the Networking Resources, by using `close()` method on their respective objects.

Database \Rightarrow Data that is stored in Tables.

DBMS \Rightarrow A Software to manage Data of a Database.

Java program always tries to communicate with DBMS but not with Database directly. It is not possible to communicate with the storage Medium.

Retrieving Data from Database :-

- JDBC Application needs to Submit SELECT Statement to DBMS in order to Retrieve Data.
- By calling executeQuery() method on the Statement Object, SELECT Statement is submitted to the DBMS for Example :

```
ResultSet rs = st.executeQuery("SELECT * FROM EMPLOYEE");
```

DBMS Returns the records to Driver, which inturn Returns them to Java Program as an Object. This is called Result Set Object.

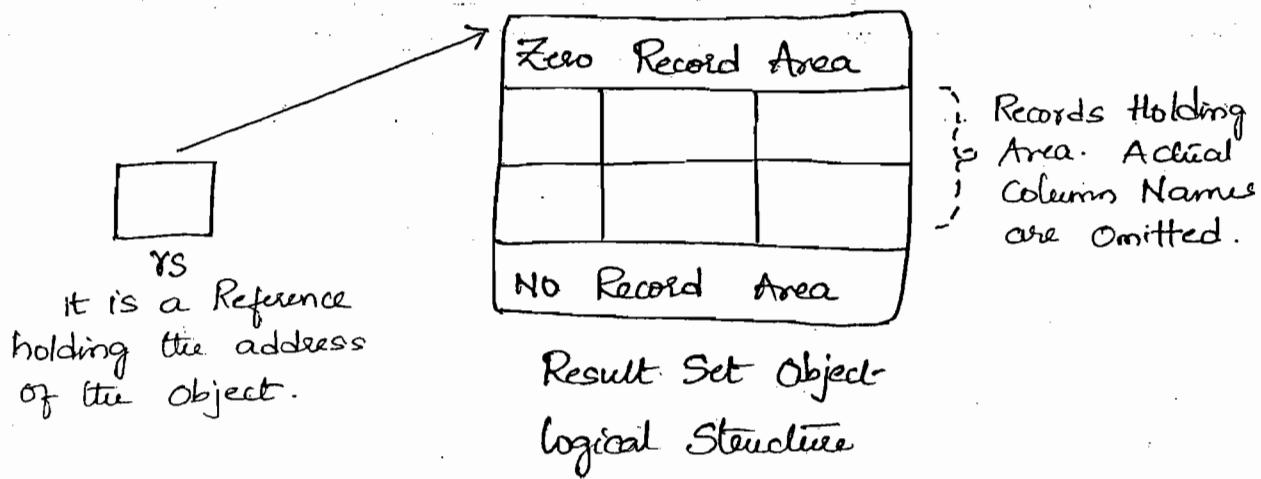
This ResultSet is also → we need not child class name
an Interface, hence its object is
Created from a child class which implements this
Interface i.e. ResultSet Interface.

Hence, a ResultSet is nothing but Object Oriented Representation of a table of data coming from DBMS.

- What is Result Set ?

Object Oriented Representation of ie. Encapsulation of a table of records that are Returned from DBMS to the JDBC Application, is nothing but ResultSet Object.

The figure below Represents internal make-up of ResultSet object.



- When the ResultSet Object is Constructed, that is ResultSet is open, cursor points to Zero Record Area.
Constructed ⇒ when `executeQuery()` method is called and then SQL Statement is Submitted, ResultSet object is created.
- The cursor is a logical pointer, pointing to Zero Record Area by default.
- To move the cursor to the Records and to Retrieve column values, `java.sql.ResultSet` provides methods.
- ResultSet Object provides getter methods to Retrieve column Values.

for Example : `int en = rs.getInt(1);`

`String nm = rs.getString(2);`

`float sal = rs.getFloat(3);`

These getter methods take column Numbers as Argument and Returns corresponding Data i.e. column Values. These

These methods get apply to the record on which the cursor is pointing. Hence cursor movement methods are first executed and then data retrieval methods i.e. the getter methods with column numbers, are applied. Therefore Row Number concept is not necessary.

→ To move the cursor in the ResultSet, we have an important 'next()' method. This method will do two things:

- 1) moving the cursor in forward direction by one record (area).
- 2) Returning True if Record is present, Otherwise Returning False.

There are around 15 methods to move the cursor i.e., for cursor movement.

When `rs.next()` method is used for first time, Cursor moves from ~~Zero~~ Record Area to first Record in Record Holding Area. If record is present, it Returns True otherwise False.

Now, when `rs.next()` method is again used, i.e., for Second time, the cursor moves to the next Record. Hence Repetition of code is occurring. To avoid this, the process is kept under loop.

`next()` method moves the cursor in forward direction only and one record at a time. These are its

→ Q) Develop A Stand Alone JDBC Application that prompts the End user to enter Employee Number and displays the Employee Details Retrieved from Database. With that number if no Employee exists, display the same.

/* Stand Alone Application that Retrieves Employee Details from Database Prompting the User to Enter Employee Number */

```
import java.lang.*;
import java.sql.*;
import java.util.*;
```

Class Retrieval1

{

```
public static void main(String[] args) throws
Exception
```

{

```
Scanner s = new Scanner(System.in);
```

```
System.out.print("Enter the Employee Number :");
int eno = s.nextInt();
```

```
Connection con = DriverManager.getConnection("jdbc:
odbc:praveen", "scott", "Tiger");
```

```
Statement st = con.createStatement();
```

```
ResultSet rs = st.executeQuery("SELECT * FROM
```

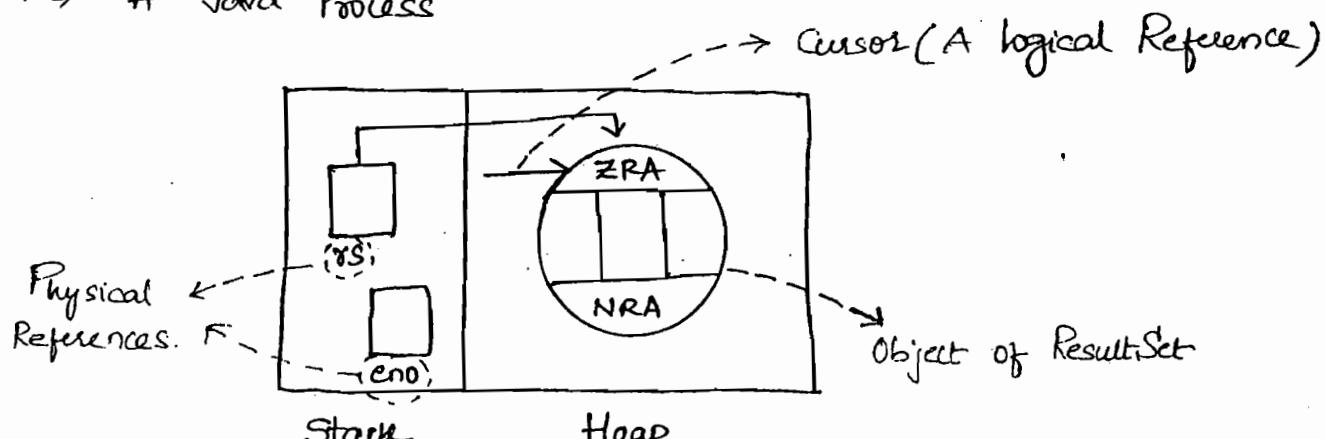
```

if (rs.next())
{
    System.out.println("Employee Name : " + rs.getString(2));
    System.out.println("Employee Salary : Rs. " + rs.getFloat(3));
}
// if
else
{
    System.out.println("Employee Does Not Exist");
    rs.close(); // closing ResultSet
    st.close(); // Closing Statement
    con.close(); // Closing Connection
}
// main
} // class.

```

Driver constructs the ResultSet object irrespective of whether Data is coming from Database or not. If no record or no Data is coming from Database, an Empty ResultSet object is created, provided a legitimate SQL Statement is submitted.

Consider the portions of RAM where the process is being Executed:
i.e A Java Process



Objects are always created in Heap Area. Their References i.e., local Variables in main method are created in Stack Area.

→ Q) Develop A Stand Alone JDBC Application that displays all the Employees Name and Salary whose Salary is more than Rs.8000.

/* Stand Alone Application That Displays All Employees Details with Salary Above Rs.8000 */

```
import java.lang.*;
import java.sql.*;
```

Class Retrieval2

{

```
public static void main(String[] args) throws
Exception
```

{

```
Connection con = DriverManager.getConnection("jdbc:
odbc:praveen", "scott", "Tiger");
```

```
Statement st = con.createStatement();
```

```
ResultSet rs = st.executeQuery("SELECT NAME,
SALARY FROM EMPLOYEE WHERE
SALARY > 8000");
```

```
if (rs.next())
```

```

System.out.println();
System.out.println("Name <|> Salary");
System.out.println("-----");
do
{
    System.out.println();
    System.out.println(rs.getString(1) + "<|>" + rs.getFloat(2));
} while(rs.next());
}

// if
else
{
    System.out.println("No Employee Exists with Salary Above Rs.800");
    rs.close(); // Closing Set i.e, Result Set
    st.close(); // closing Statement
    con.close(); // closing connection
}

// main
}

// class.

```

→ What happens within DBMS, when an SQL Statement is submitted to it ?

1) Query Compilation.

i.e, Database Engine Checks for Validity of SQL Statement i.e, whether it is Syntactically correct or not.

i.e How best the Query can be Executed is checked and Plans Generated. Many plans are generated by the Engine, in order to choose the best method i.e, the best possible way to process the Statement and give results. This includes managing I/O cycles, memory etc.

3) Query Execution.

27/07/09

→ What is the limitation of Statement Object?

Statement Object is capable of Submitting any kind of SQL Statement from the JDBC Application to the DBMS.

Whenever Same SQL Statement is required to be Submitted repeatedly (ofcourse with different set of values), Statement Object should not be used, for performance Reasons.
(i.e, if it is used, the performance Reduces)

To overcome this problem, we make use of Child of Statement Object i.e, java.sql.PreparedStatement.

Things Common between Statement & Prepared Statement:

- i) Both are Interfaces
- ii) Both belong to SQL Domain

v) Both have implementation classes in Driver Software.

Consider the following Example:

Say,

To Insert First Record,

```
st.executeUpdate("Insert Into Employee Values(1001, 'Rama', 8000);")
```

To Insert Second Record,

```
st.executeUpdate("Insert Into Employee Values(1002, 'RAMU', 9000);")
```

To Insert Third Record,

..... and so on.

Everytime, for Each Record, Syntax checking i.e Query compilation and Query Plan Generation is done for a Similar Operation with different values. hence, whenever Repetation is involved, Statement object must not be used.

A practical Example for such a process would be a Banking Application, where many customers have similar tasks like:

Balance Enquiry (SELECTING the Data)

Deposits & withdrawls (Updating the Data)

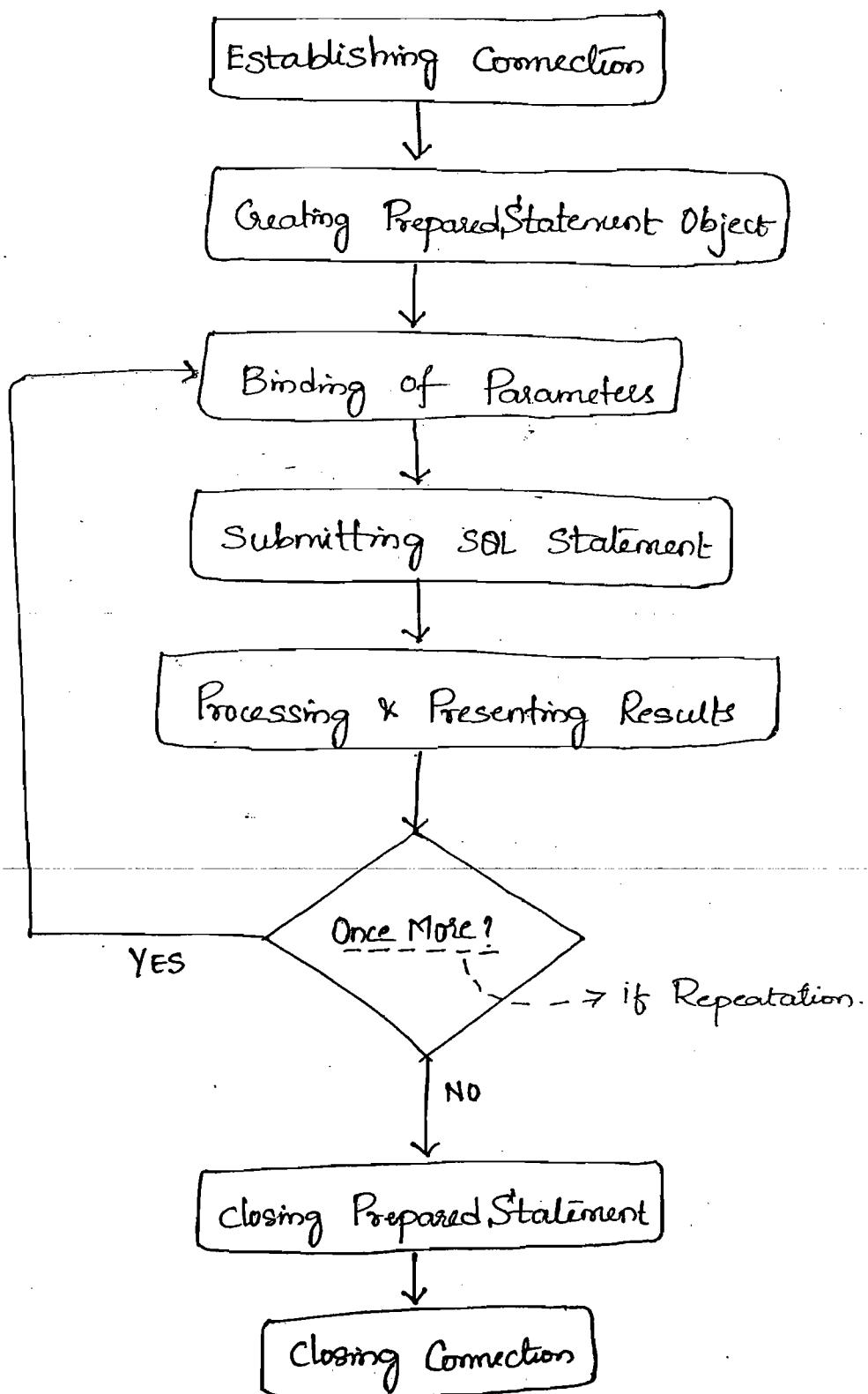
Creating New Accounts (Inserting the Data)

Closure of Accounts (Deleting the Data)

Hence, When Repetation of Similar Database Operation with Different Values is the Scenario, go for "PreparedStatement"

When there is a need submit the SQL Statement only once,

Using PreparedStatement in JDBC Application



Establishing Connection, Processing & Presenting Results, Closing Prepared Statements, Closing connection are "delet"

Creating the Prepared Statement

→ By calling `prepareStatement(String sql)` method on the Connection object, we create the Prepared Statement Object.

for Example :-

```
PreparedStatement ps = con.prepareStatement("INSERT INTO
EMPLOYEE VALUES (?, ?, ?);");
          ↓      ↓      ↓
          |      |      |
          ↓      ↓      ↓
          Place Holders.
```

Driver Receives this

Method call and translates it into DBMS understandable Statement DBMS Software, after Receiving the Statement, performs Query compilation & Query Plan Generation. It does not Execute Query as Values are not given but instead, place holders are placed.

DBMS Software, gives this precompiled SQL Statement, back to the driver. Driver converts this Statement into Prepared Statement Object. Hence, "Prepared Statement" Object is holding already prepared i.e., pre-compiled SQL Statement. The Reference of this Object is given back to Java Program, by the Driver. Hence, SQL Statement is placed again in Java Environment.

Now, the values are Bound with Place Holders. This SQL Statement with values, is again Submitted to DBMS. But this time, Query Compilation & Query Plan Generation are not done but the statement is directly run i.e. Executed

→ Note :- Encapsulation of (Object Oriented Representation of) a pre-compiled or already prepared SQL Statement is nothing but Prepared Statements Object.

In Comparison, A Statement object is empty as nothing is provided as argument when it is created i.e., `con.createStatement();`

↓
empty.

Binding of Parameters :-

→ Supplying values to the place holders (question Marks) is nothing but Binding of Parameters.

→ By calling Setter methods (mirrors to getter methods) (of ResultSet, getter methods are used to Retrieve values) on the PreparedStatement object, we bind the parameters

Running Notes ← for Example :-

```
ps. SetInt(1, 100);           Numbers
                                         corresponding to
                                         question Marks in
                                         SQL statement
                                         but not column
                                         Numbers.
```

ps. SetString(2, "Rama");

ps. SetFloat(3, 8000);

These methods take two arguments, one is the question mark number i.e., index number corresponding to the occurrence of a particular question mark (place holder)

↓ ↓ Argument passed to this

Submitting the SQL Statement

```
int re = ps.executeUpdate();
```

OR

```
ResultSet rs = ps.executeQuery();
```



No arguments are supplied because Prepared Statement Object is not empty.
It is holding an SQL Statement.
Earlier, Statement Object "st" was empty, hence an SQL statement was supplied as argument to these methods.

→ Q) Develop A Stand Alone JDBC Application, that prompts the user to enter Account Details (Accno, Name, Balance) repeatedly until the user opts out and stores the same in the Database.

/* Stand Alone JDBC Application for Repeated Account Creation */

```
import java.lang.*;
Import java.sql.*;
import java.util.*;
```

{

public static void main(String args[]) throws
Exception
{

Connection con = DriverManager.getConnection("jdbc:odbc:
praveen", "scott", "Tiger");

PreparedStatement ps = con.prepareStatement("INSERT
INTO ACCOUNT VALUES(?, ?, ?)");

Scanner s = new Scanner(System.in);

while(true)

{

System.out.println();

System.out.print("Enter Account Number : ");

int ano = s.nextInt();

System.out.print("Enter Account Holders Name : ");

String name = s.next();

System.out.print("Enter Balance : Rs. ");

float bal = s.nextFloat();

ps.setInt(1, ano);

ps.setString(2, name);

ps.setFloat(3, bal);

ps.executeUpdate();

System.out.println("One Account Record Stored Successfully");

System.out.print("Do you want to continue with

55.

```
String choice = s.next();  
if (choice.equalsIgnoreCase("no")) break;  
} // while  
ps.close(); // closing PreparedStatement  
con.close(); // closing Connection  
} // main  
} // class
```

→ Q) Develop A Stand Alone Application, that offers Continuous Deposit Service.

// Stand Alone Application That offers Deposit Service.

```
import java.util.*;  
import java.sql.*;
```

```
Class DepositService  
{
```

```
public static void main (String[] args) throws Exception  
{
```

```
Connection con = DriverManager.getConnection ("jdbc:  
odbc:praveen", "scott", "Tiger");
```

```
PreparedStatement ps = con.prepareStatement (  
"UPDATE ACCOUNT SET BALANCE =  
BALANCE + ? WHERE ACCNO = ?");
```

```
Scanner s = new Scanner(System.in);  
while(true){  
    System.out.println();  
    System.out.print("Enter Account Number : ");  
    int ano = s.nextInt();  
    System.out.print("Enter Depositing Amount : Rs. ");  
    float amt = s.nextFloat();  
    ps.setFloat(1, amt);  
    ps.setInt(2, ano);  
    int n = ps.executeUpdate();  
    if(n>0)  
        System.out.println("Amount Deposited Successfully");  
    else  
        System.out.println("Cannot Deposit. Enter correct  
                           Account Number....");  
    System.out.print("Do you want to continue  
                   with Depositing Another Amount ?  
                   (yes/no) : ");  
    String choice = s.nextLine();  
    if(choice.equalsIgnoreCase("No")) break;  
}  
// while  
ps.close(); // Closing Prepared Statement  
con.close(); // closing Connection  
}  
// main
```

→ Q) Develop A Repeated Account Deletion Application
i.e, Repeated Account Closure Application

// Stand Alone Application For Repeated Account Closure

```
import java.sql.*;  
import java.util.*;
```

```
class AccountClosure  
{
```

```
    public static void main(String[] args) throws Exception  
{
```

```
        Connection con = DriverManager.getConnection("jdbc:odbc  
praveen", "scott", "Tiger");
```

```
        PreparedStatement ps = con.prepareStatement("DELETE  
FROM ACCOUNT WHERE AccNo = ?");
```

```
        Scanner s = new Scanner(System.in);
```

```
        while(true)
```

```
{
```

```
        System.out.println();
```

```
        System.out.print("Enter Account Number : ");
```

```
        int ano = s.nextInt();
```

```
        ps.setInt(1, ano);
```

```
        int n = ps.executeUpdate();
```

```

else
    System.out.println("Account Does Not Exist");
    System.out.print("Do you want to continue with
                    the Account closure Process? (yes/no):");
    String choice = S.next();
    if (choice.equalsIgnoreCase("No")) break;
}
// while
ps.close(); // closing Prepared Statement
con.close(); // closing Connection
}
// main
}
// class

```

→ Q) Develop A Stand Alone JDBC Application that provides Balance Enquiry Service to All the customers.

// Stand Alone Application That Provides Balance Enquiry

```

import java.sql.*;
import java.util.*;

```

```

class BalanceEnquiry
{

```

```

    public static void main(String[] args) throws
        Exception
    {

```

Connection con = DriverManager.getConnection ("jdbc:odbc:
proveen", "scott", "Tiger");

PreparedStatement ps = prepareStatement ("SELECT BALANCE
FROM ACCOUNT WHERE ACCNO = ?");

while (true)

{

System.out.println();

System.out.print ("Enter Account Number : ");

int ano = s.nextInt();

ps.setInt (1, ano);

ResultSet rs = ps.executeQuery();

if (rs.next())

System.out.println ("Account Balance : " + rs.getFloat (1));

else

System.out.println ("Account Does Not Exist");

rs.close(); // for each Iteration A New ResultSet object
is created, Hence we need to close the
previous ResultSet.

System.out.print ("Do you Want to Continue the
Balance Enquiry (Yes/No) : ");

String choice = s.next();

if (choice.equalsIgnoreCase ("No")) break;

} // while

ps.close(); // closing PreparedStatement

JDBC  Part-1 (Fundamentals)
Part-2

Callable Statement

Types of Result Sets

Result Set Meta Data (RSMO)

Rowsets

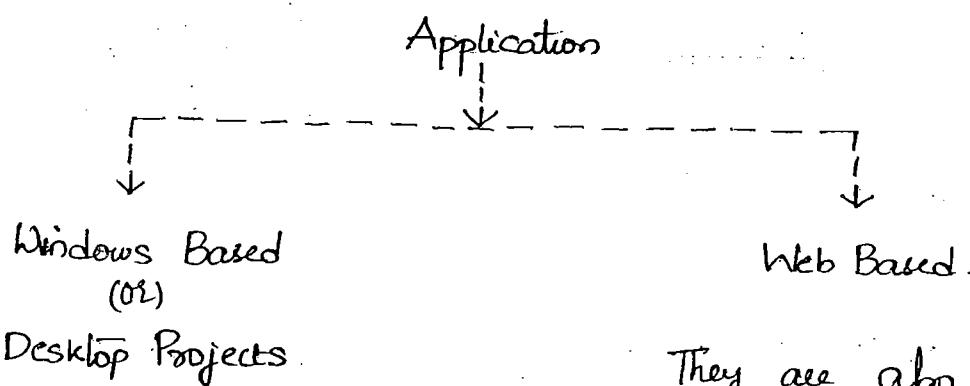
Types of Drivers

Batch Updates.

All these topics of Part-2 will
Be covered later.

WEB APPLICATION DEVELOPMENT

Any Project or Application can be :



In these projects, Customers do not have the flexibility to Access the Business Services as they like. They need to Contact the Business Organization Physically.

End Users for Such Applications would be the Staff of that Particular Organization.

The Front End (or) Screens here are developed using :

They are also known as on-line Applications. These Applications give the flexibility to customers so that, they can Access the Business Services of a Particular Organization, whenever and wherever Required.

The Front End (or) Screens here are developed using HTML for both java & .NET Based Projects.

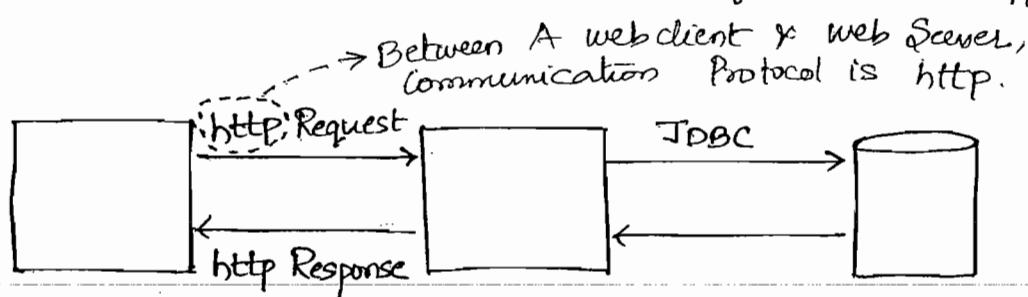
→ What is a web Application?

Any Computer Application is said to be a web Application if we can use its Services through web (internet Service).

Web Applications are meant for Providing Online Business Services to customers 24x7x365

Web Applications are also known as Online Applications (not Stand Alone).

→ What is the Architecture of A web Application?



Web Client
(Developed using
HTML (or)
JavaScript)

Web Server
(Developed using
Servlets, JSP,
JavaBeans etc..)

Database Server

All these three processes can run on Same Machine
(or) three different Machines (which is the preferred Industrial Approach)
(or) Webserver & Database on one machine and web client Process on other.

Web client is a Browser Process.

Web server process includes Servlets, JSP, JavaBeans etc.

All these processes use JDBC to communicate with Database. Hence, JDBC provides service to these technologies and is

Within browser, web pages should run & these web pages are developed using HTML & JavaScript. The HTML pages (web Pages) are stored in Server Machine and are loaded into client process i.e., browser in Client Machine on the fly. The browser executes these pages. Browser can execute HTML code, JavaScript code and also Applet code.

Web Client

→ It is a Software that requests the web server for web Resources.

Software ⇒ Browser Software

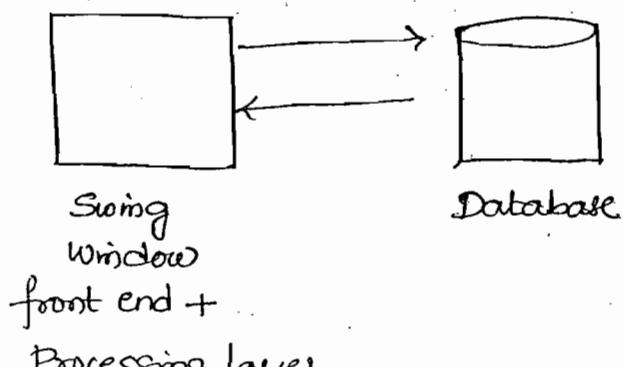
Resources ⇒ web pages, images, pdf files etc.

→ In fact, it is the Browser Software. A web client is also known as HTTP Client as it uses HTTP as Communication Protocol to interact with the web server.

HTTP ⇒ Hyper Text Transfer Protocol.

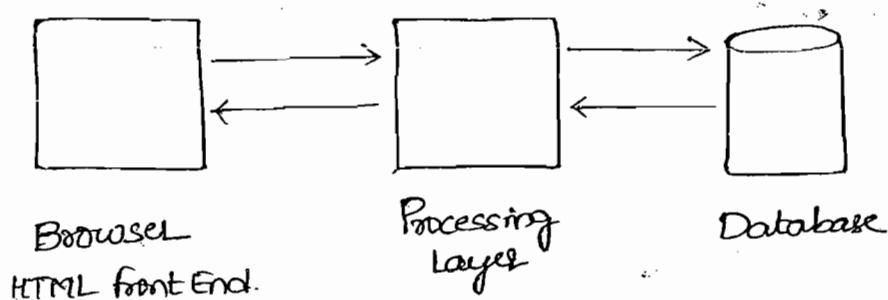
→ Duties of web client:

- * Requesting for Connectivity
- * Receiving the User Input from the webpage and forwarding to the web server
- * Receiving the HTML documents from web server
- * Executing HTML document and Rendering the web page to the user.



→ Processing code is included along with Swing code

A Desktop Application (OR) Windows Based Application Using JAVA



A Web Application Using JAVA.

Web Server

→ It is a Server Software (i.e. A Process but not Hardware). It is also known as HTTP Server.

→ Duties of Web Server are:

- * Providing Connection to the Web Clients
- * Receiving the Request from Web Client and

→ What are the limitations of Web Server?

If we use web server alone to support a website, it becomes only informative website but cannot be an interactive website.

informative website ⇒ We can perform only Read Operation on the website. User cannot Enter any input into the website. All he can do is Read the Pages.

interactive website ⇒ We can perform both Read & write Operation on the website.

A Commercial Business process always Requires Interactive Websites. Whenever user input is involved in Client-Server Process Communication, i.e., whenever user input data is sent in client request, web server can only take these inputs but it cannot process the input, cannot contact the database and it cannot create response page on its own. All it can do is to deliver, already created pages that are stored in the Server Machine. Hence web server alone is only informative and it cannot interactively support a website.

Types of Web Programming

→ In the context of a web Application (website) Development, we have two kinds of Programming:

- 1) Client Side Programming.

66.

Programming, both for one website only i.e one Application.

Client Side Programming is nothing but Javascript (Jscript, VB Script etc) Programming, which validates HTML Code i.e it forces the user to Enter Correct Input, Complete Input and Valid Input into the website or web-page.

→ Javascript code that is executed by the Browser, that performs client side validations in order to force the end-user to Enter Complete and Proper input into the web form is known as Client Side Program.

Website ⇒ A Collection of web Pages - A website with say, 30 Pages may have around 10 Pages that require input. Remaining 20 pages do not need validation as they are Read-only.

→ What is the need of Server Side Programming, in the context of a website (web Application) ?

To overcome the limitation of web server and assist it in making the website Interactive.

∴ Informative and Interactive Web Application (OI) Website =
Web Server + Server Side Programming.

→ What are the General Duties of a Server Side Program

- 1) Capturing the user input (from web Server Process)
- 2) Communicating with Database mostly, and Processing the Data.
- 3) Creating Response Page that Comprises of Processed Data and Handing over the same to web Server.

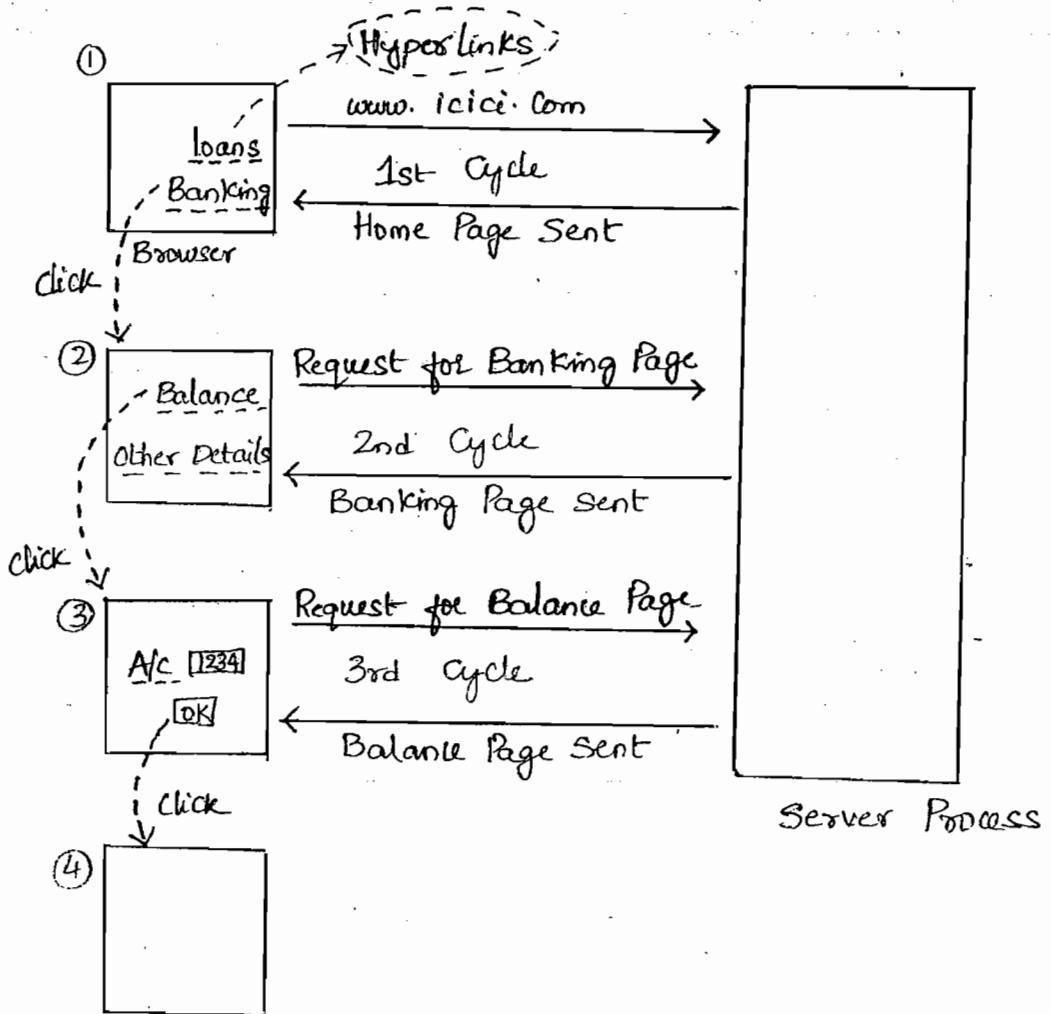
Web Server Process, alone, has Networking Module which makes connection possible i.e., this module has a Server Socket Program (similarly, at client side we have Client Socket Program) in order to communicate with client i.e., to connect with client.

Web Server Process also has an I/O Module which has the program to process client Request, Retrieve the pages required from Hardisk (this is Read operation of Server process) and hand those pages over to client (this is write operation of Server Process). This would work for pages already stored in the Server Machine. Hence web Server uses the Service of Server side Programming in order to Process the Data inputted by the user, according to Business Process and then becomes Interactive.

Types of Web Pages

→ We have two kinds of Web Pages in a website (such as -

Consider the following Process:



Pages ①, ②, ③ are Static Pages. All these pages would display same content to all the users who access them. They are already created and stored in the file system of Server Machine.

Page ④ is a Dynamic Page. It displays different contents to different users. A Server Process alone cannot generate this page. Hence Server Side Programming is needed so as to process the input data, fetch appropriate Record from database, Generate a web page dynamically according to the results from database and hand this Dynamic Page to Server Process. Server process, now, returns this Page to client i.e. browser.

→ Whether a page is Static (OR) Dynamic is decided by two criteria:

1) When is the Page Produced

i.e. Page without need for input data or
Page Produced after Data is Entered

i.e. Page already created or Page Produced
after user requests with input Data

i.e. whether informative or interactive.

2) whether its contents remain same for every
Client Request or not.

Dynamic Pages cannot be found on the Server Machine file system. They are generated programmatically.

Static Pages are HTML files that are already created and stored in file system of Server Machine.

→ What are the different technologies available to develop Server Side Coding?

CGI - Perl (Common Gateway Interface), ASP.NET,
PHP, ColdFusion, Servlets*, JSP* etc

JDBC → Service Technology

Servlets, JSP → Web Technologies.

70.

O O O O O O O O O O O O O O O O

Servlets

Before Servlets, CGI-Perl was used for Server Side Programming. In Perl, Each Client is treated as Single process. As no.of Clients increases, - no.of processes increase and hence performance decreases. To overcome this drawback, Servlets like technologies are used. In servlets, no matter how many processes are connecting (i.e, clients), all of them would be treated as One Single Process with multiple Threads. Hence performance is Better.

→ "Servlets" is one of the Web Technologies from Sun-Micro Systems

→ Servlets is a Specification.

JDBC is also a Specification i.e, a Standard Set of Rules for Driver Manufacturers to follow.

Similarly, Servlet Specification is to be followed by Server Software Manufacturers, to maintain Uniformity i.e, even though Server Changes Servlet Code need not change.

→ Servlets API is used by Web Application Developers

Hence, Specifications are for \Rightarrow Software Manufacturers
 APIs are for \Rightarrow Application Developers.

Servlets API is developed by Sun Microsystems and are implemented by web Server Manufacturers. This is Similar to the case with JDBC where Oracle Software has Implementation Classes for Several Interfaces. Servlets API is used by programmers to communicate with Web Server.

\rightarrow Servlets Technology is from Enterprise Edition of Java.

Major Part of JDBC is JSE and a minor part is J2EE

JSP, EJB, Servlets are all Technologies from Enterprise Edition.

The purpose of Servlet is to help web Server to overcome its limitations and make web Application Interactive.

\rightarrow What is a Servlet?

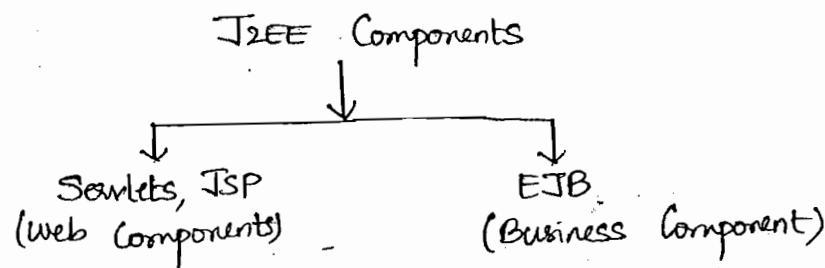
A Servlet is a (web) Serverside Piece of Code, that Enhances the functionality of a Web Server.

A Servlet is a Dynamic Web Resource in a web Application (website). It is the presence of a dynamic web resource that makes the web Application Interactive.

Static Resources \Rightarrow Already Existing HTML Pages, images, PDF

Static Resources can make a website Informative only, whereas Dynamic Resources help make a website Interactive.

→ A Servlet is a Web Component*



Servlet is not a Web Application, it is a Part of Web Application hence is a web Component. It cannot be executed as if a program is executed. It does not have a main method. Hence Servlet cannot be called as a "Program" but is only a "Component". Consider a Tubelight as an Example. It has

different components like Tube, Choke, Starter, frame etc.

Individually these components are of no use. when Assemble together only, it becomes an Application.

→ A Servlet is a Specialized Java Class

→ A Servlet produces Dynamic Web Page

→ A Servlet Contains Java code plus HTML code.

This would be something like:

Class A

§

```

S.o.println("---");
S.o.println("---");
-----}
-----}
    
```

Java Code
Functionality

```

ax("<HTML>"); }
-----}
    
```

Presentation as a HTML Page
HTML code is passed as
argument to some Special
methods Supported by Servlets.
Similar is the case in JDBC,
Since Java's power is in its
method calls alone.

Hence the page is generated dynamically i.e whenever this piece of code is called by Server process. These Pages are not stored in the file system of web Server machine as the HTML code exists as an argument supplied to some methods.

→ What are System Requirements to Develop and Run java based Interactive web Applications?

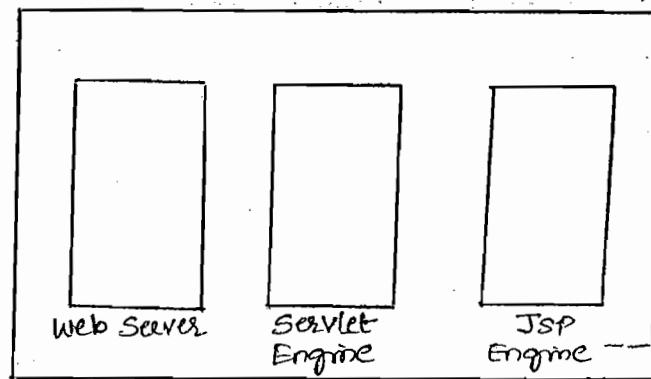
- *) Install J2SE Platform i.e. jdk Version 1.5 or preferably 1.6.
- *) Install any web container, preferably Tomcat and Set ClassPath appropriately.

→ What is a Web Container?

It is a Server Software written in Java according to Servlets and JSP Specifications.

A web Container has three Modules:

- 1) http Service / web Server
- 2) Servlet Engine/ container
- 3) JSP Engine / container



Web Container (Tomcat)

These three Modules interact with each other in Application Developments

only when the projects are developed using Java, A web Container is used. with other technologies like .Net, Perl etc Service Softwares are web Server only. In Java, web container is like a Super Set of web Server.

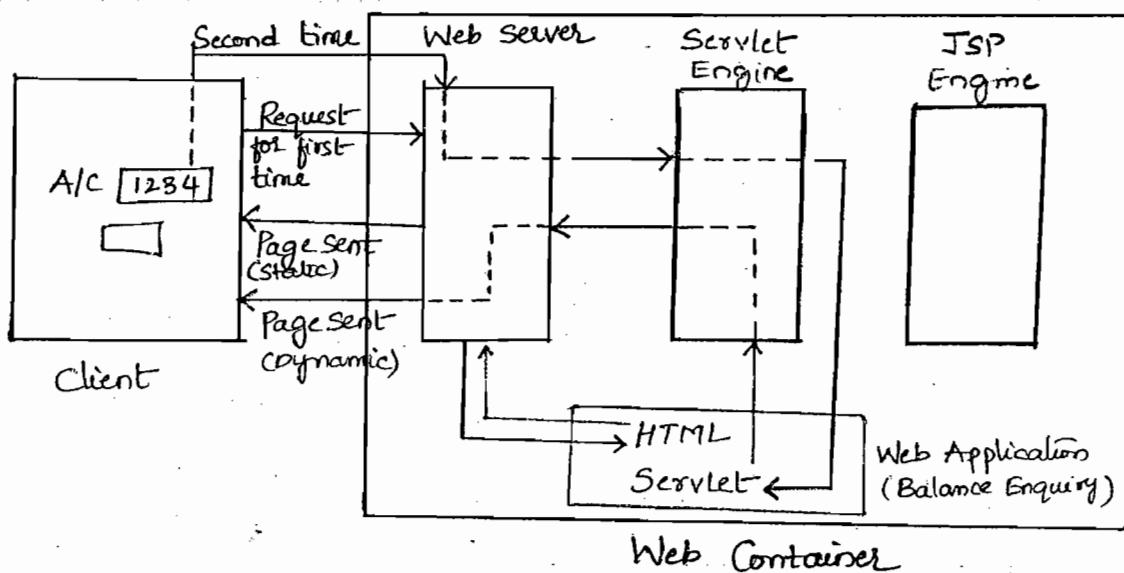
Examples for softwares which are only web Server are:
IIS from Microsoft, .Net projects work along with it,
Apache, Most of the projects in other technologies work with it, etc

These web Service Softwares do not have Servlet and JSP Engines, hence only web Servers like these two Softwares, cannot Support Servlet & JSP based websites.

There are plug-in Softwares available in market which contain only Servlet and JSP Engines. These can be connected to web Server and then, it becomes a Web Container.

Consider the following process which demonstrates how the modules inside the Container interact with each Other :

Static HTML page and delivers it to the client



Now, when the user inputs some data into the page, Client makes a request Again. Web Server once again takes the request along with the data and Servlet URL (client-side programming is done in such a way that, along with data, the address of Servlet code is also sent to web server)

As the web server is incapable of processing the data and producing the resultant page, the data is given to server side programming which is a Servlet in above case.

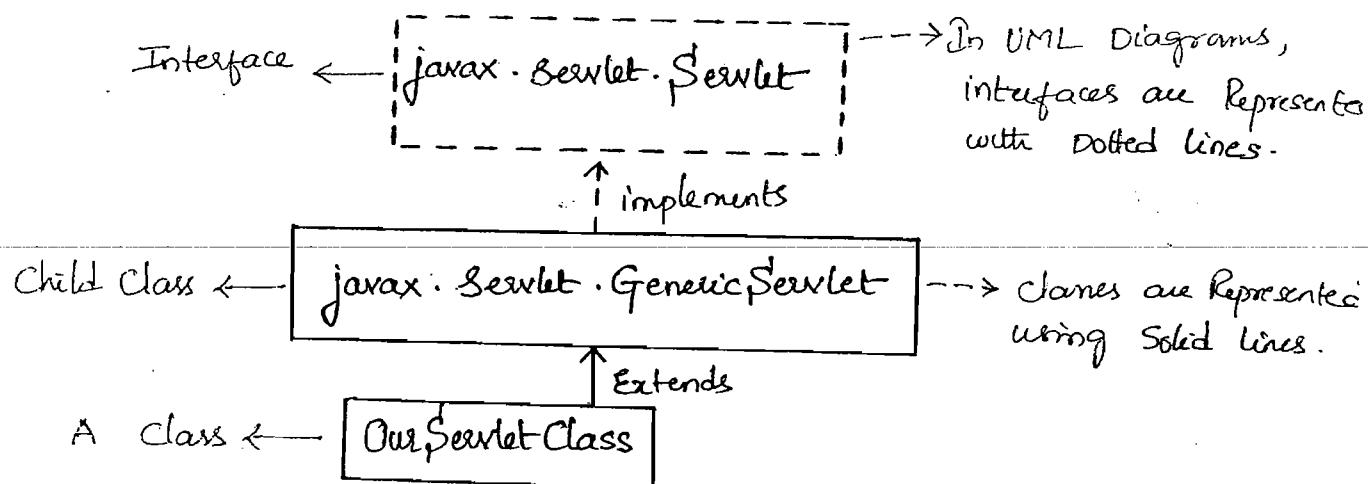
Web Server directly has no capability of communicating with Servlet, so it gives the data along with servlet URL to Servlet Engine. Depending on that address, Engine fetches servlet code and executes it. After producing the resultant page, Servlet Engine hands over the page to Web Server process. This process then sends the page to Client. This is how Server uses Server Side Programming.

Same would be the process if JSP is used instead of Servlets. Now, JSP Engine would perform its duties.

Hence, Servlet Specification and JSP Specification are literally for Web Container Manufacturer's sake i.e., for designing Servlet Engine and JSP Engine Respectively.

Servlet Development

→ A Servlet is a Container managed public Java class that implements directly or indirectly javax.servlet.Servlet Interface.



GenericServlet implements Servlet Interface. Hence we can consider GenericServlet as Servlet because of Inheritance. OurServletClass Extends GenericServlet class hence OurServletClass is Servlet because of inheritance. Hence, OurServletClass is a Java class that can be considered to implement directly or indirectly javax.servlet.Servlet Interface.

Servlet & GenericServlet both belong to same

Skeleton Structure of Our Servlet :-

i.e A Generalized Structure of a Servlet i.e,
It is Actual Format of a Servlet.

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

```
public class OurServlet extends GenericServlet
```

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

↓
Interface

* Resources Allocation Code goes here. For Example,

Database Connectivity */

---> This is like Allocating a Resource

```
public void service (ServletRequest request, ServletResponse response)
```

→ Interface ←

* Client Request Processing Code goes here (IPO
Code) */

↓
Input,
Process,
Output

```
public void destroy()
```

* Resource Releasing Code goes here. For Example,

Closing Database Connection */

---> This is like De-Allocating the Resource

GenericServlet, ServletConfig, ServletRequest, ServletResponse are

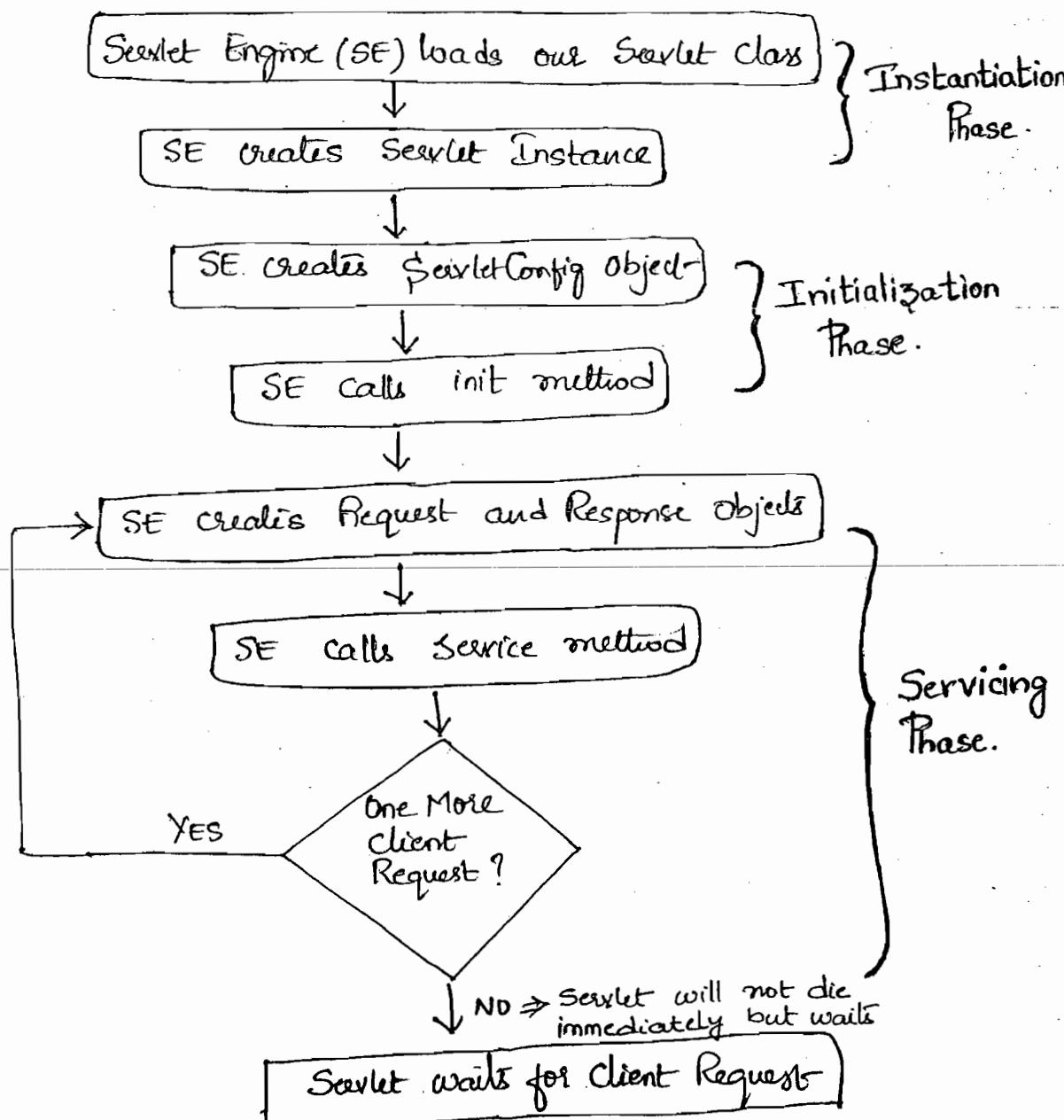
Life Cycle of Our Servlet :-

- Servlet Engine Controls the life cycle of our Servlet.
Hence, "Container Managed" implies, the Servlet Class is managed and its Objects are created and Destroyed by the Container but not by the Programmer.
- Servlet life cycle is described by 3-life cycle Methods and 4-life cycle Phases.
- init, Service and destroy methods are known as life cycle methods. Servlet Engine calls them at Appropriate Time.
- Servlet Engine calls init method soon after it has created the Servlet Instance. In the life of the Servlet Instance, init method is called only once by the Servlet Engine Implicitly.
- Just before the Servlet Instance is destroyed (garbage collected), Servlet Engine calls destroy method. This method is also called Implicitly (i.e. automatically) by Servlet Engine only once on the Servlet Instance.
- Each time client request comes, Servlet Engine calls the Service method i.e. Service method is called in times implicitly by Servlet Engine on the Servlet Instance. Therefore, we can say, a Servlet Spends most of its life in serving the Client Requests and hence the name "Servlet".

→ The 4-life cycle Phases are :

- 1) Instantiation Phase
- 2) Initialization Phase
- 3) Serving Phase
- 4) Destruction Phase

All these phase are controlled by Servlet Engine.



The Request and Response objects implies `request`, `response` references that are passed to `Service` method as can

Instantiation Phase :-

- When Servlet Engine Receives the first client Request for our Servlet, it loads the Servlet class into memory. Servlets Engine creates the Servlet Instance of the loaded (our Servlet) class.
- In this phase, Servlet instance is missing with two pieces of information
 - 1) Initialization Information
 - 2) Context Information

That is, when the object is born (created), it does not have Serving capability yet.

Context info ⇒ environmental information i.e. Exactly on which field to work on.
- Therefore, it doesn't possess 'Servletness' (i.e. Serving Nature) and hence it cannot Serve any Client Request.

Initialization Phase :-

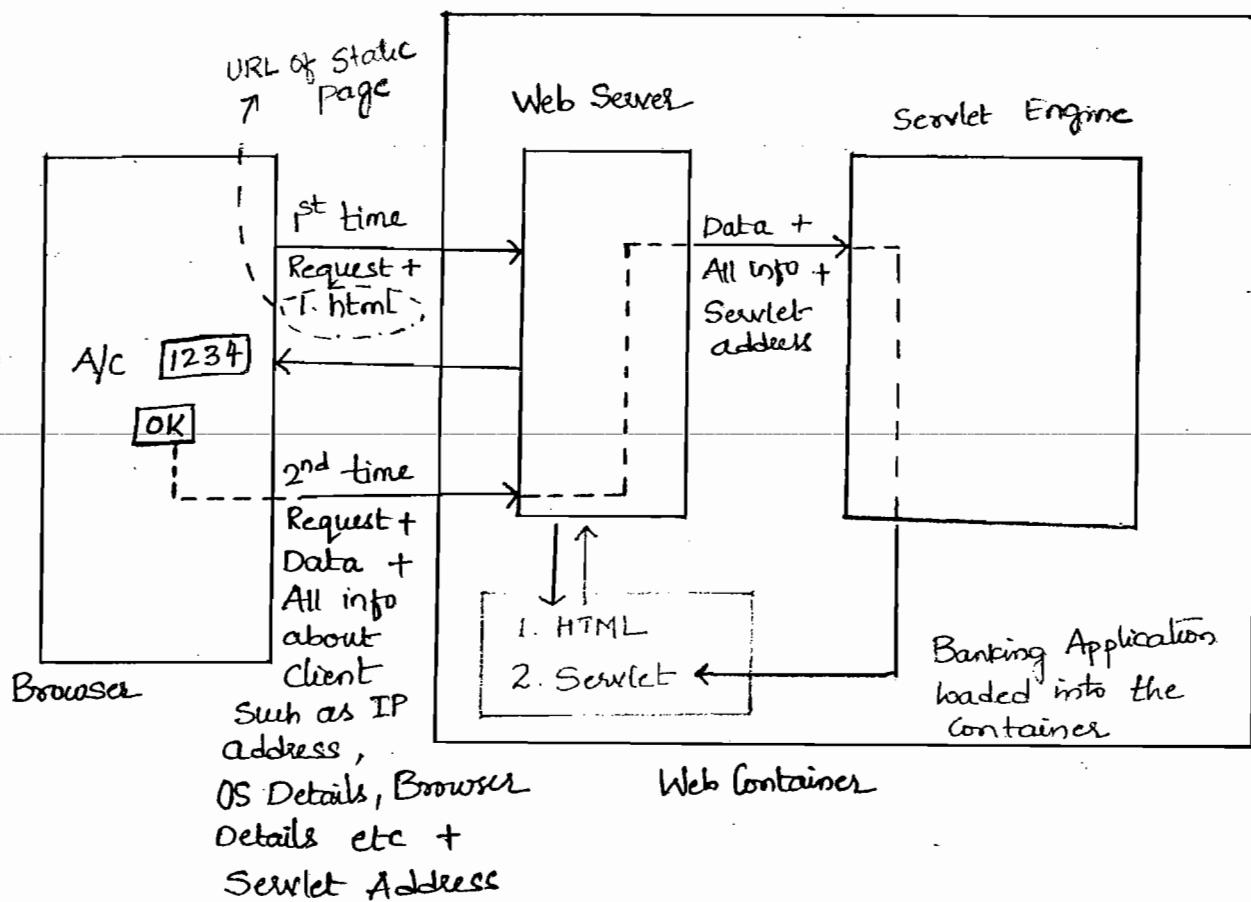
- Servlet Engine creates ServletConfig Object encapsulating the above missing information (i.e. initialization & context Information).
- ServletConfig is an interface, hence its object creation implies an object of child class that implements this interface. These child classes are written in web Container Software and they have all implementation classes for Interfaces in Servlet API just similar to regular Servlets.

→ Servlet Engine calls init method by supplying ServletConfig object (reference) as argument. Once init method is completely executed, Servlet Instance gets Servletness and it is ready to serve the client Request.

Note :- Instantiation and Initialization happens only once in the life time of Servlet.

Servicing Phase :-

Consider the following process:-

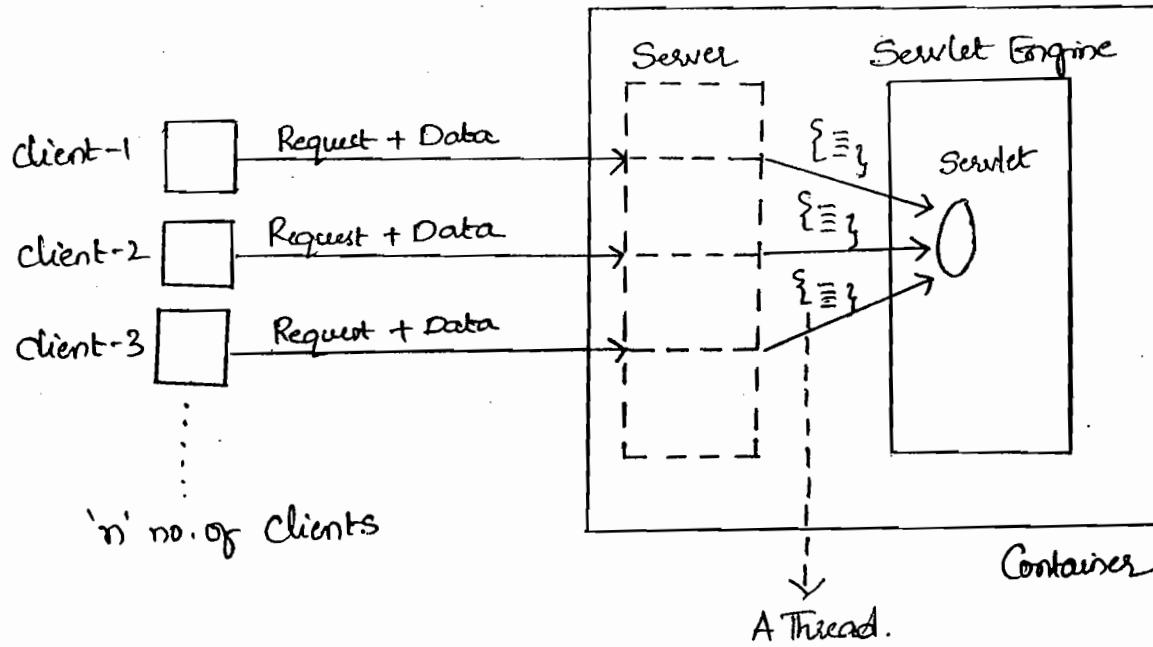


The information coming from web server (i.e., user input data + Client Information + Servlet details) is in a format which is not understandable to Servlets. Hence this raw info. is given by web server to SE, which converts it into objects. These are nothing but 'request' and 'response'.

'request' object has user data, 'response' object has whereabouts out of client. Servlet Engine passes these objects to Servlet using `service()` method.

Servlet uses 'request' object to capture input from user.
Servlet uses 'response' object to deliver output to client.

- Servlet Engine constructs `ServletRequest` object, Encapsulating the client input(user input). Servlet Engine also creates `ServletResponse` object, Encapsulating whereabouts of the client.
- Servlet Engine calls `service` method by Supplying request and response as arguments.
- Once `service` method is Completely executed, one Client Request is processed and Response is delivered i.e., the HTTP Request-Response cycle is Completed.



Each Client Request is thrown into a thread. All these threads put together is considered as one process.

If requests come one after the other, Each thread is processed one after the other.

If say, 1000 Client Requests approach at a time, the phenomenon of 'Multi-Threading' takes place. This is what actually happens in Real time. Creation of Objects for a particular client takes place in its own thread and delivering the Respective Response to a thread (i.e. to the corresponding client) is taken care without any sort of confusion or mismatch. All this process i.e. creation of threads, creation of objects within a particular thread, multi-threading and delivering the Responses to Respective clients is taken care by the Container Software and Servlet Engine inside the container. Hence, Container also deals with the Concurrency process.

03/08/09

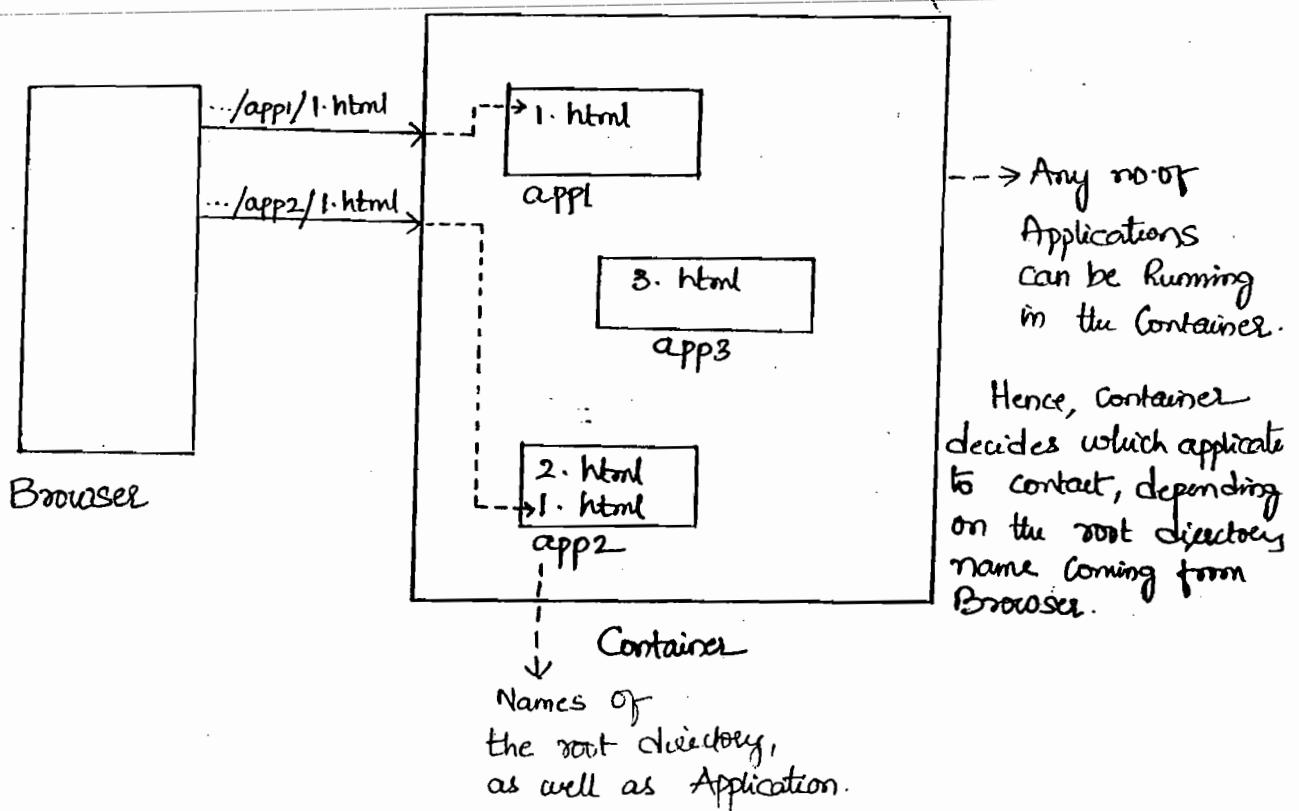
Destruction Phase :-

- When administrator (website administrator) unloads the Servlet, just before the Servlet instance is garbage collected, Servlet Engine calls `destroy()` method on the Servlet Instance.

`destroy()` method is called just before Scarlet instance is garbage collected so that, the developer has a chance to release the resources. Then the website administrator unloads the Scarlet class and destroys Scarlet instance with some kind of tool.

Steps to Develop a Web Application:

Step-1 :- Creating a Structured Hierarchy of Directories
 → We need to create a directory with any name. It becomes the application root context i.e, Container recognizes the resources of the application with this name only.
 This name would become the name of the total application.

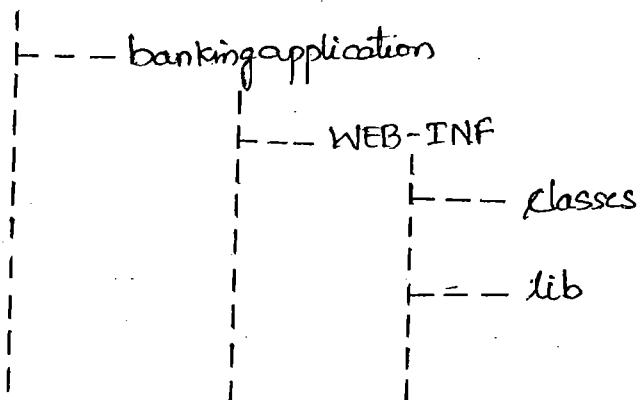


→ We need to create two more directories in WEB-INF i.e. Subdirectories

1) classes

2) lib

for Example:



A Structured Hierarchy.

Step-2 :- Developing the Web Resources.

→ We need to develop all html documents, image files, Servlets, JSPs etc.

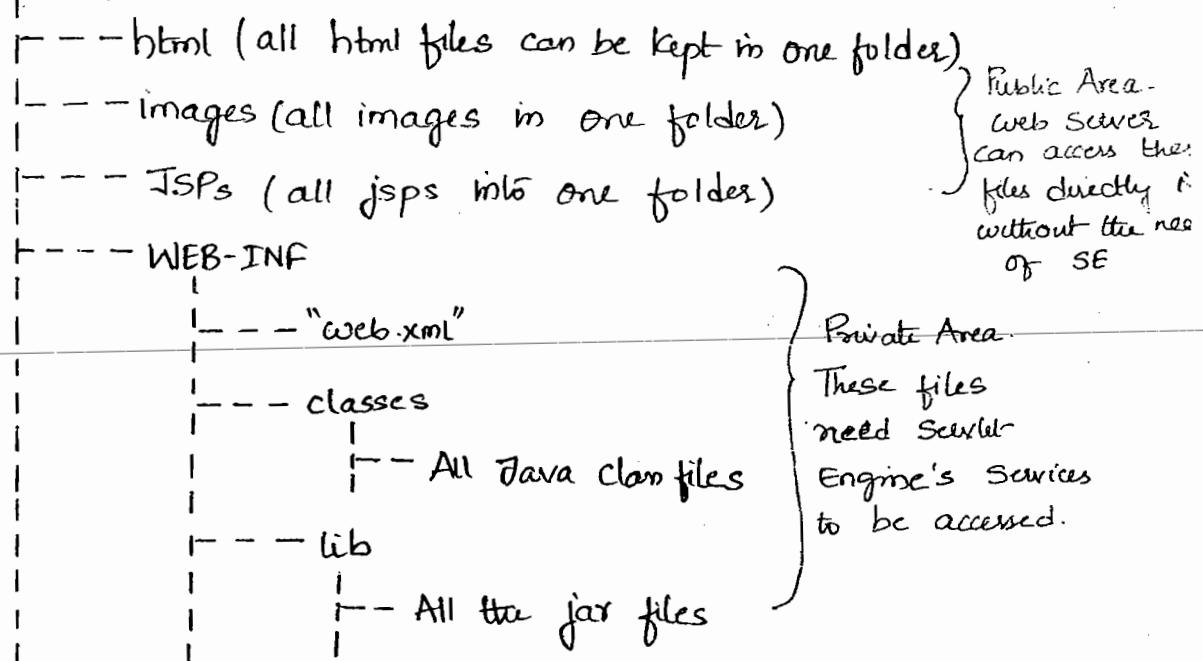
Step-3 :- Developing a Deployment Descriptor (nothing but a file), whose name and extension should be "web.xml"

Only one such file needs to be developed. It is Mandatory. It is a text file. Name should be "web" and extension should be ".xml". They cannot be changed. If changed, Container does not understand the file. We need not "www + domain file 1:1".

Step-4 :- Configuring the Application files

- All html files, jsps, image files are to be copied into root directory
- Java class files are to be copied into "classes" directory
- "jar" files of the application are to be copied into "lib" directory
- "web.xml" to be copied into WEB-INF.

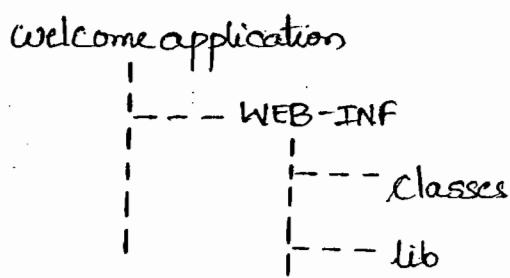
banding application



This would be the Generalized Directory Structure for any application.

- Q) Develop a web application in which a Servlet

Step 1 :- Create Directory Structure



Step 2 :- Web Resources Development

In this application, we have only one Dynamic web Resource i.e., a Servlet.

init, destroy, service are methods in Servlet Interface. They are abstract methods i.e., only declared but not defined. Hence they make Servlet Interface abstract. GenericServlet implements Servlet Interface hence, it inherits these methods. The child class should define the abstract methods of Parent's.

GenericServlet defines init() and destroy() methods but not service() method. Hence GenericServlet becomes an abstract class. Service() method is not defined in GenericServlet but is defined in its child ourServlet, since it is dependent on application.

Since init() and destroy() are already developed and defined in GenericServlet class, they may or may not be overridden by init() and destroy() of our Servlet, depending on the application. In the above application they are not overridden as the application doesn't need it, i.e., any resources allocation or

Since they are not overridden, it does not mean they are not available to the Servlet object. They are inherited from parent by default and they have default definition in parent class i.e. GenericServlet class.

`<HTML>` --> Tag and is an indication to browser i.e., indication that it is HTML code and Code Starts here.

`<BODY BGCOLOR = yellow>` --> Tag, indicating that this is Body
 ↓
 Background color.
 It is an attribute.
 Its purpose is to indicate browser that Background is yellow.

`<H1> Welcome To Our Website </H1>`

`</BODY>`

`</HTML>` --> Every Tag Opened must be closed.

The above HTML Code is Supplied to some Special methods in a Java program as arguments.

WelcomeServlet.java :-

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

public class WelcomeServlet extends GenericServlet
{
```

{

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

```
PrintWriter out = response.getWriter();
```

--> just reference name.
It can be any name
which is valid.

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

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

```
out.println("<H1> Welcome To Our Website </H1>");
```

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

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

```
} // service
```

```
} // Servlet class
```

Step 3:- "web.xml" Development

<web-app> --- root tag.

Servlet Registration Section {

- <Servlet> --- child tag.
- <Servlet-name> (One) </Servlet-name> ---> This can be any name.
- <Servlet-class> WelcomeServlet </Servlet-class>
- </Servlet>

It is
Servlet
Registration
Name

Servlet Mapping Section {

- <Servlet-mapping> --- child tag.
- <Servlet-name> one </Servlet-name>
- <url-pattern> /welcome </url-pattern>
- </Servlet-mapping>

</web-app>

--> This can be any name.
It is public url name
of Servlet.

→ What different names does a Servlet have in a Web Application?

A Servlet has three Names

- 1) Registration Name
- 2) Class Name
- 3) Public URL Name

In the above example, 'one' , 'WelcomeServlet' and '/welcome' respectively.

→ What is the purpose of <Servlet> tag in 'web.xml'?

To register a Servlet with the Web Application

→ What is the purpose of <Servlet-mapping> tag?

To give public URL Name to the Servlet and thereby indicating to the Servlet Engine to map a Client Request to a particular Servlet.

04/08/09

Step-4 :- Configuring the Application files.

welcomeapplication

-----WEB-INF

-----"web.xml" ← Copy & Paste this file
in WEB-INF.

-----lib

-----classes

-----"WelcomeServlet.class" →

Note :- while installing Tomcat, Select required directory for installation, change port Number from 8080 (which is already occupied by oracle 9i or higher) to any number and then mention location of JVM i.e., jdk1.6 path.

Note :- If we install Tomcat in our System, we need to keep 'servlet-api.jar' file in classpath

Eg:- C:\Tomcat5.0\common\lib\Servlet-api.jar

↑ Set this in classpath in Environmental Variables.

If Tomcat is not installed and then the application is run, we get errors. Even If Tomcat is installed, we get errors if above classpath is not set.

E:\ welcomeapplication>tree /f

↓
This command gives tree structure of welcomeapplication Directory.

Web Application Deployment into Tomcat :-

→ Installing / loading a web Application into the web Container so that its services are made available to web clients is known as its Deployment.

Step-1 :- copy the application (entire directory structure) into 'webapps' directory of Tomcat Installation Directory

Step-2 :- start Tomcat Server.

Upto this point, application development & deployment are done. Now we need to use the Services of Application.

Note:- To use the services of the web Application, Specify the following URL in the browser.

http://localhost:8081/welcomeapplication/welcome.

URL is a pointer to web resource.

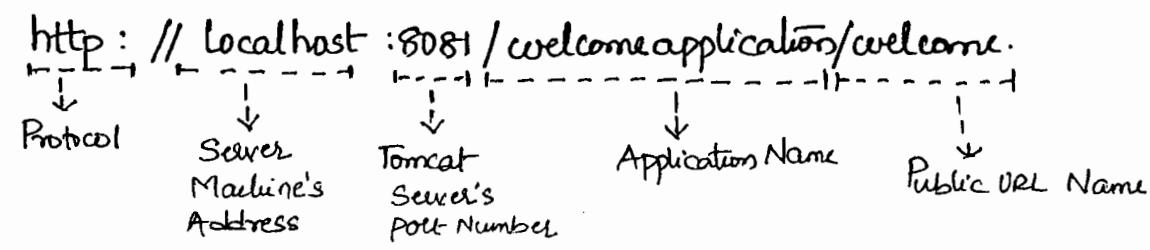
URL is divided into three parts:

how part: how to recognize the resource. It is the communication protocol i.e. http:

where part: in which machine the Tomcat Server is running i.e. IP address of machine. 'localhost' if same machine is used for browser and Server.

In this part, port number of server i.e. port-number of Tomcat that is changed from 8080 to some number, is to be mentioned.

what part: It includes public URL Name (/welcome) along with application Name (/welcomeapplication)



In the above application, control flow is as follows: 94

Since above URL does not have request for Static Resource, this client Request i.e. from browser is sent to Servlet Engine by Server. The Servlet Engine opens 'xml' file, and checks mapping Section. It compares public URL Name of Servlet in mapping Section with public URL Name coming from browser. If match is not found, an error message is sent back to browser. If match is found, then the registration name of Servlet in the mapping Section is recognized. Then the control goes to Servlet Registration Section depending on Registration name given by mapping Section.

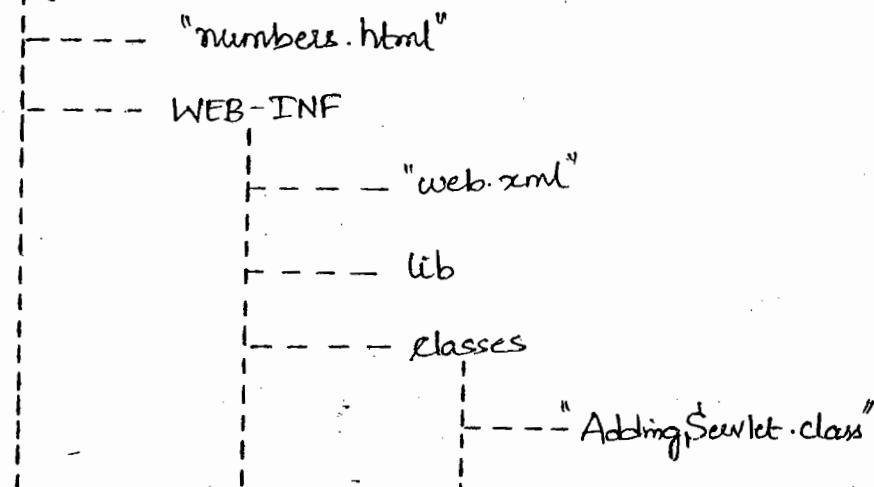
In Registration section, the actual servlet i.e. WelcomeServlet is identified and control goes to that servlet and here the Servlet lifecycle starts. It includes, loading that particular class file, creating its instance, creating ServletConfig object, executing init() method which is default method. Then service() method is executed.

This is how the control is transferred from browser to server and then to servlet. Then the output is sent back to the browser.

- Q) Develop, Deploy and Use a web Application in which, end-user should be able to enter two numbers into web form. Upon form Submission, the sum of the two numbers should be displayed to

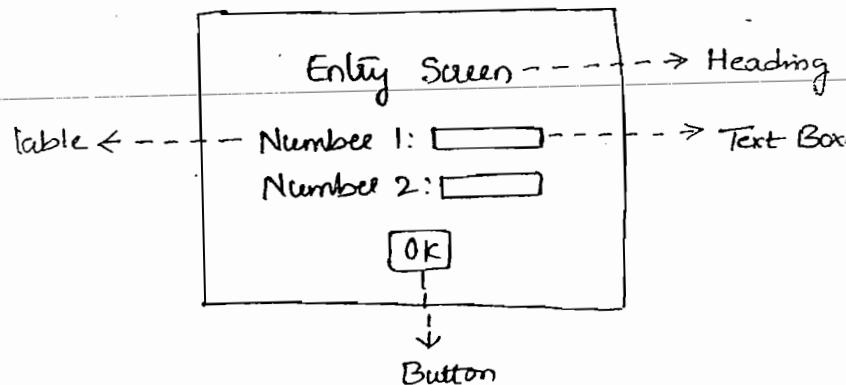
Step-1 :- Directory Structure

adding application



Step-2 :- Resources Development

It has a static Resource (html page) and a Dynamic Resource (Servlet)



Screen Should be something like shown above, and in order to develop such a Screen , we have to use FORM tag

numbers.html :-

<HTML>

<BODY BGCOLOR = "cyan">

Number One : <INPUT TYPE = "text" NAME = "t1">

Number Two : <INPUT TYPE = "text" NAME = "t2">

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

</FORM>

</ CENTER>

</BODY>

</HTML>

Save the above file as "numbers.html" in root directory

Adding Servlet.java :-

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

```
public class AddingServlet extends GenericServlet
{
```

```
  public void service(ServletRequest request, ServletResponse
  response) throws ServletException, IOException
  {
```

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

```
    PrintWriter out = response.getWriter();
```

String a = request.getParameter("t1"); Data comes from
 the browser in the
 String b = request.getParameter("t2"); form of text i.e.
 String

int n1 = Integer.parseInt(a); → static method of
 Integer class.

```

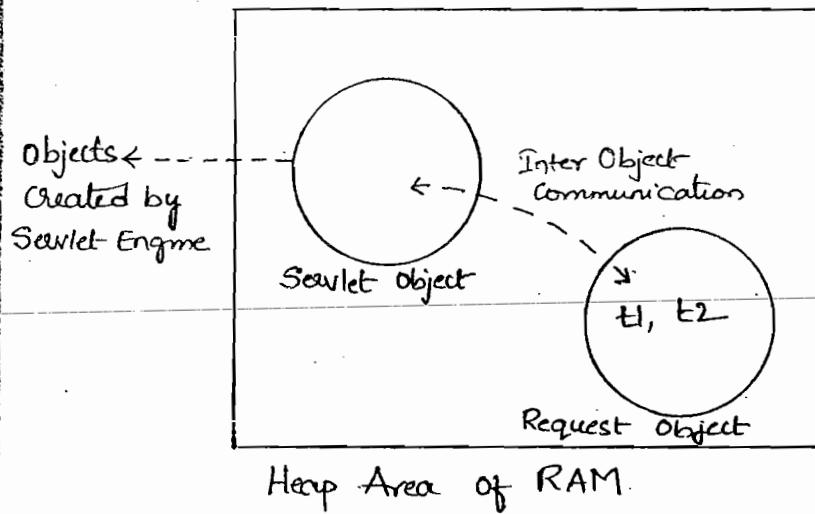
int sum = n1+n2;

out.println("<HTML>");
out.println("<BODY BGCOLOR = wheat>");
out.println("<H1> The Sum Is: " + sum + "</H1>");
out.println("</BODY>");
out.println("</HTML>");
out.close();
}

} // Service

} // Servlet class

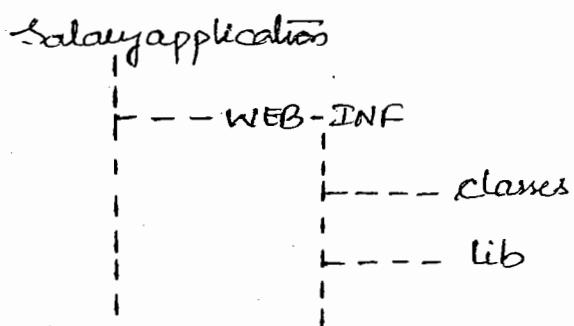
```



Inter Object Communication is a regular phenomenon in Java. `Service()` method is calling `getParameter()` method. Hence Servlet object requires data from the request object. Thus Servlet class object is talking to Request object. This is possible by message passing i.e., by method calls. This is the power of Java.

→ Q) Develop and Deploy a web Application in which end user should be able to enter basic Salary of an employee and get take home salary details (i.e, Net Salary)

Step-1 :- Directory Structure Creation:



Step-2 :- Web Resources Development:

In this web application, we need to have a static Resource (html document) to create the web form and a Dynamic Resource (Servlet) to provide interaction for the user.

basic.html :-

```

<HTML>
<BODY BGCOLOR = wheat>
<CENTER> <H1> Salary Details Screen </H1>
<FORM ACTION = ". / basic" >
Basic Pay : <INPUT TYPE = "text" NAME = "t1"> <BR> <BR>
<INPUT TYPE = "submit" VALUE = "Get SALARY Details">
</FORM> </CENTER>
</BODY>
    
```

SalaryServlet.java :-

```

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

public class SalaryServlet extends HttpServlet
{
    public void service(ServletRequest request, ServletResponse response
            throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        // capturing Input i.e, I-section of IPO
        float basic = Float.parseFloat(request.getParameter("t1"));

        // Processing Section i.e, P of IPO
        float da = basic * 0.5f; // whenever a decimal literal
        float hra = basic * 0.2f; // is used, compiler treats
        float gross = basic + da + hra; // it as double. To indicate
        float deductions = gross * 0.15f; // that it is float, we
        float net = gross - deductions;

        // Output Section i.e, O of IPO
        out.println("<HTML>");
        out.println("<BODY BGCOLOR = yellow>");
        out.println("<H1> Take Home Salary IS: " + net + "</H1>");
        out.println("</BODY>");
    }
}

```

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

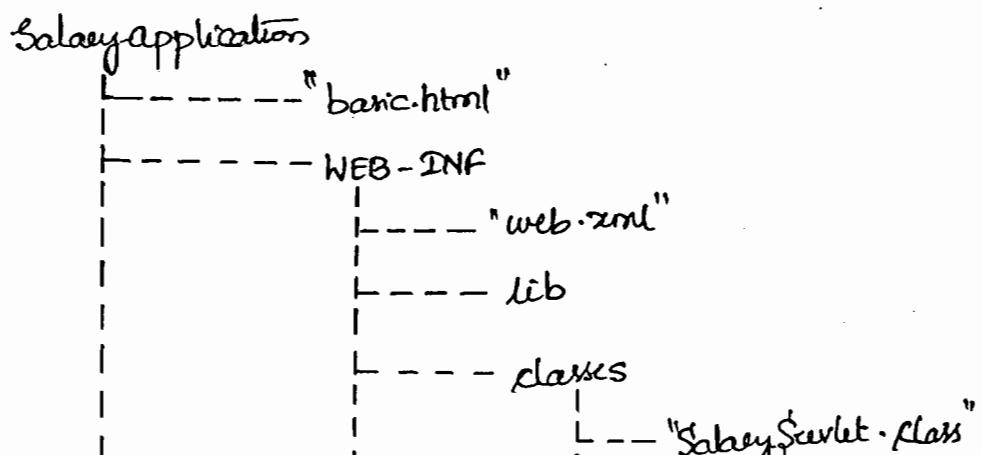
⁹ // web Component i.e., Servlet

Step-3 :- Deployment Descriptor Development

web.xml :-

```
<web-app>
  <Servlet>
    <Servlet-name> three </Servlet-name>
    <Servlet-class> SalaryServlet </Servlet-class>
  </Servlet>
  <Servlet-mapping>
    <Servlet-name> three </Servlet-name>
    <url-pattern> /basic </url-pattern>
  </Servlet-mapping>
</web-app>
```

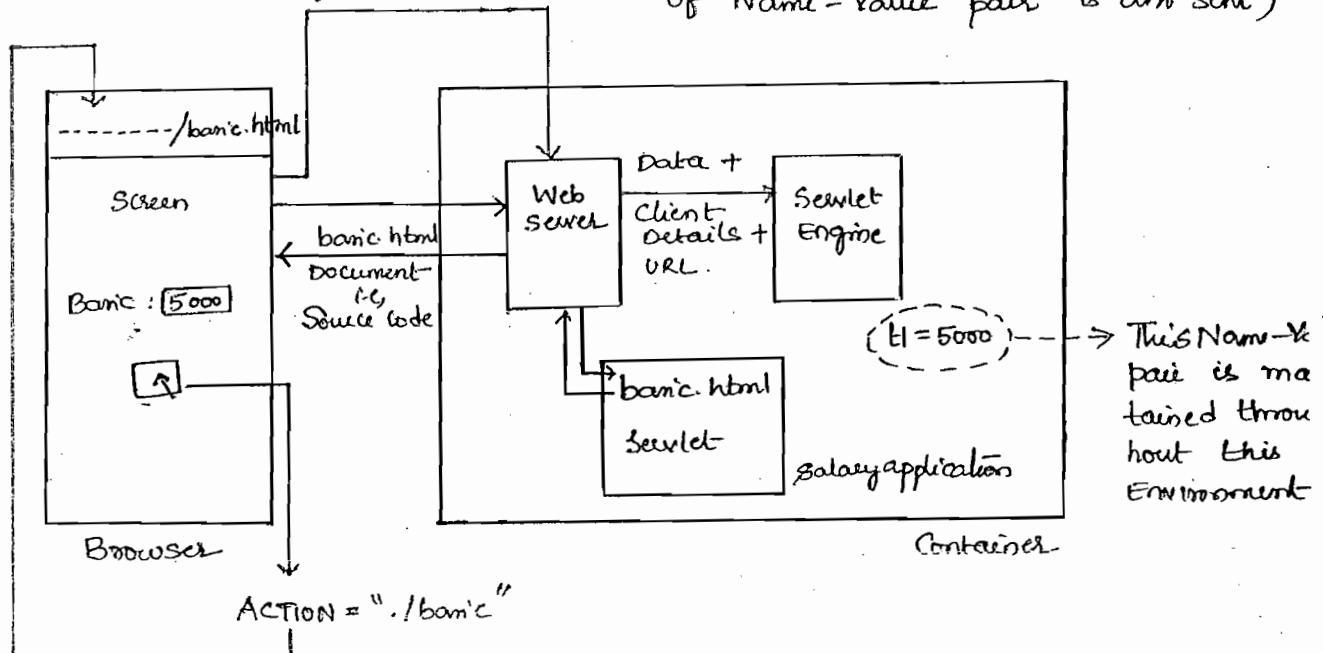
Step-4 :- Configuring the Application Files



URL :-

http://localhost:8081/Salaryapplication/basic.html

-----/basic?t1=5000 (along with URL, Client-Details, Data in the form of Name-Value pair is also sent)



in place of :'
Previous address is
added upto the
point of public
URL name of
Servlet i.e. /basic

When first time Request is Sent, It fetches basic.html Static page developed by us. When data is inputted and Submit button is clicked, as ACTION was Specified as "./basic", the previous URL of application excluding basic.html is directly copied into the browser. Along with it, public URL name of Servlet is appended. This becomes new URL for Second Request. To this URL data that is inputted is added and the request is sent again to web Server. Data is sent as Name-Value pair. Since it is a request for Dynamic Resource, it is sent to Servlet Engine along with

web.xml page and so on. Then, Servlet Instance is created. After that, request & response objects are created with data in request object as Name-Value pair. Client details are in response object

06/08/09

→ What are different init() methods defined in the GenericServlet class?

In GenericServlet class, two init() methods are defined

- 1) init(ServletConfig c) --> Parameterized init() method
- 2) init() --> Zero Argument init() method

GenericServlet gets Parameterized method from Servlet Interface but is defined here.

The two init methods are defined in GenericServlet as:

public abstract class GenericServlet implements -----

{

ServletConfig sc;

Zero argument init method.
It has no code with it.

{ }

If we override init
that method called by
parameterized init

----- public void init(ServletConfig c) throws ServletException

↓

This comes
from parent.

{ sc = c; }
 init(); }

It is defined here.

Note :- It is recommended to override `Zero argument init` method in a Servlet whenever `init` method is required.

A child method in order to override its parent method, 4-rules are to be satisfied.

- 1) Signature of both should be same i.e. name and no. of arguments i.e. parameters
- 2) Return Type Should be Same
- 3) Access Specifier of Child can be decreased or kept at same level as parent but can never be increased
- 4) No. of Exceptions and their type in Child can be Equal to parent's or less, but can never be more or different when compared to parent's.

So use `Zero argument init` method in Application Development generally. This could have some added advantage. Consider the following Example:-

```
public class MyServlet Extends GenericServlet
{
```

```
    public init() throws ServletException
    {
```

```
        System.out.println("-----");
```

```
    }
```

```
    -----
```

```
    -----
```

```
}
```

This method is not called by Servlet Engine.

SE always calls Parameterized init method only. Here we did not declare Parameterized init method.

we did not declare Parameterized init method.

Hence this parent's init method is called. This method internally calls zero argument init method. Since two init methods are there with zero arguments, one in GenericServlet and one in MyServlet, init method of MyServlet overrides init method of GenericServlet.

→ How are Servlets classified?

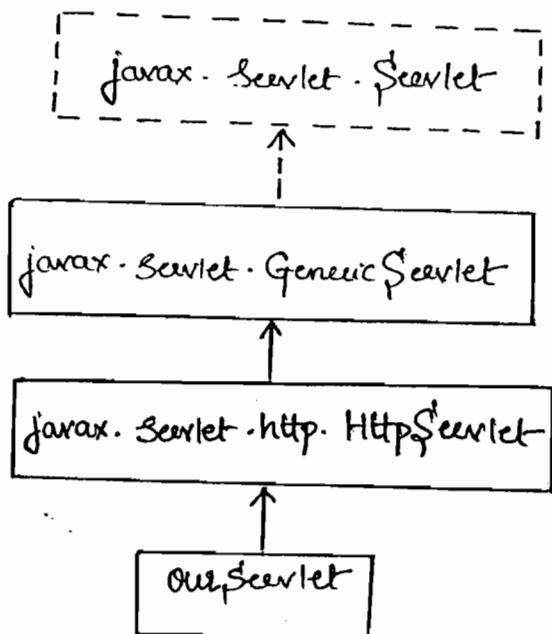
Servlets are divided into 2 categories:

- 1) Protocol Independent Servlets
- 2) Protocol Dependent Servlets

Our Servlet is said to be protocol independent if it extends GenericServlet.

Our Servlet is said to be protocol dependent if it extends HttpServlet.

Note :- we must always create a protocol dependent Servlet



GenericServlet provides say 70% of functionality in website

along with the 70% functionality inherited from GenericServlet. GenericServlet is developed by Sun-Microsystems so that it can offer its services to any kind of protocol (i.e. http or ftp or smtp etc). Hence it is generic to all the protocols.

Therefore it misses that 30% specific functionality. HttpServlet was also developed by Sun-Microsystems. For other protocols classes were not developed because of commercial viability i.e., they are not that commercially used.

Skeleton Structure of a Protocol Dependent Servlet

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

```
public class MyServlet extends HttpServlet
```

```
{
```

```
    public void init() / init(ServletConfig c)
```

```
{
```

```
    // Resources Allocation Code
```

```
}
```

```
    public void destroy()
```

```
{
```

```
    // Resources Releasing Code
```

```
}
```

or

```
    public void doGet / doPost(HttpServletRequest request,
```

order of defining the methods has no importance.

Order of calling the methods is only important.

// Client Request Processing Code (IPO)
}

? // MyServlet Class.

GenericServlet is an abstract class not concrete class because of the abstract method service() i.e. service() method is not defined in GS. HttpServlet class is also an abstract class, but no method in it is abstract. It is kept abstract i.e. declared abstract so as to make the user not to create object of this class. It has only library related code but no application related code. Hence, creating its object would be of no use.

(If a method is abstract, class will be abstract. If the class is abstract, methods inside it need not be abstract.)

Order of method calls :-

SE calls implicitly or explicitly always

↓
init(ServletConfig)

↓
init()

↓ then SE calls

public Service(ServletRequest request, ServletResponse response)

↓

Protected Service(HttpServletRequest , HttpServletResponse)

↓

doGet / doPost (HttpServletRequest , HttpServletResponse)

public Service() method first converts normal objects to Http compatible objects. It then calls protected service() method. protected service() method first recognizes what kind of incoming Http request i.e either get request or post request. It then calls doGet() or doPost() methods accordingly.

Http Request Methods :-

→ From the web client, request can come to the web server in any one of two ways / methods :

- 1) GET
- 2) POST

→ These two methods are known as HTTP request methods.

→ What are the differences between GET and POST HTTP Request methods ?

GET

1) It is the default request method.

2) In three different ways this request method will be used:

- i) User specifies the URL

POST

1) we need to explicitly specify

2) In only one way the request is made i.e. only when form is submitted.

- 3) It is meant for getting data from Server but not for posting data to Server for database modifications.
- 4) It has query String
- 3) It is meant for posting data to Server for database modifications.
- 4) No query String.

`http://localhost:8081/addingapplication/add? t1=10 & t2=20
Data.`

This is Query String.

When such an information is sent to server, it is visible in the bar as it is appended to the URL.

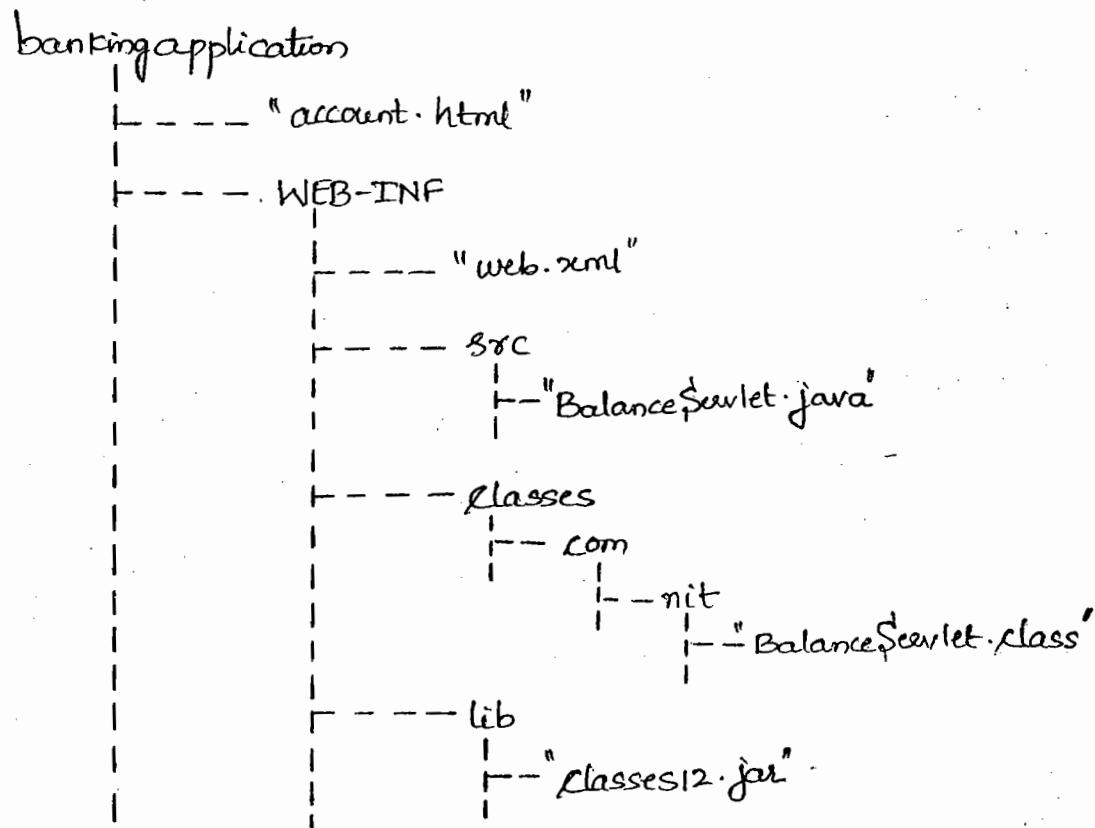
If login screens are developed using 'get' request method, privacy would be lost. They have to be developed using 'post' request only but should not affect the Database.

- 5) Sensitive information like passwords should not be sent.
- 5) Can be sent
- 6) limited data transfer from Client to Server
- 6) No limit.

07/08/09

→ Q) Develop and Deploy a web Application in which end user should be able to enter Account Number into the web form and get Balance Details.

Directory Structure :-



'src' directory is used to store Source Code. It is not mandatory but used in industry for flexibility. When Application is given to client, this directory is removed and only class file is given.

User defined package is kept in 'classes' directory. It is also optional.

"classes12.jar" file is copied into 'lib' directory. By doing so, its path need not be specified in Classpath. For Stand Alone Applications, mentioning Classpath is mandatory. For Servlets i.e. Web Resources, keeping the file in 'lib' directory is mandatory.

URL :-

We can make the HTML file default. For this, home page must be configured. It is done in web.xml page.

account.html :-

```

<HTML>
<BODY BGCOLOR = "wheat">
<CENTER> <H1> Balance Enquiry Screen </H1>
    <FORM ACTION = "./balance">
        Account Number : <INPUT TYPE = "text" NAME = "t1">
        <BR> <BR>
        <INPUT TYPE = "submit" VALUE = "Get Balance">
    </FORM> </CENTER>
</BODY>
</HTML>

```

BalanceServlet.java :-

```

package com.nit;
-----> www.nit.com
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.sql.*;
public class BalanceServlet extends HttpServlet
{
    -----> This is written in Reverse
          i.e. Reverse Domain Name.
          It is Naming Convention in
          Industry. The website for
          which they register,
          that name is written in
          Reverse. It is not manda-
          tory but optional.
}
-----> An instance variable. It has

```

```
public void init(ServletConfig c) throws ServletException  
{  
    try  
    {  
        Class.forName("oracle.jdbc.driver.OracleDriver");  
        Con = DriverManager.getConnection("jdbc:oracle:  
                                         thin:@localhost:1521:orcl", "scott", "tiger");  
    } // try  
    catch(Exception e)  
    {  
        e.printStackTrace();  
    } // catch  
} // init
```

```
public void destroy()  
{
```

```
    try  
    {  
        Con.close();  
    }
```

```
    catch(Exception e)  
    {  
        e.printStackTrace();  
    }
```

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

```

response.setContentType("text/html");

PrintWriter out = response.getWriter();

out.println("<HTML>");
out.println("<BODY BG-COLOR = grey>");

try
{
    Statement st = con.createStatement();

    ResultSet rs = st.executeQuery("SELECT
        BALANCE FROM ACCOUNT WHERE ACCNO
        =" + request.getParameter("t1"));

    if (rs.next())
        out.println("<H1> Balance is : Rs. " + rs.getFloat(1)
            + "</H1>");

    else
        out.println("<H1> Account Does Not Exist </H1>");

    rs.close();
    st.close();
}

// try

catch (Exception e)
{
    e.printStackTrace();
}

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

out.close();

} // doGet.

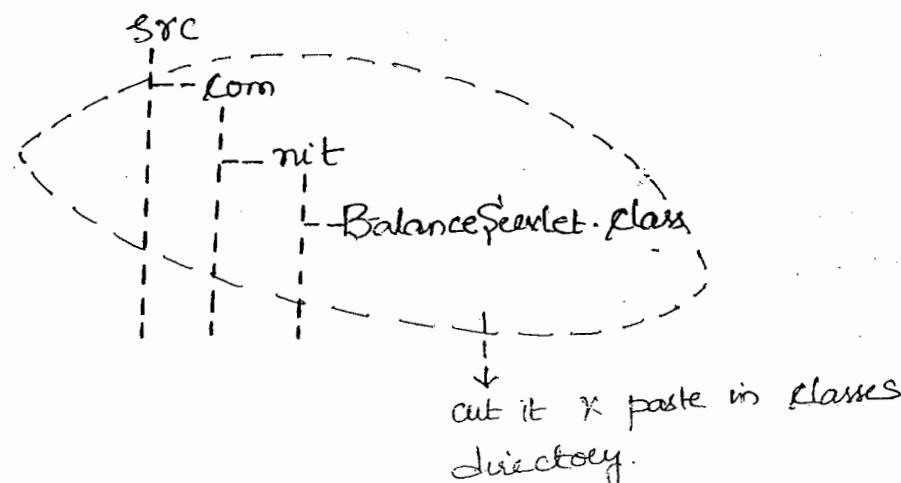
```

Note :- Compile the Servlet Source Code as follows:

.....\src> javac -d BalanceServlet.java
Spaces.

↓-----→

This makes a directory in src i.e,



Web.xml :-

<web-app>

<Servlet>

<Servlet-name> bal </Servlet-name>

<Servlet-class> com.nit.BalanceServlet </Servlet-
class>

</Servlet>

<Servlet-mapping>

<Servlet-name> bal </Servlet-name>

<url-pattern>/balance</url-pattern>

</Servlet-mapping>

<welcome-file-list>

</web-app>

Server Administrators can make even application name default and port Number can be made default too. 'localhost' i.e, IP address is converted into domain name by administrator after making these values default and that name (Eg:- www.nit.com) is delivered into market.

If Sun-Driver is used instead of pure driver, dsn configuration should be changed for above program.

Control Panel → Admin Tools → Data Sources → System dsn

↓
Add ODBC Driver Here
(instead of in user dsn)

08/08/09

using this the DB details and connection, we are ^{with DB details} mentioning in web.xml. if any changes are made we need not alter source code so we need not compile source code again. Just we modify this in web.xml file

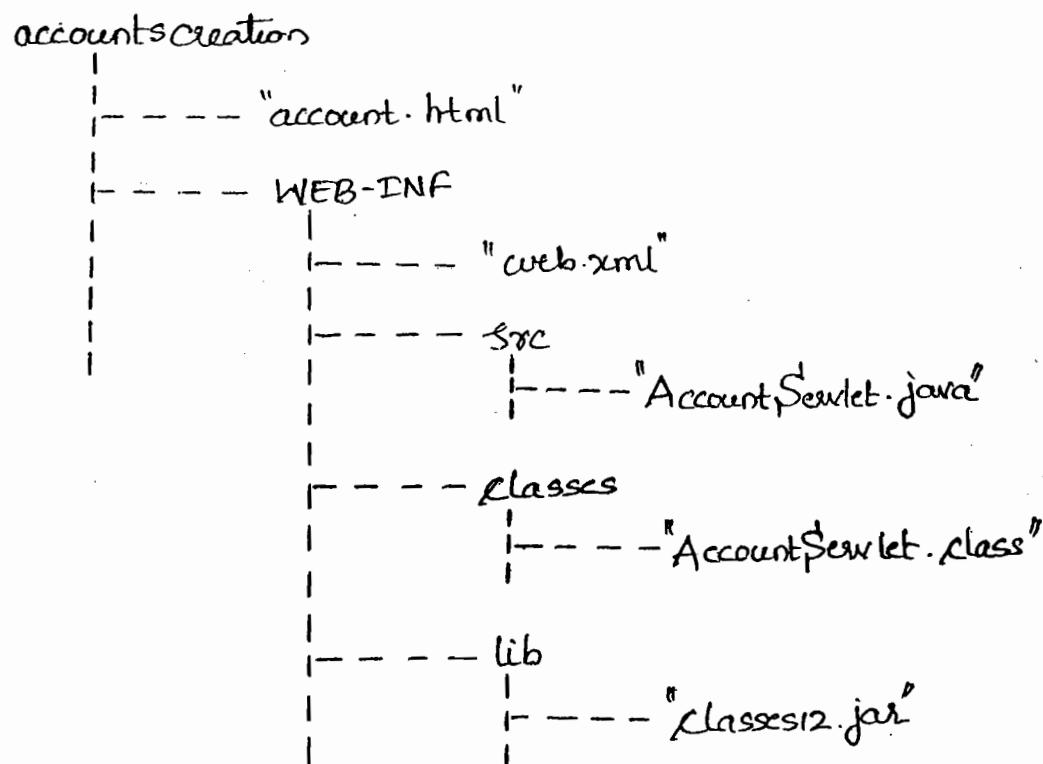
→ Q) Develop & Deploy a Web Application in which End User should be able to repeatedly Enter account details into the web form, that are to be stored in the database. Don't hard code database connection details in Servlet Source Code.

Creation of Connection, PreparedStatement is done in init() method. Connection & PreparedStatement Objects are created only once in the application hence are written in init() method, since init() method is called only once.

Driver Name, Connection String, Username, Password

is they should not be literals but variables. These values should be supplied in 'xml' file. Therefore, whenever they need to be changed, only 'xml' file can be edited without touching the Source Code. Also, by hardcoding, we are making an application specific to those details. If hardcoded, Source Code should also be given to the client company who purchases the application so that they can change these details whenever required. That is if driver needs to be replaced or username or password is to be changed. But, the Source Code is never given to the clients. Hence, Supplying these details in 'xml' file is a safer thing to do.

Directory Structure :-



account.html :-

```

<HTML>
<BODY BGCOLOR = GOLD>
<CENTER> <H1> Account Creation </H1>
        <FORM ACTION = "./Create" METHOD = "POST">
            Account Number : <INPUT TYPE = "text" NAME = "t1">
                            <BR><BR>
            Account Holders Name : <INPUT TYPE = "text" NAME = "t2">
                            <BR><BR>
            Balance : <INPUT TYPE = "text" NAME = "t3">
                            <BR><BR>
            <INPUT TYPE = "Submit" VALUE = "Create Account">
        </FORM> </CENTER>
</BODY>
</HTML>

```

Note:- To avoid the Hardcoding of database Connection related information, we make use of initialization parameters concept.

→ What are initialization parameters? How to Supply them to a Servlet? How the Servlet can Receive them?

Name - Value pairs of textual information Supplied to the Servlet during its initialization phase are known as initialization parameters or Servlet 'init' parameters.

We supply init params from 'web.xml'. By calling the method `getInitParameter()` on the `ServletConfig` object, init parameters are retrieved.

AccountServlet.java :-

```

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

public class AccountServlet extends HttpServlet
{
    Connection con;
    PreparedStatement ps;

    public void init(ServletConfig c) throws ServletException
    {
        String driver = c.getInitParameter("p1");
        String connstr = c.getInitParameter("p2");
        String user = c.getInitParameter("p3");
        String pwd = c.getInitParameter("p4");

        try
        {
            Class.forName(driver);
    
```

This method takes init parameter name as argument

----- Returns corresponding value into this variable.

Class.forName(driver);

```
ps = con.prepareStatement("INSERT INTO ACCOUNT
    VALUES(?, ?, ?)");
```

} // try

Catch (Exception e)

{

e.printStackTrace();

} // catch

} // init

public void destroy()

{

try

{

ps.close();

con.close();

}

Catch (Exception e)

{

e.printStackTrace();

}

} // destroy

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

{

```

PrintWriter out = response.getWriter();
    // capturing Input i.e, I section

    int ano = Integer.parseInt(request.getParameter("t1"));
    String name = request.getParameter("t2");
    float bal = Float.parseFloat(request.getParameter("t3"));

    // Processing Input i.e, P section

    try {
        ps.setInt(1, ano);
        ps.setString(2, name);
        ps.setFloat(3, bal);
        ps.executeUpdate();
    } // try

    catch(Exception e) {
        e.printStackTrace();
    } // catch

    // Presenting Output i.e, O section

    out.println("<HTML>");
    out.println("<BODY BGCOLOR = GOLD>");
    out.println("<H1> Account Created Successfully <H1>");
    out.println("<A HREF = account.html> One More To Create?
                </A>");

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

} // doPost

```

10/08/09

120.

web.xml :-

<web-app>

<Servlet>

<Servlet-name> Seven </Servlet-name>

<Servlet-class> AccountServlet </Servlet-class>

<init-param>

<param-name> p1 </param-name>

<param-value> oracle.jdbc.driver.OracleDriver

</param-value>

</init-param>

<init-param>

<param-name> p2 </param-name>

<param-value> jdbc:oracle:thin:@localhost:1521:
orcl </param-value>

</init-param>

<init-param>

<param-name> p3 </param-name>

<param-value> scott </param-value>

</init-param>

<init-param>

<param-name> p4 </param-name>

<param-value> tiger </param-value>

</init-param>

</Servlet>

<Servlet-mapping>

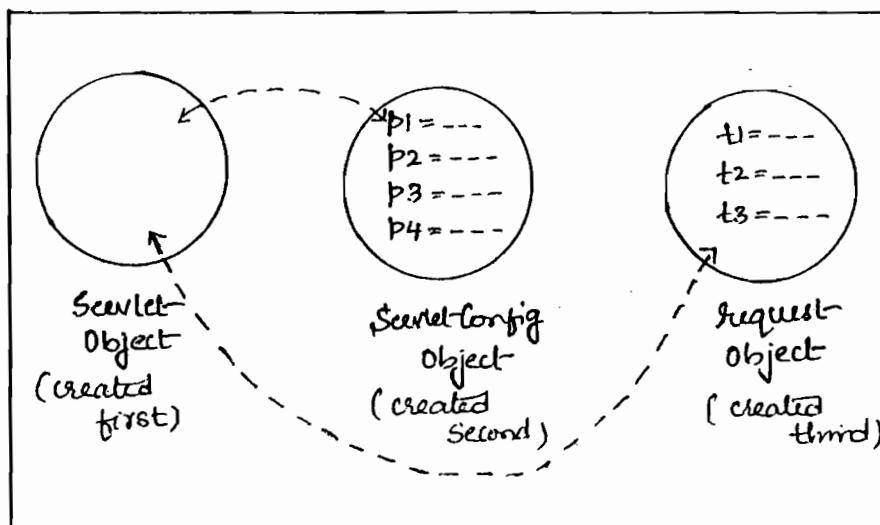
<Servlet-name> Seven </Servlet-name>

</servlet-mapping>
</web-app>

If more than one Servlet is present, these values belong to AccountServlet because, they are written in the Registration section of this Servlet alone i.e., they are made local to this servlet. Other servlets cannot access these values.

Servlet life cycle: AccountServlet class is loaded. Its instance created. Then ServletConfig Object is created. This Object contains details given in web.xml file i.e., when the Servlet Engine opens web.xml to check Servlet Registration then the init parameters which are nothing but Name-Value pairs are stored in Config Object.

Hence, init parameters are supplied to Servlet Engine by us through web.xml. Servlet Engine creates ServletConfig object with these values and provides these values to servlet through init method.

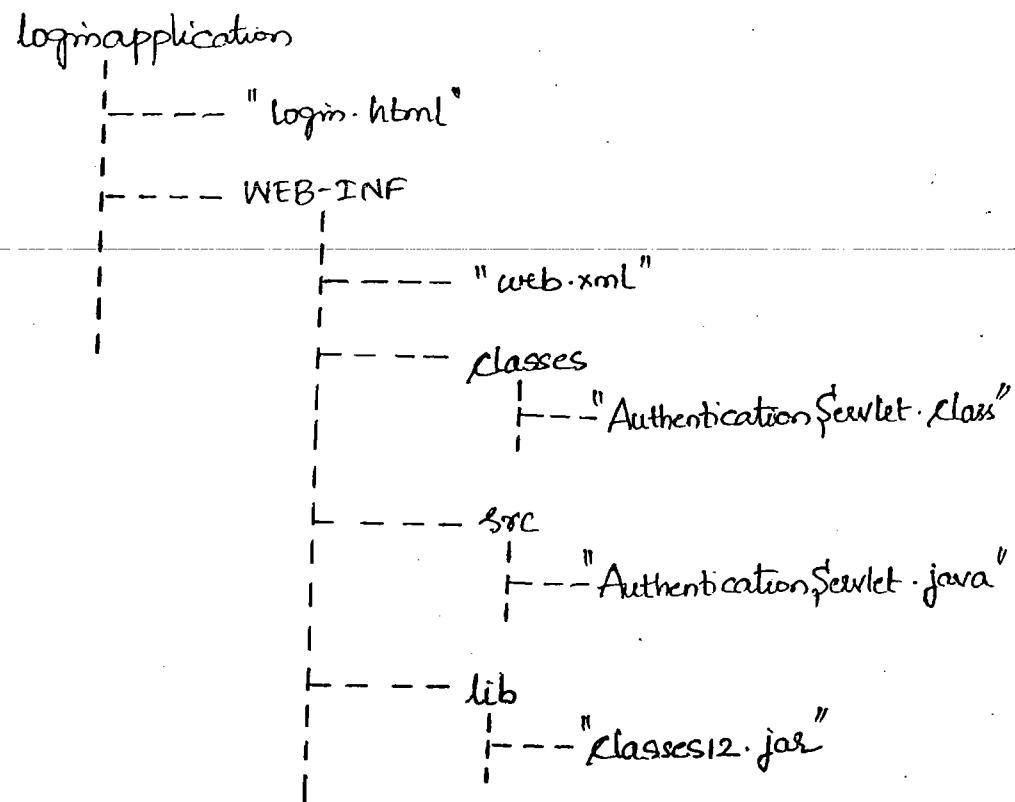


`init()` method is calling `getInitParameter()` method. Hence `Servlet` class object is talking to `ServletConfig` Object

When `init()` method calls `getParameter()` method, `Servlet` class object needs data from `request` object. Hence `Servlet` object is now talking to `request` object. This phenomenon is message passing, which is nothing but simple method calls in Java

→ Q) Develop & Deploy a login web Application

Directory Structure :-



Login.html :-

<HTML>

<CENTER> <H1> Login Screen </H1>

<FORM ACTION = "./Login" METHOD = "POST">

User Name : <INPUT TYPE = "text" NAME = "t1">

Password : <INPUT TYPE = "password" NAME = "t2">

<INPUT TYPE = "submit" VALUE = "Login">

</FORM> </CENTER>

</BODY>

</HTML>

AuthenticationServlet.java :-

import javax.servlet.*;

import javax.servlet.http.*;

import java.io.*;

import java.sql.*;

public class AuthenticationServlet extends HttpServlet

{

Connection con;

PreparedStatement ps;

public void init() throws ServletException

{

String driver =

getInitParameter("p1");

→ No reference is used. Hence it is a self calling process. Therefore, it also belongs to Servlet class object i.e., in this case, called method belongs to same object as calling

*As this class is inherited
no class, so no constructor
is defined*

over class Servlet class (not object) from Servlet

is inherited by gets some methods mainly

1) public void init(ServletConfig config)

throws ServletException.

2) public void service(ServletRequest request,
ServletResponse response)

throws ServletException, IOException.

3) public void destroy()

4) public String getServletInfo()

5) public ServletConfig getServletConfig()

so in which, when we are overriding & implementing, we directly implement, when Servlet engine creates

this object and from that object

reference Servlet engine calls

these methods are of any Variables

(variables into

two Servlet classes and two Methods

to obtain create & info to Servlet

engine class object create & ref of Servlet automatically no call

of Servlet class object

for Servlet class life

cycle Methods init & service

(do Servlet's E.g.) & destroy

```

String comstr = getInitParameter("p2");
String user = getInitParameter("p3");
String pword = getInitParameter("p4");

try
{
    Class.forName(driver);
    con = DriverManager.getConnection(comstr, user, pword);
    ps = con.prepareStatement("SELECT * FROM OURUSERS
                            WHERE UPPER(MYUSER) = UPPER(?) AND
                            PASSWORD = ?");

    } // try

    catch (Exception e)
    {
        e.printStackTrace();
    } // catch

} // init

public void destroy()
{
    try
    {
        ps.close();
        con.close();
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}

```

```

public void doPost (HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException
{
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    // capturing Input i.e, I section

    String user = request.getParameter("t1");
    String prod = request.getParameter("t2");
    boolean flag = false;
    // processing Input i.e, P section

    try
    {
        ps.setString(1, user);
        ps.setString(2, prod);
        ResultSet rs = ps.executeQuery();
        if (rs.next) flag = true;
        rs.close();
    } // try

    catch (Exception e)
    {
        e.printStackTrace();
    } // catch

    // Presenting Output i.e, O section

    out.println("<HTML>");
    out.println("<BODY BGCOLOR = CYAN>");
}

```

```

out.println("<H2> Welcome To Our Website </H2>");
else
{
    out.println("<H2> Login Failed </H2>");
    out.println("<A HREF=login.html> <H3>Try Again?
                </H3></A>");
    out.println("</BODY></HTML>");
    out.close();
}
} // doPost
} // class

```

web.xml :-

```

<web-app>
    <servlet>
        <servlet-name> eight </servlet-name>
        <servlet-class> AuthenticationServlet </servlet-class>

        <init-param>
            <param-name> p1 </param-name>
            <param-value> oracle.jdbc.driver.OracleDriver
        </init-param>                                </param-value>

        <init-param>
            <param-name> p2 </param-name>
            <param-value> jdbc:oracle:thin:@localhost:
                           1521: ORCL </param-value>
        </init-param>
    
```

```
<init-param>
    <param-name>p3</param-name>
    <param-value>scott</param-value>
</init-param>

<init-param>
    <param-name>p4</param-name>
    <param-value>tiger</param-value>
</init-param>

</servlet>

<servlet-mapping>
    <servlet-name>eight</servlet-name>
    <url-pattern>/login</url-pattern>
</servlet-mapping>

</web-app>
```

URL :-

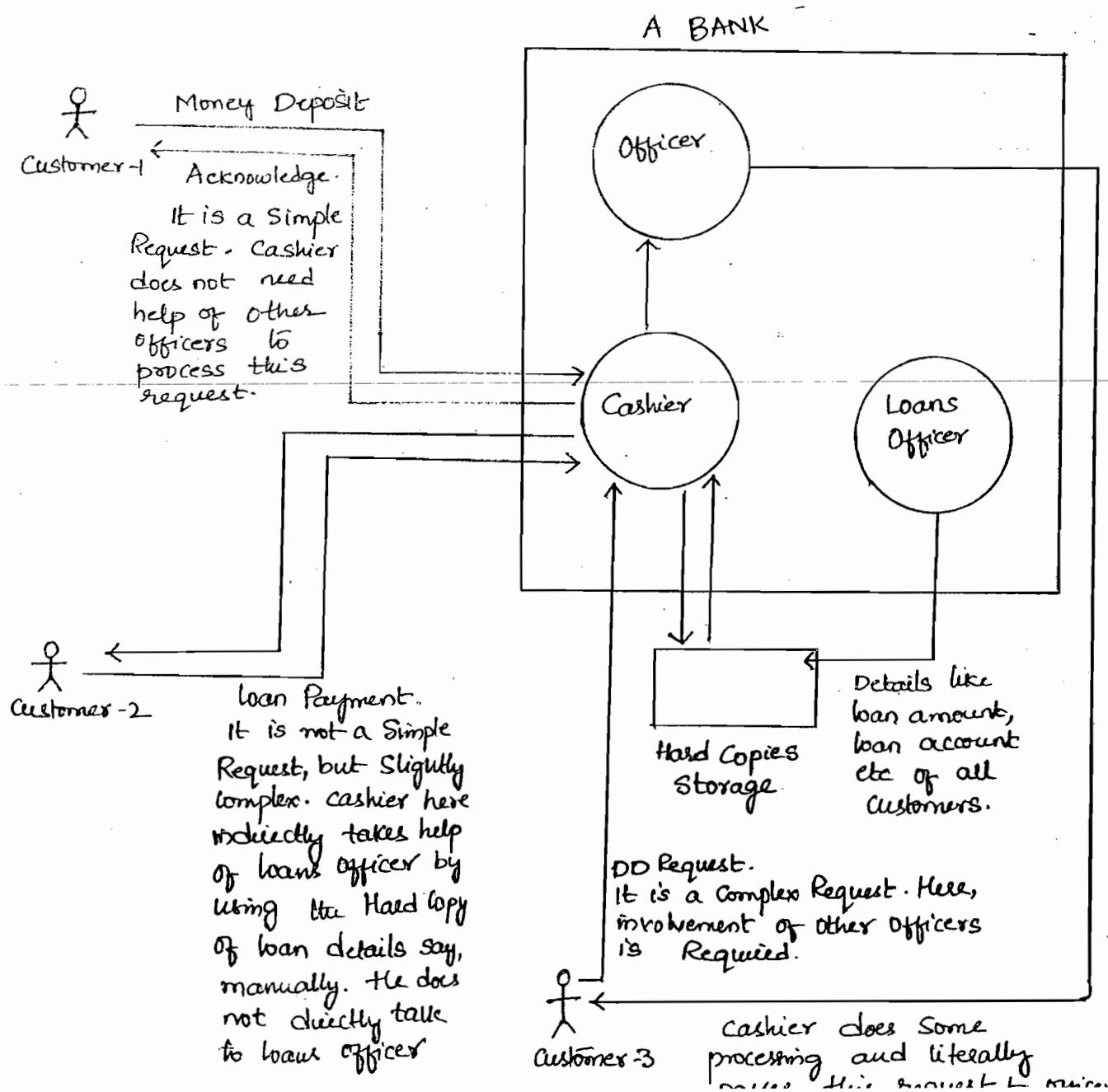
http://localhost:8081/loginapplication/login.html

11/08/09

Servlet Collaboration :-

→ One Servlet assisting another Servlet in processing a client Request is known as Servlet Collaboration

Consider the following Scenario :-



→ Servlets collaborate with one another in two ways:

- 1) By Sharing Data (loan Payment Example)
- 2) By Sharing Control (DD issuance Example)

→ In the first case, only one servlet will actually be involved in processing the client request. The other servlet shares data and therefore, the first servlet could finish the job.

→ In the second case, more than one servlet directly participates in processing the client Request.

→ How does a servlet share data with other servlet?

*) Through Attribute Concept

*) An Attribute is a Name-Value pair of Representation of a Data Item.

*) Servlet API provides methods to deal with Attributes.

*) `setAttribute(String name, Object value)` :- This method is used to store a particular data item in a given scope.

*) `getAttribute(String name)` :- This method retrieves the data item stored in a given scope and returns the same with that name if no attribute exists, this method returns NULL.

*) `removeAttribute(String name)` :- It is used to remove a data

*) we have three scopes for an attribute

- 1) request scope
- 2) session scope
- 3) application scope

javax.Servlet.ServletContext

ServletContext, ServletConfig, HttpServletRequest, HttpServletResponse are all interfaces in Servlet API with implementation classes in Container (Tomcat or Weblogic etc)

No. of Config Objects per Servlet is one.

No. of request & response objects per Servlet depends on No. of client Requests

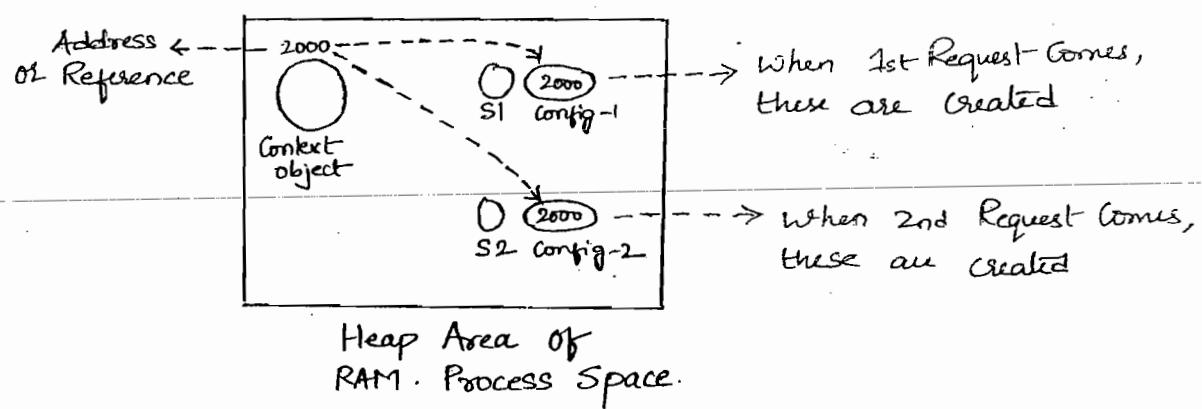
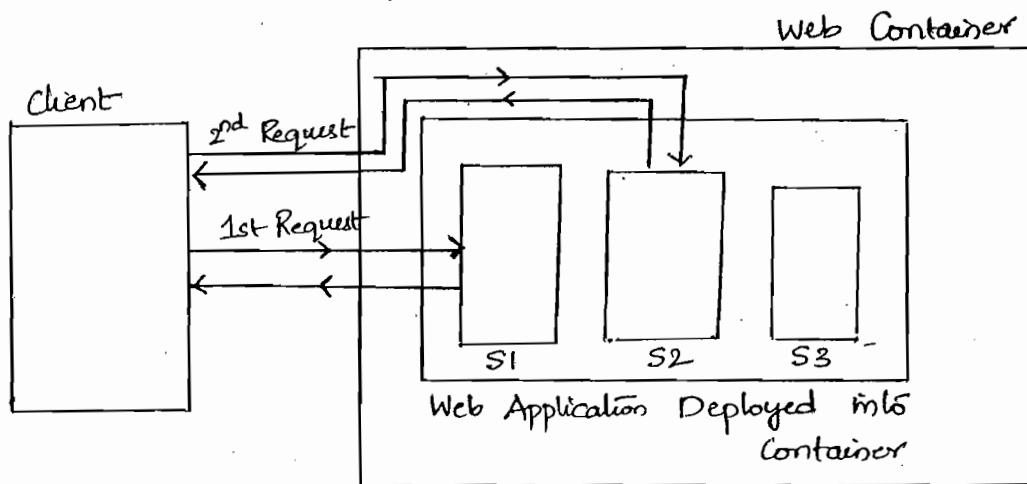
→ ServletContext is an interface. Container has implementation class for this interface. As soon as the web Application is deployed, Servlet Engine creates ServletContext object i.e. object of child class implementing this interface

→ ServletContext object is one per web application.

As soon as the application is deployed into Tomcat and Server is started, the first object to be created is ServletContext object, even before the client Request comes.

→ Every Servlet of the web Application shares this unique ServletContext object.

→ Servlet Engine encapsulates `ServletContext` reference in the `ServletConfig` object during the initialization phase of each Servlet of the Application

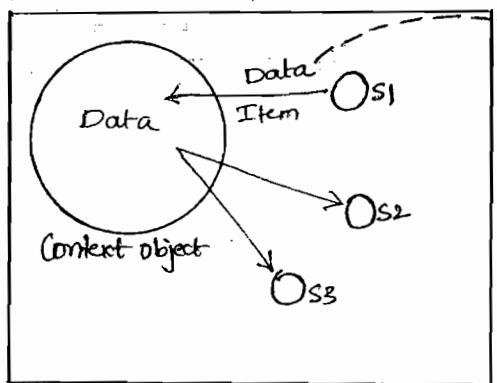


→ In a Servlet, `ServletContext` reference is captured as follows:

`ServletContext sc = Config.getServletContext();` ← if Parameterized
(OR) `init()` is used.

`ServletContext sc = getServletContext();` ← if Zero Argument
`init()` is used.

→ `ServletContext` Object acts as Shared Transformation Reference



This Data can be Shared by Other Servlet objects.

If Data need not be Shared, it is never kept in ServletContext Object

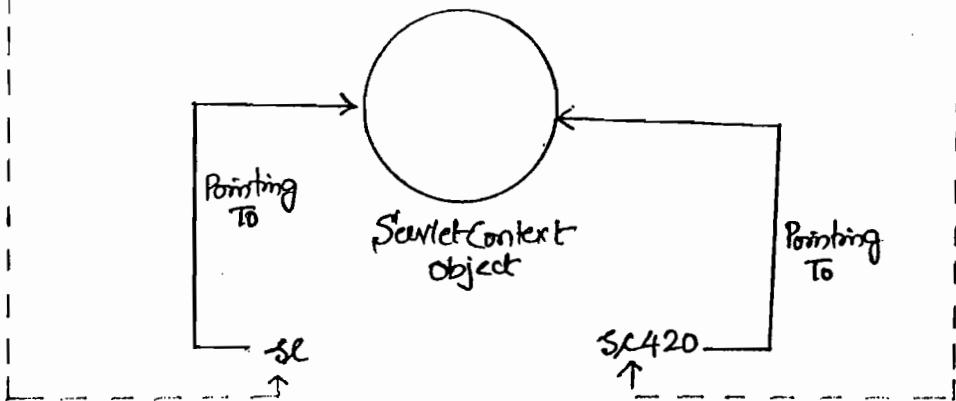
Note:- A Data Item is said to have Application Scope if it is stored in ServletContext Object.

```
=====
doGet()
{
    String v = response.getParameter();
    ServletContext sc = getServletContext();
    sc.setAttribute("Hello", v);
}
=====
```

Servlet-1 code

```
=====
doGet()
{
    ServletContext sc420 = getServletContext();
    sc420.getAttribute("Hello");
}
=====
```

Servlet-2 code



→ `ServletContext` object provides a method to retrieve Context parameters

for Example :

```
String value = sc.getInitParameter("p1");
```

→ What are Context Parameters?

Initialization parameters supplied to the whole Application from the Deployment Descriptor are known as Context parameters.

At the time of Application Deployment, Servlet Engine encapsulates context parameters in the `ServletContext` object.

We supply context parameters as follows :-

`<web-app>`

`<context-param>`

`<param-name> p1 </param-name>`

`<param-value> 123 </param-value>`

`</context-param>`

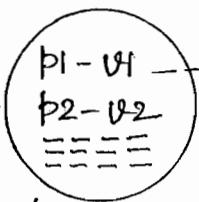
`<Servlet>`

`</Servlet>`

`</web-app>`

Hence, any no. of Servlets can access these parameters.

These parameters are shared by all the Servlets.



They are stored as Name-value pairs only

Servlet Context Object

init parameters are available only to one Servlet. Context Parameters are available to total application. Hence in 'web.xml' it is written above all Servlet Registration Sections

Therefore, Context parameters can be used for Storing Database information and Driver Details ie, Driver class, Conn String, Username, password

Servlet Life cycle \Rightarrow Servlet Engine loads Servlet Class

\downarrow then

Servlet Class instance is created

It misses Context or initialization info

\downarrow then

ServletConfig object is created. Here init-params and ServletContext reference is encapsulated.
Hence the missing info is provided

\rightarrow ServletContext object provides a method to create 'RequestDispatcher' object
for Example :-

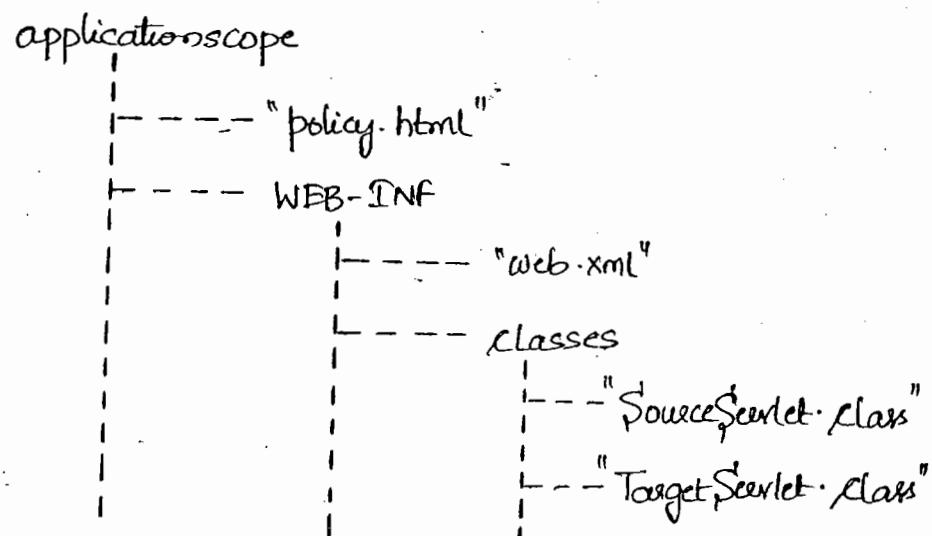
RequestDispatcher rd = sc.getRequestDispatcher(String target);

\rightarrow RequestDispatcher object is used for inter-Servlet Communication

12/08/09

- Q) Develop & Deploy a Web Application in which, a Servlet shares data in Application Scope

Directory Structure :-



URL :-

http://localhost:8081/applicationscope

web.xml :-

<web-app>

 <Context-param>

 <param-name>adminmail</param-name>

 <param-value>Admin@nit.com</param-value>

 </Context-param>

```

<Scenlet-class> SourceScenlet </Scenlet-class>
</Scenlet>

<Scenlet>
    <Scenlet-name> two </Scenlet-name>
    <Scenlet-class> TargetScenlet </Scenlet-class>
</Scenlet>

<Scenlet-mapping>
    <Scenlet-name> one </Scenlet-name>
    <url-pattern> /Source </url-pattern>
</Scenlet-mapping>

<Scenlet-mapping>
    <Scenlet-name> two </Scenlet-name>
    <url-pattern> /target </url-pattern>
</Scenlet-mapping>

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

</web-app>

```

policy.html :-

```

<HTML>
    <BODY BGCOLOR = "CYAN">
        <CENTER><H1> WWW. INSURANCE . COM </H1>
        <FORM ACTION = "./Source">

```

```

<INPUT TYPE = "submit" VALUE = "Send">
</FORM></CENTER>
</BODY>
</HTML>

```

SourceServlet.java :-

```

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

public class SourceServlet extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String pno = request.getParameter("policy");
        ServletContext context = getServletContext();
        context.setAttribute("np", pno);
        out.println("<HTML>");
        out.println("<BODY BGCOLOR = MAROON>");
        out.println("<H1>" + context.getInitParameter("admini-

```

```

    out.println("<A HREF = ./target> Get Your Policy
Number </A>");

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

    out.close();
}

} // doget

} // class

```

TargetServlet.java :-

```

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

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

    PrintWriter out = response.getWriter();

    ServletContext context420 = getServletContext();
    String pno = (String)context420.getAttribute("mp");
    ↑
    ↴ return type is java.lang.
    object.
}

```

```

    out.println("<HTML>");
    out.println("<BODY BGCOLOR = FUSHIA>");
    out.println("<H1>" + context420.getInitParameter("adminmail")
               "</H1>");
    out.println("<H1> Your Policy Number Is :" + pno + "</H1>");
    out.println("</BODY>");
    out.println("</HTML>");
    out.close();
}

// doget
}

// class

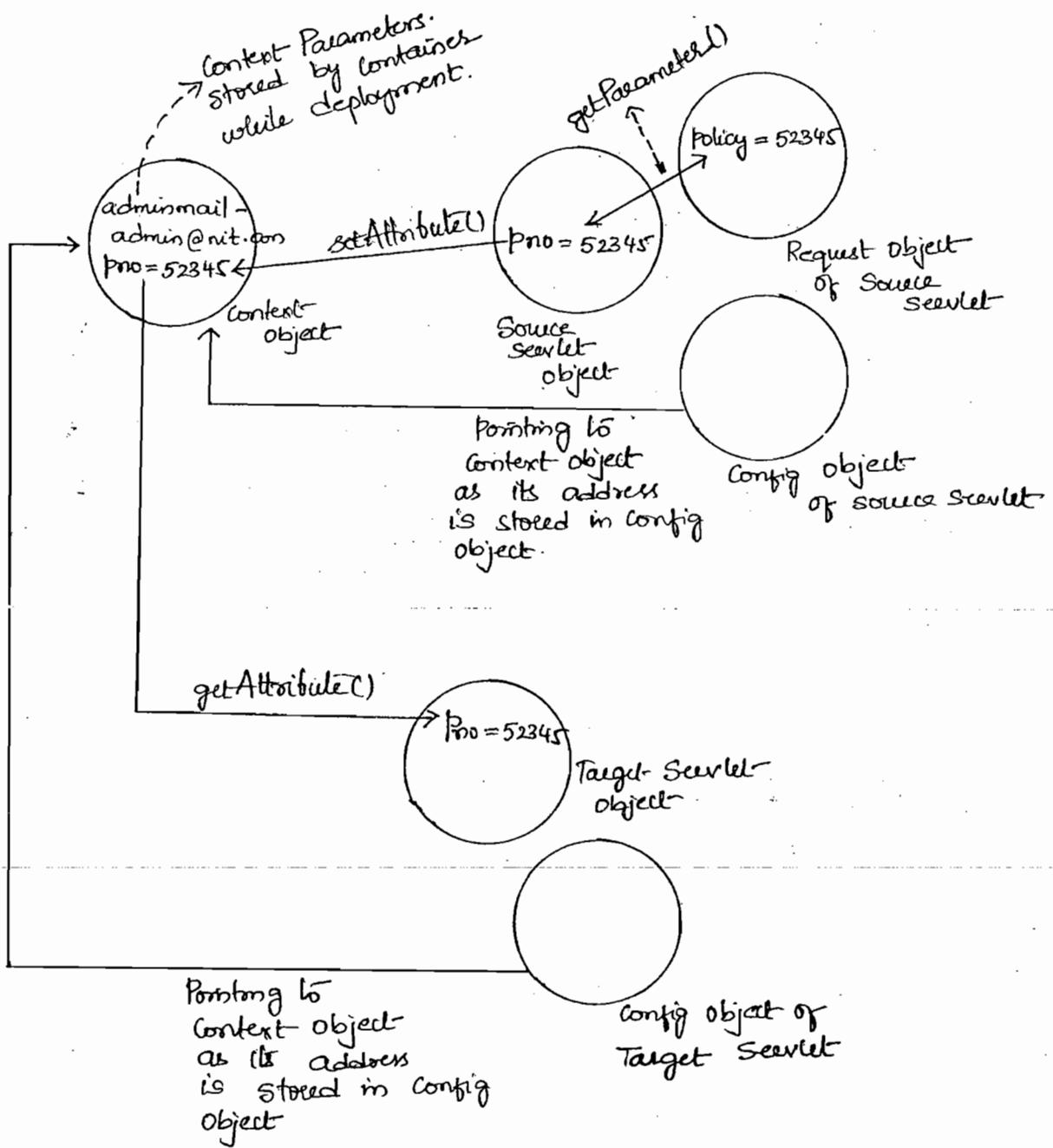
```

In the above Example, only Sharing of data took place. ScwL did not communicate with each other but only collaborated i.e. inter-Scwlet Communication has not happened.

As soon as Application is Deployed, Web Container reads the total Contents of web.xml file. That is, whatever information or instructions are given in the file is readed first and is stored as Java Objects in RAM. This is an XML process. When a request comes from the browser the URL Checking is done on these Objects but again web.xml is not fetched from Hardisk. If done so, performance decrease.

In the above Example, as soon as the Application is Deployed, Scwlet Engine creates ScwletContext Object and Name-Value pairs i.e. Content info is stored in this Object. These are read from web.xml file as mentioned above.

page is retrieved. When data is inputted, the SourceSevlet life cycle starts



The user inputted data is captured from Request Object by the Source Sevlet Object. This is by message passing i.e., by using `getParameter()` method. Then address of the Context Object is fetched. Here also object to object talking takes place. The fetched data is then stored into Context object using this address. The output page that was

Upto this point, two cycles have been completed between Client and Server. When the link to TargetServlet is clicked, the Third Cycle Starts. Now the control goes to TargetServlet. This class is loaded into RAM, its instance created, its own ServletConfig object is created in which the address of ServletContext object is stored. This Address is fetched by method calls and is used to retrieve the data which was originally stored in Context object by SourceServlet. Then, the corresponding output page is sent to the browser as Response with this, the Third cycle ends. This is how the control flows.

Request Dispatching :-

This is another Servlet collaboration method through Sharing of control.

→ One Servlet Delegating (dispatching) request processing duty to another Servlet (can also be a JSP too) is known as Request Dispatching.

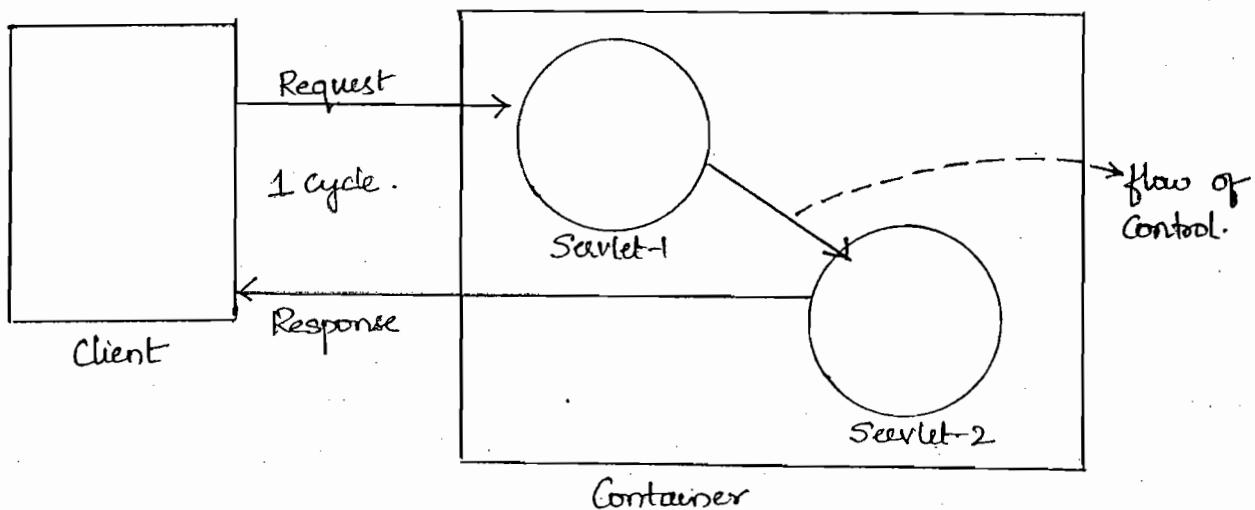
This is done when a request is too complex to process by one Servlet hence the duty is Shared.

13/08/09

→ What is the purpose of Request Dispatching? How is it implemented?

Request Dispatching is used to implement

During Request Dispatching, Control is Shared between Servlets within one Request-Response cycle.



Request Dispatching is implemented as follows:-

Step-1 :- `ServletContext sc = getServletContext();`

Step-2 :- `RequestDispatcher rd = sc.getRequestDispatcher("Other_Servlet/JSP Name");`

↓
To which Servlet/JSP
we want to switch the
control.

The switching is not done yet, but object for that purpose is created.

Step-3 :- `rd.forward(request, response);--`
OR

`rd.include(request, response);--`

At this point,
inter Servlet
communication
is implemented.

Whether Source Servlet or Target Servlet is producing the response, is decided by forward /

```

<INPUT TYPE = "submit" VALUE = "Get Take Home Salary">
</FORM> </CENTER>

</BODY>
</HTML>

```

web.xml :-

```

<web-app>
    <Servlet>
        <Servlet-name> one </Servlet-name>
        <Servlet-class> SourceServlet </Servlet-class>
    </Servlet>

    <Servlet>
        <Servlet-name> two </Servlet-name>
        <Servlet-class> TargetServlet </Servlet-class>
    </Servlet>

    <Servlet-mapping>
        <Servlet-name> one </Servlet-name>
        <url-pattern> /source </url-pattern>
    </Servlet-mapping>

    <Servlet-mapping>
        <Servlet-name> two </Servlet-name>
        <url-pattern> /target </url-pattern>
    </Servlet-mapping>

```

SourceServlet.java :-

146.

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

```
public class SourceServlet extends HttpServlet  
{
```

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

```
{
```

```
        float basic = Float.parseFloat( request.getParameter("basic") );
```

```
        float da = basic * 0.5f ;
```

```
        float hra = basic * 0.2f ;
```

```
        float gross = basic + da + hra ;
```

```
        request.setAttribute("gr", new float(gross));
```

The Data gross,
is to be maintained
for one cycle only.
Hence, it is kept
under request Scope
but not Application
Scope

SetAttribute takes Second
argument as Object. Hence,
a primitive data is converted
into object type using
"Wrapper Classes" concept here.
This is called 'Boxing'. It
happens automatically from 1.5
version of jdk onwards and need
not be mentioned explicitly as
above.

```
    ServletContext sc = getServletContext();
```

```
    RequestDispatcher rd = sc.getRequestDispatcher("/target");
```

```
    rd.forward( request, response );
```

```
} // doGet
```

TargetServlet.java :-

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

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

```
import java.io.*;
```

```
public class TargetServlet extends HttpServlet
```

```
{
```

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

```
{
```

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

```
    PrintWriter out = response.getWriter();
```

Float g = (Float) request.getAttribute("gr");

Downcasting..

It is in
Object form.

Hence, cannot be

used in Mathematical
calculations directly.

It has to be converted
into primitive type. This
is 'unboxing'. It is done

below

Since Return Type of
getAttribute is java.lang.Object.

float gross = g.floatValue();

UnBoxing. It is done automatically
from jdk 1.5 Version onwards.

we can combine

and float net = gross - gross * 0.2f;

float gross =

(float)request.getAttribute("gr");

out.println("<HTML>");

```

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

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

    out.close();
  
```

} // doGet

} // class

URL :-

http://localhost:8081/dispatchingapplication/basic.html.

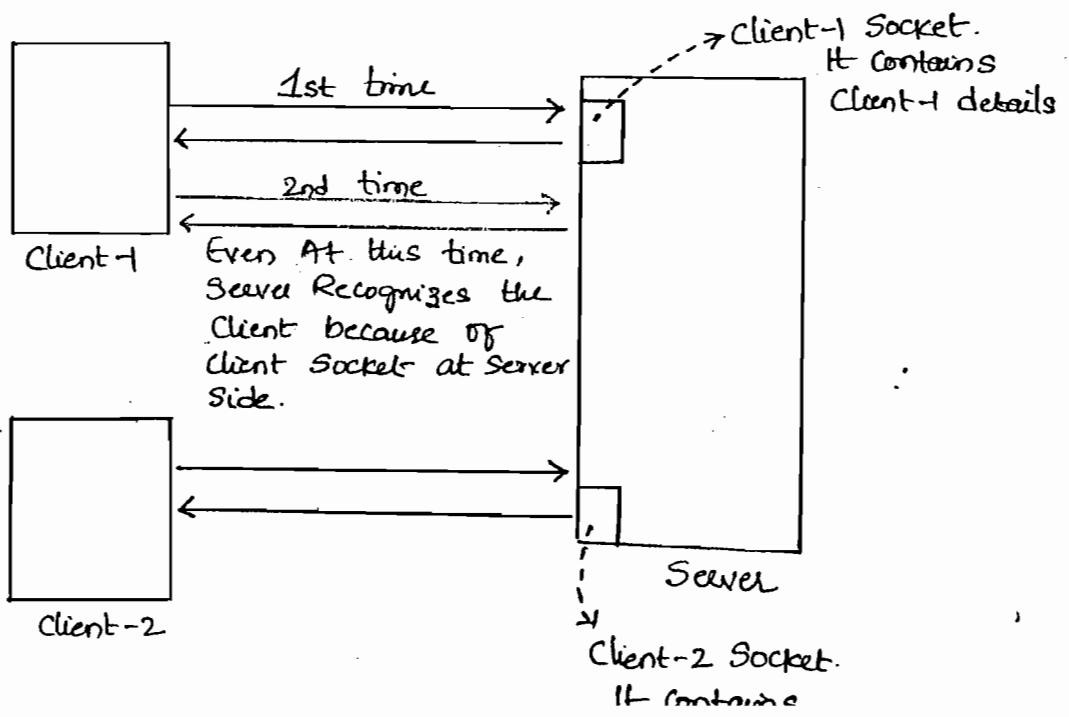
17/08/09

Session Tracking :-

→ We have two kinds of Protocols

1) Stateful / connection Oriented

2) Stateless / Connectionless



If after one request-response cycle, the client is disconnected i.e. Client Socket destroyed at Server Side, it is Connectionless protocol.

If connection is maintained throughout the process i.e., as many request-response cycles as required, and is disconnected only explicitly, it is Connection Oriented.

→ A protocol is said to be stateful if the Server has got the memory of prior Client Connections and is able to distinguish one Client Request from that of others.

Example : FTP (File Transfer Protocol)

Disadvantage in using FTP like protocols is that they reduce performance, as number of connections i.e., requests increase. A single socket of Client at Server Side eats away a bulk of resources. Therefore it is obvious that as no. of Client Requests increases performance decreases. An FTP protocol can support a maximum of say 200 connections with ease but as the number increases performance decreases.

Even if no. of Client Requests increase, the performance of Server should not decrease. Such a Server is Scalable.

→ A Server that has Stateful protocol implementation is not Scalable.

→ A protocol is said to be stateless/ Connectionless if the Server has got no memory of prior Client Requests.

the other in a Series of Client-Server interactions.

Example :- Http.

→ A Http Server is Scalable.

Stateless protocols even though are performance oriented, for a commercial website, the protocol should act as stateful only, since user request does not end in one request-response cycle. This needs to be managed by the programmer.

→ What is the limitation for a web Application/website as HTTP protocol is Stateless?

Almost all the cases, no online business service is possible with one client-server interaction. Because of the nature of HTTP, web server disconnects the client every time. As a result, user's input in the previous request for that business service cannot be kept track of at the server side, i.e., user-website interaction becomes Stateless.

User (Client) - Web Application interaction should be Stateful in order to provide Online Business Services to the clients.

→ What is a Session?

It is the time period during which, the Server is able to recognize the client uniquely in a Series of Client-Server interactions.

→ What is Session Tracking?

It is the ability of the Server (Container) to recognize a client uniquely and associating a particular request to a particular client in a series of Client-Server interactions.

18/08/09

Steps to implement Session Tracking

Step 1 :- Getting the reference of HttpSession object
i.e., Starting A session

`HttpSession session = request.getSession();`

Step 2 :- Dealing with user (client) data by calling attribute methods on Session Object

Step 3 :- Ending the Session

A Session can End in two ways:

1) User Explicitly logs-out.

Then we call `session.invalidate();` to end the Session

2) If Session Time Out Occurs

In this case, Container implicitly destroys the

→ What happens in the background when the following statement is executed?

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

When Servlet's Service method (doGet/doPost) calls the getSession() method on HttpServletRequest object, the following things happen:

- 1) Container checks if any Session ID came from the client.
- 2) If Session ID does not come from the Client, Container creates a new instance of HttpSession. It also creates a unique SessionID for that Session object. Container writes that Session ID into the response object. It returns the reference of the HttpSession object to the Servlet.
- 3) If Session ID is found in the client request, Container does not create the brand new HttpSession object. It returns the reference of the already existing Session object to the Servlet.

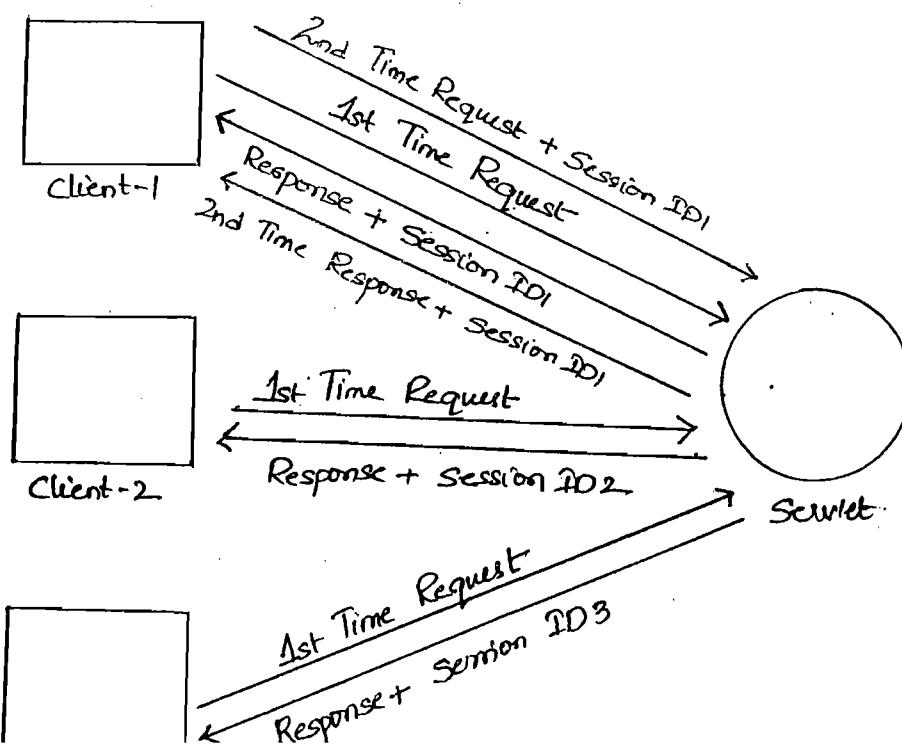
When first time request comes from browser, it does not send the Session ID.

When doGet/doPost calls getSession() method on the request object, the Container checks whether request has Session

If there is no session ID coming from browser, the container creates a brand new Session object and a unique Session ID assigned to it. These two are stored in a table in the background.

This Session ID is literally written into response Object. The reference of this Session object is given to the servlet. When Response is sent to the client, this Session ID is also sent which is not visible. Hence, browser is able to capture this ID which comes from the Server.

When Second time the request comes from same Client - i.e. browser say, for the same Servlet, this Session ID is also sent. Container checks request object again for any Session ID. This ID is verified in the table and when the match is found, the corresponding Session object's reference is sent to servlet and also this Session ID is again written into response object.



Session ID	Session Objects
Session ID1	SessionObject-1
Session ID2	SessionObject-2
Session ID3	SessionObject-3
-----	-----

Servlet Stores the Data i.e. Dealing with the Data is done using Attribute methods. Hence, previous request's data is not lost.

Per client one session object is created, which has a unique session ID assigned to it. Even though the socket of client at server side is destroyed, its session object is not destroyed.

When a Session is Complete, i.e. when user opts to log-out, the session must be ended. When a session is ended, session object is destroyed and all the data it is holding is lost and even the session ID is destroyed. This is how a session is ended Explicitly.

If the user does not explicitly end the session by logging-out and chooses to close the browser instead, the session does not end. It stays active for some stipulated period of time and then container which realizes that no request has come, it automatically ends the session implicitly.

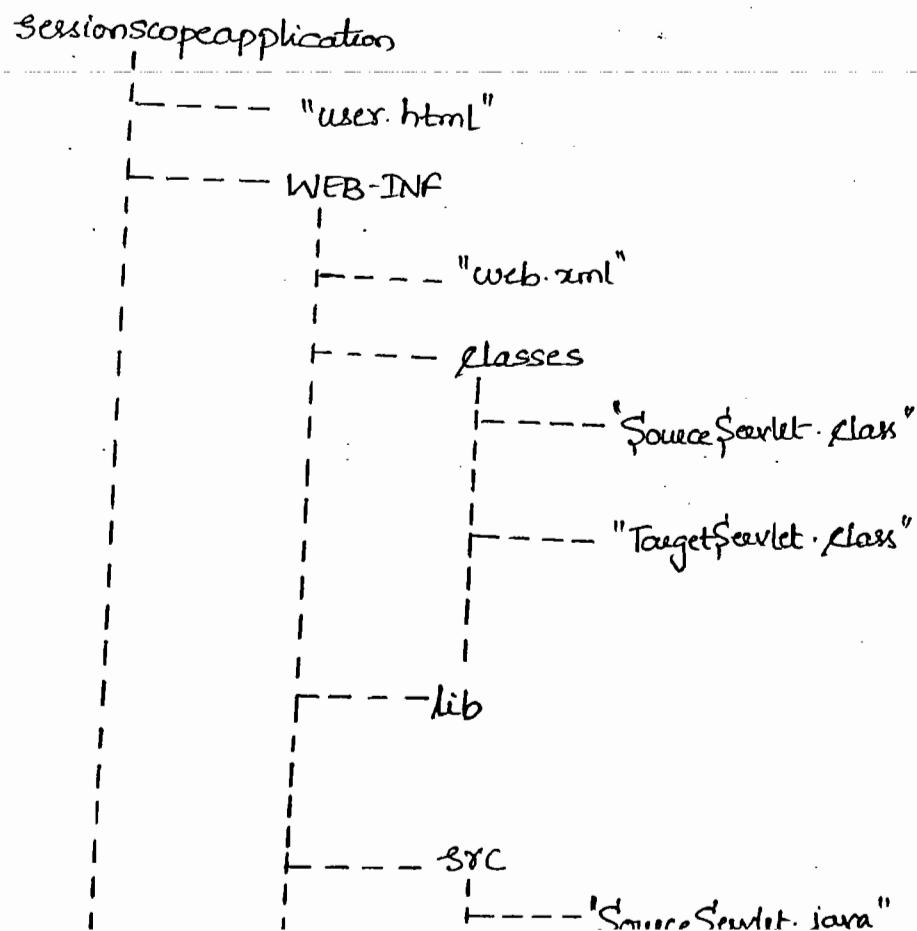
If the user does not opt to log-out nor does he close the browser, but stays idle after performing say two requests browser has a ...

If no request has come, the Container implicitly ends the Session.

The above two cases come under "Session Time out". The Container after ending the Session, destroys the Session object and its associated data. If we explicitly ends Session, in the background "session.invalidate()" method should be executed. If implicitly Session is ended, we need not do anything i.e. when Session Time out happens.

- Q) Develop & Deploy a Web Application in which, Two Servlets Share data in Session Scope.

Directory Structure :-



URL :- http://localhost:8081/sessionScopeApplication/user.html

19/08/09

web.xml :-

```

<web-app>
    <Servlet>
        <Servlet-name> One </Servlet-name>
        <Servlet-class> SourceServlet </Servlet-class>
    </Servlet>

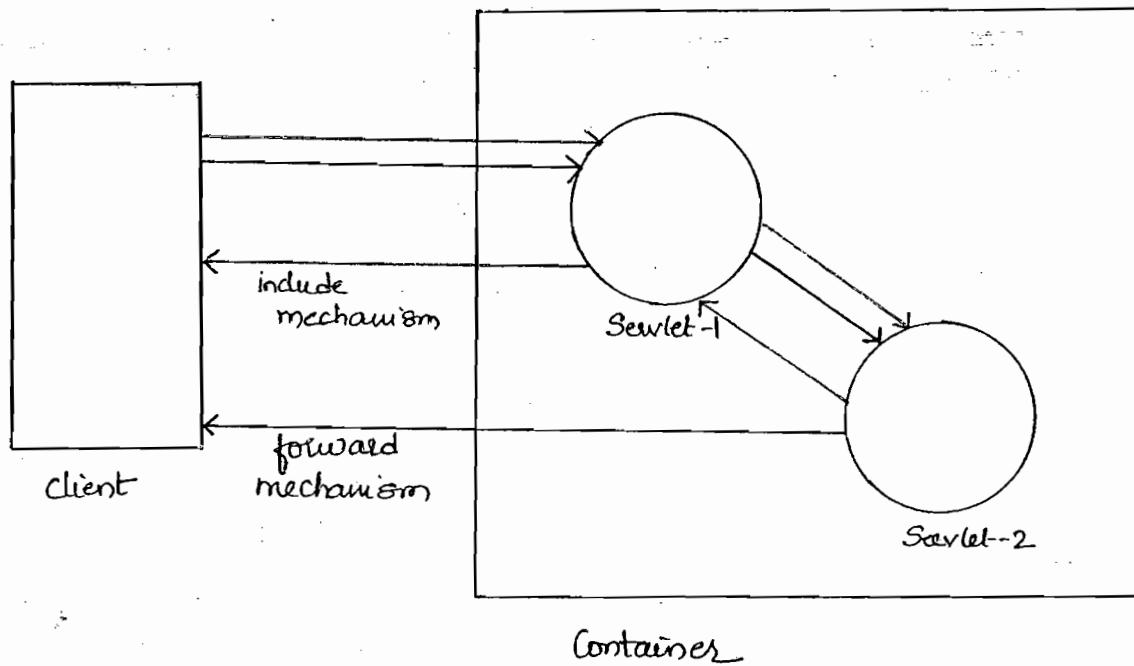
    <Servlet>
        <Servlet-name> Two </Servlet-name>
        <Servlet-class> TargetServlet </Servlet-class>
    </Servlet>

    <Servlet-mapping>
        <Servlet-name> One </Servlet-name>
        <url-pattern> /Source </url-pattern>
    </Servlet-mapping>

    <Servlet-mapping>
        <Servlet-name> Two </Servlet-name>
        <url-pattern> /target </url-pattern>
    </Servlet-mapping>

</web-app>

```



Whatever be the Mechanism, Accepting the request and switching the Control is done, by the first Servlet only. Upto this point 'I' is over, Some part of 'P' may be over. Remaining part of 'P' is done in Second Servlet. 'O' is left out.

If Servlet-2 is sending the output to client, it is forward mechanism. If output is to be given by Servlet-1 to the Client, then control has to come back to Servlet-1 from Servlet-2 after some processing. Now Servlet performs 'O' operation. This is include mechanism.

In Inter Servlet Communication, Include mechanism is least used. Forward mechanism is mainly used.

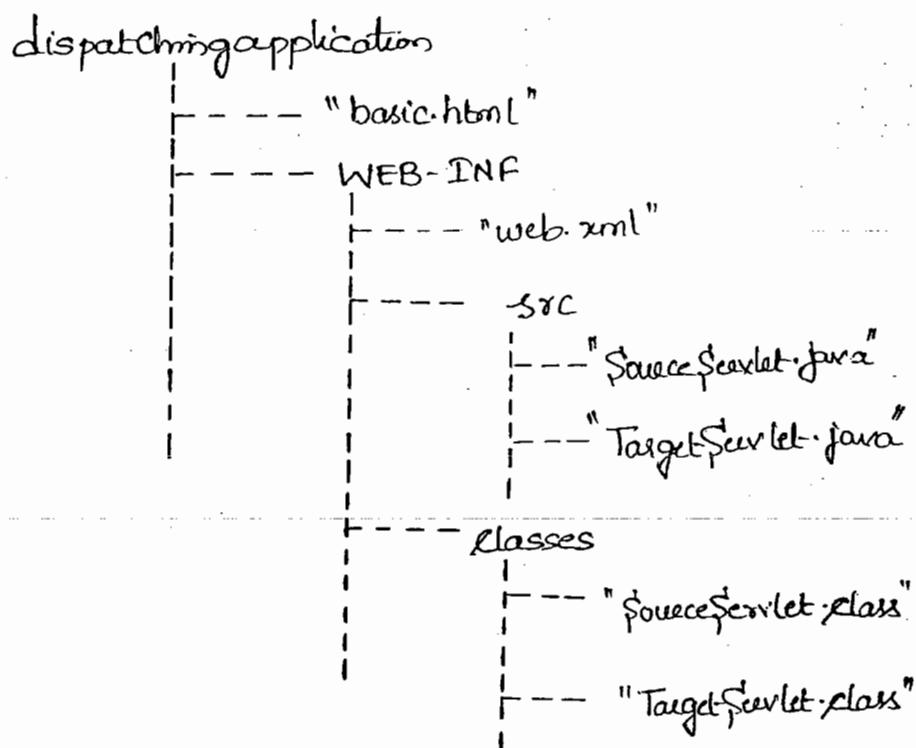
In Servlet-JSP Communication, forward mechanism is least used. Include mechanism is mostly used.

Note:- No matter how many Servlets participate in Request Dispatching, all those Servlets Share the Same request.

Response objects die only when one cycle is completed.

- Q) Develop & Deploy a web Application in which Request Dispatching is implemented.

Directory Structure :-



basic.html :-

<HTML>

<BODY BGCOLOR = OLIVE>

<CENTER><H1> Net Salary Query Screen </H1>

<FORM ACTION = "./Source">

Basic Pay : <INPUT TYPE = "text" NAME = "basic">

User.html :-

```

<HTML>
  <BODY BGCOLOR = "CYAN">
    <CENTER><H1> User Screen </H1>
    <FORM ACTION = ". /source">
      User Name : <INPUT TYPE = "text" NAME = "user"><BR><BR>
      <INPUT TYPE = "submit" VALUE = "Send">
    </FORM> </CENTER>
  </BODY>
</HTML>

```

SourceServlet.java :-

Lecture 10

```

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

public class SourceServlet extends HttpServlet
{
  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
  {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    // capturing Input i.e., I section
  }
}

```

This request
is applica
tive in enti
e application the
one achieves

HTTPSession receives details maintained at server and if client wait to again connect, then no need to create socket (is session scope) 158.

// Processing Input i.e., P Session

HttpSession Session = request.getSession();

Session.setAttribute("USR", user);

// Presenting i.e., O Session

out.println("<HTML>");

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

out.println(" Get User Name Here ");

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

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

out.close();

System.out.print(session.getId());

This message is print at client browser but important point is along with that message, invisibly HttpSession reference is arrived at client browser

The control goes to Target servlet but the important point here is ~~also along with, session~~ ~~target~~ ~~servlet~~ ~~reference~~ also sent to target, so at that servlet class no need to create new HttpSession object

] // doGet

] // class

TargetServlet.java :-

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

```
public class TargetServlet extends HttpServlet
```

```
{
```

```
    public void doGet(HttpServletRequest request, HttpServletResponse
```

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

If recognise the request until some session so this is Session scope

```
PrintWriter out = response.getWriter(); // No need of I-section  
// P-section
```

```
HttpSession session420 = request.getSession();
```

After here if recognise the old HttpSession object and give that same object address to session 420

```
System.out.println(session420.getId()); // print after some time user intentionally lose session id
```

If can old

```
String user = (String) session420.getAttribute("usr");
```

// O-section

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

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

```
out.println("<H1> User Name is : " + user + "</H1>");
```

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

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

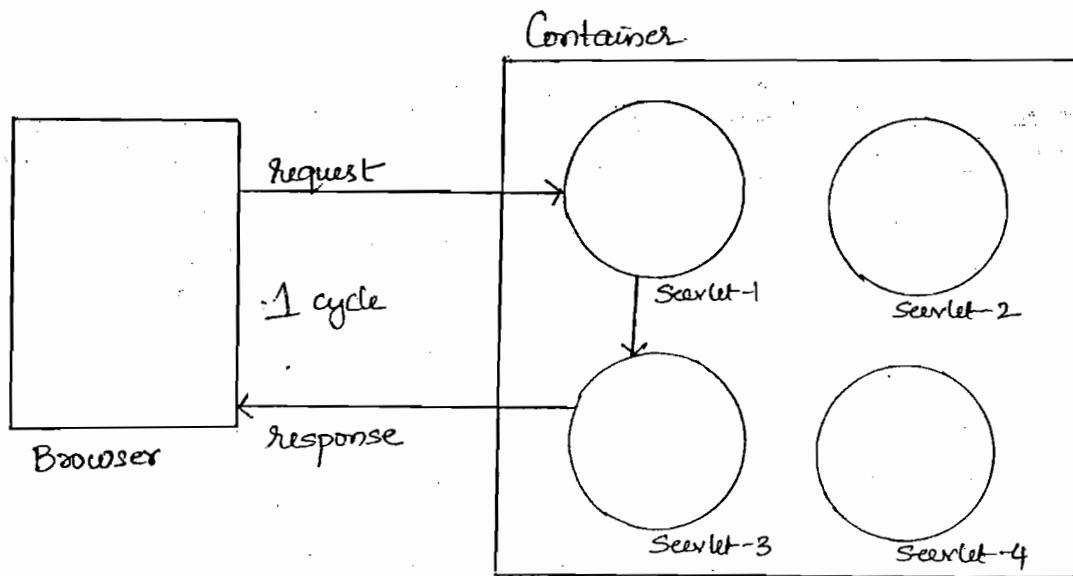
? // doGet

? // class.

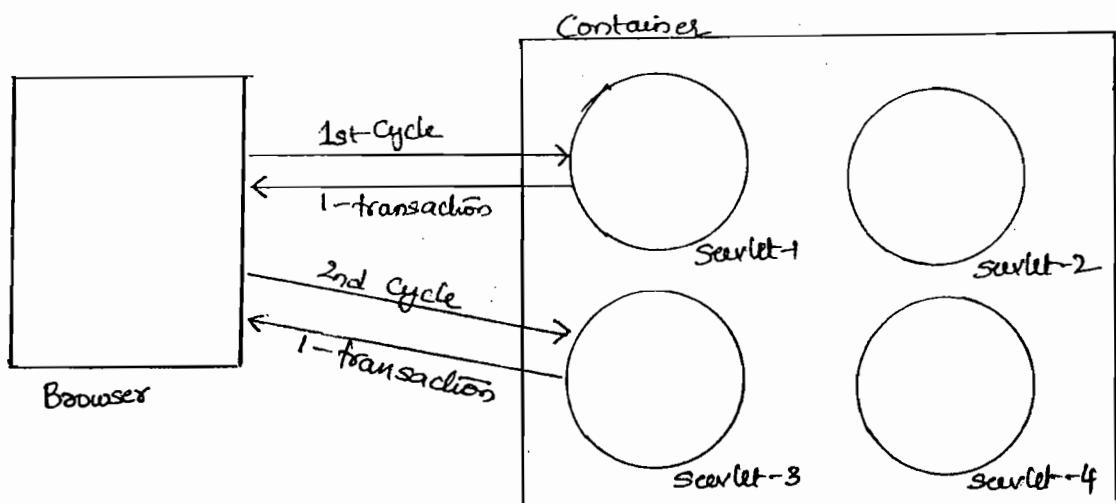
C:\> javac *.java < This would compile all the java files in that directory

The data stored in Session object is also in the form of Name-Value pairs

Till now, we have seen three different scopes of data storage i.e., Request Scope
Session Scope



Servlet-1 has some data. It needs to share that data with only one **Servlet**, say, **Servlet-3** in order to process the request. That too, the sharing would be enough for 1-request-response cycle. In this case keeping that data item in the request scope would be sufficient i.e. in the **Request Object**. This data item can be kept in **Context Object** too but it would not be a good practice to do so. This scenario is mostly evident in implementation of inter-**Servlet** or **Servlet-JSP** communication.



When Request comes, **Servlet-1** takes this request and corresponding response is sent. First cycle is complete and 1-transaction is done. In the second cycle it proceeds.

22/09/09

Types of ResultSets

→ we have 4 kinds of ResultSets

- 1) Scrollable and Non Updatable
- 2) Scrollable and Updatable
- 3) Non Scrollable and Updatable
- 4) Non Scrollable and Non Updatable. (default)

Mostly used
in Industry Strength
Applications

} Less frequently used
in Industry

→ A ResultSet is said to be Scrollable if we can move the cursor in forward direction, backward direction, and to any particular record of the ResultSet.

→ We can move the cursor only in forward direction in case of non-Scrollable ResultSet. We cannot move the cursor to a specified record directly.

→ If we can only read the column values from the ResultSet and if we cannot modify its contents, it is known as non-updatable ResultSet

→ If we can manipulate the ResultSet Content in addition to reading its contents, it is known as updatable ResultSet

→ How to create different kinds of ResultSet objects?

At the time of ResultSet object creation we don't do anything special. But we create Statement or

produce different kinds of ResultSets.

ResultSet Interface has 2 kinds of Constants

- 1) Scrollability Specifying Constants.
- 2) Updatability Specifying Constants.

→ Scrollability Specifying Constants are three:

- 1) TYPE_FORWARD_ONLY
- 2) TYPE_SCROLL_SENSITIVE
- 3) TYPE_SCROLL_INSENSITIVE

The first constant makes the ResultSet non-scrollable.

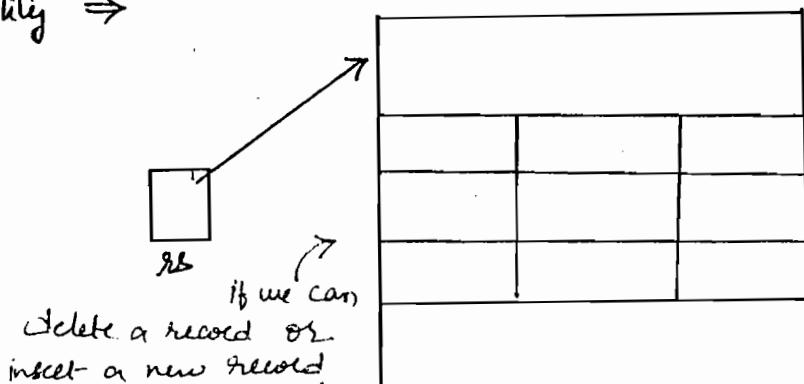
The other two constants make a ResultSet object scrollable.

→ There are two Updatability Specifying Constants

- 1) CONCUR_UPDATABLE
- 2) CONCUR_READ_ONLY

The first constant makes the ResultSet updatable and the other one makes it non-updatable.

After Submitting SQL Statement to DBMS, either using Statement or Prepared Statement, we get a ResultSet Object.
updatability →



These contents are read only. These column values cannot be modified i.e. they cannot be updated or deleted and also a new record cannot be inserted here. Hence this

transaction, which may require data from previous transaction. Such a data item is kept in Session Object. Hence, when data is needed for atleast one complete Business Transaction, it is kept in session scope.

If a data item is needed to be used during entire life of application and is to be shared by all the Servlets in the application, then keep that data item in Application scope i.e., in the Context Object.

→ What are different types of Web Application Deployments?

We have two kinds of deployment for a Web Application

1) Cold Deployment

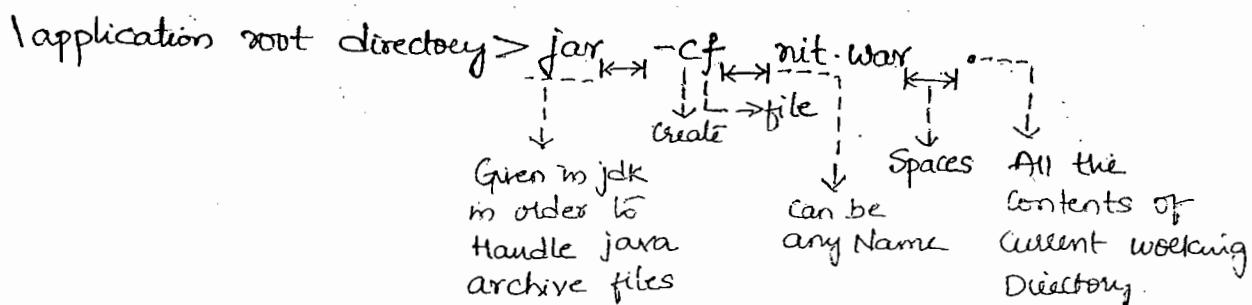
2) Hot Deployment

If we deploy a web application when the Container (Server) is not ^{start} up and Running, it is known as Cold Deployment. It is preferred only while designing & testing the web application.

If we deploy a web application when the Container (Server) is Up and Running, it is known as Hot Deployment. It is generally used in Real time i.e., Industry, because, these Servers in industry are called Production Servers and they cannot be Shutdown as easily as it is done while Cold Deployment. If done so, other users who are

Steps to Perform Hot Deployment on Tomcat Server:

Step-1 :- Create war (web archive) file of the web application only after developing the web application in Routine fashion (i.e. using four Mandatory Steps discussed earlier) and testing it in Cold Deployment fashion (it is done on an independent module of the designer), war file should be created



Now, all the Contents of Root directory (excluding Root directory) are archived i.e. Zipped into one file. After developing the entire Project, it is converted into a 'war' file and is given to the client.

Step-2 :- Start Tomcat Server

Step-3 :- Get the "Tomcat Web Application Manager" which is a Graphical Tool, by specifying the following URL in the browser

`http://localhost:8081/manager/html`

Step-4 :- Using the above tool, deploy the war file by specifying its location graphically

Note :- war file name becomes the application context root.

i.e. `http://localhost:8081/nit/user.html`

This should have been session scope application.

Here, the name of war file is only used in Hot Deployment.

Thus it is beneficial to give "source" in Screen Development. If full address of Root Context is given in place of `lit`, then every screen needs to be modified when war files are to be created.

Consider the following Scenario:

`<INPUT TYPE = "Submit" Value = "add" Name = "s">` on the screen
`<INPUT TYPE = "Submit" Value = "delete" Name = "s">` we get two Buttons.

```

doGet
{
    String sb = request.getParameter("s");
    if( sb.equals(add))
    {
        // do something
    }
    else
    {
        // do some other thing
    }
}

```

Tell now we have been using only one Submit button. When clicked on that button, Control goes to Servlet.

In the above case, two Submit buttons are used. When clicked on one button, Servlet should perform some operation. When clicked on other button Servlet should perform a different operation. This can be managed by an if-else block in Servlet. The difficulty would be to distinguish the control flow i.e. which Submit goes to which block in the Servlet. For this, we use naming conventions i.e. naming the buttons, so that internally they travel with the request as name value pairs. Then, the flow of control in Servlet can be distinguished easily. It is shown above in the Example.

20/08/09

→ What is Session Time Out (or Session Expiry)? How to Specify Session Time out?

After a Session is created for a client, if the client is inactive for more than a specified period of time, Container destroys the Session implicitly. This is known as Session Timeout OR Session Expiry

We can specify Session Time out period to the Container in two ways:

- 1) Programmatically
- 2) Declaratively (in web.xml)

By calling setMaxInactiveInterval(int seconds) method on Session object, time out period is specified programmatically

With the following elements, Session Time Out is Specified Declaratively:-

<session-config>

<session-timeout> n </session-timeout>

</session-config>

Where 'n' is a number in minutes.

Note:- Even if we don't specify the Session Timeout Period, Container by default will have some period.
for Example :- 30 minutes for Tomcat Server

Socket Objects are heavy weight i.e. they occupy more space and consume more Resources. Whereas, Session Objects are light i.e. they consume very few Resources. Even though they are not that resource consuming, when not used, they must be released. Hence, for this purpose session time out concept is used.

Specifying Session Time Out:

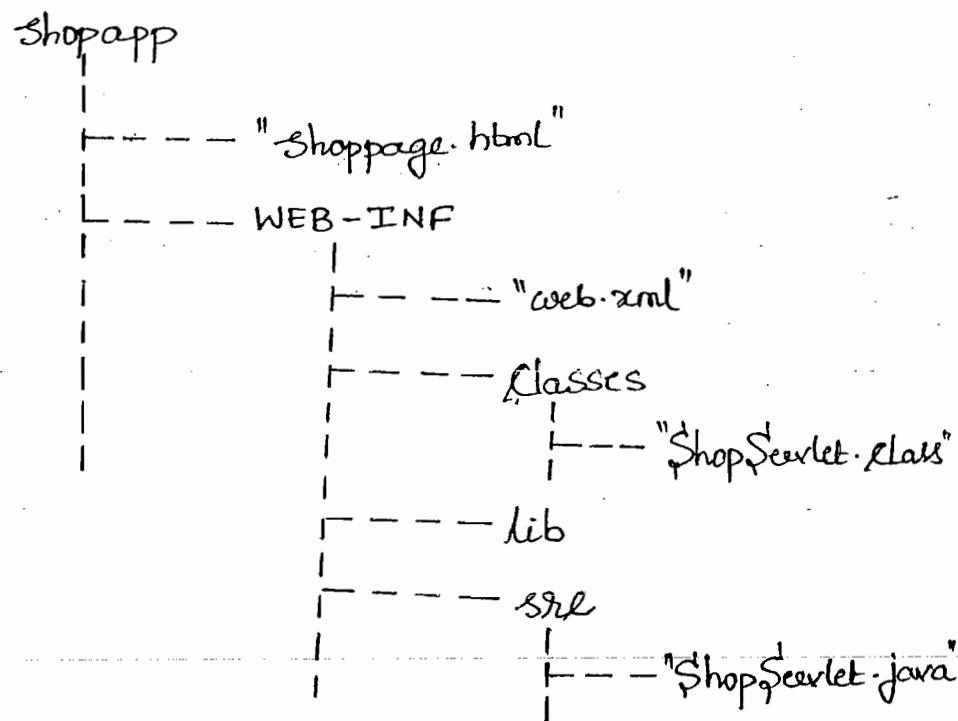
programmatically \Rightarrow Instructions through method calls

declaratively \Rightarrow Instructions through web.xml

Declarative method of Session time out is better since, any future modifications in Session time period can be done easily on web.xml file and hence there would be no need to touch the Source Code, however Source Code is never given to the client. Hence, specifying Programmatically would be troublesome

- Q) Develop & Deploy a web-application that implements an Online Shopping Cart.

Directory Structure :-



URL :-

http://localhost:8081/Shopapp

Shoppage.html :-

<HTML>

```

<BODY BGCOLOR = WHEAT>
<CENTER><H2> Welcome To Shopping Mall </H2>
<FORM ACTION = "./ShopSession" METHOD = "Post">
    SELECT Product Code:
  
```

```

<OPTION VALUE = "101">101
<OPTION VALUE = "102">102
<OPTION VALUE = "103">103
<OPTION VALUE = "104">104
<OPTION VALUE = "105">105

```

</SELECT>

PRODUCT QUANTITY : <INPUT TYPE = "text" NAME = "t1">


```

<INPUT TYPE = "SUBMIT" NAME = "s" VALUE = "ADD ITEM">
<INPUT TYPE = "SUBMIT" NAME = "s" VALUE = "REMOVE ITEM">
<INPUT TYPE = "SUBMIT" NAME = "s" VALUE = "SHOW ITEMS">
<INPUT TYPE = "SUBMIT" NAME = "s" VALUE = "LOGOUT">

```

</FORM> </CENTER>

</BODY>

</HTML>

Say $\Rightarrow \left\{ \begin{array}{l} pCode = 101 \\ t1 = 3 \\ s = \text{ADD ITEM} \end{array} \right.$

At a time, all these
three would travel to
Servlet

web.xml :-

<web-app>

<Servlet>

<Servlet-name> Shopping </Servlet-name>

<Servlet-class> ShopServlet </Servlet-class>

</Servlet>

```

<url-pattern> /shopSession </url-pattern>
</servlet-mapping>

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

</web-app>

```

ShopServlet.java :-

```

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

public class ShopServlet extends HttpServlet
{
    HttpSession session;
    String code, qnty, sb;

    public void doPost(HttpServletRequest req, HttpServletResponse
        res) throws IOException, ServletException
    {
        res.setContentType("text/html");
        PrintWriter pw = res.getWriter();
        session = req.getSession();
        System.out.println("Session ID : " + session.getId());
        System.out.println("Is it A New Session : " + session.isNew());
    }
}

```

Cookies :-

- A cookie is a name-value pair of textual information Exchanged between Http Server and Http Client.
- It is programmatically created at Server side by Server Side programming. It is sent to browser first by web Server along with response Headers. Browser has capability of capturing and resending this cookie back to Server again, along with next request.
- We have two kinds of Cookies
 - 1) Session Cookies.
 - 2) Persistent Cookies.
- A Session cookie is not stored in the file system of the Client Machine. It dies as soon as the Browsing Session Ends.
- A Persistent cookie is stored in the Browser's Machine. Even if the browser is closed, that cookie will not die. Any time later, if the request goes to the same Server from where the cookie is originated, browser is capable of retrieving that cookie from the file system, adding the same to the HTTP request headers and sending to the Server.

→ How do Cookies Serve in a Web Application?

- *) To identify a client (user) uniquely in a series of Client-Server Interactions
- *) To keep track of user preferences
- *) To promote targeted advertisement in Web

(This session ID has non-unique pairs encrypted & it travels invisibly b/w client & server)

A Session ID is exchanged as a cookie, hence because of that, Server is able to recognize the client uniquely. It is exchanged as Session Cookie.

Random Advertisement ⇒ Sending any advertisement randomly to any user.

Targeted Advertisement ⇒ Sending an advertisement specific to some product to its related user.

Eg:- Sending an advertisement of a medical eBook to a Doctor.

This is possible only by using Cookies which may have all the related information of the user.

Steps To Implement Persistent Cookies in a Web Application :-

Step 1 :- Create a cookie by instantiating javax.scarlet.http.Cookie class

Eg:- `Cookie c1 = new Cookie("sni", "sv1");`

```

session.setMaxInactiveInterval(120);
sb = req.getParameter("s");
if (sb.equals("ADD ITEM"))
{
    code = req.getParameter("pCode");
    qty = req.getParameter("q");
    session.setAttribute(code, qty);
    res.sendRedirect("shoppage.html");
}

else if (sb.equals("REMOVE ITEM"))
{
    code = req.getParameter("pCode");
    session.removeAttribute(code);
    res.sendRedirect("shoppage.html");
}

else if (sb.equals("SHOW ITEMS"))
{
    Enumeration e = session.getAttributeNames();
    if (e.hasMoreElements())
    {
        pw.println("<H2><FONT COLOR=BLUE> Your Shopping
Cart Items </FONT></H2>");
        while (e.hasMoreElements())
        {
}
}

```

170.

```
pw.println("<H2>Product Code :" + c);
pw.println(" Quantity is :" + session.getAttribute(c) +
           "</H2>");
} // while
} // if

else
{
    pw.println("<BODY BGCOLOR = CYAN>");
    pw.println("<H2><FONT COLOR = RED> No Items
In Cart </FONT> </H2>");
}
pw.println("<A HREF = shoppage.html> Want To
Shop ! </A>");
}

// outer if

else if (sb.equals("LOGOUT"))
{
    session.invalidate();
    pw.println("<BODY BGCOLOR = CYAN>");
    pw.println("<A HREF = shoppage.html> Want To
Shop Again ! </A>");
}

pw.close();
}

// doGet

}

// class
```

Step 2 :- Making the Cookie Persistent

c.setMaxAge(int seconds);

for Example :-

c.setMaxAge(365 * 24 * 3600);

Step 3 :- Sending the Cookie to the Client

response.addCookie(c);

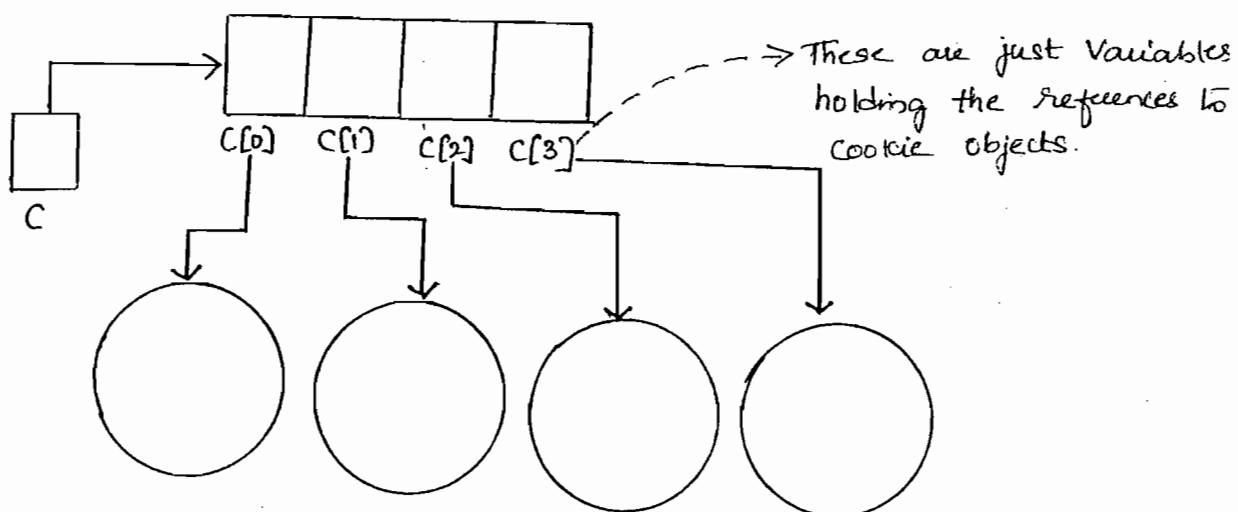
This is just added to response object i.e., response headers. It is sent to browser by web server along with Response.

Step 4 :- Capturing the Cookies from the Client Request

Cookie c[] = request.getCookies();

↓
This is used even though
there is one cookie.

If there are four cookies that are coming from Browser, then



Hence, when there are four cookies, five objects are created

Note:- Cookie class has two important methods to capture name and value

- 1) String getName()
- 2) String getValue()

To print the Name-Value pairs of cookies, the code would be something like :

```
Cookie c[] = request.getCookies();
if (c != null)
{
    for (int i=0; i<c.length; i++)
    {
        System.out.println(c[i].getName());
        System.out.println(c[i].getValue());
    }
}
```

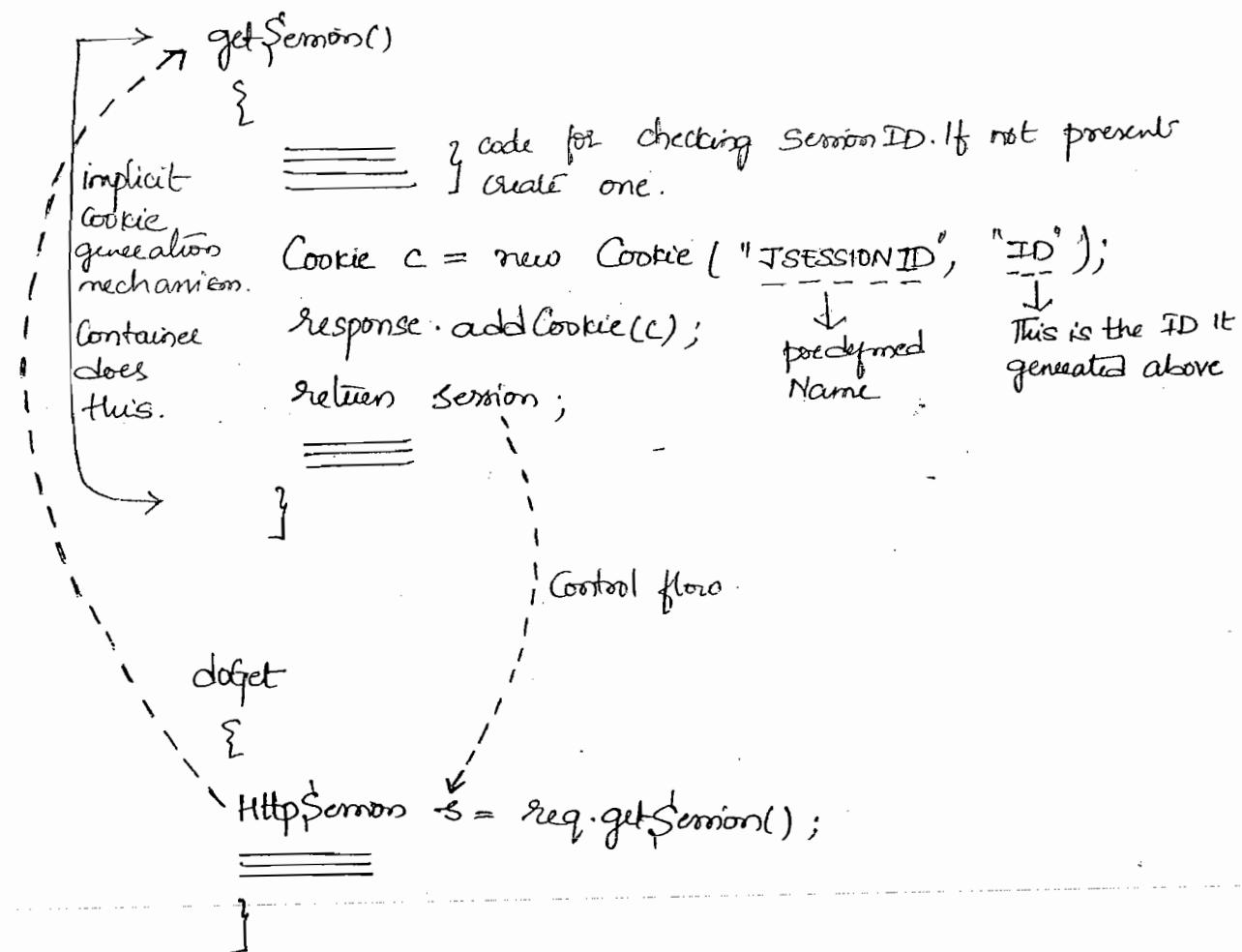
24/08/09

In persistent cookie implementation, if Step-2 is removed, it becomes implementation of Session cookie.

getSession() method internally implements this Session cookie.

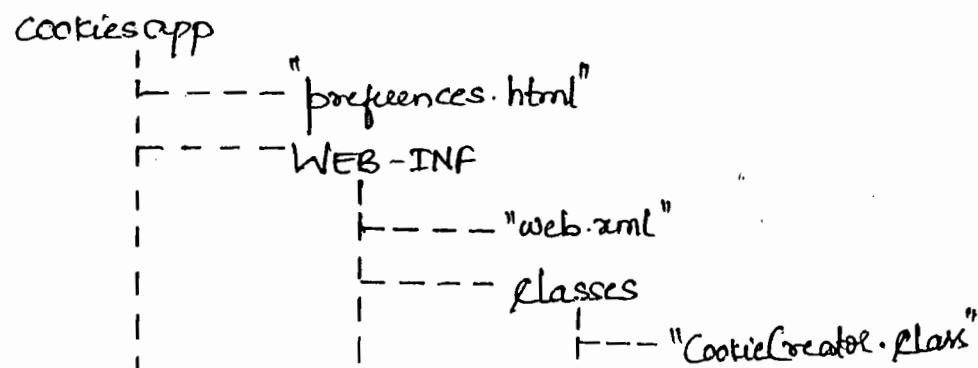
Not Every application i.e website needs cookies to be generated. Those websites who need the user settings and preferences to be stored, only those websites need cookies to be created and that too stored at client side only.

`getSession()` method would be defined in a way as shown:



→ Q) Web Application on Implementation of Persistent Cookies.

Directory Structure :-



URL :-

176

http://localhost:8081/cookiesapp/preferences.html

preferences.html :-

<HTML>

<BODY BGCOLOR = PINK>

<CENTER><H1> Welcome To WWW.Nit.Com </H1>

<FORM ACTION = ". /source">

Select Background Color : <SELECT NAME = "t1">

<OPTION VALUE = "GREEN"> Green

<OPTION VALUE = "RED"> Red

<OPTION VALUE = "YELLOW"> Yellow

</SELECT>

Select The Sport :

→ Radio Button

 <INPUT TYPE = "RADIO" NAME = "t2"

↓ gives one space

VALUE = "CRICKET"> Cricket

 <INPUT TYPE = "RADIO" NAME = "t2"

VALUE = "FOOTBALL"> Football

 <INPUT TYPE = "RADIO" NAME = "t2"

VALUE = "BASKETBALL"> Basketball

<INPUT TYPE = "SUBMIT" VALUE = "SEND">

<FORM> .

</CENTER>

<BODY>

</HTML>

web.xml :-

<web-app>

<Servlet>

<Servlet-name> one </Servlet-name>

<Servlet-class> CookieCreator </Servlet-class>

</Servlet>

<Servlet>

<Servlet-name> two </Servlet-name>

<Servlet-class> CookieReceiver </Servlet-class>

</Servlet>

<Servlet-mapping>

<Servlet-name> one </Servlet-name>

<url-pattern> /source </url-pattern>

</Servlet-mapping>

<Servlet-mapping>

<Servlet-name> two </Servlet-name>

<url-pattern> /target </url-pattern>

</Servlet-mapping>

</web-app>

--> Registration Name is also known
as Private Name or Secret Name
or Alias Name

CookieCreator.java :- // implementation of first 3-steps of Persistent Cookie

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

```

public class CookieCreator extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ScarletException, IOException
    {
        String bctr = request.getParameter("t1");
        String spt = request.getParameter("t2");
        Cookie c1 = new Cookie("bc", bctr);
        c1.setMaxAge(300);
        response.addCookie(c1);

        Cookie c2 = new Cookie("s", spt);
        c2.setMaxAge(300);
        response.addCookie(c2);

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<A HREF = ./target> Get Your Preferences
Here </A>");
        out.close();
    }
}

```

CookieReceiver.java :- // implementation of 4th Step of Persistent Cookie.

```

import javax.scarlet.*;
import javax.scarlet.http.*;

```

```

public class CookieReceiver extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse
        response) throws ServletException, IOException
    {
        Cookie c[] = request.getCookies();
        String color = null;
        String sport = null;
        if (c != null)
        {
            for (int i=0; i < c.length; i++)
            {
                String name = c[i].getName();
                if (name.equals("bc"))
                    color = c[i].getValue();
                else
                    sport = c[i].getValue();
            } // for-loop
        } // if-block
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HTML>");
        out.println("<BODY BGCOLOR = " + color + ">");
        out.println("<MARQUEE> <FONT SIZE = 6>" + sport +
    
```

out.close();

} // doGet;

} // class.

All the 4-Steps of

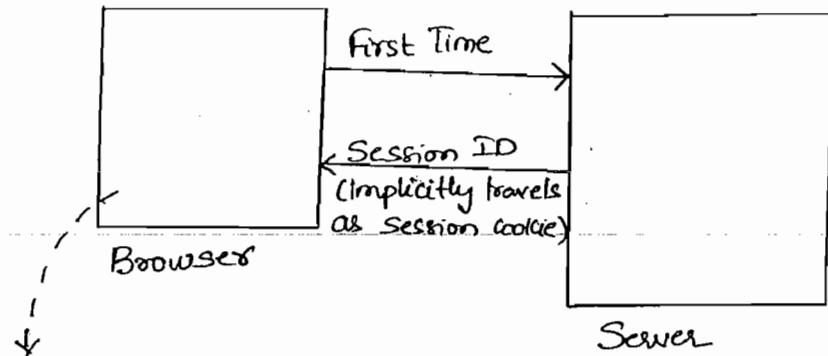
180

persistent Cookie implementation
can be done with in same
Servlet also using If-else
blocks.

URL Rewriting :-

- Appending Session ID to the URL is known as URL Rewriting
- We implement URL Rewriting by calling the following method
 - response.encodeURL(String url);

Need for URL Rewriting :-



Browser has Options to Block cookies from getting created.

This option is used to Block cookies if some sort of Malicious Code is coming from Server. If disabled, Session cookies are also not sent by Browser to Server hence Session ID is not sent. Hence for every request a new Session ID is created and hence Session Tracking fails and website cannot be used properly.

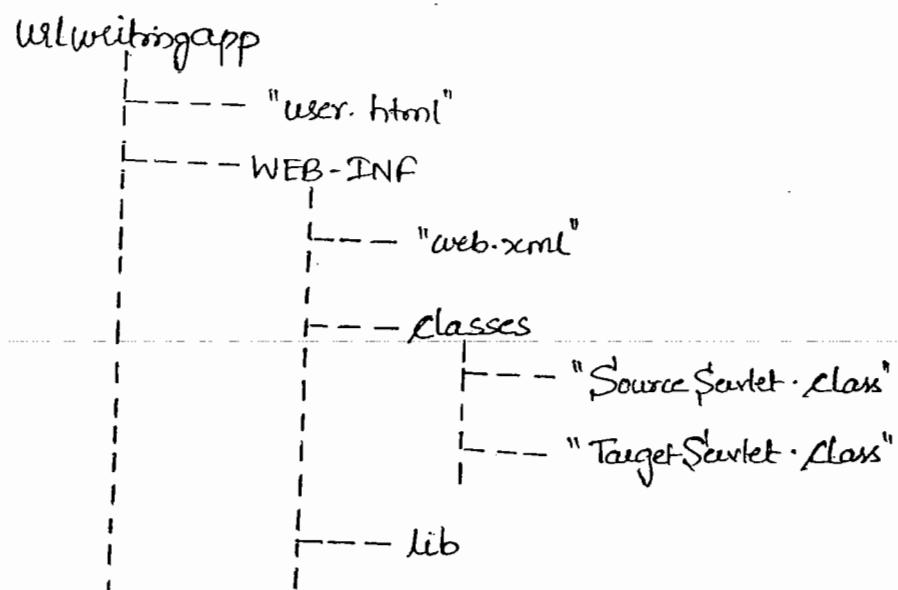
As an alternative to this is, if the cookies are blocked, we go for URL Rewriting. We should always assume that-

must be implemented by calling the method `encodeURL()` on response object.

Note :- If Cookies are disabled in the browser, Session Tracking fails. To overcome this problem, we go for URL Rewriting.

→ Q) Web Application on URL Rewriting.

Directory Structure :-



web.xml :-

<web-app>

<Servlet>

<Servlet-name> one </Servlet-name>

<Servlet-class> SourceServlet </Servlet-class>

```

<Scenlet>
    <Scenlet-name> two </Scenlet-name>
    <Scenlet-plan> TargetScenlet </Scenlet-plan>
</Scenlet>

<Scenlet-mapping>
    <Scenlet-name> one </Scenlet-name>
    <url-pattern> /source </url-pattern>
</Scenlet-mapping>

<Scenlet-mapping>
    <Scenlet-name> two </Scenlet-name>
    <url-pattern> /target </url-pattern>
</Scenlet-mapping>

</web-app>

```

User.html :-

```

<HTML>
<BODY BGCOLOR = PINK>
<CENTER><H1> Welcome To www.Nit.Com </H1><BR>
<FORM ACTION = ". /source">
User Name : < INPUT TYPE = "text" NAME = "t1" ><BR><BR>
<INPUT TYPE = "SUBMIT" VALUE = 'Send' >
</FORM>
</CENTER>
</BODY>

</HTML>

```

→ Give some portions of implementation of GenericServlet.

public abstract class GenericServlet implements Servlet,
ServletConfig, Serializable

{

ServletConfig config; // Variable with class scope

public void init() throws ServletException {}

public void init(ServletConfig c) throws ServletException

{

config = c;

init();

}

public ServletContext getServletContext()

{

ServletContext sc = config.getServletContext();

return sc;

}

=====

]

Hence, internally, config object is only used to get Context object reference, even though we don't mention this 'config' while calling getServletContext() in our Servlet code.

In the above GenericServlet class, 'c' is the reference of object of implementation class of ServletConfig interface that is created by Servlet Engine even before calling the init method. This reference has method scope i.e., localized, hence for Globalizing, it is supplied to the reference 'config' which has class scope. We are just increasing the scope of object from method scope to

that is created by the Container.

188.

Consider :-

```
public class MyServlet extends HttpServlet  
{  
    public void init()  
    {  
        System.out.println("Hello");  
    }  
    public void init( ServletConfig sc )  
    {  
        System.out.println("Hello");  
    }  
}
```

Even though not visible, all the methods of GenericServlet are literally written into MyServlet because of inheritance. Hence, for MyServlet class object, 4 init() methods are available i.e. 2 - coming from parent (HttpServlet, which inherits them from GenericServlet) and 2 - that are defined in MyServlet.

When control comes to this Servlet, Servlet instance is created, ServletConfig object is created, both by Servlet Engine and then the Engine calls parameterized init() method. Always Servlet Engine calls parameterized init() method only. Now, parameterized init() method of MyServlet gets executed since it overrides parameterized init() method of HttpServlet.

In MyServlet, If parameterized init() is removed, then for MyServlet Object 3 init() methods are available. Now non-parameterized init() is only called but

SourceServlet.java :-

```

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

public class SourceServlet extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse response
    throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String name = request.getParameter("t1");
        HttpSession s = request.getSession();
        s.setAttribute("user", name);

        out.println("<HTML>");
        out.println("<BODY BGCOLOR = MAROON>");
        out.println("<A HREF = " + response.encodeURL("./target") +
                   "> Get User Name Here </A>");;
        out.println("</BODY>");

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

        out.close();
    } // doGet
}

```

↖
URL Rewriting
terminology

TargetServlet.java :-

```

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

public class TargetServlet extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse
                      response) throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        HttpSession s = request.getSession();
        String usr = (String) s.getAttribute("user");
        out.println("<HTML>");
        out.println("<BODY BGCOLOR = RED>");
        out.println("<H1> User Name Is: " + usr + "</H1>");
        out.println("</BODY>");
        out.println("</HTML>");
        out.close();
    }
}

} // class

```

parameterized init() method internally calls zero argument init() method. Again there are two init() methods with zero arguments. The zero argument init() method of MyScarlet overrides zero argument init() method of parent's in this case.

→ what happens when the following doget method is execute

```
public class MyScarlet extends HttpServlet
```

```
{
```

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

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

```
        ServletContext sc = getServletContext();
```

```
        System.out.println("Hello");
```

```
}
```

```
}
```

⇒ A NullPointerException is raised.

MyScarlet inherits from HttpServlet, HttpServlet inherits from GenericServlet hence, all the methods of GenericServlet discussed earlier are literally copied into MyScarlet

When MyScarlet object is created, at this point, instance variable 'config' which has class scope is allocated with memory. It is by default initialized with 'null'.

It has no definition.

Now `doGet()` is called, which inturn calls the method `getServletContext()`. The control goes to `getServletContext()` coming from `GenericServlet`. Since the contents of 'config' object are not changed by parameterized `init()` of `GenericServlet` i.e., not changed from 'null' to 'c'-reference of `ServletConfig` object created by Servlet Engine, but parameterized `init` of `MyServlet` is executed as it overrides the parent's parameterized `init()` method. Hence 'config' still holds 'null' value. Calling `getServletContext()` on a null holding object results in `NullPointerException` to be raised.

To overcome this,

Solution 1 :- Never override the Parameterized `init()` method in our Servlet code.

Solution 2 :- If overridden, atleast specify '`super.init(ServletConfig config)`' inside the definition of Parameterized `init()` method of `MyServlet` i.e., our Servlet code. By doing so, control goes to parameterized `init()` of parent's class and it gets executed.

Hence, always, overriding the `Zero argument init()` method is the best practice. Overriding the parameterized `init()` method is useless because, if overridden, the actual reference of `ServletConfig` object - 'c' which would be created by Servlet Engine is lost and hence we cannot provide the initialization info and context info to the Servlets, which is only possible through the actual `ServletConfig` Object. Hence we need not declare and define an

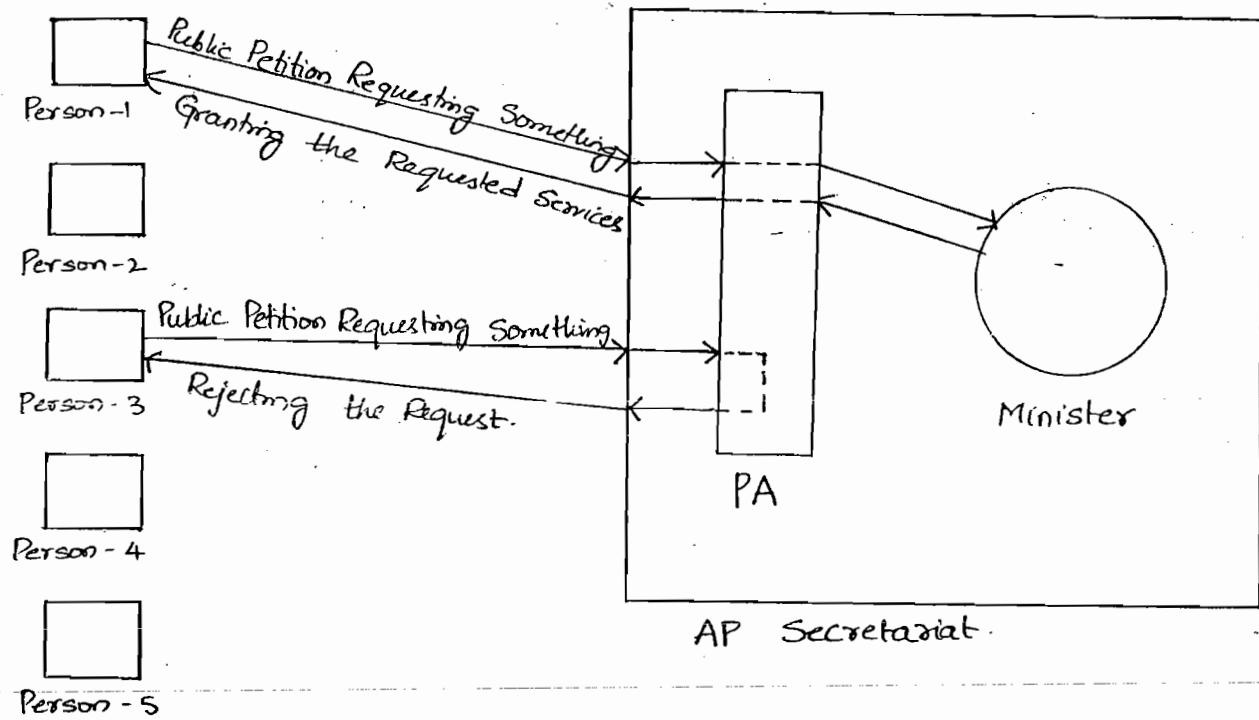
In case an init() method is required in Project Development so as to perform certain operations such as Database Connectivity, creating Statement or Prepared Statement Object, then override the Zeer argument init() method. By doing so, Seerlet Engine calls parameterized init() method inherited from GenericSeerlet, the actual reference of SeerletConfig object 'c' is assigned to a Global Variable with class scope and then, this parameterized init() method calls Zeer argument init() method. Since there are two Zeer argument init() methods one in our Seerlet code and one inherited from GenericSeerlet, the Zeer argument init() method of child class that is our seerlet code is only executed. Hence, database related operations i.e. resources allocation is done apart from not losing the actual SeerletConfig object's reference that was created by Seerlet Engine even before the init() method is called.

→ What are the methods of Seerlet interface?

- 1) init(SeerletConfig)
- 2) Service(SeerletRequest, SeerletResponse)
- 3) destroy()
- 4) SeerletConfig getSeerletConfig()
- 5) String getSeerletInfo()

Filters

Consider the following Scenario:



Say Person-1 files a Petition for Requesting Something, to the minister. The Request first comes to Secretariat, it then moves to PA of the Minister but not directly forwarded to minister. Here, the PA checks whether the request is legitimate, proper and is fulfilling all other requirements. Then only, this request from that Person is forwarded to the Minister. The request is processed by Minister only and the corresponding response is handed over to PA only. This PA makes sure that the response is also proper and then is sent back to the Person-1.

→ What are the methods of `ServletConfig` Interface?

There are 4 Abstract methods in `ServletConfig` Interface:

1) `getInitParameters`

2) `getServletContext`

3) `getServletName()`: This method returns the Registration Name of Servlet

4) `getInitParameterNames()`: This method returns an Enumeration of init param Names as String Objects

Container has implementation class for `ServletConfig` interface, hen object of this child class is created. This child implementation class has definitions of all the above mentioned methods which are Abstract in `ServletConfig` interface

→ How many interfaces does `GenericServlet` implement? why?

`GenericServlet` implements 3 interfaces:

1) `Servlet`: It has 5 methods, 3-lifecycle methods, 2-Nonlifecycle methods.

2) `Serializable`: It is null interface i.e, it has no methods.

3) `ServletConfig`: It has 4 methods.

To get Eligibility to act as a servlet and thereby making its sub classes also servlets, `GenericServlet` implements `Servlet`

All `Servlet` instances can be made Persistent, as `GenericServlet` implements `java.io.Serializable` interface. If any class implements `Serializable` interface, the objects of that class can be stored in the file system or can be transported in Network. Hence, it can make a `Servlet` instance persistent. Therefore, to store any object in secondary memory rather than in primary memory, the class needs to implement `Serializable` Interface.

Without the need of `ServletConfig` object directly, `ServletConfig` methods can be called on the `Servlet` instance as `GenericServlet` implements `ServletConfig` interface.

Consider:

```

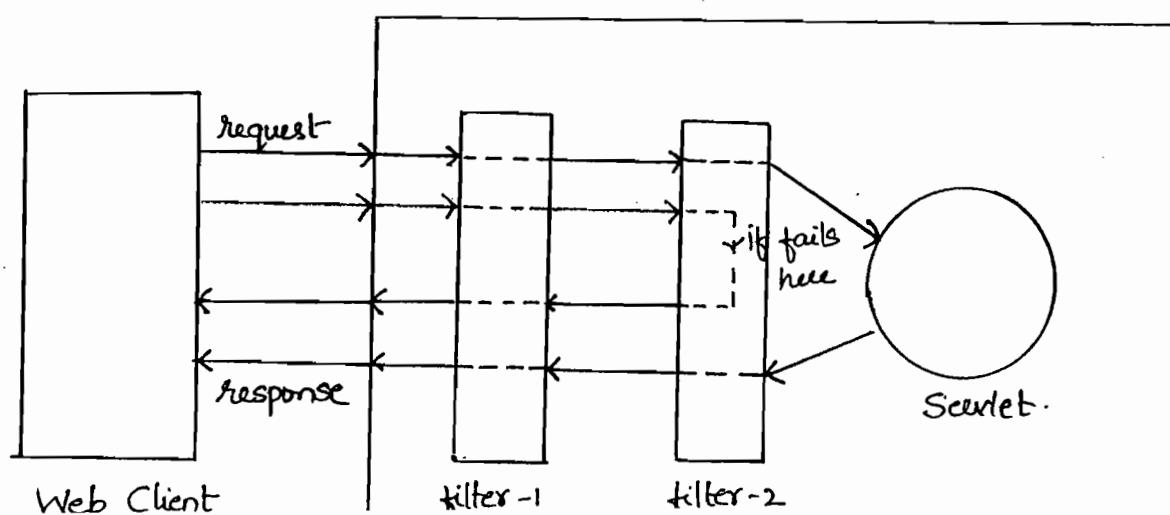
MyServlet
{
    doGet() ---> Calling method
    {
        ServletContext sc = getServletContext();
    }
}
                                ↓
                                called method.
    }
```

Both the methods `doGet()` & `getServletContext()` belong to same object i.e. `MyServlet` class object since no reference is used. `MyServlet` inherits from `HttpServlet`. `HttpServlet` inherits from `GenericServlet` which implements `ServletConfig` interface. Hence, the 4 methods of `ServletConfig` interface are directly inherited to `MyServlet` i.e., they are virtually written inside `MyServlet`. Therefore there is no need to use `Config` object reference to call these methods. If `GenericServlet` does not implement `ServletConfig` interface, then the 4 methods of `ServletConfig` need to be called using reference.

found illegitimate, this request never reaches the Minister and PA Sends back the respective response to that Person

In this analogy, PA is Filter, Secretariat is Container, Minister is Servlet and People are Browsers (client processes)

- A filter is a Container Managed Java class whose instance intercepts the calls of the web components (servlets/jsp) in request-response cycle.
- If a filter is applied to a Servlet, every client request that addresses the Servlet will be passed through filter. Similarly, every response.
- Filters are meant either for enriching the Servlets OR Monitoring the Servlets.
- If we apply more than one filter to a Servlet, it is known as filter Chaining.



Filter API :-

→ We have 3-interfaces in filter API

- 1) Filter
- 2) FilterConfig
- 3) FilterChain

→ Filter interface provides life cycle methods to our filter classes

- 1) init(FilterConfig)
- 2) doFilter(ServletRequest, ServletResponse, FilterChain)
- 3) destroy()

→ FilterConfig interface has the following methods

- 1) getInitParameter
- 2) getInitParameterNames
- 3) getServletContext
- 4) getFilterName() :- It returns the registration name of the filter

→ FilterChain interface has only one method

doFilter(ServletRequest, ServletResponse)

This method is used to switch the control from one filter to other. If it is the only filter or the last filter in the chain, this method switches the control to the servlet/jsp.

Skeleton Structure of one filter:

```

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

public class MyFilter implements Filter
{
    public void init(FilterConfig config) throws ServletException
    {
        // resources allocation code
    }

    public void doFilter(ServletRequest request, ServletResponse response,
                         FilterChain chain) throws ServletException, IOException
    {
        // actual duty of the filter is implemented here
    }

    public void destroy()
    {
        // resources deallocation code
    }
}

// filter class

```

Serlet Interface Contribution:-

- 1) It makes any class implementing it, act as Serlet.
- 2) It provides life cycle methods for Serlet.

act as a filter

196.

2) It provides life cycle methods for filter

dofilter() of filter Interface

- * It is meant for performing filtering or monitoring operation only
- * It has three arguments
- * It is a lifecycle method which is called by Container

dofilter() of FilterChain Interface

- * It is meant for switching the control only
- * It has two arguments
- * It is method called by programme through object

Filter Life cycle

Filter life cycle is similar to that of a Servlet

→ As soon as the web application is deployed, all the filters of the application are instantiated and initialized by Container (only once)

As soon as web application is deployed, `ServletContext` object is created and then every filter in `web.xml` is identified and their classes are loaded, their instances created, for every filter a `FilterConfig` object is created.

Hence, as soon as the application is deployed, for each and every filter, the initialization & instantiation are done with even before the request comes from the Client.

- When the application is undeployed, filter instances are destroyed. Just before the garbage collection of the filter instance, Container calls the destroy method on the filter instance.
- Whenever client request comes for the Servlet to which the filter is applied, container calls doFilter method of the filter instance.

How many times client requests come for the Servlet, those many times doFilter method is called. doFilter method is called either for equal number of times or for more number of times than service method of Servlet.

→ How to apply a filter to a Servlet?

*.) Register the filter

*) Map the filter to the Specified Servlet in web.xml

For Registration we say

<filter>

<filter-name> SomeName </filter-name>

<filter-class> OurFilter </filter-class>

</filter>

For Mapping we say

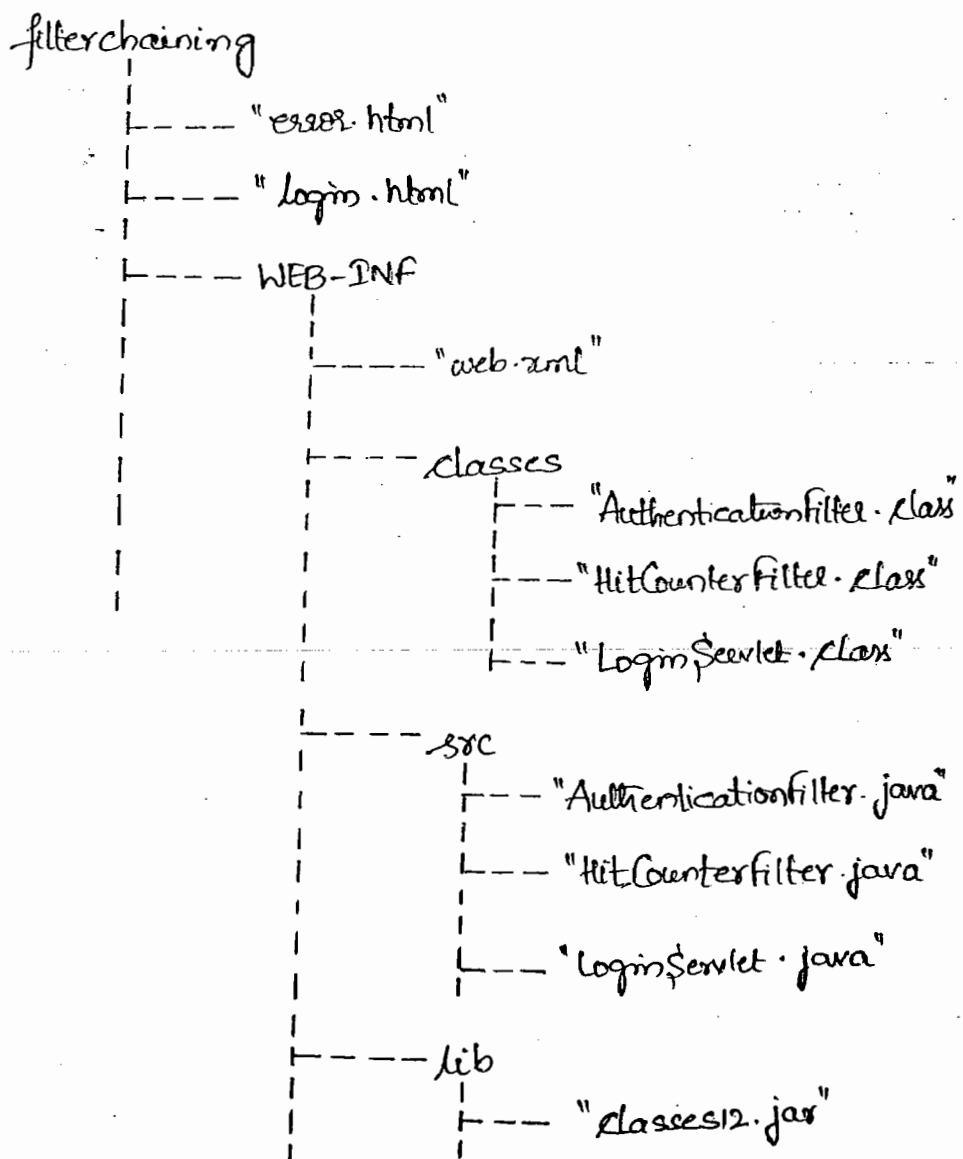
<filter-mapping>

<filter-name> SomeName </filter-name>

<url-pattern> /servlet/OurFilter </url-pattern>

- Q) Develop and Deploy a web Application in which, filter chaining is implemented.

Directory Structure :-



URL :- <http://localhost:8081/filterchaining/login.html>

In this Example, Authenticationfilter is to assist the Servlet,

HitCounterfilter \Rightarrow Counts how many times the Servlet is used

Authenticationfilter \Rightarrow Helps the Servlet by checking whether Username and Password matches or not. If matches, control is passed to HitCounterfilter otherwise sends a error message to the Screen.

This filter also proves useful if multiple Servlets need authentication. So, the duplication of authentication code can be eliminated by applying this filter to all those Servlets.

web.xml :-

The Order of Tags i.e., filter, Servlet - registration, mapping is decided by Sun Microsystems according XML. It has to be done in the order only.

The order of filter Registration & Mapping is important. The order in which they are written in web.xml, the control moves in the order among the filters

<web-app>

<filter>

<filter-name> authenticate </filter-name>

<filter-class> AuthenticationFilter </filter-class>

<init-param>

<param-name> driver </param-name>

<param-value> oracle.jdbc.driver.OracleDriver </param-value>

<init-param>

<param-name> url </param-name>

<param-value> jdbc:oracle:thin:@localhost:1521:orcl </param-value>

</init-param>

<init-param>

<param-name> user </param-name>

<param-value> scott </param-value>

</init-param>

<init-param>

<param-name> pass </param-name>

<param-value> tiger </param-value>

</init-param>

</filter> ← one filter registration

<filter>

<filter-name> hitcount </filter-name>

<filter-class> HitCounterFilter </filter-class>

</filter>

<filter-mapping>

<filter-name> authenticate </filter-name>

<url-pattern> /init </url-pattern>

</filter-mapping>

<filter-mapping>

<filter-name> hitcount </filter-name>

<url-pattern> /init </url-pattern>

```

<scroller>
    <scroller-name> login </scroller-name>
    <scroller-class> LoginScroller </scroller-class>
</scroller>

<scroller-mapping>
    <scroller-name> login </scroller-name>
    <url-pattern> /init </url-pattern>
</scroller-mapping>

</web-app>

```

At the time of Scroller Registration, whatever init parameters are encountered by the Container, are all stored in ScrollerConfig object.

At the time of filter Registration, whatever init parameters are encountered by the Container, are all stored in FilterConfig object.

login.html :-

```

<HTML>
    <BODY BGCOLOR = "CYAN">
        <CENTER> <H1> Login Screen </H1>
        <FORM ACTION = "./init" METHOD = "POST">
            Username: <INPUT TYPE = TEXT NAME = "user"><BR><BR>
            Password: <INPUT TYPE = PASSWORD NAME = "password"><BR><BR>
            <INPUT TYPE = SUBMIT VALUE = "Login">
        </FORM>
    </CENTER>
</BODY>

```

error.html :-

```

<HTML>
  <BODY BGCOLOR = CYAN>
    <CENTER> <H2> Invalid Login OR Password </H2>
    <FORM ACTION = ". / init" METHOD = "POST">
      Username: <INPUT TYPE = "text" NAME = "user"> <BR> <BR>
      Password: <INPUT TYPE = "password" NAME = "password"> <BR> <BR>
      <INPUT TYPE = "Submit" VALUE = "Login">
    </FORM>
  </CENTER>
</BODY>
</HTML>

```

LoginScirlet - java :-

```

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

public class LoginScirlet extends HttpServlet
{
  public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
  {
    response.setContentType("text/html");
    PrintWriter pw = response.getWriter();
    pw.println("<BODY BGCOLOR = RED> Welcome To Our Website");
    pw.close();
  }
}

```

HitCounterFilter.java :-

```

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

public class HitCounterFilter implements Filter {
    ServletContext sc;
    int Count;

    public void init(FilterConfig f) throws ServletException {
        System.out.println(f.getFilterName() + " Filter Initialized");
        sc = f.getServletContext();
    } // init

    public void destroy() {}

    public void doFilter(ServletRequest request, ServletResponse response
        FilterChain chain) throws ServletException, IOException {
        chain.doFilter(request, response);
        Count++;
        sc.log("Number of Times Request Came to LoginServlet is "
            + count);
    } // doFilter
} // HCF Class

```

AuthenticationFilter.java :-

204.

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

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

```
import java.io.*;
```

```
import java.sql.*;
```

```
public class AuthenticationFilter implements Filter
```

```
{
```

```
Connection con;
```

```
ServletContext se;
```

```
public void init(FilterConfig f) throws ServletException
```

```
{
```

```
System.out.println(f.getFilterName() + " Filter Initialized");
```

```
String d = f.getInitParameter("driver");
```

```
String url = f.getInitParameter("url");
```

```
String usr = f.getInitParameter("user");
```

```
String pwd = f.getInitParameter("pass");
```

```
try
```

```
{
```

```
Class.forName(d);
```

```
con = DriverManager.getConnection(url, usr, pwd);
```

```
System.out.println("Connection Established");
```

```
} // try
```

```
catch (Exception e)
```

```
{ e.printStackTrace(); }
```

```
er = f.getServletConfig().getInitParameter("errorPage");
```

```

public void doFilter(ServletRequest request, ServletResponse response,
    FilterChain chain) throws ServletException, IOException
{
    Statement st = null;
    ResultSet rs = null;

    String user = request.getParameter("user");
    String pword = request.getParameter("password");

    try
    {
        st = con.createStatement();
        String sql = "SELECT * FROM OURUSERS WHERE
                    UPPER(MYUSER) = UPPER('" + user + "') AND
                    PASSWORD = '" + pword + "'";

        rs = st.executeQuery(sql);

        if(rs.next())
            chain.doFilter(request, response);
        else
            sc.getRequestDispatcher("/error.html").forward(request,
                response);
    }
    // instead of getRequestDispatcher(), sendRedirect() can also be used
    // //try
    catch(Exception e)
    {
        e.printStackTrace();
    }
}

```

```

try
{
    if(rs!=null) rs.close();
    if(st!=null) st.close();
}
catch(Exception e)
{
    e.printStackTrace();
}

// finally
}

// dofilter()

```

```

public void destroy()
{
try
{
    if(con!=null) con.close();
}
catch(Exception e)
{
    e.printStackTrace();
}

// destroy()
}

// AF class.

```

28/08/09

Performance filters can also be designed in the same way.
 If a performance filter is to be developed i.e. to calculate
 the time taken by Scarlet to process each client

Class PerformanceFilter implements Filter

{ dofilter()

{

long t1 = System.currentTimeMillis(); --> This returns current
System time in milliseconds

chain.doFilter(req, res);

long t2 = System.currentTimeMillis();

long t = t2 - t1; --> This gives time consumed.

}

Event Handling in Web Applications :-

Generalized Examples for Event Handling:

event	Power	Lecture
source	Switch	Faculty
listener	Fan	Student
registration	Switch to Fan Connection Switch Connection(fan)	Payment of Fee
event handler	rotation(Power P)	To understand(Lecture L)

Defining appropriate listener class(es), implementing corresponding event handlers, writing task performing code in those event handlers and registering the listener(s) with the web application put together is known as event handling in web applications.

Event Handling is also done in AWT, SWING and XML but-

Steps to Implement Event Handling in Web Applications

Step 1:- Defining our own listener class that implements the xxxListener interface and implementing its methods

Step 2:- Registering the listener class with the web Application declaratively (i.e, in web.xml file)

i.e, <listener>

```
<listener-class> MyListener </listener-class>
</listener>
```

XXX Listener Interface's Contribution is to

↓
depends on
what type
of event is
Raised.

⇒ Make a class implementing it act as a
listener

⇒ To provide methods for Event handling

29/08/09

An Exception is also an event. Hence it is handled using Exception Handlers.

Creating a ServletContext object as soon as the application is deployed ⇒ an event is raised.

Destorying ServletContext object as soon as the application is Undeployed ⇒ an event is raised.

In a web application, Everything is an event

etc. For each of these events, there is a corresponding listener interface. For example, `ServletContext` related events can be handled using `ServletContextListener`, `HttpSession` related events can be handled using `HttpSessionListener` etc.

The best way to know the methods present in a particular listener interface is to define our own listener class that implements that listener interface. Then compile the class without any definition. Compile raises errors saying that a particular method is not defined, since they are abstract in the parent interface. Hence by looking at those errors we can know the method names and they can be defined in our own listener with some functionality required to perform a task.

For example, a listener can be designed to keep track of number of users logged-in to a particular website. This is possible if we can handle the events raised during session object creation.

→ Q) Develop a listener class that keeps track of number of active sessions

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

```
public class MyListener implements HttpSessionListener
{
```

```
    int count;
```

```

        count++;
    } // event handler -1

    public void sessionDestroyed(HttpSessionEvent e)
    {
        count--;
    } // event handler -2

} // MyListener class

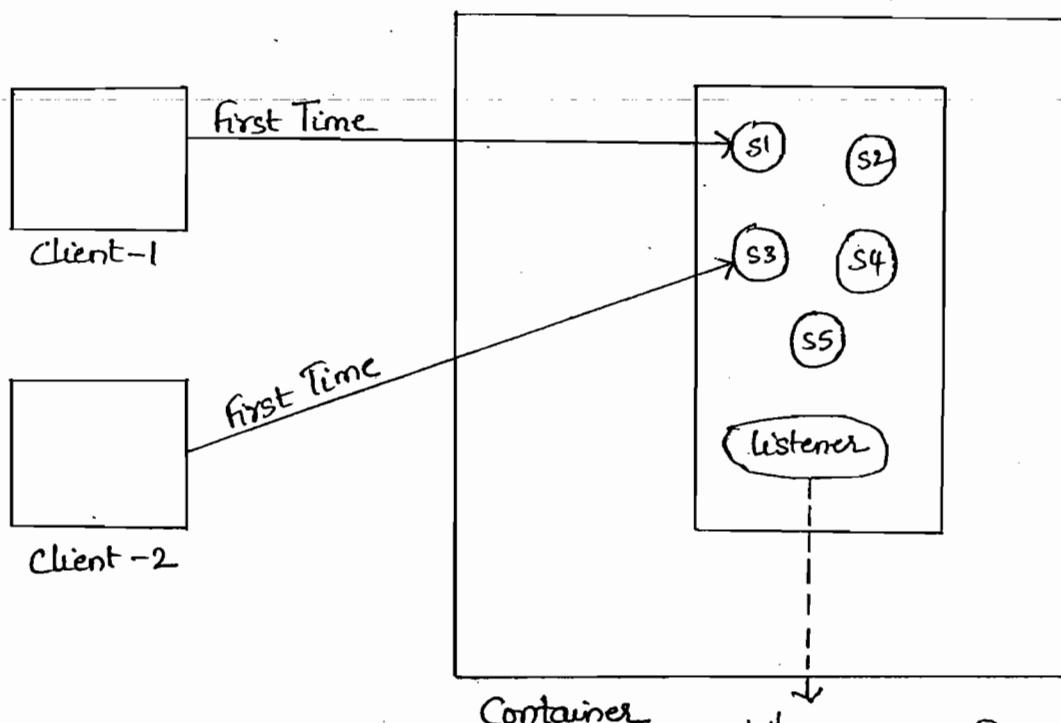
```

Note :- We need to register this Listener as follows :

<listener>

<listener-class> MyListener </listener-class>

</listener>



Whenever a Request Comes for the first time, a new Session object is created, then MyListener's SessionCreated() method is called and whatever code is written inside it gets executed. Similarly whenever a session object is

and whatever code is written inside the method, gets Executed. Hence, from the above 'MyListener' Class we can keep track of number of active users for a website at any given time.

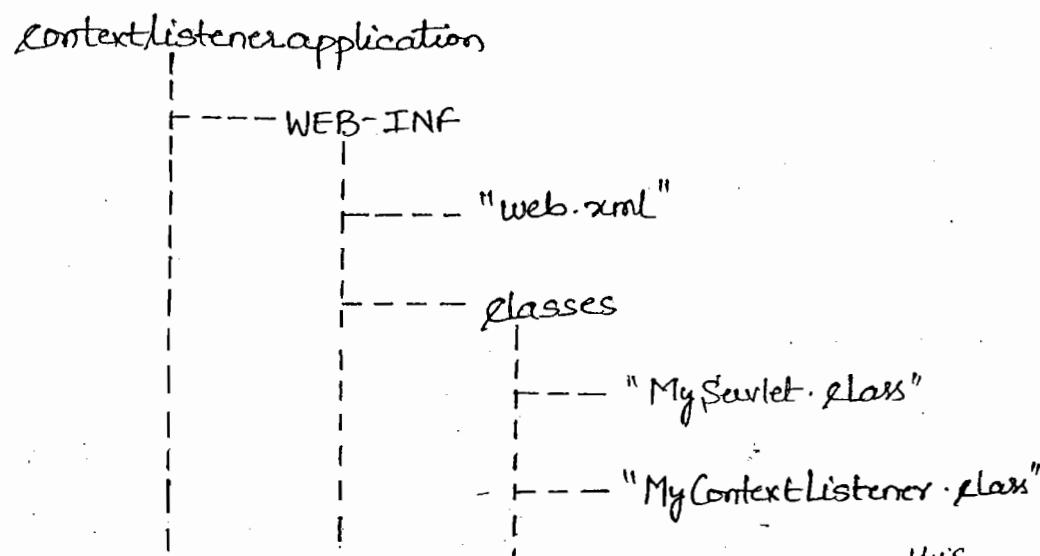
→ Q) Web Application on `ServletContextListener`.

Consider a Scenario, Where in an application, there are 10-servlet and all of them need Database Connectivity. The Connection should be established as soon as the Application is Deployed.

Then, without hardcoding database details, we can provide them to all the Servlets using the concept of init-param. But in each Servlet Registration section, 4-pairs of init-param are to be declared in web.xml. This would not satisfy our need i.e, if Connection is to be Established at the time of Application Deployment. In order to perform such a task declaring the context-param would be better, because as soon as the application is deployed the first object to be created is the `ServletContext` object. After creating the context object the Container encapsulates the context-param into this object.

Creating `ServletContext` object implies an event is raised and hence a Listener can be designed to handle this event. The connection can be established within the methods of the Listener. These methods override the Abstract methods of `ServletContextListener`. Hence, database Connectivity can be achieved as soon as the application is deployed into the Container. This Example i.e. Web Application on

Directory Structure :-



URL :-

`http://localhost:8081/contextlistenerapplication/listen`

web.xml :-

`<web-app>`

`<listener>`

`<listener-class>MyContextListener</listener-class>`

`</listener>`

`<context-param>`

`<param-name>p1</param-name>`

`<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>`

`</context-param>`

`<context-param>`

`<param-name>p2</param-name>`

`<param-value>the other vendor driver value`

As soon as this application is deployed, Container reads the xml file. It checks if there are any context parameters. If found, they are encapsulated in `ServletConfig` object which was created at the time of deployment.

```

<context-param>
    <param-name>p3</param-name>
    <param-value>scott</param-value>
</context-param>
<context-param>
    <param-name>p4</param-name>
    <param-value>tiger</param-value>
</context-param>

<Servlet>
    <Servlet-name> Share </Servlet-name>
    <Servlet-class> MyServlet </Servlet-class>
</Servlet>

<Servlet-mapping>
    <Servlet-name> Share </Servlet-name>
    <url-pattern> /listens </url-pattern>
</Servlet-mapping>

</web-app>

```

ServletContext object is created implies an Event is raised. It is handled inside the listener

MyContextListener.java :-

```

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

```

```

public class MyContextListener implements ServletContextListener
{

```

Connection con;

) indirectly contextInitialized() method is called.

```

ServletContext sc = e.getServletContext();
String driver = sc.getInitParameter("p1");
String cs = sc.getInitParameter("p2");
String user = sc.getInitParameter("p3");
String pword = sc.getInitParameter("p4");

try {
    Class.forName(driver);
    Con = DriverManager.getConnection(cs, user, pword);
    System.out.println("Connected");
    sc.setAttribute("cn", con);
} // try
catch (Exception eo) {
    System.out.println(eo);
} // contextInitialized()

public void contextDestroyed(ServletContextEvent e) {
    try {
        con.close();
        System.out.println("connection closed");
    }
    catch (Exception es) {}
} // contextDestroyed()

} // MyContentListener

```


 The code is executed up to this point after the deployment of application is done i.e even before the request comes.

During translation phase, Java expression is placed in Service method of the container generated servlet

Scriptlet :- A JSP scriptlet looks as follows

```
<%  
    // java code  
%>
```

During translation phase, the above Java code will be placed in Service method.

Expression & Scriptlet are similar in certain respects:

- *) Both are used for Embedding java code directly
- *) Both are kept in service method
- *) Both are Executed as many number of times the request comes from the client

They are different in certain respects :

- *) Expression has only one line of code whereas Scriptlet has many lines
- *) Expression prints the output i.e, the value it returns is printed on browser. Whereas in Scriptlet the output is not printed unless it is mentioned

02/09/09

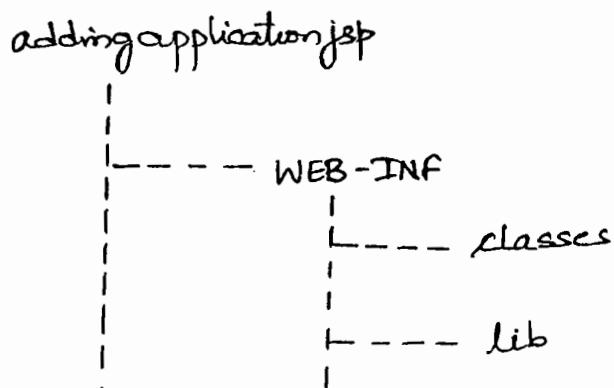
→ What are JSP implicit objects?

implicitly available in jsp's and are known as implicit objects

<u>Object Name</u>	<u>Type</u>
request	HttpServletRequest
response	HttpServletResponse
session	HttpSession
application	ServletContext
config	ServletConfig
page	java.lang.Object
out	JspWriter
pageContext	PageContext
exception	Throwable

- Q) Develop, deploy and use a web Application in which, End User should be able to enter two numbers into web form and get the sum of those two numbers. A jsp has to provide the interactive capability to the web Application

Step -1 : Directory Structure Creation



MySeerlet.java :-

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

when the request
comes from client

```
public class MySeerlet extends HttpServlet
```

{

```
    public void doGet( HttpServletRequest request, HttpServletResponse response
        throws IOException, SeerletException
    {
```

```
        SeerletContext sc = getSeerletContext();
```

```
        Connection con = (Connection) sc.getAttribute("en");
```

```
        System.out.println(" Connection is Retrieved");
```

}

}

If the application is
undeployed, contextDestroyed()
method of MyContextListener is called
and code inside it is executed

→ Explain the following Statement of a Seerlet

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

↓
MIME Type

In the above statement, we are specifying the MIME Type as "text/html".

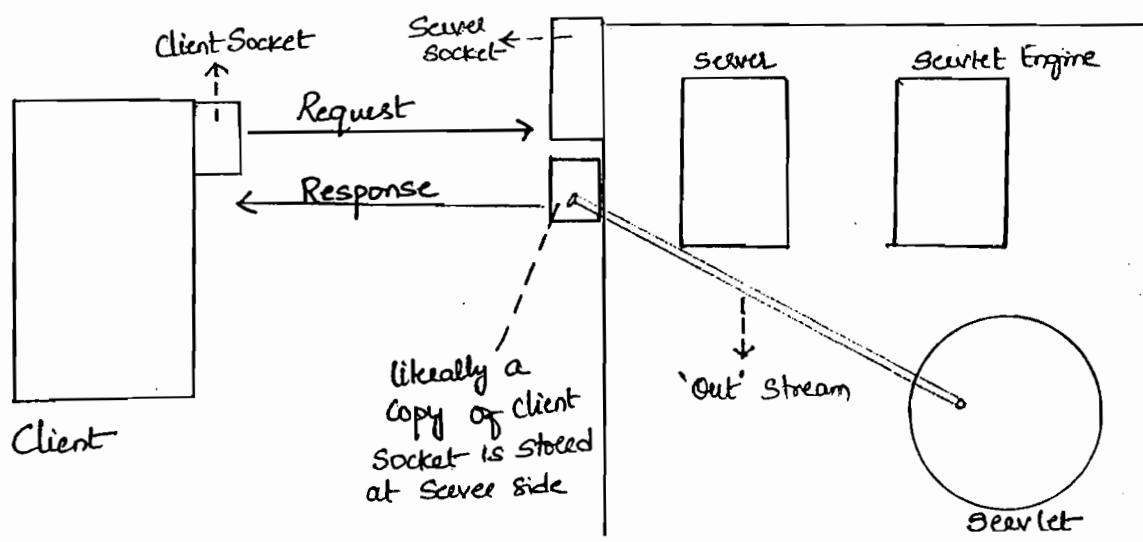
The request from Browser to Server can be for any file such as MS Word Document (or) PDF file (or) Image file etc. MIME Type \Rightarrow Server's Responsibility to indicate the browser which type of file it is delivering to it.

"text/html", "text/plain", "image/gif" etc are some MIME Types. There are around 250 different MIME Types.

Purpose of `PrintWriter out = response.getWriter();`

↓
A Stream

'out' is a stream which acts as pipe for transmission of data. Once the data is given to Client socket at Server side, the duty of Servlet and Servlet Engine are over. This data, which is the output is delivered to browser by web server from Client socket at Server side. Once response is given to browser, this Client socket at Server side is destroyed but Session object remains intact till the Session is ended implicitly or explicitly.



Java Server Pages (JSP)

- JSP is one of the two Web Technologies from Sun Microsystem
- JSP is a J2EE (JEE) technology.
- JSP is a Specification for Web Container Manufacturers.
- JSP API is used by web Application Developers to develop interactive Web Applications.

JSP (in capital letters) implies whole Technology

jsp (in small letters) implies one java server page, developed by the programmer.

- A jsp is a server side piece of code that enhances the functionality of the web Server.
- A jsp is a dynamic web resource.
- A jsp is a web Component.
- A jsp produces dynamic web Content i.e, a dynamic web page
- A jsp contains HTML Code (template text) and Java code (directly / indirectly)
in Servlets, HTML code is written inside java methods but
in jsp, java code is written inside HTML tags.

Limitations of Servlets :-

- Servlets are excellent in request handling but very poor in presentation.
- Major problem with Servlets is tight coupling of Context generation code and presentation code. Such tight coupling leads to following problems
 - 1) parallel development is not possible.
 - 2) Code generation tools cannot be used effectively and hence development time increases and hence cost effectiveness is lost.
 - 3) Development, Administrative and Maintenance problems occur.

Advantages of JSP :-

- Almost all the problems of Servlets are addressed by JSPs.

Note :- JSP is not a replacement for Servlets

01/09/09

Key Points about a JSP :-

- A JSP is a pure text file with any name but with ".jsp" Extension.

- A JSP is nothing but a simple text file with some special tags.

- A jsp on its own cannot process a client request. It will be translated into a Servlet by JSP engine. That Container generated Servlet serves the client request.
- Developing a jsp is nothing but another style of developing a Servlet. Instead of we writing a Servlet, we instruct the Container to write a Servlet for us.

Then why we need a jsp if it is internally converted into a Servlet?

— Because Servlet is good at handling a request, it is (i.e., a jsp) internally converted into a Servlet.

Then why dont we use only Servlet in our application instead of jsp?

- A Servlet is tightly coupled hence some drawbacks are unavoidable
- Using only Servlet interactive Dynamic pages cannot be developed with ease.

Therefore we just use the Servlet where it is advantageous and where a Servlet lacks, there the advantage of a jsp is used (i.e. a jsp is loosely coupled, it is good at presenting output these are the advantages of a jsp)

A jsp is not a program, it is a page and hence its developer is not called as a programmer but is known as jsp writer or jsp author or jsp developer

jsp life cycle :-

→ Life cycle methods are:

- 1) jspInit()
- 2) jspDestroy()
- 3) - jspService(request, response)

→ Life cycle Phases are:

- 1) Translation Phase
- 2) Compilation Phase
- 3) Instantiation Phase
- 4) Initialization Phase
- 5) Servicing Phase
- 6) Destruction Phase

Translation Phase :- When first Client request comes for the JSP, JSP Engine makes full read of the JSP and translates it into Servlet Source code.

Compilation Phase :- Servlet Source Code is compiled to "class" file.

Note :- Remaining four phases are similar to that of a normal Servlet.

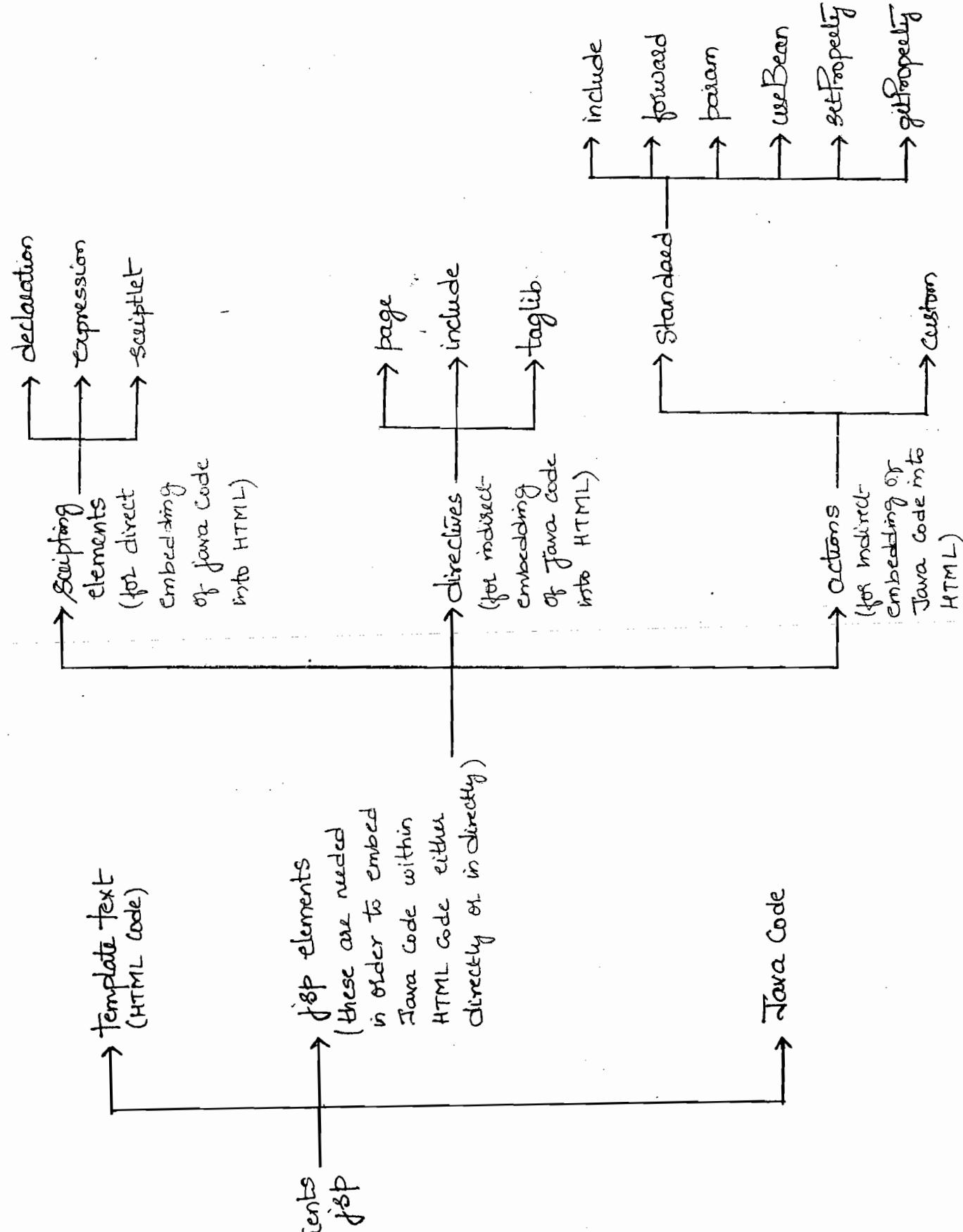
Tabular Representation of JSP life cycle :-

Case	Request Number	Translation Phase	Compilation Phase	Instantiation Phase	Initialization Phase	Servicing Phase
JSP developed for first time and deployed	R1	✓	✓	✓	✓	✓
	R2 - Rn	✗	✗	✗	✗	✓
Container Restarted (with same JSP code)	R1	✗	✗	✓	✓	✓
	R2 - Rn	✗	✗	✗	✗	✓
JSP modified (due to some ill-logic, JSP needed to be modified)	R1	✓	✓	✓	✓	✓
	R2 - Rn	✗	✗	✗	✗	✓

This is as good

→ What are the Constituents of a JSP?

22



Scripting Elements :-

- JSP Scripting elements are those using which, Java code is directly embedded into the (html portion of) jsp.
- Scripting elements are of three types :
 - 1) declaration
 - 2) Expression
 - 3) Scriptlet

Declaration : A JSP declaration is as follows

```
<%!
    // class scope variables declaration
    // method definitions
%>
```

During translation phase, Java methods and Variables of JSP declaration are placed in Container generated Servlet at class scope i.e as instance variables and instance methods i.e, as direct data members and member functions

Expression : A JSP expression is as follows

```
<% = only one Java Expression %>
```

for Example :-

```
<% = a+b %>
```

```
<% = request.getParameter("t1") %>
```

Step-2 : Web Resources Development

In this application, we have two web Resources

- 1) Static Web Resource (html)
- 2) Dynamic Web Resource (jsp)

numbers.html :-

```
<HTML>
<BODY BGCOLOR = "cyan">
<CENTER> <H1> Numbers Entry Screen </H1>
<FORM ACTION = "add.jsp">
Number One : < INPUT TYPE = "text" NAME = "t1" ><BR><BR>
Number Two : < INPUT TYPE = "text" NAME = "t2" ><BR><BR>
< INPUT TYPE = "submit" VALUE = "Add" >
</FORM> </CENTER>
</BODY>
</HTML>
```

add.jsp :-

```
<%
String a = request.getParameter("t1");
String b = request.getParameter("t2");
int n1 = Integer.parseInt(a);
int n2 = Integer.parseInt(b);
int sum = n1 + n2;
%>
```

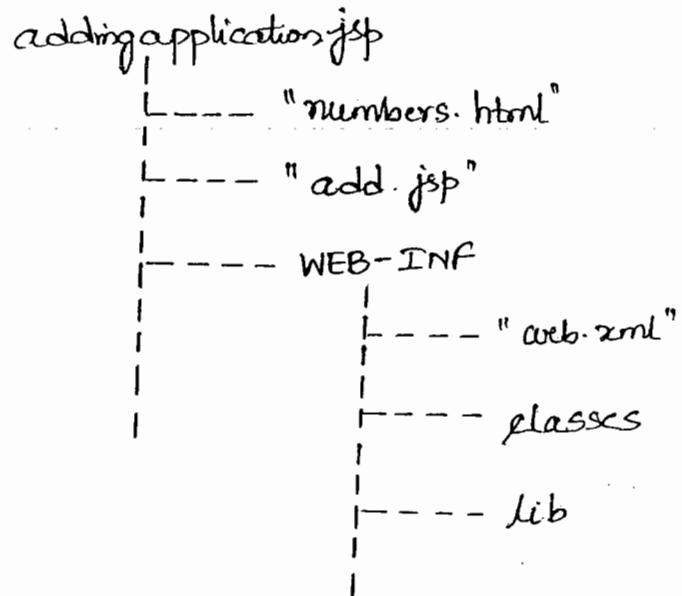
226

```
<BODY BGCOLOR = "RED">  
<H1> The Sum Is : <% = sum %> </H1>  
</BODY>  
</HTML>
```

Step-3: Deployment Descriptor Development (web.xml)

```
<web-app> ←  
</web-app> A jsp need not be Registered (can also be  
Registered but not required). A Servlet Should  
be Registered
```

Step-4: Configuration of Application files



URL :- http://localhost:8081/addingapplication.jsp/numbers.html

When control comes to jsp it gets executed. After its execution
jsp will not come to the browser, but a dynamic HTML

application is run.

In order to see the Script which was internally created

⇒ Tomcat 6.0 → work → addingapplication.jsp

↑ check in this directory

for every jsp, four packages are implicitly available, they are:

- java.lang.*
- javax.servlet.*
- javax.servlet.http.*
- javax.servlet.jsp.*

In the above application i.e addingapplication.jsp, the entire code in "add.jsp" can be replaced with one line shown below:

<H1> The Sum Is: <% = Integer.parseInt(request.getParameter("t1"))
+ Integer.parseInt(request.getParameter("t2")) %> </H1>

for this,
look the Container
generated Script.

if a ; is placed
here, inside an
expression, the
translation phase is
successful but
compilation fails.

→ Q) Develop a jsp that produces a dynamic web page indicating how many times user visited that page

Directory Structure:-

countingapplication.jsp

---- "Count.jsp"

---- WEB-INF

---- "web.xml"

URL :- <http://localhost:8081/countingapplication/jsp/count.jsp>

Count.jsp :-

```
<%! int count; %>
<HTML>
<BODY BGCOLOR = YELLOW>
<H1> The Number of Times This Page Is Visited : <%= ++count %>
<A HREF = "count.jsp"> Click Here </A>
</BODY>
</HTML>
```

web.xml :-

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

05/09/09

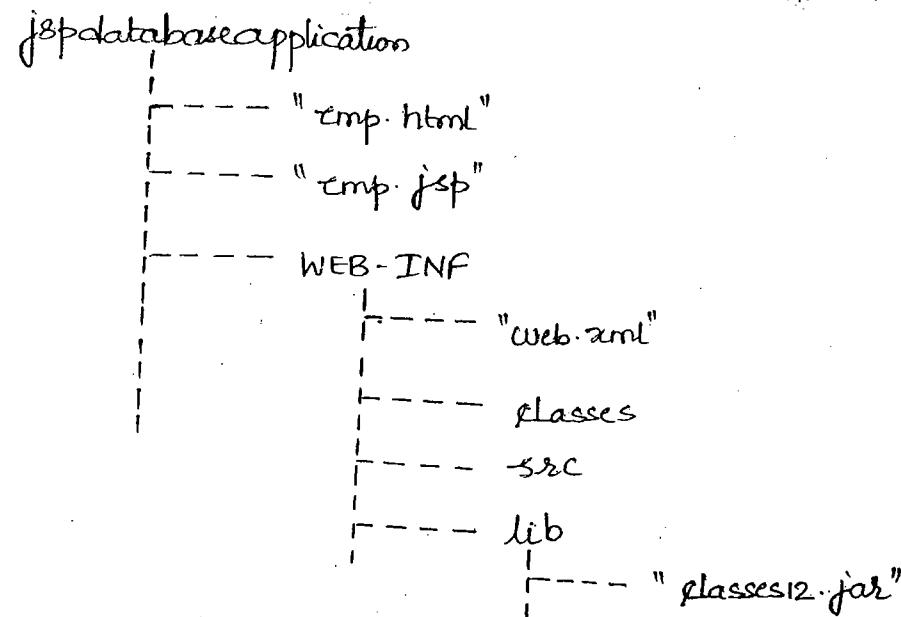
→ Q) Develop and Deploy a Web Application in which,
End User Should be able to enter employees data

URL :- <http://localhost:8081/jspdatabase>

↑
A WAR file

Directory Structure :-

229.



Emp.html :-

```
<HTML>
<BODY BGCOLOR = "wheat">
<CENTER><H1> Employee Creation </H1>
<FORM ACTION = "emp.jsp" METHOD = "Post">
    Empno : <INPUT TYPE = "text" NAME = "t1"><BR><BR>
    Name : <INPUT TYPE = "text" NAME = "t2"><BR><BR>
    Salary : <INPUT TYPE = "text" NAME = "t3"><BR><BR>
    <INPUT TYPE = "submit" VALUE = "Create Employee">
</FORM> </CENTER>
</BODY>
</HTML>
```

web.xml :-

```
<web-app>
```

</welcome-file-list>
</web-app>

Emp.jsp :-

According to MVC Architecture, a JSP should not communicate with database. In normal projects (which are not industry oriented) it can be done so.

A package cannot be imported using Scripting elements in a JSP. Hence, apart from the four-implicitly-available packages, other packages must be imported into a JSP using indirect methods like directives. In directives, 'page' directive is used for this purpose. It is as follows:

<%@ page import = "java.sql.*" %>

<%@ page import = "java.util.*" %>

(OR)

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

↑
A comma separated
list can also be used.

<%@ page import = "java.sql.*" %>

<%!

Connection con;

PreparedStatement ps;

public void jspInit()

try
{

```
Class.forName("oracle.jdbc.driver.OracleDriver");
con = DriverManager.getConnection("jdbc:oracle:thin:@localhost
                                :1521:Sever", "scott", "tiger");
ps = con.prepareStatement("INSERT INTO EMPLOYEE
                           VALUES(?, ?, ?)");
```

{}

```
catch (Exception e)
{ e.printStackTrace(); }
```

} // jspInit()

```
public void jspDestroy()
{
```

try
{

```
ps.close();
con.close();
}
```

```
catch (Exception e)
{ e.printStackTrace(); }
```

} // jspDestroy()

% >

<%

```
float sal = float.parseFloat(request.getParameter("t3"));
```

```
ps.setInt(1,eno);
ps.setString(2,nm);
ps.setFloat(3,sal);
ps.executeUpdate();
%>
```

even though this code is not written inside a try-catch block, no problem will rise because - jspService() method implicitly places this code inside a try-catch block.
But, this is not true for jspInit() and jspDestroy() methods.

```
<%@ include file = "emp.html" %>
```

↑
'include' directive performs include mechanism so that the same page is displayed again.
In this place, forward mechanism can also be used.

WAR file :-

```
C:\> jar -cf jspdatabase.war .
```

JSP Directives :-

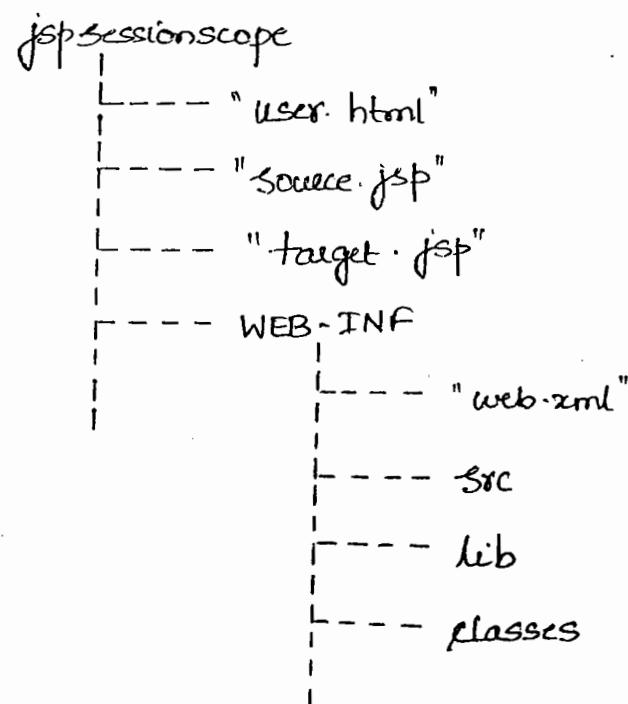
- A JSP Directive is a translation time instruction to the JSP Engine.
- We have three kinds of Directives
 - 1) page directive
 - 2) include directive
 - 3) taglib directive
- Any JSP Directive Starts with <%@ and ends with %>

- 'taglib' directive is used to import tag libraries into the jsp
- 'include' directive is used to include other pages into the current jsp inline. inline means wherever the include directive is written, at that point.
- 'page' directive is used to import java packages into the jsp, in configuring error pages, specifying the content type etc.

07/09/09

- Q) Develop a Web Application in which, two jsp share data in session scope.

Directory Structure :-



User.html :-

```

<HTML>
<BODY BGCOLOR = CYAN>
<CENTER><H1> User Screen </H1>
<FORM ACTION = "source.jsp">
User Name : <INPUT TYPE = "text" NAME = "user"><BR><BR>
<INPUT TYPE = "submit" VALUE = "Send">
</FORM> </CENTER>
</BODY>
</HTML>

```

web.xml :-

```

<web-app>
    <welcome-file-list>
        <welcome-file> user.html </welcome-file>
    </welcome-file-list>
</web-app>

```

Source.jsp :-

<%

```

String user = request.getParameter("user");
session.setAttribute("usr", user);
%
```

↑ implicit object. The creation of session object & travelling
of session ID takes place
implicitly (automatically) in
a jsp

<HTML>

<BODY BGCOLOR = "Yellow">

 Click Here To Get User Name

target.jsp :-

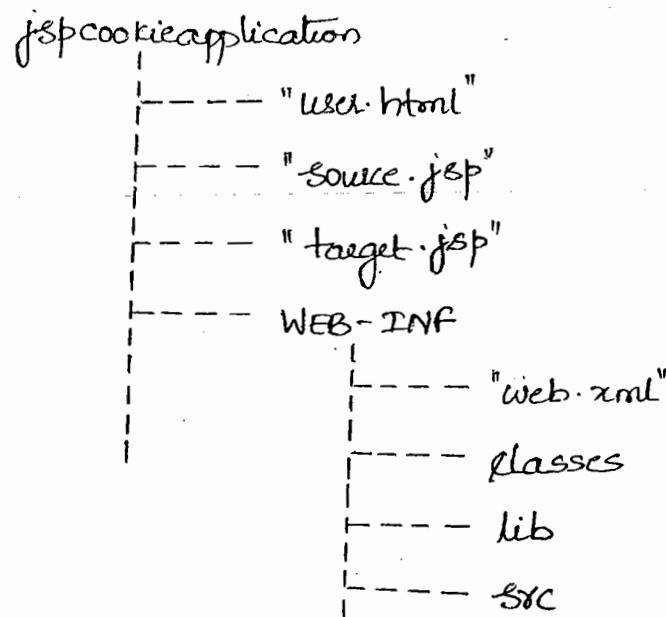
```

<HTML>
<BODY BGCOLOR = "wheat">
<H1> User Name Is : <% = (String) session.getAttribute("usr") %></H1>
</BODY>
</HTML>

```

→ Q) Develop a Web Application in which, persistent cookie concept is implemented

Directory Structure :-



URL :- http://localhost:8081/jspcookieapplication

user.html :-

```

<CENTER><H1> User Screen </H1>
<FORM ACTION = "source.jsp">
User Name : <INPUT TYPE = "text" NAME = "user"> <BR><BR>
<INPUT TYPE = "submit" VALUE = "Send">
</FORM> </CENTER>
</BODY>
</HTML>

```

web.xml :-

```

<web-app>
    <welcome-file-list>
        <welcome-file> user.html </welcome-file>
    </welcome-file-list>
</web-app>

```

Source.jsp :-

```

<%
String user = request.getParameter("user");
Cookie c = new Cookie("usr", user);
c.setMaxAge(300);
response.addCookie(c);
%>

```

<HTML>

<BODY BGCOLOR = "wheat">
 Click Here To Get User Name
</BODY>

target.jsp :-

<%

Cookie c[] = request.getCookies();

String user = c[0].getValue();

%>

<HTML>

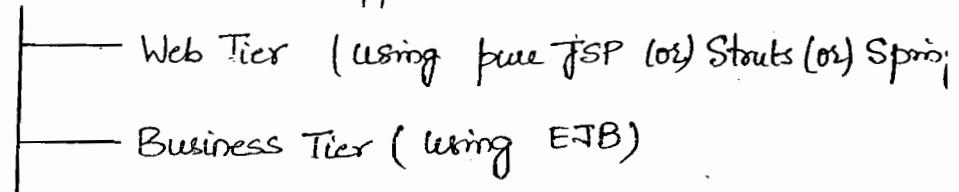
<BODY BGCOLOR = RED>

<H1> User Name Is : <%= user %> </H1>

</BODY>

</HTML>

J2EE professional Business Applications



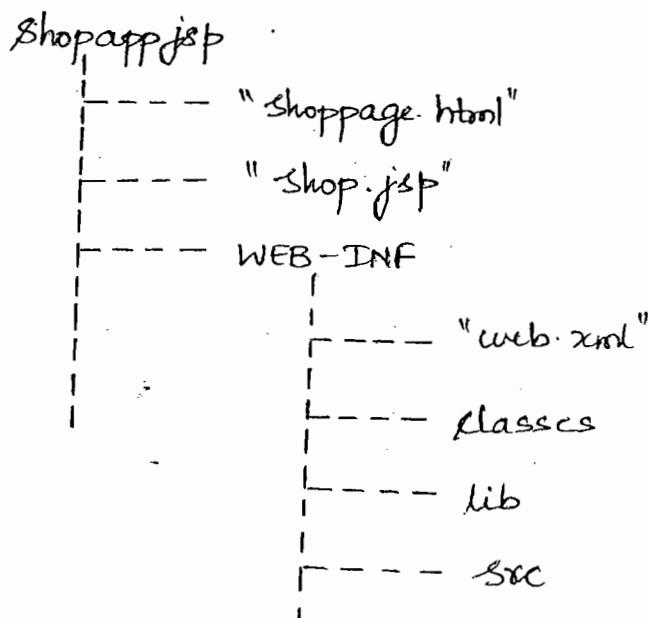
Major disadvantages of writing the java code in jsp would be (i.e. mixing java code with HTML) :

- Source code & its logic is directly exposed to the business client
- If Syntactical errors are present, they are thrown at runtime rather than at compile time as java code in jsp is compiled at runtime i.e. when application is run in the browser
- No tool can be used to develop java code and HTML together as a jsp.

Hence, using some method java code and HTML Should be Sepa

→ Q) Shopping Cart Application Using JSP.

Directory Structure :-



URL :- <http://localhost:8081/Shopapp.jsp/shoppage.html>

web.xml :-

```

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

Shoppage.html :-

```

<HTML>
<BODY BGCOLOR = "WHEAT">
<CENTER><H2> Welcome To Shopping Mall </H2>
<FORM ACTION = "shop.jsp" METHOD = "POST">
  
```

```

<OPTION VALUE = 101>101
<OPTION VALUE = 102>102
<OPTION VALUE = 103>103
<OPTION VALUE = 104>104
<OPTION VALUE = 105>105
</SELECT><BR><BR>

```

Product Quantity : <INPUT TYPE = "text" NAME = "t1">
<input type = "submit" name = "s" value = "ADD ITEM">

<input type = "submit" name = "s" value = "REMOVE ITEM">

<input type = "submit" name = "s" value = "SHOW ITEMS">

<input type = "submit" name = "s" value = "LOGOUT">

</FORM> </CENTER>

</BODY>

</HTML>

Shop.jsp :-

```

<%@ page import = "java.util.*" %>
<%! String code, qty, sb; %>

<%
    session.setMaxInactiveInterval(300);
    sb = request.getParameter("s");
    if (sb.equals("ADD ITEM"))
    {
        code = request.getParameter("pCode");
        qty = request.getParameter("t1");
    }
%>

```

```

}
else if (sb.equals("Remove ITEM"))
{
    code = request.getParameter("pCode");
    Session.removeAttribute(code);
    response.sendRedirect("shoppage.html");
}

else if (sb.equals("SHOW ITEMS"))
{
    Enumeration e = session.getAttributeNames();
    if (e.hasMoreElements())
    {
        %>
<HTML>
<BODY BGCOLOR=WHEAT>
<H2><FONT COLOR=BLUE> Your Shopping Cart Items </FONT></H2>
<%
    while (e.hasMoreElements())
    {
        String c = (String)e.nextElement();
        %>
<H2> Product Code : <% = c %> </H2>
<H2> Quantity Is : <% = session.getAttribute(c) %> </H2>
<%
    } //while
%>
</BODY>

```

<%

} // if

else
{

%>

<HTML>

<BODY BGCOLOR = WHEAT>

<H2> No Items In the Cart </H2>
</BODY>

</HTML>

<%

} // else

%>

<H3> Want To Shop! </H3>

<%

} // outer If

- else if (sb.equals("LOGOUT"))
{

Session.invalidate();

%>

<HTML>

<BODY BGCOLOR = RED>

<H2> Want To

Shop Again! </H2>

</BODY>

</HTML>

<%

// Alternate Way of writing Shop.jsp :-

242

```
<%@ page import = "java.util.*" %>
<%! String code, qLty, sb; %>
<%
```

```
Session. setMaxInactiveInterval(300);
```

```
sb = request.getParameter("s");
```

```
if (sb.equals("ADD ITEM"))
```

```
{
```

```
    code = request.getParameter("pCode");
```

```
    qLty = request.getParameter("t1");
```

```
    Session. setAttribute(code, qLty);
```

```
    response. sendRedirect("shoppage.html");
```

```
}
```

```
else if (sb.equals("REMOVE ITEM"))
```

```
{
```

```
    code = request.getParameter("pCode");
```

```
    Session. removeAttribute(code);
```

```
    response. sendRedirect("shoppage.html");
```

```
}
```

```
else if (sb.equals("SHOW ITEMS"))
```

```
{
```

```
    Enumeration e = Session.getAttributeNames();
```

```
    if (e.hasMoreElements())
```

```
{
```

```
    out.println("<H2><FONT COLOR=BLUE> Your
```

```

while ( e.hasMoreElements() )
{
    out.println( "<BODY BGCOLOR = CYAN>" );
    String c = (String)e.nextElement();
    out.println( "<H2> Product Code: " + c );
    out.println( " Quantity Is: " + session.getAttribute(c)
                + "</H2>" );
}

// while

} // if

else
{
    out.println( "<BODY BGCOLOR = CYAN>" );
    out.println( "<H2> <FONT COLOR = RED> No Items In "
                + "The Cart </FONT></H2>" );
}

// else

out.println( "<A HREF = Shoppage.html> Want To "
            + "Shop! </A>" );
}

// outer if

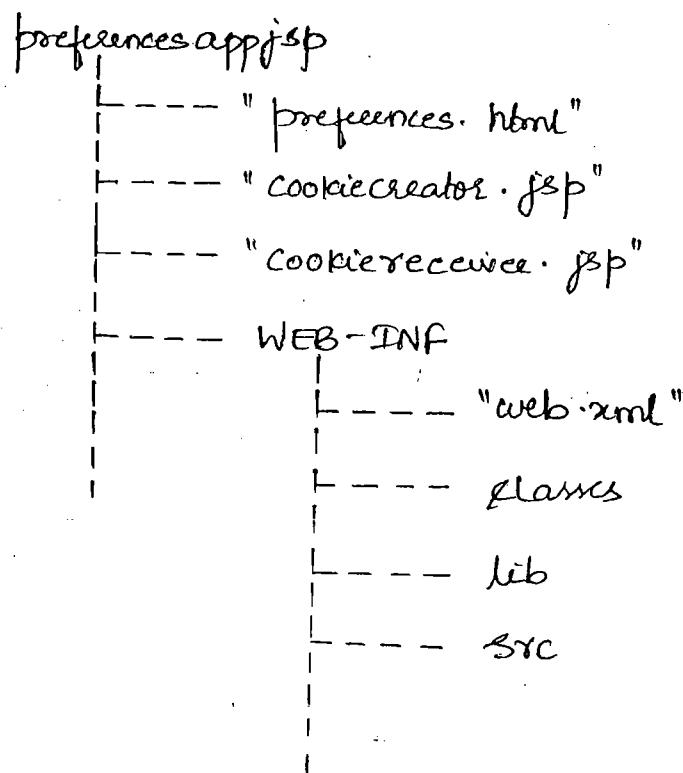
else if ( sb.equals( "LOGOUT" ) )
{
    session.invalidate();
    out.println( "<BODY BGCOLOR = CYAN>" );
    out.println( "<A HREF = Shoppage.html> Want To "
                + "Shop Again! </A>" );
}

```

%>

→ Q) Use Preferences Application Using jsp

Directory Structure :-



URL :- http://localhost:8081/preferencesappjsp/preferences.html

web.xml :-

```

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

preferences.html :-

```

<HTML>
    
```

<CENTER><H1> Welcome To www.Nit.Com </H1>

 <FORM ACTION = "cookiecreator.jsp" METHOD = "POST">
 Select Background Colour : <SELECT NAME = "t1">
 <OPTION VALUE = "GREEN"> Green
 <OPTION VALUE = "RED"> Red
 <OPTION VALUE = "YELLOW"> Yellow
 </SELECT>

 Select The Sport :

 <INPUT TYPE = "RADIO" NAME = "t2" VALUE = "CRICKET"> Cricket

 <INPUT TYPE = "RADIO" NAME = "t2" VALUE =
 "FOOTBALL"> Football
 <INPUT TYPE = "RADIO" NAME = "t2" VALUE =
 "BASKETBALL"> Basketball

 <INPUT TYPE = "SUBMIT" VALUE = "Send">
 </FORM> </CENTER>
 </BODY>
 </HTML>

CookieCreator.jsp :-

```

<%
String bclr = request.getParameter("t1");
String sport = request.getParameter("t2");
Cookie c1 = new Cookie("bc", bclr);
c1.setMaxAge(300);
response.addCookie(c1);
  
```

Response.addCookie(c2);

%>

<HTML>

<BODY BGCOLOR = WHEAT >

<H3> Get Your Preferences Here </H3>
</BODY>
</HTML>

CookieReceiver.jsp :-

<%

Cookie c[] = request.getCookies();

String color = null;

String spost = null;

if (c != null)

{

for (int i=0; i < c.length; i++)
{

String name = c[i].getName();

if (name.equals("s"))

spost = c[i].getValue();

else if (name.equals("bc"))

color = c[i].getValue();

}

}

%>

<HTML>

<BODY BGCOLOR = <% = color %>>

<MARQUEE> <% = spost %> </MARQUEE>

Include Mechanism in a jsp

- Including other jsps into the current jsp inline, is known as Include Mechanism.
- Include Mechanism is meant for reusability of presentation code and avoiding duplication of same presentation code in Multiple jsps of the web Application

Consider the following Scenario:

In an application, there are 80 jsps

1.jsp

header code = 40 lines →



footer code = 60 lines →

Say, these two are required for rest of the 79 jsps, then these 100 lines must be duplicated in all the 80 jsps. This results in duplication of presentation code. This can be avoided by developing 82 jsps with 81.jsp as header.jsp & 82.jsp as footer.jsp.

Now,

1.jsp

include 81 →



include 82 →

These two can be written in

rest of 79 jsps, hence

duplication of presentation code is eliminated.

jsps always have to implement include mechanism only at industry standard

→ Include Mechanism is implemented in a jsp in two ways:

- 1) Using include directive
- 2) Using include standard action

08/09/09

→ What are the differences between include directive and include standard action?

Before we look into differences, the similarities between include directive and include standard action are as follows:

- 1) Both are instructions to container
- 2) Both have same purpose i.e., to implement Include Mechanism and to avoid duplication of code
- 3) Both perform Include Mechanism i.e., including other pages (could be either HTML or jsps), 'inline' only.

inline means at what point these instructions are written in the current jsp, exactly at that place the other jsps or HTML pages are included.

teria

include Directive

→ attribute

<%@ include file = "jsp" %>

Eg:- <%@ include file = "header.jsp" %>

include Standard Action

→ attribute

<jsp: include page = "jsp" />

Eg:- <jsp: include page = "header.jsp" />

of Instruction

Translation time Instruction

i.e. A Static Instruction

Request Processing time Instruction

i.e. A Dynamic Instruction. It is also known as Runtime Instruction

is included?

Other page Source Code is included,
inline

Other page's Response is included, inline

or of Servlets produced

1 *

2 ---

Productive Performance

Higher

implies, when request comes
how fast it is processed and response
is sent.

changes are made to the
included page, do they reflect

the Current page response

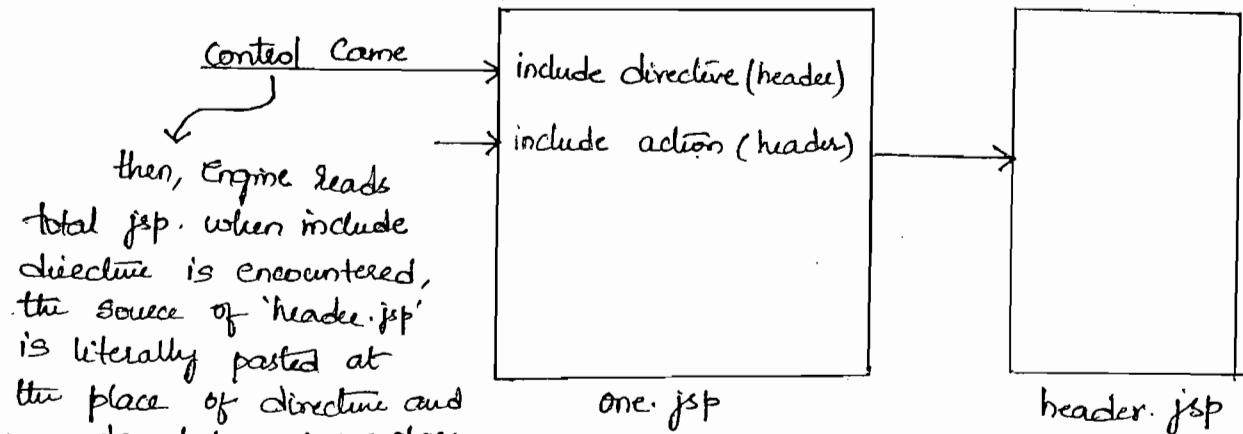
Lower

*

Yes

Consider the following Scenario

250.



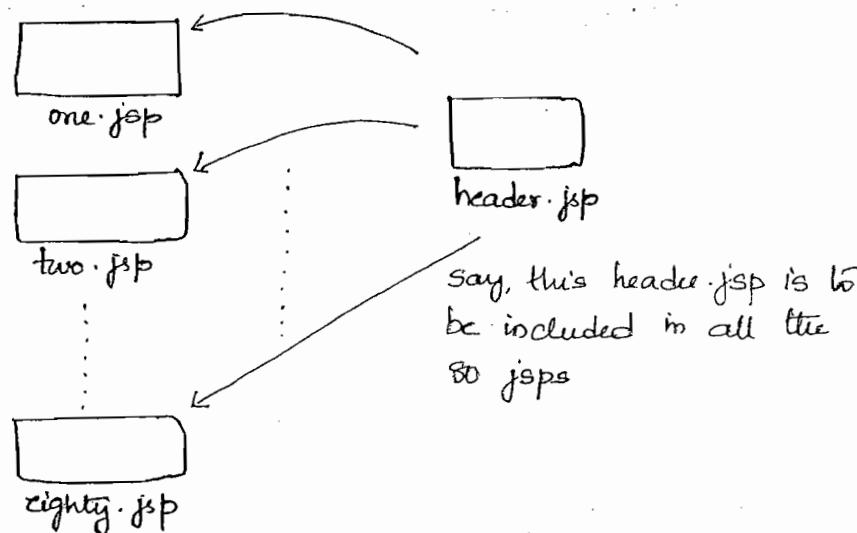
In case of include action, whenever request comes to 'one.jsp' the control transfers to 'header.jsp' as soon as include standard action is encountered. In the background, request dispatching takes place i.e. control is transferred to 'header.jsp' and its response is added to response of 'one.jsp'. Response of 'header.jsp' implies its Servlet has to be generated.

Always static mechanism has better performance compared to dynamic mechanism. Dynamism gives flexibility but is poor in performance.

Since directive is a static instruction, its performance is higher. When first time request comes for 'one.jsp', the contents of 'header.jsp' are included and translation into a Servlet takes place. For the next n-times, this translation is not done hence next n-requests are processed faster and hence performance is high.

Since action is a dynamic instruction, its performance is lower. Each time request comes, the control is switched from 'one.jsp' to 'header.jsp' and whenever an 'header.jsp' is added to 'one.jsp'

Consider the following scenario



Say, this header.jsp is to be included in all the 80 jsp's

If directive is used to include 'header.jsp', when first time request comes for 'one.jsp' all the process is done and response is sent. When second time request comes for 'one.jsp', the request is processed and response is directly sent, but again translation is not done, hence performance is not lost. This would be the case for rest of jsp's. This would be true as long as the jsp's are not modified.

Now, consider a case where 'header.jsp' needs to be modified. That means the changes are made to the included page. After this modification, when request comes to 'one.jsp' directly servicing phase is done but again translation is not done since 'one.jsp' is not modified i.e. unless 'one.jsp' is not modified, translation is not done once again. Hence, modifications in included page do not reflect in current pages.

If 'two.jsp' is modified, the translation of this jsp is done again and the modifications done in 'header.jsp' reflect in the response of 'two.jsp'. This is the drawback in using include directive.

But when include action is considered, any changes

response of 'header.jsp' is added to the current page's response.

Therefore, when to use directive and when to use action?

⇒ It depends.

- If performance is needed, go for include directive. This requires the pages not to be modified throughout
- If pages need to be continually modified throughout the life of application and these modifications should reflect, then go for include standard action but at the cost of losing performance

→ Example jsp in which, include mechanism is implemented using both directive and standard action.

In 'one.jsp', both 'header.jsp' and 'footer.jsp' are included. Their bodies are according to the application requirement.

one.jsp

<jsp:include page = "header.jsp" />



<%@ include file = "footer.jsp" %>

when response goes to browser, only one page goes but in the background, three pages are added together.

In the above example, it was assumed that 'header' may

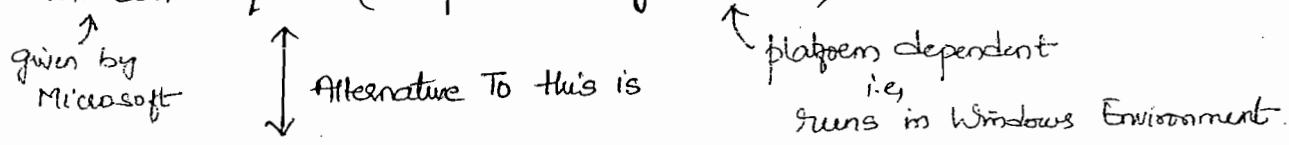
Using A Java Bean In A jsp

→ What is a Java Bean?

A Java Bean is a specialised Java class that is defined according to Java Beans Specification given by Sun Microsystems.

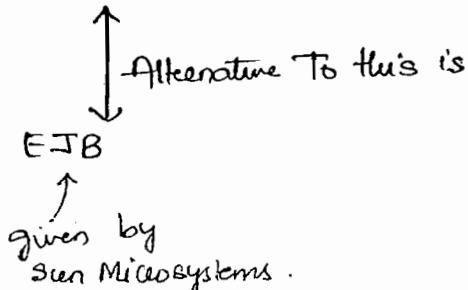
A Bean is a platform independent reusable software component. Java Bean is used in Web Tier (i.e., presentation tier) but an Enterprise Java Bean is used in Business Tier.

Activex controls / COM (component Object Model)



Java Bean
given by Sun Microsystems

DCOM (Distributed COM)



EJB
given by Sun Microsystems.

Before COM / Java Bean, applications were developed as a single piece of code i.e., Monolithic Development. Disadvantage in such a Development is that any erroneous logic or ill logic or future modifications at one portion means that, total application code needs to be modified. To overcome this, an application

these modules (components) are assembled (linked) together. This is similar to the Assembly in hardware. A Generalized Example of hardware assembly could be a "Tubelight". If all the parts in a Tubelight like starter, choke, frame, bulb etc are moulded as a single piece then if problem occurs with any one of the part, total Tubelight needs to be changed. If these parts are individually developed and then assembled together later, then if any problem with a specific part, only that part can be changed. This concept would also increase productivity with less infrastructure. This process in software is known as COM. This was introduced in Java Bean.

09/09/09

→ Define an Employee Bean.

Any Java class can act as a Java Bean if it follows the given rules:

- *) Class is public
- *) Class implements Serializable Interface
- *) Properties are private
 - *) Class has public default Constructor
 - *) Each bean field (property) has one public setter method and one public getter method.

They are
also known
as bean
fields

These are written because the

Employee.java

package empbean; *not mandatory as a specification but in order to use it (bean) in a web Application it is mandatory*
 public class Employee implements java.io.Serializable

```
private int empro;
private String name;
private float salary;
public void setEmpro( int empro ) { this.empro = empro; }
public int getEmpro() { return empro; }
```

Variable name as it is written, but initial letter must be in capital -

public void setName(String nm)

```
{ this.name = nm; }
```

not required as these two are different

public String getName()

```
{ return this.name; }
```

public void setSalary(float salary)

required here { this.salary = salary; }

public float getSalary()

```
{ return salary; }
```

}

Note :- Setter methods are meant for giving data to the bean fields. getter methods are meant for retrieving data

- When we are using a Java Bean in a jsp, it is mandatory to package it in a user defined package.
- Compile the bean as follows and copy the user defined package along with the bean class file into "classes" directory of our web Application

`javac -d Employee.java`
 ↗
 ↗
 ↗ space

functionally reusable, physically replaceable Software piece of code is known as a component.

Component Technology is an Extension to object orientation but not a replacement

functionally reusable

↓ consider

class A

{
}

A a = new A();

↓

This object cannot be directly used in another application unless a special code like inheritance code is written.

for a component, this is not necessary. A component can be reused without any specialized code. This is functional reusability

Java Bean was initially used at client side / GUI, like development of buttons, password fields, screens etc, so that they are reusable. This improves the development time.

Nowadays, Industry requires Java Bean to be developed at Server Side.

Java Beans → Visible Bean (used in GUI Development)

Consider A Bean:

```
public class Account implements Serializable
{
```

```
    private int accno;
```

```
    public Account() {}
```

This is a 'default constructor'.
It is another name given
to a zero argument constructor.

```
    public void setAccno(int accno)
```

```
    { this.accno = accno; }
```

Even if this default constructor
is not written, compiler writes it.

```
    public int getAccno()
```

```
    { return accno; }
```

```
}
```

Consider:

```
class A
{
    ===
}
```

forac A.java

Inside the source code no constructor is available. Even before compilation takes place, the compiler reads the source code and if no constructor is found, it literally writes a constructor. This would be the default constructor. Hence, default constructor is given by compiler but not by JVM.

It is like:

Class A extends B

```
{
```

A()

```
{ super() }
```

```
===
}
```

↑
If inheritance
then,

default constructor is like this

why class should be public?

→ we never create instance of bean in our Web Application.

Container creates this instance. Hence, for this to happen, the class should be public

why to implement Serializable interface?

→ Nature of Java Bean Should be persistent. Hence, what instance created in RAM, should also be stored in secondary memory i.e. Hard-disk. For this, the class should implement Serializable Interface.

But when a Java Bean is used in Web Applications, this rule is not mandatory.

why properties should be private?

→ Bean State i.e. Bean Data Should not be accessed or modified by External Entities

why setter & getter methods?

→ These methods are mandatory because, in order to store or retrieve data into / from the private properties (bean fields), these methods are helpful.

Naming conventions are to be followed for these methods so that the Container can call these methods implicitly.

If not followed, this implicit calling nature is lost & some other means should be used to call them, by the programmeer.

why a default constructor?

is required so that the instance of the bean class can be created by container. If a parameterized constructor is used, this would not be possible.

Using A Java Bean In A jsp :-

→ To make use of a Java Bean in a jsp, three standard actions (tags) are used:

↑
Synonymous.
They are instructions.

- 1) useBean
- 2) setProperty
- 3) getProperty

useBean :-

This tag is used to instruct the container to create the bean instance OR to get the already available instance (if any).

Syntax :- <jsp:useBean id = "beanref" class = "fully qualified name of bean class" />

for Example:- <jsp:useBean id = "e" class = "empbean.Employee" />

↑
After creating
bean instance,
its reference is
stored in this
'e'. Hence it acts
as a pointer.

↑
package
name
↑
class
name
—————
fully qualified name
of bean class.

SetProperty :-

This standard action is used to instruct the container to populate the specified bean field(s).

Syntax:- <jsp:setProperty name = "beanref" property = "bean
variable name" value = "Some Value" />

for Example :- <jsp:setProperty name = "e" property = "empno" value =
"1001" />

↑
in the background
setters methods are only
called. we do not call
them but container does.

↑
This is static
way of assigning
values.

it is equivalent to $\Rightarrow e.\text{SetEmpno}(1001)$

Note:- If data is coming from web form to populate the bean fields,
"Value" attribute should not be used. We have to use "param"
attribute.

for Example :-

```
<jsp:setProperty name = "e" property = "empno" param = "t1" />
<jsp:setProperty name = "e" property = "name" param = "t2" />
<jsp:setProperty name = "e" property = "salary" param = "t3" />
```

Note:- Instead of using setProperty tag 3-times, we can do the same job with one entry of the tag but request parameter names and bean field names should match.

for Example:-

```
<jsp:setProperty name = "e" property = "*" />
```

In this case, "param" & "value" attributes are

getProperties :-

This Standard action is used to retrieve data from the bean field and write to the browser Stream. Hence it performs two things i.e., data retrieval and writing data to browser stream.

Syntax :- `<jsp:getProperty name = "beanref" property = "bean field"`
 ↓
 This is equivalent to `out.print(e.get())`

for Example :- `<jsp:getProperty name = "e" property = "empno" />`

↓
 This is equivalent to
 `out.print(e.getEmpno())`

Once the above action is executed, Employee number stored in the bean instance is retrieved (internally by calling getEmpno() method only) and written to the browser Stream.

- useBean tag \Rightarrow To instantiate Bean,
- setProperty tag \Rightarrow To provide data to Bean
- getProperty tag \Rightarrow To retrieve data from the Bean

→ Q1 Develop a Web Application in which a jsp makes use of a Java Bean

Directory Structure :-

```

jspbeanapplication
    |
    +-- "accountinfo.html"
    +-- "account.jsp"
    +-- WEB-INF
        |
        +-- "web.xml"
        |
        +-- src
            |
            +-- "Account.java"
        |
        +-- lib
        |
        +-- classes
            |
            +-- "com"
    
```

URL :- <http://localhost:8081/jspbean>

→ no html file, implies home page is configured in web.xml
not root directory name, hence should be a war file.

262

web.xml :-

```
<web-app>
  <welcome-file-list>
    <welcome-file> accountinfo.html </welcome-file>
  </welcome-file-list>
</web-app>
```

Account.java (Java Bean Source Code)

```
package com.mit.beans;
```

```
public class Account implements Serializable
```

```
{
```

```
  private int accno;
```

```
  private String name;
```

```
  private float balance;
```

```
  public void setAccno(int ano)
```

```
  { accno = ano; }
```

```
  public int getAccno()
```

```
  { return accno; }
```

```
  public void setName(String n)
```

```
  { name = n; }
```

```
  public String getName()
```

```

public void setBalance(float bal)
{
    balance = bal;
}

public float getBalance()
{
    return balance;
}
}

```

accountinfo.html :-

```

<HTML>
<BODY BGCOLOR = "CYAN">
<CENTER><H1> Account Creation Screen </H1>
<FORM ACTION = "account.jsp">
    Accno : <INPUT TYPE = "text" NAME = "accno"><BR><BR>
    Name : <INPUT TYPE = "text" NAME = "name"> <BR><BR>
    Balance : <INPUT TYPE = "text" NAME = "balance"><BR><BR>
    <INPUT TYPE = "submit" VALUE = "Send">
</FORM> </CENTER>
</BODY>
</HTML>

```

This is same as bean field name so that '*' would work at the time of setProperty.

request parameter names 'accno', 'name' and 'balance' (which were t1, t2, t3 earlier) match with bean field names so that, while populating the bean i.e. while using `setProperty` tag, '*' would work and automatic synchronization could take place. Otherwise if t1, t2, t3 are only used, then while populating, three different `setProperty` tags are to be written with 'param' attribute.

account.jsp :-

→ not mandatory
to import. If not imported, this JSP may
work or may not depending
on the container 264

<%@ page import = "com.nit.beans.Account" %>

<jsp:useBean id = "acc" class = "com.nit.beans.Account" />

<jsp:setProperty name = "acc" property = "*" />

<HTML>

 <BODY BGCOLOR = "Yellow">

 <CENTER><H1> Account Details </H1>

 <TABLE BORDER = "2" >

 <TR>

 <TH> ACCNO </TH>

 <TH> NAME </TH>

 <TH> BALANCE </TH>

 </TR>

 <TR>

 <TD> <jsp:getProperty name = "acc" property = "accno" /> </TD>

 <TD> <jsp:getProperty name = "acc" property = "name" /> </TD>

 <TD> <jsp:getProperty name = "acc" property = "balance" /> </TD>

 </TR>

 </TABLE>

 </CENTER>

 </BODY>

</HTML>

In the above application, the bean (i.e., the bean instance) cannot be shared among two or more JSPs since scope was not mentioned. The `<useBean>` tag has an attribute named 'scope' which can be used to specify the scope for the bean.

If scope is not mentioned in `<useBean>` tag, the default scope would be 'page' i.e., the bean is available only in that JSP, which instantiates it. This bean cannot be shared.

To increase the scope from default 'page' scope to say, 'request scope', the attribute 'scope = "request"' must be used in the `<useBean>` tag. The following application demonstrates this fact.

→ Q) Web Application in which, a bean is shared between two JSPs in request scope.

Directory Structure :-

```

requestScopeBeanSharing
├── accountinfo.html
├── account.jsp
└── target.jsp
├── WEB-INF
│   ├── web.xml
│   ├── src
│   │   └── Account.java
│   ├── lib
│   └── classes
        └── com
    
```

URL :-

http://localhost:8081/request-scope-bean-sharing

web.xml :-

```
<web-app>
    <welcome-file-list>
        <welcome-file>accountinfo.html</welcome-file>
    </welcome-file-list>
</web-app>
```

Account.java :-

```
package com.nit.beans;

public class Account implements java.io.Serializable
{
```

```
private int accno;
private String name;
private float balance;
```

```
public void setAccno(int ano)
{
    accno = ano;
}
public int getAccno()
{
    return accno;
}
```

```
public void setName(String n)
{
    name = n;
}
public String getName()
```

```

public void setBalance(float bal)
{
    balance = bal;
}

public float getBalance()
{
    return balance;
}
]

```

accountinfo.html :-

```

<HTML>
<BODY BGCOLOR = "CYAN">
<CENTER><H1> Account Creation Screen </H1>
<FORM ACTION = "account.jsp">
    Accno: <INPUT TYPE = "text" NAME = "accno"><BR><BR>
    Name: <INPUT TYPE = "text" NAME = "name"><BR><BR>
    Balance: <INPUT TYPE = "text" NAME = "balance"><BR><BR>
    <INPUT TYPE = "submit" VALUE = "Send">
</FORM></CENTER>
</BODY>
</HTML>

```

account.jsp :-

```

<%@ page import = "com.nit.beans.Account" %>

<jsp:useBean id = "acc" class = "com.nit.beans.Account" scope = "request" />
<jsp:setProperty name = "acc" property = "*" />

```

target.jsp :-

268

```
<%@ page import = "com.nit.beans.Account" %>
<jsp:useBean id="acc" class = "com.nit.beans.Account" scope = "request" />

<HTML>
<BODY BGCOLOR = "Yellow">
<CENTER><H1> Account Details </H1> <BR>
<TABLE BORDER = "2">
<TR>
<TH> Accno </TH>
<TH> Name </TH>
<TH> Balance </TH>
</TR>
<TR>
<TD><jsp:getProperty name = "acc" property =
"accno"/> </TD>
<TD><jsp:getProperty name = "acc" property =
"name"/> </TD>
<TD><jsp:getProperty name = "acc" property =
"balance"/> </TD>
</TR>
</TABLE>
</CENTER>
</BODY>
</HTML>
```

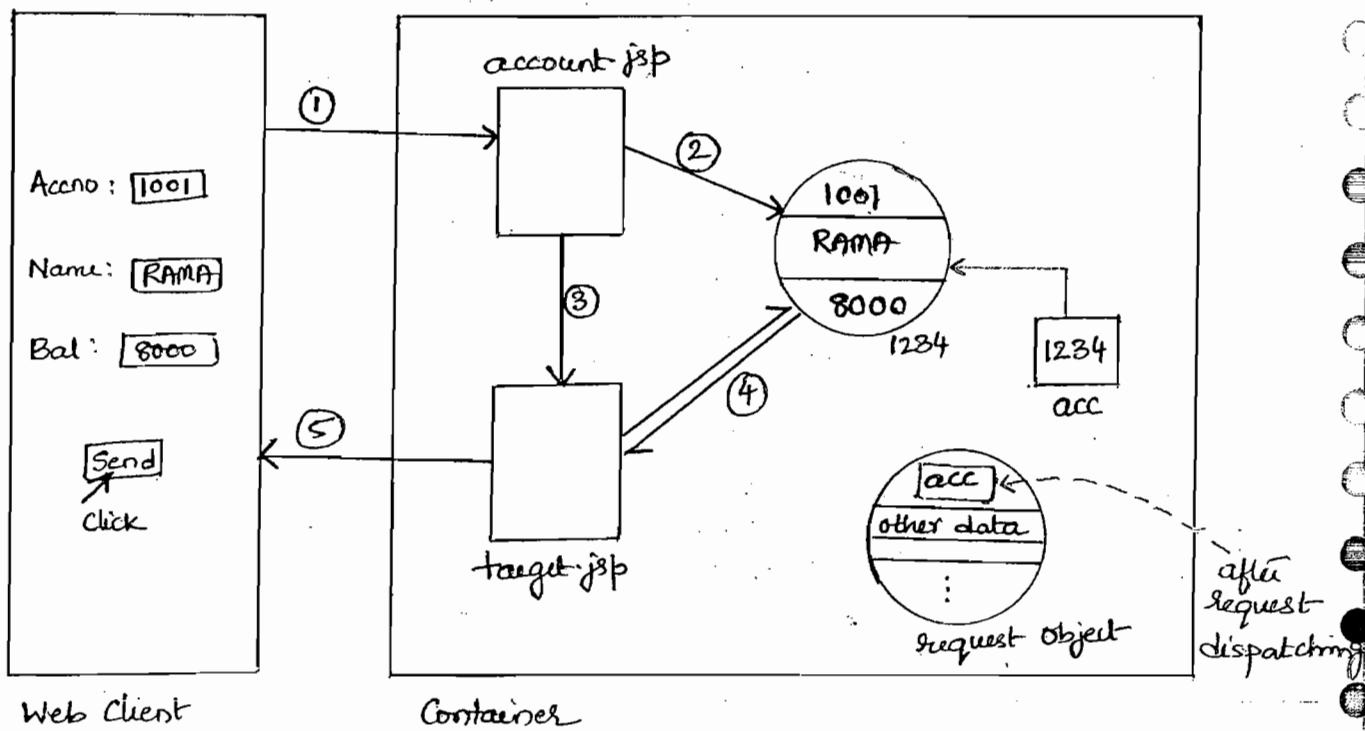
As soon as the user inputs some data and clicks on submit button, control is transferred to "account.jsp". When useBean tag is encountered in "account.jsp", as the scope was mentioned to be in "request-scope", the container checks the request object, for any bean reference. Since the request was for the first time no bean reference comes and hence a new bean instance is created with a reference as "acc". Then, the bean is populated and after that the request is dispatched to "target.jsp"

When request is dispatched, the reference of this bean is also sent along, within the request object. In "target.jsp" when useBean tag is encountered, the container once again checks the request object if there is any bean instance's reference. Since the request object contains the reference "acc" which was actually created by "account.jsp", this reference is only used in "target.jsp" to retrieve the values present in bean fields.

If the "request" attribute is not written in both the jsp's, after the Request Dispatching, control comes to "target.jsp" and in "target.jsp" when useBean is encountered, instead of using the already created bean instance and its reference, a new bean instance is created with reference as "acc". Since in "target.jsp" no setProperty tag is used, the properties of bean are initialized to default values i.e, integer property with zero, String property with null, float property with 0.0 and so on. When getProperty tag is encountered, these default values are only retrieved and pointed on to the browser Stream.

In request scope, the bean is shareable among

the bean instance too.



- ① \Rightarrow Request comes to "account.jsp" for the first time along with user inputted data
- ② \Rightarrow "account.jsp" instantiates the bean and populates it with user inputted data
- ③ \Rightarrow "account.jsp" switches the control to "target.jsp" i.e, Request Dispatching takes place
- ④ \Rightarrow "target.jsp" contacts the bean through the reference "acc" and retrieves the data
- ⑤ \Rightarrow Response is sent to browser by "target.jsp".

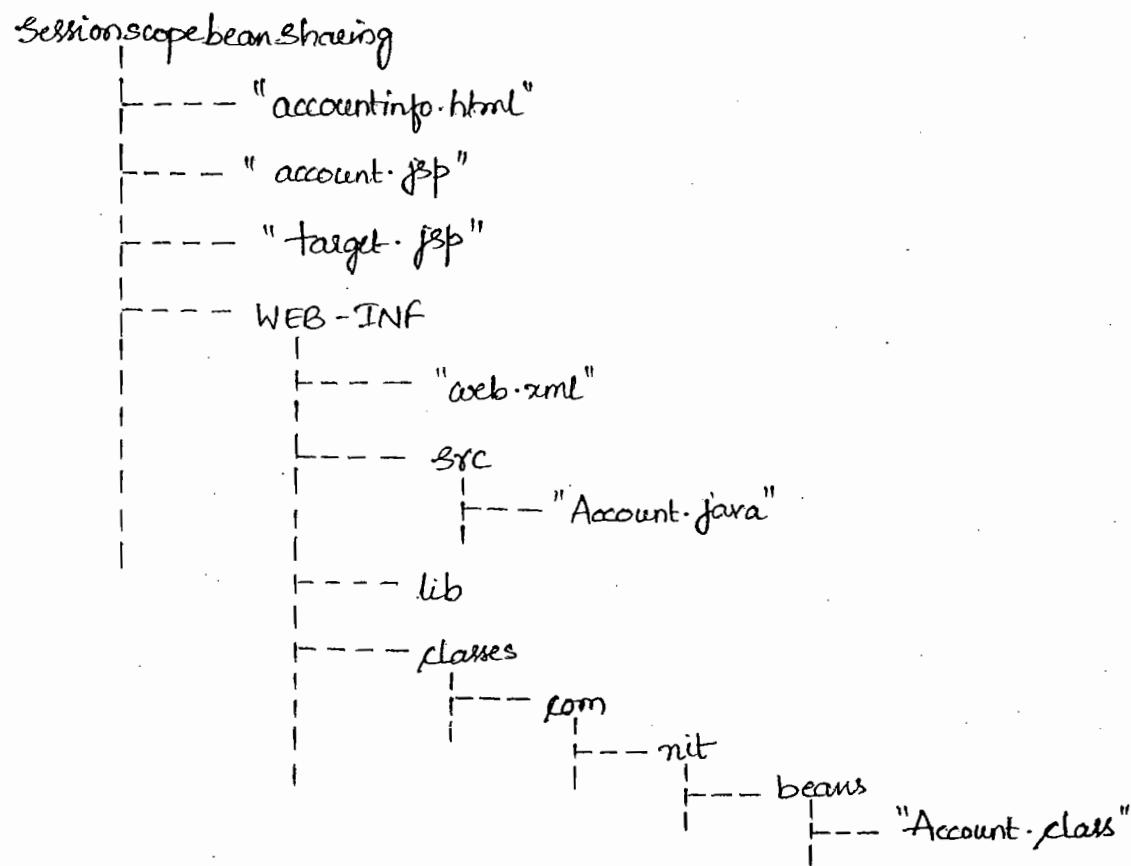
The Scope can be further increased to Session Scope. In this Scope, the bean instance can be shared among JSPs for an entire Session.

11/09/09

- Q) Develop and Deploy a Web Application in which a

Directory Structure :-

2:



URL :-

http://localhost : 8081 / sessionscopebeansharing

web.xml :-

```
<web-app>
    <welcome-file-list>
        <welcome-file> accountinfo.html </welcome-file>
    </welcome-file-list>
</web-app>
```

Account.java :-

public class Account implements java.io.Serializable

}

private int accno;
private String name;
private float balance;

public void setAccno(int ano)

{ accno = ano; }

public int getAccno()

{ return accno; }

public void setName(String n)

{ name = n; }

public String getName()

{ return name; }

public void setBalance(float bal)

{ balance = bal; }

public float getBalance()

{ return balance; }

}

accountinfo.html :-

<HTML>

<BODY BGCOLOR = "CYAN">

<CENTER> <H1> Account Creation Screen </H1>

Accno : <INPUT TYPE = "text" NAME = "accno">

 Name : <INPUT TYPE = "text" NAME = "name">

 Balance : <INPUT TYPE = "text" NAME = "balance">

 <INPUT TYPE = "submit" VALUE = "Send">
 </FORM> </CENTER>
 </BODY>
 </HTML>

account.jsp :-

```

<%@ page import = "com.nit.beans.Account" %>
<jsp:useBean id = "acc" class = "com.nit.beans.Account" scope = "session" />
<jsp:setProperty name = "acc" property = "*" />

<HTML>
<BODY BGCOLOR = "WHEAT">
<A HREF = "target.jsp"> Get Account Details Here </A>
</BODY>
</HTML>
  
```

target.jsp :-

```

<%@ page import = "com.nit.beans.Account" %>
<jsp:useBean id = "acc" class = "com.nit.beans.Account" scope = "session" />

<HTML>
  
```

```

<TABLE BORDER = "2">

    <TR>
        <TH> ACCNO </TH>
        <TH> NAME </TH>
        <TH> BALANCE </TH>
    </TR>

    <TR>
        <TD> <jsp:getProperty name="acc" property =
                            "accno" /> </TD>
        <TD> <jsp:getProperty name="acc" property =
                            "name" /> </TD>
        <TD> <jsp:getProperty name="acc" property =
                            "balance" /> </TD>
    </TR>

</TABLE>
</CENTER>
</BODY>

</HTML>

```

In the above application, Session Scope was implemented. The following process happens in the background:

When the request comes for the first time to "account.jsp", on encountering useBean tag, the container checks in session scope i.e. Session object, whether any bean reference is available. As it is the first time request, no session object and no bean instance. Hence a session object is created, a session ID is associated with it and bean is instantiated. The reference in the bean instance is stored in the session object.

If data coming from user is stored in the bean as soon as setProperty tag is encountered and an Hypelink is sent to the browser. Along with this Response, Session ID also travels. When this Hypelink is clicked, session ID travels again from client to Server. Control comes to "Target.jsp", when the useBean tag is encountered, the Container checks the Session object for any bean reference, as it is already available, that bean reference is taken and the corresponding bean instance is contacted and data is retrieved using getProperty tags.

Hence, we can conclude that:

- If the bean is to be shared among jsp's, then increase the scope from default "page" scope, by using "scope" attribute in useBean tag.
- If scope is increased to "request", then the bean is available for two or more jsp's for that Request-Response cycle only. Once the Response is sent to browser, the bean reference and bean instance are garbage collected.
- If the scope is further increased to "session", then the bean instance and bean reference are available for two or more jsp's for that entire session. Only when the session expires, they are garbage collected.

→ Scope can be further increased to "application" scope. Then, the bean would be available to all the jsp's in the application as long as the application is running in the container. As soon as the application is undeployed, both the bean instance and its reference are garbage collected.

JSP Architectural Models :-

→ Sun Micro Systems Suggested two Architectural Models for developing Web Applications:

- 1) JSP Model I Architecture (Page Centric)
- 2) JSP Model II Architecture (MVC)

Any enterprise Web Application (real time online project), needs to have three different logics:

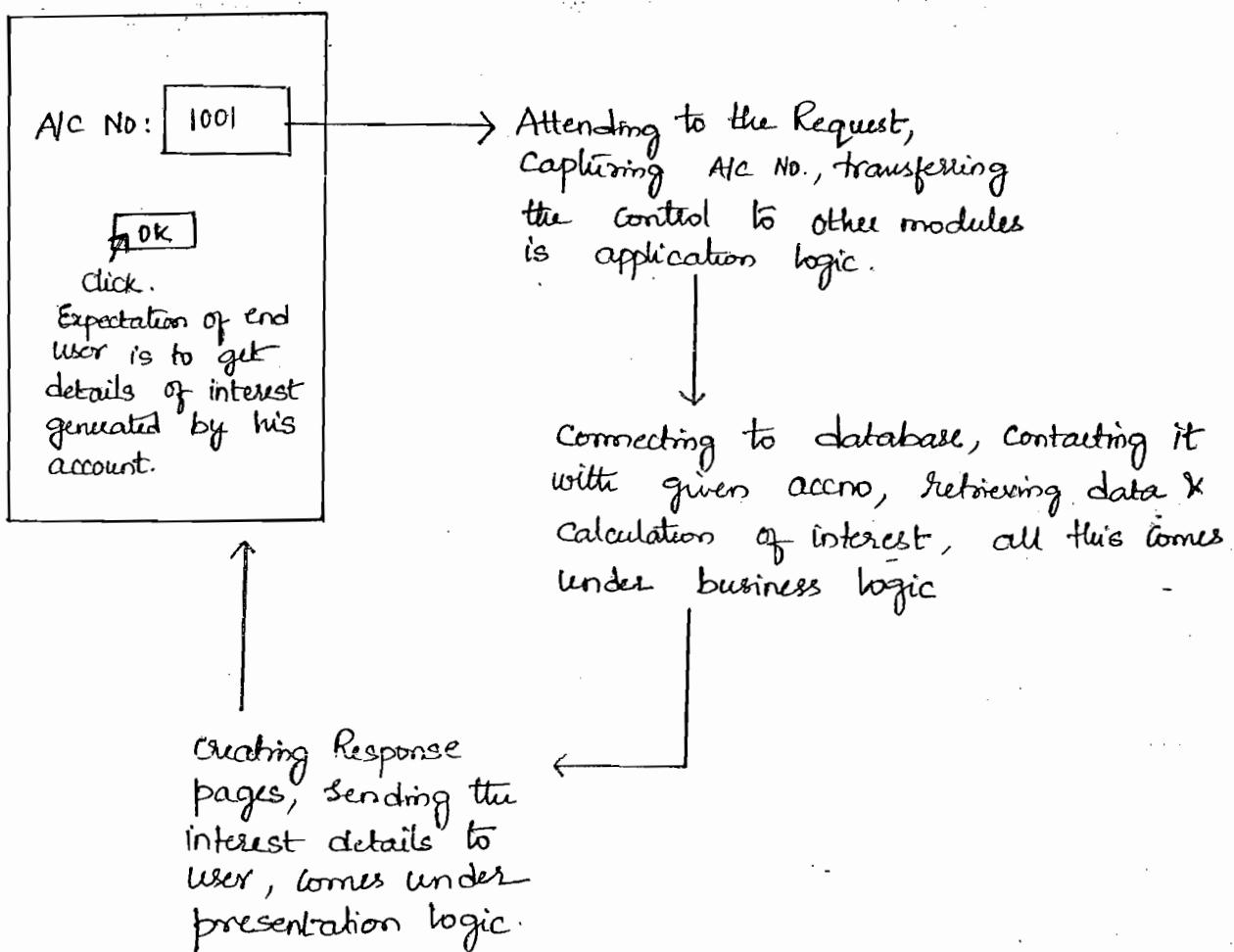
- 1) Application Logic
- 2) Business logic
- 3) Presentation Logic

* Receiving the request & capturing the user input, performing validation (if required) on user input (Server side validation only, not client side. For client side Validation Javascript is used) and controlling the flow in the application ; code written for the above tasks is nothing but Application logic.

* Communicating with database and processing data by applying the business rules ; code written for these tasks is known as Business logic. This is also known as Persistent Logic according to J2EE terms.

* Presentation logic is multi. +

Consider the following scenario:



In an application development all these logics must be developed independently but must be linked together. whichever technology is good at a particular area, we have to use that technology in that area i.e, JSP is good at presentation, Servlet is good at request handling etc.

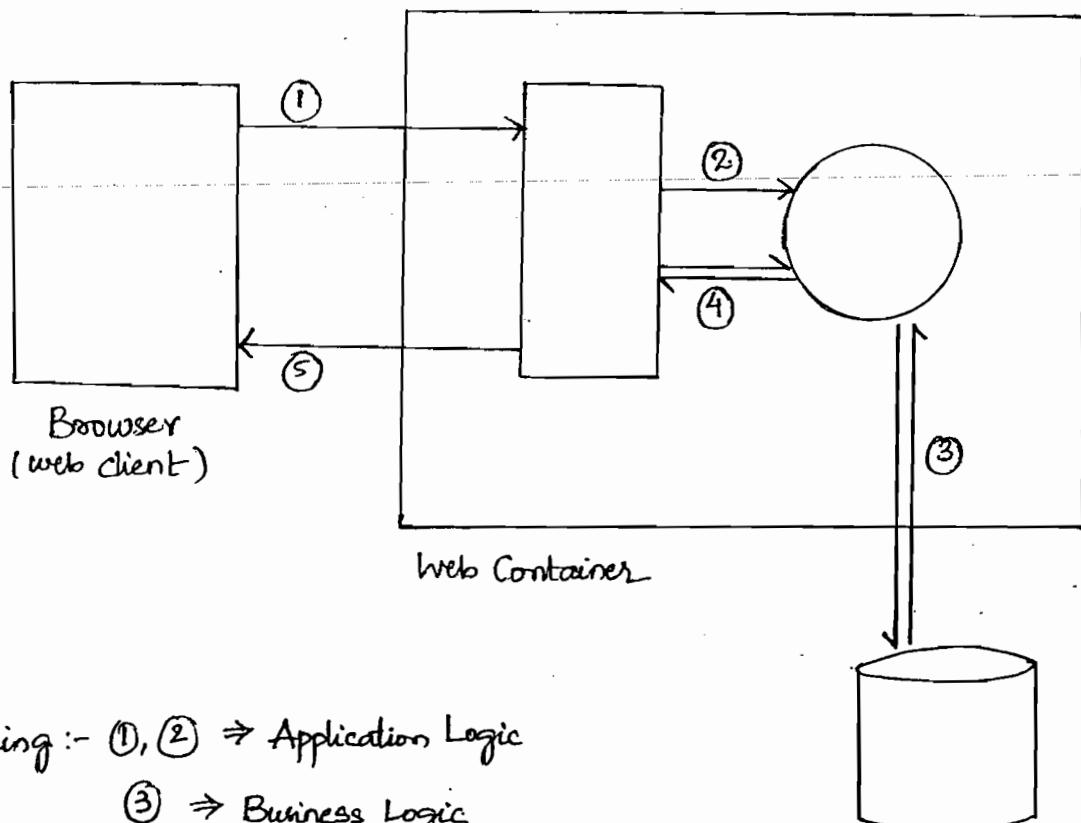
Implementation of such process is Divide & Rule (or) Divide & Conquer, is the main objective of the two Architectures i.e, MVC and Page Centric Architectures of JSP gives by sun Microsystems.

In JSP Model $\hat{I} \Rightarrow$ Page Centric \Rightarrow Everything revolves around jsp.
a jsp implements both Application logic and

Small sized Projects but not Suitable for Large scale projects

In JSP Model 2 \Rightarrow MVC \Rightarrow further improvement is achieved and all the three logics are separated from each other i.e. mixing of the logics is eliminated and pure application logic, pure business logic and pure presentation logic are developed as separate components. This is valid for large scale and very large scale projects.

Schematic Representation of Page Centric (JSP Model 2) style of a Web Application



Performing :- ①, ② \Rightarrow Application Logic

③ \Rightarrow Business Logic

④, ⑤ \Rightarrow Presentation Logic

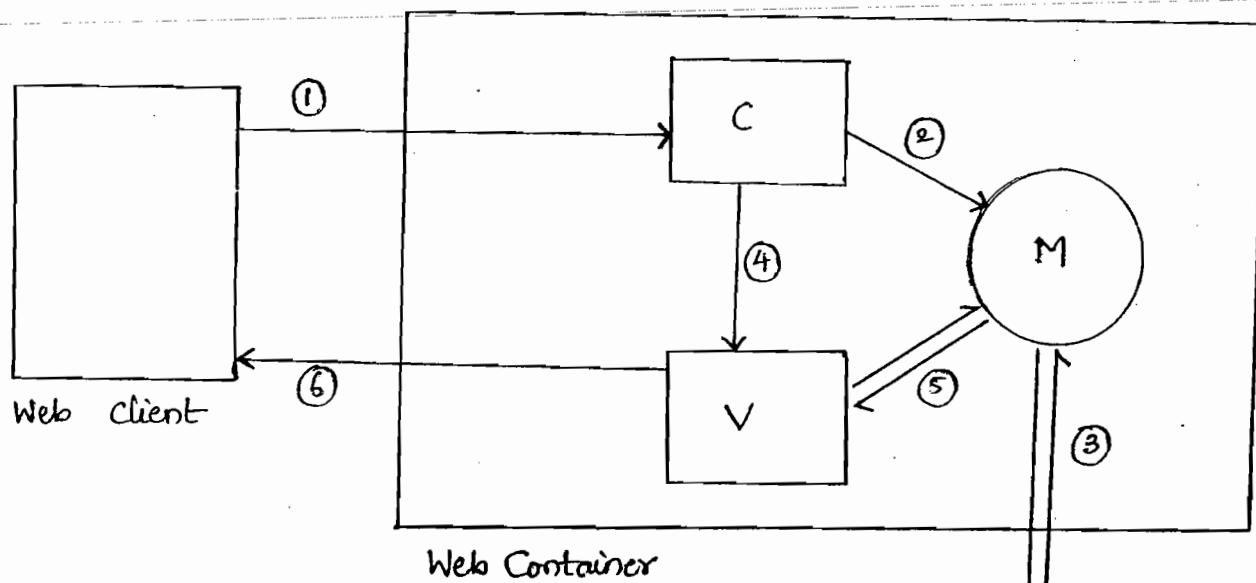
Database Server

- 1 :- jsp receives the client request and captures the user input
- 2 :- jsp instantiating the bean and populating the bean with form dat
- 3 :- bean contacting the database and getting data and processing that data by applying business rules
- 4 :- jsp contacting the bean for processed data
- 5 :- jsp generating the response and sending the same to the client

Hence, this model utilizes bean in separating business logic from other logics.

Note :- In this Architectural Model, business logic could be separated by application and presentation logic could not be. Therefore, it is not suitable for Larger and very Large Applications.

Schematic Representation of MVC (JSP Model II) style of a Web Application



Performing :- ①, ②, ④ \Rightarrow Application Logic

③ \Rightarrow Business Logic

④, ⑤ \Rightarrow P

- In this Architecture, application logic is implemented in the Controller, which is a Servlet
- Business logic is implemented in Model, which is a Java Bean (can also be an EJB OR Spring Bean)
- Presentation logic is implemented in View, which is a jsp.

14/09/09

- 1 :- Controller (Servlet) receiving the client request and capturing the form data
- 2 :- Controller instantiating the model and populating the model with form data.
- 3 :- model (Java Bean) contacting the database, retrieving data, processing data according to the business rules.
- 4 :- Controller switching the control to the view (jsp)
- 5 :- View(jsp) contacting the bean and retrieving processed data from the bean(model)
- 6 :- View(jsp) generating the response and sending the same to the client.

→ What are the benefits of MVC in a Web Application?

If a Web Application is developed according to the Model, View, Controller design pattern, clear separation of 3 concerns is achieved i.e, loose coupling between Application logic, Business logic and Presentation Logic is achieved.

It offers the following benefits.

- 2) Without Affecting the other layers, we can make modifications to any layer. Layer implies M or V or C layers
- 3) Code optimization is possible as we use suitable technology for the task.
- 4) No maintenance problems, development problems and administrative problems.
- 5) Code generation tools can be used for Application development which increases productivity and offers cost effectiveness for the applications.

View \Rightarrow jsp. 95% of jsp. is HTML. That 5% is to get data from model which may be a Bean or EJB or Spring Bean.

Model \Rightarrow A Bean, which is a pure Java class (i.e. independent of Servlet i.e. independent on request, response, PrintWriter etc) to perform database operations.

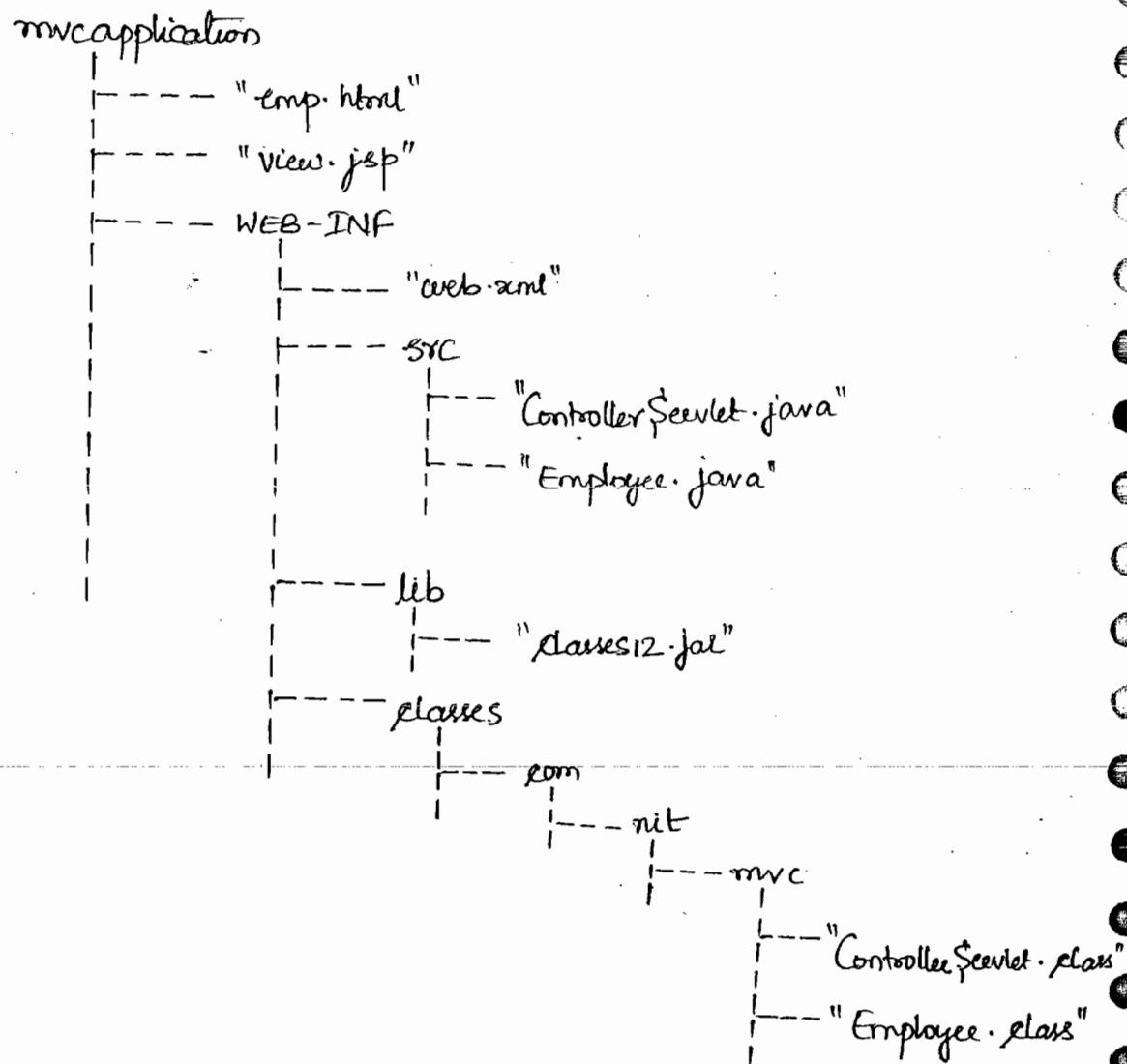
Controller \Rightarrow It is a Servlet and purpose of Servlet is only performed i.e. a pure Servlet operation. This is nothing but requests handling and switching control.

RAD \Rightarrow Rapid Application Development. This can be achieved if code generation tools are used instead of manually developing the code. Using these tools productivity improves.

Drawback in MVC :- If not designed in a proper manner i.e. when MVC for small projects is wasting the

→ Q) Develop & Deploy a Web Application that follows MVC.

Directory Structure :-



Use this command for Compiling both the Servlet & Bean:

javac -d *.*.java;
 ↓
 Spaces.

URL:-

http://localhost:8080/mvapplication

emp.html :-

```

1. <HTML>
2.   <BODY BGCOLOR = "cyan">
3.     <FORM ACTION = "./mvc">
4.       <CENTER>
5.         Employee Number : <INPUT TYPE = "text" NAME = "eno"> <BR> <BR>
6.         <INPUT TYPE = "Submit" VALUE = "Send">
7.       </CENTER>
8.     </FORM>
9.   </BODY>
10. </HTML>

```

web.xml :-

```

11. <web-app>
12.   <servlet>
13.     <servlet-name> controller </servlet-name> → from here to
14.     line : 14
15.     <servlet-class> com.mkt.mvc.ControllerServlet </servlet-class>
16.   </servlet>
17.   <servlet-mapping>
18.     <servlet-name> controller </servlet-name> → from here to
19.     line : 13
20.     <url-pattern> /mvc </url-pattern>
21.   </servlet-mapping>
22. </web-app>

```

↓
from here to
line : 70

↳ from this line the
control goes to line : 17

Employee.java :-

```

23. public class Employee implements java.io.Serializable
24. {
25.     Connection con;
26.     private int empno;
27.     private String name;
28.     private float salary;
29.     public Employee()
30.     {
31.         try
32.         {
33.             Class.forName("oracle.jdbc.driver.OracleDriver");
34.             con = DriverManager.getConnection("jdbc:oracle:thin:
35.                                         @localhost:1521:orcl", "scott", "tiger");
36.         }
37.         catch(Exception e) {}
38.         public void setEmpno(int empno) ← Control comes here from
39.         { this.empno = empno; results(); } → line: 74
40.         public int getEmpno()
41.         { return this.empno; } → from here, the control
42.         moves to line: 50
43.         public void setName(String name)
44.         { this.name = name; } → from here,
45.         public String getName()
46.         { return this.name; } → the control
47.         public void setSalary(float salary)
48.         { ... }

```

→ What is the purpose of param Standard Action?

- *) Param Standard action is a very rarely used Standard action.
- *) It is used to pass request parameters from one JSP to other JSP during include OR forward mechanisms

for Example :-

- this piece
of code is
written in,
say, "source.
jsp" {

```

<jsp:forward page = "target.jsp" > no '}' here.
<jsp:param name = "t1" value = "s01" /> It is not ended
<jsp:param name = "t2" value = "s02" />
</jsp:forward>
      ended here.
    
```

- *) Now, in "target.jsp" two request parameters are available in the request object.

Custom actions (tags)

- An action is a request processing time or runtime instruction to the container.
- We have two kinds of actions
 - 1) standard actions.
 - 2) custom actions.

Similarities between these types of actions are :

- *) both are actions i.e., tags

- * both have same purpose i.e., to use only tags in the foreground and to eliminate Java code from foreground. But in the background Java code only operates.
- * these tags follow XML syntax and are case sensitive.

→ What are the differences between standard actions and custom actions?

Standard actions

- 1) These are pre-created tags
- 2) Shipped along with the container
- 3) Meaning and functionality known to the container

It means that, tag is pre created & its corresponding functionality (Java code) is also pre defined.

Custom actions

- 1) user defined tags
- 2) we need to develop them
- 3) we need to explain to the container

Hence, we need to create the tag and corresponding functionality (Java code), that is to run in the background is also developed by us.

Hence, by sending the Java code to the background, we have several advantages like, parallel developing of Java code & presentation code, source code is invisible etc.

4) These tags will have "jsp" as prefix.

Eg:- <jsp:useBean>
 ↓ ↓
 prefix standard tag.

prefix ⇒ alias name or short

name to a Java library,

just like a package in Java language.

- prefix implies: "that particular tag belongs to which library".

4) can have any prefix**

* Any prefix implies : following some Standard set of rules these are developed.

5) Limited in functionality

Not flexible

5) Extensible

Much flexible

6) Common/ Standard in any application

whatever may be the application, the meaning of these tags remains the same.

6) application specific

These are developed according to the application and once developed for an application, cannot be used for another application.

tag :- It is an instruction to JSP Engine

A tag has the following Syntax (similar to XML) :-

<prefix:tagname attribute1="value1" attribute2="value2" ...>

tld :- Tag Library Descriptor. It is an XML file with ".tld" extension. In this file, we specify the properties of the tags. We associate the tag with functionality providing code. We

Tag handler :- It is a specialized Java class that provides functionality for the tag.

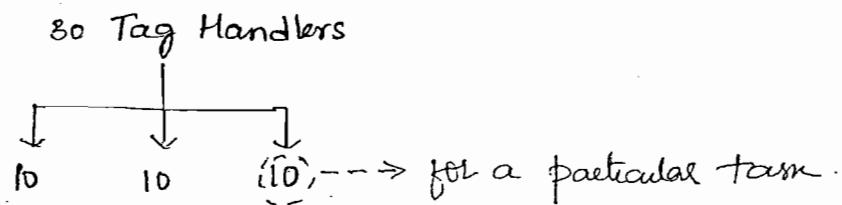
i.e. it is the functionality providing code that runs in the background.

Tag library :- It is a collection of tag handlers and tld file.

tld file is one per library.

It is a collection of all the tag handlers and the unique tld.

Consider :



⇒ 30 Java classes

3 TLDs

3 tag libraries.

⇒ 1 TLD + 10 Java classes = 1 tag library.

Steps to Develop & Use a Custom Tag :-

Step 1 :- Develop the tag handler (a Java class)

Step 2 :- Develop the tld file

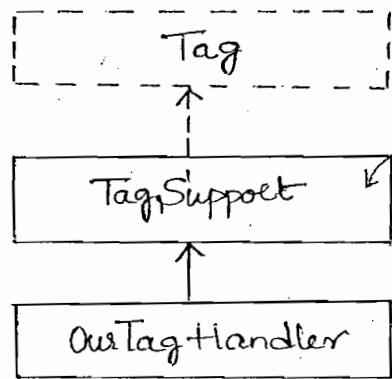
Step 3 :- Deploy the tag library (".class" file into classes directory,
".tld" file in WEB-INF directory)

Step 4 :- Register the tag library with the web application (in
web.xml by using <taglib>...</taglib>)

Step 5 :- Import the tag library into the jsp and make use

Tag Handler Development:

→ A tag handler is a Container Managed public java class that implements javax.servlet.jsp.tagext.Tag interface (directly/ indirectly)



its life cycle is managed by JSP engine

Adapter class implements Tag interface. It has dummy definitions for all methods coming from Tag. This is just as in the case of GenericServlet. Hence whatever methods are required to be overridden by our Tag Handler, only those methods are defined.

Contribution of Tag interface:

- ⇒ It makes a class implementing it act as a Tag Handler
- ⇒ It provides Tag Handler life cycle methods

→ Tag interface provides the following life cycle methods:

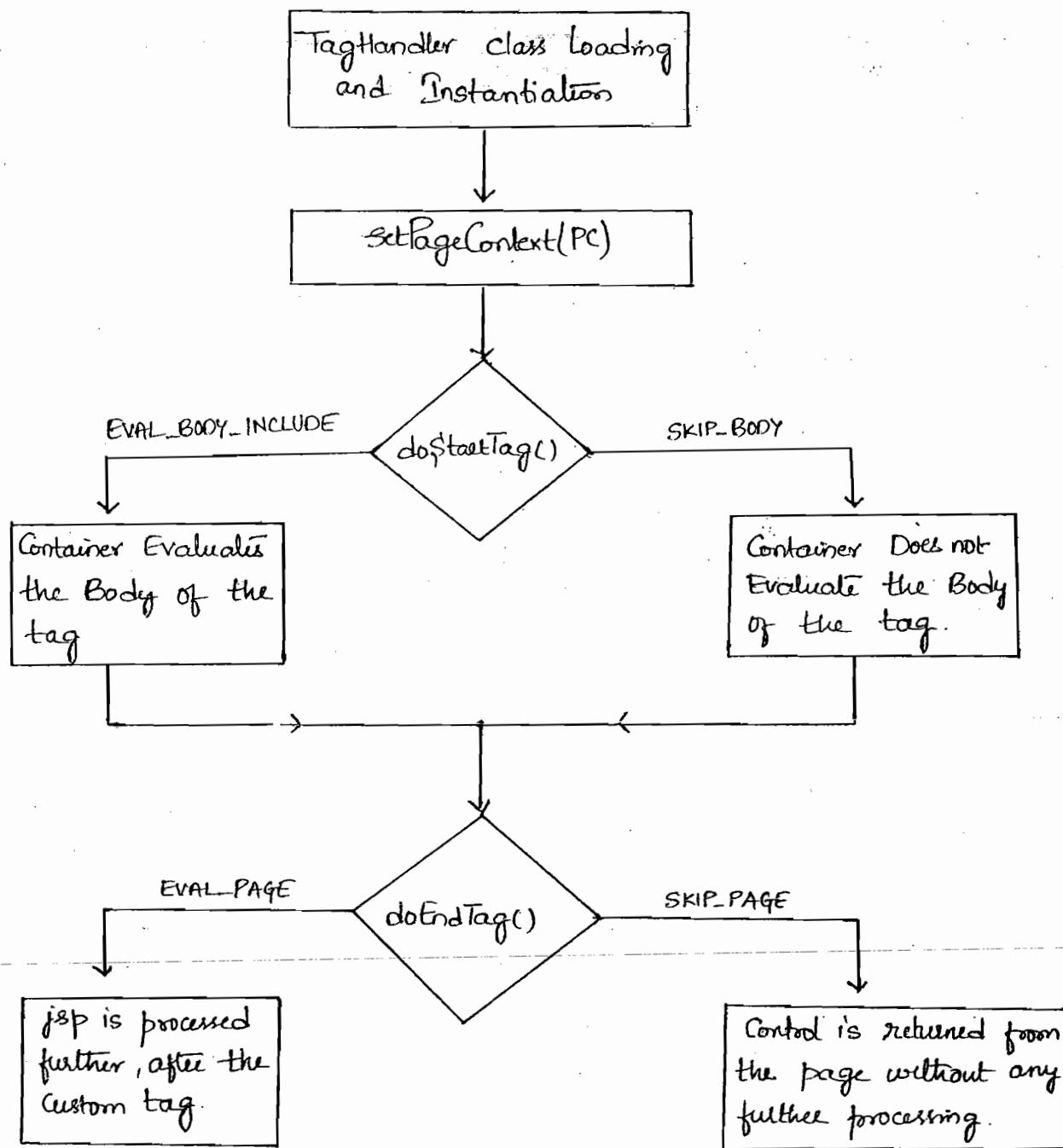
- 1) setPageContext (PageContext)
 - 2) int doStartTag()
 - 3) int doEndTag()
 - 4) release()
- Abstract
in Tag
interface

have a
dummy
definition in
TagSupport class.

Note:- for all these methods, implementation is given in TagSupport class.

→ In our tag handler class, we don't define the first method. we

- Whenever custom tag is encountered, doStartTag() method is implicitly called. In this method, we implement the functionality providing code for the custom tag.
- If we want to perform any task when container encounters the end tag, we implement that code in doEndTag(). Releasing the resources is done in release() method. Just before Tag Handler instance is garbage collected, the release() method is called.
- In Tag interface, we have the following 4 constants in addition to the life cycle methods
 - 1) EVAL_BODY_INCLUDE
 - 2) SKIP_BODY
 - 3) EVAL_PAGE
 - 4) SKIP_PAGE
- All these are integer constants
- doStartTag() returns any one of the first two constants to the container
- doEndTag() returns any one of the last two constants to the container.
- These constants act as indicators to the JSP engine what to do next.



Consider the following scenario :

my.jsp.jsp

A Custom Tag

```

<myprefix:mytag> hello </myprefix:mytag>
      |           |
      ↓           ↑
     body of      end of the Tag.
    tag
  
```

→ As soon as this is encountered

↓ next

Complete Environment is encapsulated and is handed over to TagHandler (just as initializing the Scarlet using ScarletConfig object)

↓ next

doStartTag is implicitly called. Its definition is executed. This method returns a number to the container. This number indicates the container, what to do next.

If we feel that after this, the body of tag (hello) should not be evaluated, we say: return SKIP_BODY;

If we feel that the body should be evaluated, we say:
return EVAL_BODY_INCLUDE.

↓ next

end of tag is encountered. doEndTag is implicitly called. The body of this method is executed.

If we feel that the remaining jsp is to be executed

after the custom tag, we say: return EVAL_PAGE;

If we feel that the remaining jsp should not be executed
after the custom tag, we say: return SKIP_PAGE;

TagSupport class would be something like:

```
public class TagSupport implements Tag
{
```

 PageContext pageContext;

```
    public void setPageContext(PageContext pc)
    { pageContext = pc; }
```

*just as pageConfig in
call of GenericServlet*

*This is created by container.
It is local here and hence
available inside the container.*

```

public int doStartTag()
{
    return SKIP_BODY;
}
public int doEndTag()
{
    return EVAL_PAGE;
}
}

```

```

public class MyTagHandler extends TagSupport
{

```

// all the 4-methods with Dummy definitions are literally placed here.

```

public int doStartTag()
{
    // task performing code
    return SKIP_BODY;
}
}

```

This method overrides
the doStartTag() coming
from parent.

```

}

```

→ What is PageContext?

- * Object Oriented representation of the entire environment of the jsp is nothing but PageContext object
- * PageContext object encapsulates all other implicit objects
 - It provides getter methods for each implicit object

for Example:

```

MyTagHandler
{

```

```

    int doStartTag()
    {

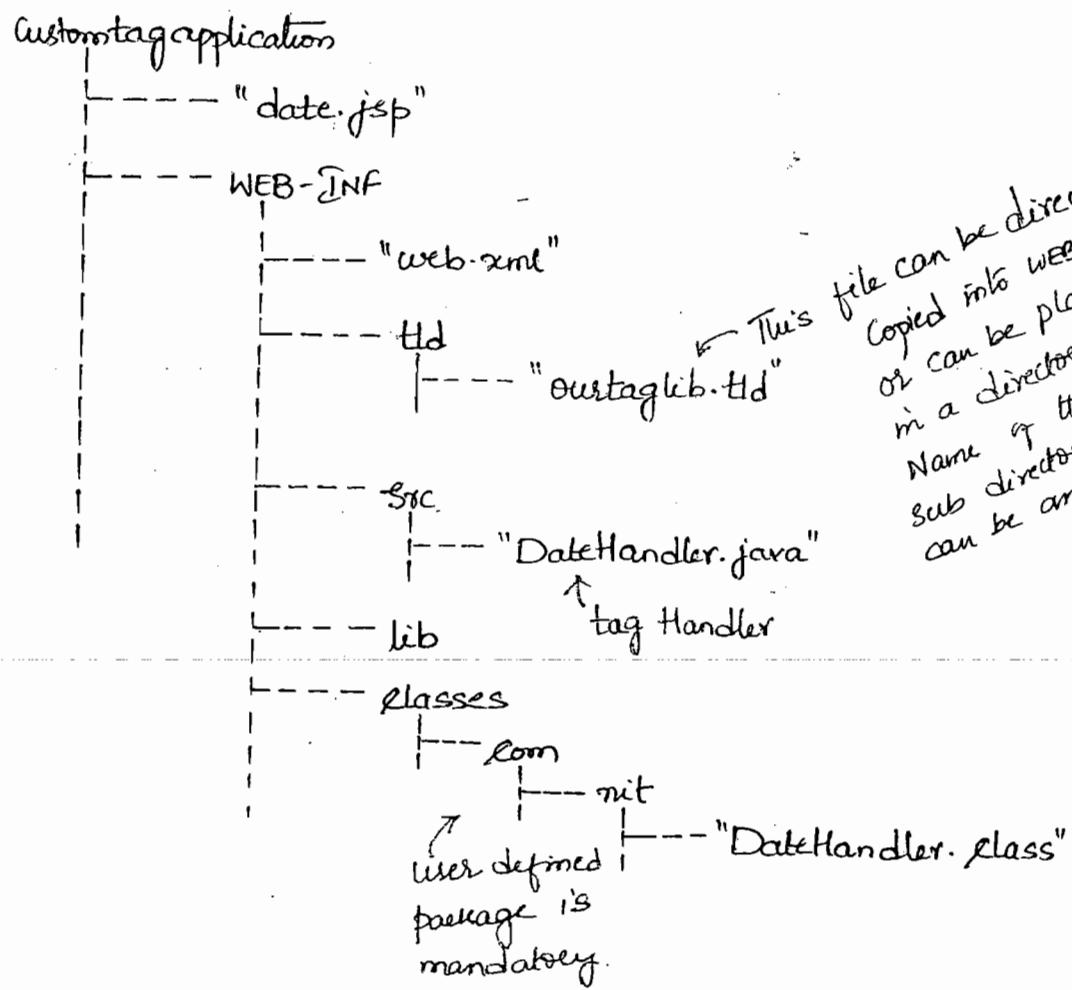
```

pageContext.getRequest().getParameter("...");

→ getter method.

→ Q) Develop & Deploy A Web Application in which a custom tag is developed and used in a jsp.

Directory Structure :-



URL :-

`http://localhost:8081/Customtagapplication/date.jsp`

DateHandler.java (our own tag handler Development) :-

package com;

```

import javax.servlet.jsp.tagext.*;
import java.util.Date;
import java.text.DateFormat;
import java.io.*;

public class DateHandler Extends TagSupport
{
    public int doStartTag() throws JspException
    {
        Date today = new Date();
        DateFormat dft = DateFormat.getDateInstance(DateFormat.LONG);
        String date = dft.format(today);
        try
        {
            JspWriter out = pageContext.getOut();
            out.println(date);
        }
        catch (IOException e)
        {
            System.out.println(e);
        }
        return SKIP_BODY; // since there is no body in the
                        // custom tag, we use SKIP-BODY.
    } // doStartTag()
} // class.

```

If EVAL-BODY-INCLUDE is written,
it throws Exception.

Note:- Before we compile the Tag Handler, we need to place "jsp-api.jar" file in classpath
for Example:

ourtaglib.tld (tag library descriptor development) :-

→ This file has two sections

- 1) tags description Section.
- 2) tag library description Section

<taglib>

```

<tlib-version> 1.0 </tlib-Version>
<jsp-version> 2.0 </jsp-Version>
<short-name> nit </short-name>
<tag>
  <name> date </name>
  <tag-class> com.nit.DateHandler </tag-class>
  <body-content> empty </body-content>
</tag>

```

</taglib>

web.xml :-

```

<web-app>
  <taglib>
    <taglib-uri> http://www.nit.com/customtags
    <!-- This can be any name.
        Here, industry standard
        is being used. -->
    <taglib-location> /WEB-INF/tld/ourtaglib.tld
  </taglib>
</web-app>

```

date.jsp :-

<HTML>

<BODY BGCOLOR = WHEAT>

<%@ taglib prefix = "nit" uri = "http://www.nit.com/customtags" %>

using taglib
directive to import
a tag library

two
attributes of
taglib directive

<H2> Today's Date Is : <nit:date /> </H2>

</BODY>

custom tag

</HTML>

18/09/09

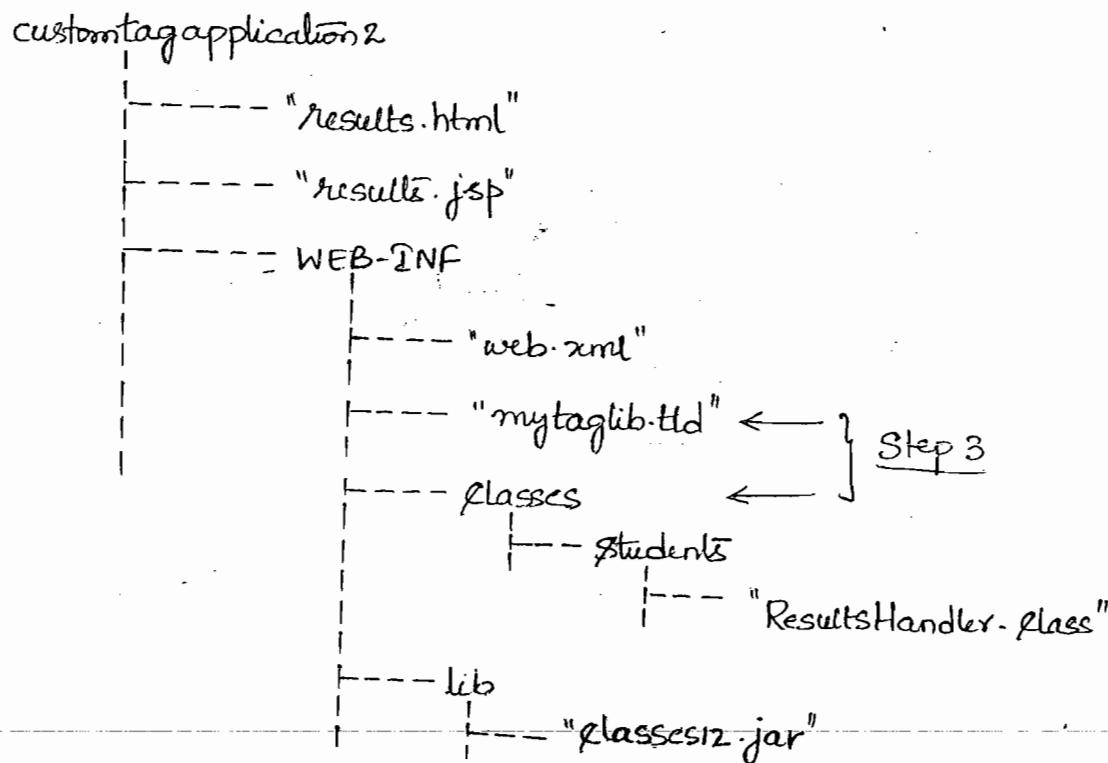
→ What happens in the background when jsp engine encounters the custom tag?

- 1) Container picks up the prefix of the custom tag.
- 2) This prefix is compared with taglib directive's prefix.
- 3) When two prefixes match, Container picks up the tag uri value.
- 4) Container compares it with web.xml entry.
- 5) Once the match is found, Container picks up the tld location.
- 6) Container looks for the encountered custom tag match in the tld.
- 7) Container starts the life cycle of the corresponding tag

This can be any name.
Industry standard is used in this case. This should match with web.xml entry

→ Q) Develop & Deploy a web Application in which, a custom tag is developed and used in a jsp.

Directory Structure :-



URL :-

http:// localhost:8081/ customtagapplication2/ results.html

results.html :-

<HTML>

<BODY BGCOLOR = CYAN>

<H2> Entrance Exam Results </H2>

<FORM ACTION = "results.jsp">

Hall Ticket No: <INPUT TYPE = "text" NAME = "htNo">

<INPUT TYPE = "Submit" VALUE = "Get Results Here">

results.jsp :-

1. <HTML>
2. <BODY BGCOLOR = CYAN>
3. → <%@ taglib prefix="nit" uri="customtags" %>
4. Step 5 <H2> Entrance Score Card </H2>
5. → <H3> <nit: results /> </H3> ← prefix of this line &
prefix of line: 3 are
compared. If matches
pick up uri value
from line: 3 and
control goes to web.xml
line: 10
6. </BODY>
7. </HTML>

web.xml :-

8. <web-app>
9. <taglib> ← Step 4
uri value of line: 3 & line: 10
are compared. If match four
pick up location from
line: 11
10. <taglib-uri> customtags </taglib-uri>
11. <taglib-location> /WEB-INF/mytaglib.tld </taglib-locate
↑
from here
the control goes to
mytaglib.tld i.e. line: 14
12. </taglib>
13. </web-app>

mytaglib.tld :- ← Step 2

14. <taglib> ← from here
control goes to
line: 17
15. <tlib-version> 1.0 </tlib-version>
16. <jsp-version> 1.2 </jsp-version>
17. <short-name> nit </short-name> ← this 'short name'
is compared with
line: 5 prefix and
then control goes to
line: 19
18. <tag>
19. ... > <name> results </name>

21. <body-content> empty </body-content>
 22. </tag>
 23. </taglib>

ResultsHandler.java :- ← Step 1

```

24. package students;
25. import javax.servlet.jsp.*;
26. import javax.servlet.jsp.tagext.*;
27. import java.sql.*;
28. public class ResultsHandler extends TagSupport
29. {
30.     Connection connection;
31.     public ResultsHandler()
32.     {
33.         try
34.         {
35.             Class.forName("oracle.jdbc.driver.OracleDriver");
36.             System.out.println("Driver Loaded");
37.             connection = DriverManager.getConnection("jdbc:oracle:
38.                 thin:@localhost:1521:Sequel", "scott", "tige");
39.             System.out.println("Connection Established");
40.         }
41.         catch (Exception e) { }
42.     }
43.     public int doStartTag() throws JspException
  
```

```

44.     try
45.     {
46.         JspWriter out = pageContext.getOut();
47.         String htNo = pageContext.getRequest().getParameter("htNo");
48.         Statement st = connection.createStatement();
49.         ResultSet rs = st.executeQuery("SELECT * FROM ENTRANCE
50.                                         WHERE HTNO = " + htNo);
51.         out.println("HALL TICKET NUMBER :" + htNo);
52.         out.println("CANDIDATE NAME :" + rs.getString(2));
53.         out.println("MARKS :" + rs.getString(3));
54.         rs.close();
55.         st.close();
56.     }
57.     catch(Exception e) {}
58.     return SKIP_BODY;
59. } // doStartTag()

```

public void release() {
 This method is
 called only when
 application is undeployed.
 After this method is
 called, all the instances
 are garbage collected.

```

60.     {
61.         try
62.         {
63.             if (connection != null)
64.                 connection.close();
65.             System.out.println("Connection closed");
66.         }
67.         catch(Exception e) {}
68.     } // release()

```

Note:- Before compiling the above class:

SET CLASSPATH = D:\Tomcat 5.5\common\lib\servlet-api.jar; D:\
Tomcat 5.5\common\lib\jsp-api.jar; %classpath%

Compile as: javac -d ~~bin~~ ResultHandler.java
 ↪
 ↓
 Spaces

Custom Tag with Attributes :-

Dealing with attributes in a JSP involves the following steps:

Step 1:- In the Tag Handler class, declare one private variable and one public setter method for each attribute of the Custom Tag.

Step 2:- In the tld file, make an entry for each attribute within the <tag> element.

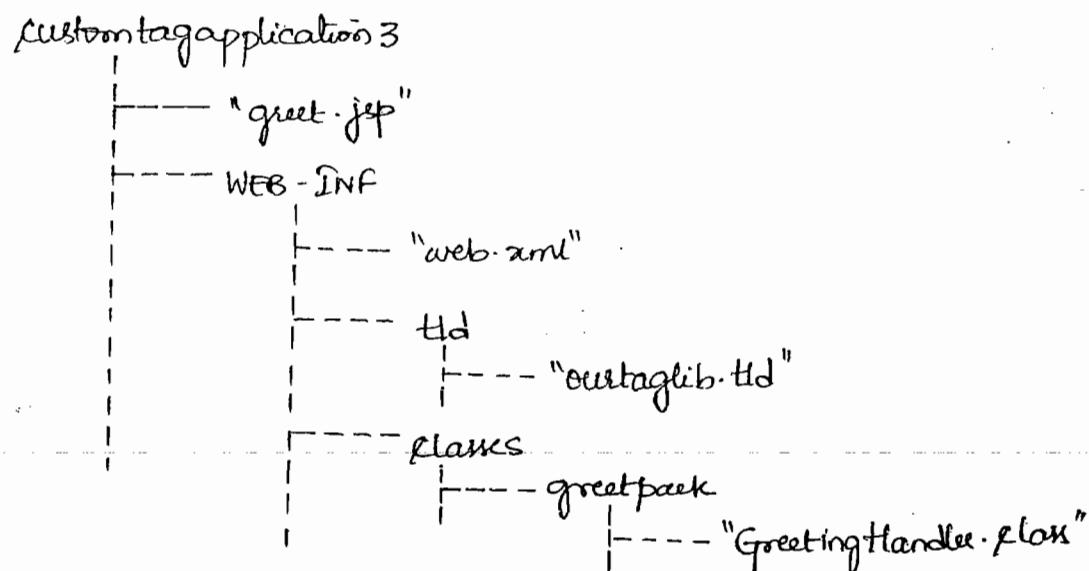
<attribute> element is used for this purpose. This element has three sub elements

- 1) <name> :- It specifies the name of the attribute
- 2) <required> :- It specifies whether the attribute is mandatory or optional. If we place "true" value for it, it means that whenever that tag is used in a JSP, this attribute should be used for that tag
- 3) <rtexprvalue> :- It is used to specify the way we can supply value to this attribute. If "true" is specified, we can use

for this attribute. If the value is "false", we can supply value to the attr only statically.

-) Web Application in which a jsp makes use of a custom tag that has Attributes.

Directory Structure :-



URL :-

http://localhost:8081/customtagapplication3/greet.jsp

greet.jsp :-

<HTML>

<BODY BGCOLOR=WHEAT>

<%@ taglib prefix="nit" uri="http://www.nit.com/customtags" %>

<H2> <nit:greet name="Ram" age="25" /></H2>

</body>
</HTML>

The values for the attributes can be dynamically supplied using a JSP expression OR JSTL. The values coming from client can be supplied to these, which is a dynamic procedure

web.xml

```
<web-app>
  <taglib>
    <taglib-uri> http://www.nit.com/customtags </taglib-uri>
    <taglib-location> /WEB-INF/tld/ourtaglib.tld </taglib-location>
  </taglib>
</web-app>
```

ourtaglib.tld :-

```
<taglib>
  <tlib-Version> 1.0 </tlib-Version>
  <jsp-Version> 2.0 </jsp-Version>
  <short-name> nit </short-name>
  <tag>
    <name> greet </name>
    <tag-class> greetback.GreetingHandler </tag-class>
    <body-content> empty </body-content>
    <attribute>
      <name> name </name>
      <required> true </required>
      <rtpvalue> true </rtpvalue>
    </attribute>
  </tag>
</taglib>
```

307.

```

        → by default it
        would be false.

<required>(true)</required>
<rtexprvalue> true</rtexprvalue>
</attribute>
</tag>
</taglib>

```

GreetingHandler.java :-

```

package greetpack;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.io.*;

```

```

public class GreetingHandler extends TagSupport
{

```

```

private String name; } first these variables are created
private int age; } × setPageContext() method is called
                     implicitly

```

```

public void setName(String name)
{
    this.name = name;
}

```

```

public void setAge(int age)
{
    this.age = age;
}

```

These two methods are
called automatically, before
doStartTag() method
just after setPageContext()
method is called

```

public int doStartTag() throws JspException
{

```

key
{}

```
        catch( IOException e )
        {
            System.out.println(e);
        }
        return SKIP_BODY;
    }
} // class
```

1) When using RequestDispatcher, the use of which methods can lead to IllegalStateException?

- a) getWriter
- b) getOutputStream
- c) flush ← If this method is used before RequestDispatcher, this Exception occurs.
- d) getNamedDispatcher

Ans:- (c)

2) Which types define Attribute methods?

- a) HttpSession
- b) ServletConfig
- c) ServletRequest
- d) ServletContext

Ans:- (a), (c), (d)

3) GET request is idempotent. Justify.

Ans:- "Even if repeated requests are sent, no side effects at the server side" — nature is known as Idempotency. GET request is idempotent. Post request is not idempotent because, with it, repeated requests have side effects at server side.

4) How to deal with Servlets and thread safety?

- a) Write a servlet that implements SingleThreadModel ← Old Solution
- b) Use local variables and if you have to use instance variables, synchronize access to them

Ans :- (a), (b)

5) Which of the following is not thread Safe?

- a) HttpServletRequest
- b) ServletContext
- c) HttpSession
- d) HttpServletResponse

Ans :- (b), (c)

6) How to Register a listener in a Web Application?

Ans :- <listener>

<listener-class> ListenerClassName </listener-class>

</listener>

7) Which Statement about HttpSession Object is true?

- a) A session whose timeout period has been set to -1 will never expire
- b) Session becomes invalid as soon as the user closes the browser
- c) A session will become invalid after session timeout period is lapsed
- d) Session object is not thread safe

Ans :- (a), (c), (d)

8) Which of the following is used in implementation of Cookies?

- a) ServletRequest ←
- b) HttpServletRequest
- c) HttpServletResponse
- d) Cookie

It is of GenericServlet X
GS cannot implement
Session Tracking nor
cookies. They are concepts
of HttpServlet

Ans :- (b), (c), (d)

9) Which of the following creates a new session if one does not exist?

- a) getSession() → Both have same
- b) getSession(true) → functionality
- c) getSession(false) →
- d) createSession() → It returns 'null'

Ans:- (a), (b)

10) What should be the name of the cookie that carries session from server?

Ans:- "JSESSIONID"

11) How to know about number of active users for the web Application?

Ans:- By implementing HttpSessionListener or HttpSession event handler

12) What is the limit for the size of a cookie?

Ans:- 4K

13) What is the maximum number of cookies that can be stored in a client machine?

Ans:- 300 is the maximum number. If crossed, old ones will be deleted

14) When is ServletContextEvent fired? → means raised (or) generated.

Ans:- As soon as the application is deployed into the container

15) How to Configure a listener in a web application?

Ans:- It is same as Registering a listener

i.e. <listener>

```
<listener-class> ClassName </listener-class>
</listener>
```

16) What is the difference between getRequestDispatcher() method of request and context?

Ans:- getRequestDispatcher() method of request expects from roots and context does not

17) What is the difference between doFilter method of FilterChain and filter?

Ans:- doFilter of filter is a life cycle method whereas doFilter of FilterChain just passes the control from filter to filter (or) filter to Servlet.

18) Which is the interceptor in request response cycle of a Servlet?

Ans:- A filter is the interceptor

19) How to know in a Servlet about the type of HTTP request method?

Ans:- String m = request.getMethod();

20) How to convert a session cookie into a persistent cookie?

Ans:- By using cookie.setMaxAge();

```

48.     public float getSalary()
49.     { return this.salary; }

50.     public void results()
51.     {
52.         try
53.         {
54.             Statement s = con.createStatement();
55.             ResultSet rs = s.executeQuery ("SELECT * FROM EMPLOYEE
56.                                         WHERE EMPNO = " + empno);
57.             rs.next();
58.             bean = new Employee();
59.             bean.name = rs.getString ("name");
60.             bean.salary = rs.getFloat ("salary");
61.             bean //try
62.             catch (Exception e) {} }
63.         } // results() → from here control goes to line: 39
64.     } // class

```

ControllerServlet.java :-

```

63. package com.nit.mvc;
64. import javax.servlet.*;
65. import javax.servlet.http.*;
66. import com.nit.mvc.Employee;
67. import java.io.*;

68. public class ControllerServlet extends HttpServlet
19

```

70. → public void doGet (HttpServletRequest request,
 Control comes here HttpServletResponse response) throws IOException,
 ServletException

71. }

72. int empno = Integer.parseInt (request.getParameter ("eno"));

73. Employee ebean = new Employee(); ^{**} instantiating bean

74. from here → ebean.setEmpno (empno); ^{**} populating bean

75. goes to line: 38 getServletContext(). setAttribute ("ebean", ebean);

76. control moves to line: 79 getServletContext(). getRequestDispatcher ("/view.jsp"). forward (request, response);

77. is JSP ? }

78. }

^{**} If a bean is used in Servlet directly (ie, not in a JSP), we only need to instantiate & populate it instead of container doing them.

View.jsp :-

<HTML>

<BODY BGCOLOR = "Yellow">

EMPLOYEE DETAILS

<%@ page import = "com.nit.mvc.Employee" %>

<jsp:useBean id = "ebean" class = "com.nit.mvc.Employee"
 scope = "application" />

for these, { EMPNO : <jsp:getProperty name = "ebean" property = "empno" />

 get the NAME : <jsp:getProperty name = "ebean" property = "name" />

 values are called SALARY : <jsp:getProperty name = "ebean" property = "salary" />
 retrieved </BODY>

</HTML>

79.

80.

81.

82.

83.

84.

85.

86.

87.

88.

At line: 75, Servlet is storing the bean reference in
 + ... - - - which means that bean instance is never

29/08/09

Workshop : Servlets

1) What is the outcome of the following code of doGet method?

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

```
PrintWriter out = response.getWriter();
```

out.flush(); → The response is committed i.e. sent to client

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

System.out.println(response.isCommitted()); → it returns 'true' as the response was already committed, hence true

```
response.setContentType("name/value"); is printed on console.
```

a) IllegalArgumentException is thrown.

b) Blank page is written into client.

→ Since no content is sent back to browser i.e. no out.println("") is written in the code.

c) true is printed on server console

d) false is printed on server console

Ans :- (b), (c)

2) What is the outcome of the following code of doGet method?

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

```
response.setContentLength(4); ← After first 4 characters, output  
PrintWriter out = response.getWriter(); is committed i.e. sent to browser.
```

```
out.println("What is printed?");
```

```
out.println("is response committed :" + response.isCommitted());
```

```
System.out.println("is response committed :" + response.isCommitted());
```

a) Blank page is returned to the client

b) "What" is printed to the client

c) is response committed : true is printed to the client

d) is response committed : false is printed to the client

3) If we want to send an image from Seerlet to the client, what are the wrong statements in `doGet` method?

- `response.setContentType("text/html");` this should be "image/gif"
- `PrintWriter out = response.getWriter();` Character oriented Stream. Hence only plain text can be sent or
- `ServletOutputStream out = response.getOutputStream();` HTML text.
- `response.setContentType("text/plain");`

Ans :- (a), (b), (d)

4) Which of the following are likely to be found as request header fields?

- `accept`
 - `accept-language` c) None
 - `Client-agent` would have been correct if it was user-agent. user \Rightarrow Brower Manufacturers like net-scape, iexplore, mozilla etc
 - `rely-after` It is a response header
- Ans :- (a), (b)

5) Which of the following statements are incorrect?

- Seerlet Container creates seerlet instance at application Start up
 - Seerlet Container creates Seerlet instance at application Seever Start up This can be achieved using the tag <load-on-startup>
 - Seerlet Container creates Seerlet instance when the client request comes for the first time This is true
 - Seerlet Container creates Seerlet instance for each client request
- Ans :- (b), (d)

6) Which of the following are correct in web.xml?

a) <filter-mapping>

```
<filter-name> hello </filter-name>
<servlet-name> HelloServlet </servlet-name>
```

</filter-mapping>

} filter can be applied
this way also, by
using the registration
name of Servlet

b) <filter-mapping>

```
<filter-name> hello </filter-name>
```

```
<url-pattern> /HelloServlet </url-pattern>
```

</filter-mapping>

} Applying the filter, by
using the public URL
name of Servlet

c) <filter-mapping>

```
<filter-name> hello </filter-name>
```

```
<url-pattern> HelloServlet </url-pattern>
```

</filter-mapping> " is missing

d) <filter-mapping>

```
<filter-class> MyFilter </filter-class>
```

```
<url-pattern> /HelloServlet </url-pattern>
```

</filter-mapping>

filter class is not used

Ans:- (a), (b)

Q Which of the following statements is wrong?

- The order of filters in chain is arbitrarily determined by web container
- Only URL patterns can be used by filters to target specific web resources
- We must define doFilter(request, response) method to pass the control to other filter or Servlet
- filter cannot send the response to the client

order is determined by the
order in which they
are registered in
web.xml

but Registration names of
Servlets can also be used
as can be seen in
the question

8) Which of the following statements will make the session expire after 1 minute?

→ Expires in '1' second

a) `session.setMaxInactiveInterval(1);`

b) `<session-config>` → Expires in '1' minute

`<session-timeout> 1 </session-timeout>`

`</session-config>` → Expires in '60' seconds

c) `session.setMaxInactiveInterval(60);`

d) `<session-config>` → Expires in '60' minutes

`<session-timeout> 60 </session-timeout>`

`</session-config>`

Ans:- (b), (c)

9) What is the default implementation of Service method in GenericServlet

Ans:- In GenericServlet, Service method is Abstract. It does not have any implementation

10) What is the default implementation of doGet method in HttpServlet?

Ans:- It has a code, printing Error message saying that Get request is not supported by this URL

Eg:- public class MyServlet extends HttpServlet {

 public void doGet() {

 System.out.println("Error: ---");

}

}

incoming request is POST
but the method defined
is Get, hence, in
this case the default
implementation is
executed.

<form ACTION = ".abc" METHOD = "POST">

11) Which method of HttpServlet is abstract?

Ans:- No method in HttpServlet is abstract.

12) Why to implement event handling in Web Applications?

Ans :- To perform a customized task whenever some event is raised in the Application.

13) How to implement event handling in Web Applications?

Ans :- Defining A listener class and Registering it with Web Application.

14) Servlets are by default pre-initialized (true/false)

Ans :- false.

before
client request
comes.

Servlets are initialized only after
request comes from client.

15) Filters are by default pre-initialized (true/false)

Ans :- true.

filters are initialized as soon as the
application is deployed i.e even before
client request comes from the client.

16) As soon as the Web Application is deployed, I want to do
Something in the Web Application. Which Listener Should be
implemented?

Ans :- ServletContextListener.

17) What are the methods of Servlet interface?

Ans :- init (ServletConfig)

service (req, res)

... .

`getServletConfig()`

18) Which object is used by a Servlet to communicate with the web container?

Ans:- Servlet uses `ServletContext` object to communicate with Container i.e., using `log()`, `getServletInfo()` methods on `ServletContext` object

19) Which of the following statements are true?

- `ServletConfig` object is one per web Application
- `ServletConfig` object is one per Servlet
- `ServletContext` object is one per web Application
- `ServletContext` object is one per Servlet
- `request` object is one per Servlet

Ans:- (b), (c)

20) What is a Session cookie?

Ans:- Cookie that is not stored in Client Machine's file system.
It dies as soon as Browsing Session ends.

As soon as user login happens, Session object is created
hence an event is raised. It can be handled by `HttpSessionListener`

31/07/09

ResultSetMetaData (RSMD)

- Q) Develop A Stand Alone JDBC Application that takes table Name from the Command line and Displays All the Records of that table along with Column Names.

Since the ResultSet Object directly does not give the column details i.e, there are no methods to retrieve column Name No.of Columns, Precision etc, we go for ResultSetMetaData and use its Object. This object is created by calling getMetaData() method on ResultSet object. ResultSetMetaData has methods to manipulate column Details.

// Stand Alone Application for Dynamic Table Retrieval.

```
import java.lang.*;  
import java.sql.*;
```

```
Class DynamicTableRetrieval
```

```
{
```

```
public static void main( String[] args ) throws Exception  
{
```

```
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

```
    Connection con = DriverManager.getConnection("jdbc:  
        odbc:praveen", "scott", "Tiger");
```

```
ResultSet rs = st.executeQuery("SELECT * FROM " + args[0]);
```

```
ResultSetMetaData rmd = rs.getMetaData();
```

```
int cc = rmd.getColumnCount();
```

→ This method
Returns no. of columns
present in the
Table.

↓
Table Name
is taken from Command
line since we
don't know the Table
Name at the moment
of writing this
Application.

```
for (int i=1; i<=cc; i++) // loop to print column names.
```

```
System.out.print(rmd.getColumnLabel(i) + "\t");
```

```
System.out.println();
```

```
System.out.println("-----");
```

```
while (rs.next()) // outer loop to fetch records one after another
```

```
{
```

```
for (int i=1; i<=cc; i++) /* inner loop to fetch column values  
of a particular record */
```

```
System.out.print(rs.getString(i) + "\t");
```

```
System.out.println();
```

```
}
```

```
// metaData need not be closed.
```

```
rs.close(); // closing ResultSet
```

```
st.close(); // closing Statement
```

```
con.close(); // closing Connection
```

```
} // main
```

```
} // class
```

Batch Updates:

→ What is a "Batch Update" in JDBC? Give an Example Program on Batch Updates.

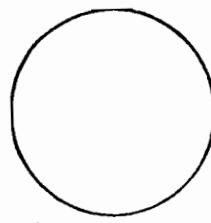
'Batch Updates' is a facility given by JDBC, using which, multiple SQL Statements (only DML) are grouped physically and submitted at a time to the DBMS for processing, as a Batch.

'java.sql.Statement' has two methods to implement Batch Updates in one JDBC Application :

1) addBatch(String dml) : This method is used to add an SQL Statement to the Batch.

2) executeBatch() : To submit at a time all the SQL Statements of the Batch to DBMS, this method is used.

When a Statement Object is created, in the background a list is associated with it. Initially the list is empty. When each SQL Statement is encountered with addBatch() method, the statements are piled up in the list.



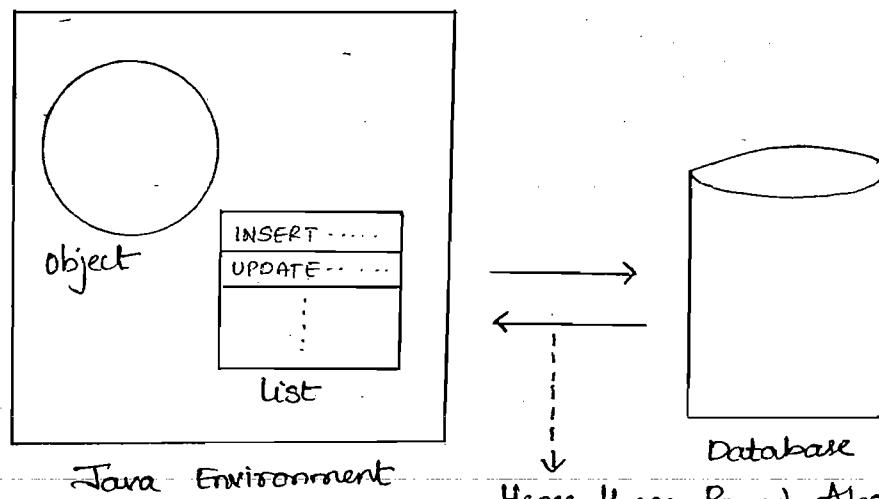
INSERT.....

st.addBatch("INSERT.....");

st.addBatch("UPDATE.....");

Hence, a Batch i.e., a Group of SQL Statements is formed. This Batch is not Submitted to DBMS as long as `executeBatch()` method is not encountered in the Program. When this method is called, at a time, all the SQL Statements of the batch are submitted to Server.

But, at Server side, only one by one Statement is executed i.e., only individual statements.



Hence, these Round-About trips from client to Server and back to Client are Reduced, as only time process of, Submitting Several SQL Statements and getting back corresponding results, is happening once.

The Return type of `executeBatch()` method is an integer Array.

A	B
<code>rec[0]</code>	<code>rec[1]</code>

The size of Array depends on no. of SQL Stmt's Submitted to DBMS.

Here, 'A' would be no. of records affected by SQL Stmt-1 of batch, 'n' be no. of records affected

// Stand Alone Application To Demonstrate Batch Updates

```
import java.lang.*;  
import java.sql.*;
```

```
class BatchUpdateExample
```

```
{
```

```
public static void main(String[] args) throws Exception
```

```
{
```

```
Class.forName("sun.jdbc.Odbc.JdbcOdbcDriver");
```

```
Connection con = DriverManager.getConnection("jdbc:odbc:  
praveen", "scott", "Tiger");
```

```
Statement st = con.createStatement();
```

```
st.addBatch("INSERT INTO EMPLOYEE VALUES(1004,  
'Prasad', 9800));
```

```
st.addBatch("UPDATE EMPLOYEE SET SALARY = SALARY +  
1500");
```

```
int re[] = st.executeBatch();
```

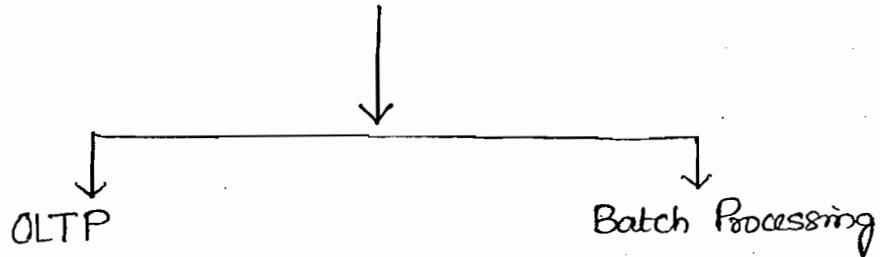
```
System.out.println("Number of Rows Affected By first  
SQL Statement : "+re[0]);
```

```
System.out.println("Number of Rows Affected By second  
SQL Statement : "+re[1]);
```

```
st.close(); // Closing Statement
```

```
con.close(); // Closing Connection
```

Business Processing i.e. Customer Request Processing is divided into:



It does not mean Internet Based Application.

Eg:- End user using a computer, needing the results immediately.

Eg :- ATM. Here, every time Java program should communicate with Database.

Here, user is not waiting for immediate Results i.e, immediate Response.

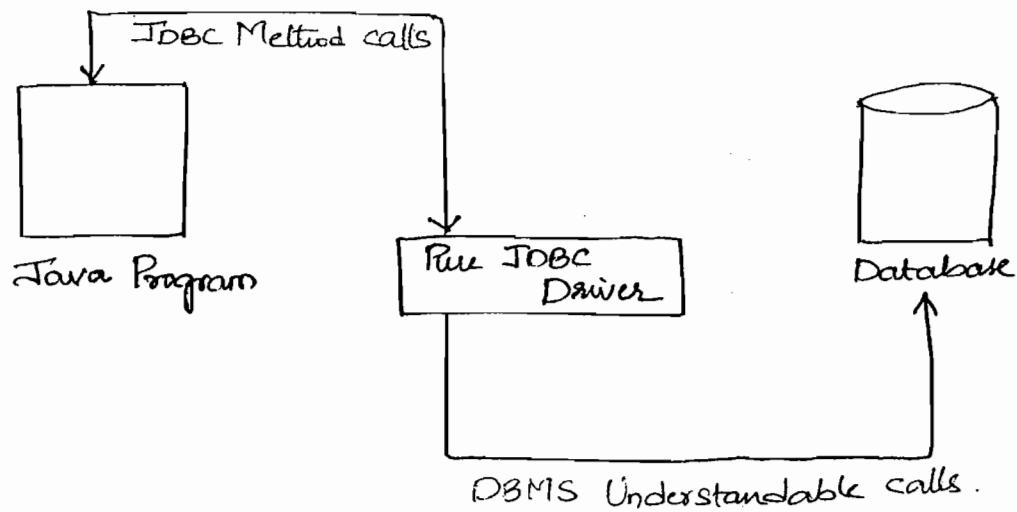
Since user is not waiting for System Response immediately, all the Queries can be piled up and can be Submitted at a later time to the Database Server, as a Batch.

Pure JDBC Approach of Connectivity

- Q) Develop A Stand Alone JDBC Application that gets connected to the Oracle Database Server using Pure JDBC Approach.

Pure JDBC Approach is used in industry for Performance Reasons. As we know that, in JDBC-ODBC Approach, translation is done in two stages: First, JDBC method calls are translated to ODBC Function calls. Next, ODBC Function calls are translated to DBMS Understandable calls. Similar procedure is Reverse direction. This involves two Drivers - Sun Driver & ODBC Driver. Hence, performance is obviously lost.

In pure JDBC Approach, there is only one Driver - the JDBC Driver. This driver directly translates JDBC method calls to SQL Statements i.e., DBMS understandable calls and vice versa.



Hence, Driver class file and Connection String should be changed.

- In this approach of Java-Database Connectivity, we need to have only JDBC Driver. There is no need of ODBC Driver and hence, no need of DSN creation.
 - In this approach, Driver Class will differ and Connection String will differ.

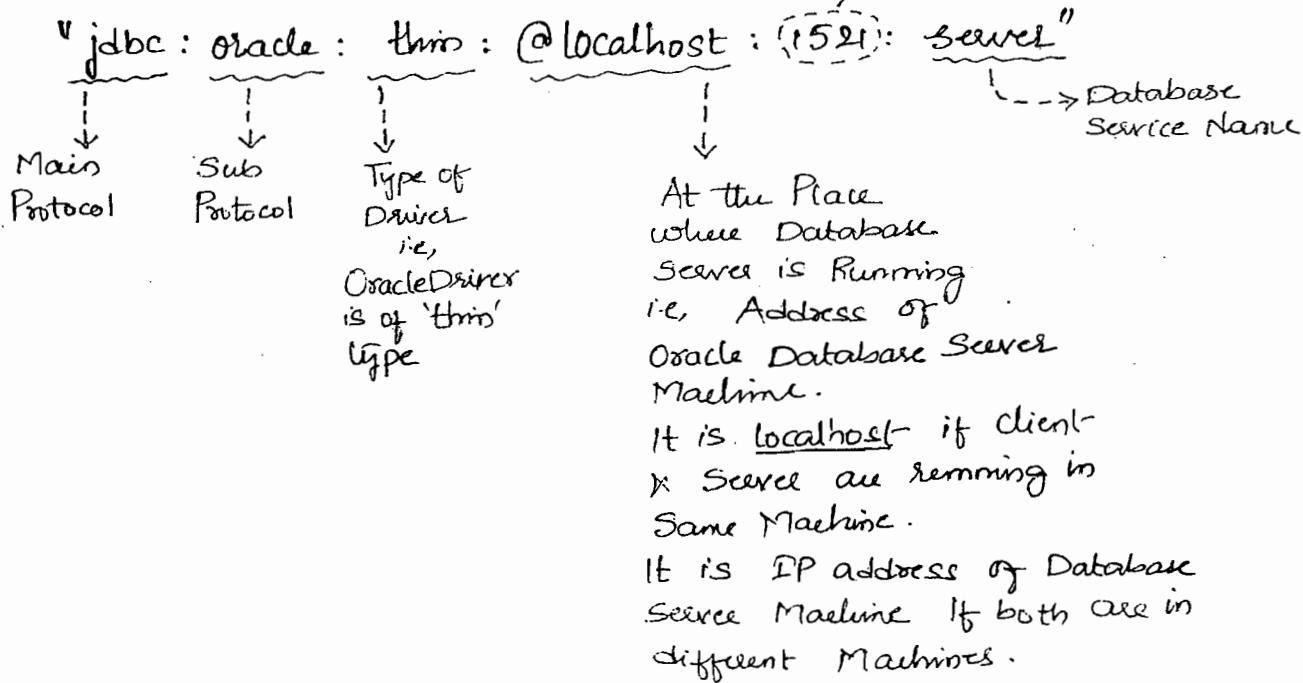
```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

- We need to keep classes12.jar file for this Driver to load properly.

Eg:- D:\oracle\ora90\jdbc\lib\classes12.jar

Set this in classpath, in Environmental variables.

- For this driver, Connection String is as follows:



```
// Stand Alone Application Demonstrating Pure JDBC Approach.  
import java.lang.*;  
import java.sql.*;  
  
class PureConnection  
{  
    public static void main(String[] args)  
    {  
        try  
        {  
            Class.forName("oracle.jdbc.driver.OracleDriver");  
  
            Connection con = DriverManager.getConnection(  
                "jdbc:oracle:thin:@localhost:1521:orcl",  
                "scott", "tiger");  
  
            System.out.println("Connected to Database");  
            con.close();  
        } // try  
  
        catch (ClassNotFoundException e)  
        {  
            System.out.println("Driver class cannot be found");  
            // e.printStackTrace();  
        } // catch-1  
  
        catch (SQLException e)  
        {  
            System.out.println("Unable to connect to Database");  
            // e.printStackTrace();  
        }  
    }  
}
```

? // class.

In the above program, instead of just throwing Exceptions, some handling is done. But in real time, this would not be sufficient. Consider the discussion below:

In the program, if an Exception occurs in try block, control transfers to corresponding catch block and then out of main() method. The remaining statements in the try block are not executed. Hence connection that was opened will never be closed. So, in order to make the 'connection closing' mandatory, we can call 'close()' method on 'con' object within a 'finally' block. This block would be the last block.

Now, if we compile, compiler shows an error:
Cannot find symbol : Variable con.

Since 'con' was declared in try block and is used outside it in finally block, scope problem occurs i.e., 'con' is local to try block alone. This could be resolved by making 'con' Global i.e., declaring con above all blocks.

On Compiling now, compiler shows two errors:

Error 1: Variable 'con' might not have been initialized.

Error 2: Unreported Exception java.sql.SQLException; must be caught or declared to be thrown.

Error 1 can be resolved by initializing variable 'con' to 'null'.
i.e., Connection con = null;

Since close() method is a library method and it was

in the form of Exception, so that it could be handled. This could be Resolved by placing close() method in another try-catch block in the finally block.

This could lead to another problem. If Something went wrong and an Exception has occurred in Driver loading, its corresponding catch block is Executed and Exception is handled. Then, while Executing finally block, as content of 'con' is null and further updation of 'con' has not happened, calling a method on object that is holding null value would lead to Null Exception i.e., NullPointerException.

This could be Resolved by placing the close() method in 'if' block. This 'if' block is placed inside a try-catch block. This is how in application development, Exception Handling is done. Finally the code would be like :

```
import java.lang.*;  
import java.sql.*;  
  
class PureConnection  
{  
    public static void main(String args[])  
    {
```

Connection con = null;

try

{

```
con = DriverManager.getConnection("jdbc:oracle:thin:  
@localhost:1521:orcl", "scott", "tiger");  
System.out.println("Connected to Database")  
} // try.  
catch (ClassNotFoundException e)  
{  
    System.out.println("Driver class cannot be found");  
    // e.printStackTrace();  
} // catch -1  
catch (SQLException e)  
{  
    System.out.println("Unable to Connect to Database");  
    // e.printStackTrace();  
} // catch -2  
finally  
{  
    try  
{  
        if (con != null)  
            con.close();  
    } // try  
    catch (SQLException e)  
{  
        System.out.println("Error Occurred while Closing the  
connection");  
        // e.printStackTrace();  
    } // catch  
} // finally  
} // main
```

19/09/09

Transaction Management

→ What is a transaction?

A single unit of interaction with the database is known as a transaction.

In a transaction, a set of SQL statements are grouped logically into a unit and submitted to the DBMS whose changes are made permanent OR undone only as a unit.

→ What is transaction management?

Every transaction has two boundaries

1) beginning of the transaction

2) end of the transaction

A mechanism of controlling the transaction boundaries is known as transaction management

→ How to implement transaction management in a JDBC Application

java.sql.Connection interface has three methods to deal with transaction management in a JDBC Application

1) setAutoCommit(boolean)

2) commit()

3) rollback()

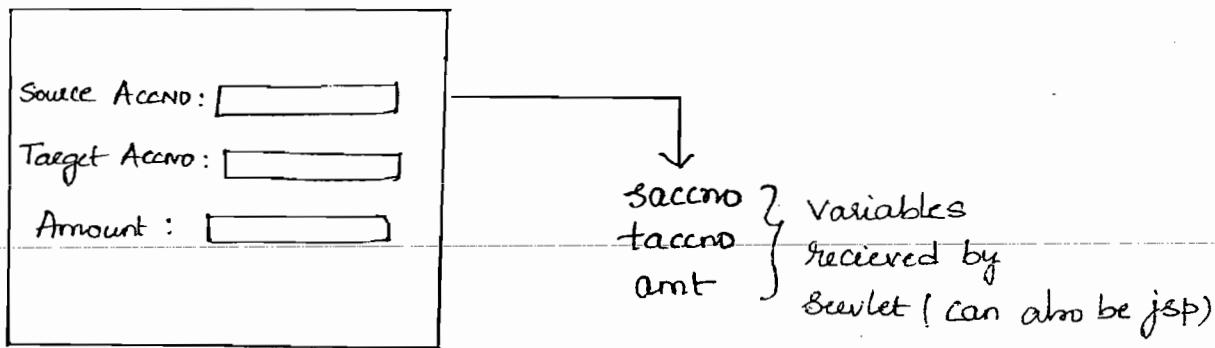
The first method is used to begin a transaction

→ why to implement transaction management in a JDBC Application?

To safeguard data integrity and consistency of enterprise data, we need to implement transaction management.

Consider a Scenario :-

Online funds transfer i.e., transferring funds from one account to another account through internet. Hence, amount from account is to be reduced and amount in another account must be increased.



Two SQL statements are needed to perform above task.

1st Stmt : UPDATE SACCOUNT SET BALANCE = BALANCE - amt
WHERE ACCNO = saccno

2nd Stmt : UPDATE CACCOUNT SET BALANCE = BALANCE + amt
WHERE ACCNO = tacno

Assume that 1st Stmt is executed successfully and due to

This has resulted because, we treated the statements as individual statements but not as a single logical unit (not physical as physical \Rightarrow Batch Updates). As a logical unit implies, control transfers twice from Servlet to database only, but the changes made on the database must be committed i.e., made permanent only if both the statements are successful. Even if one of the statement fails, the changes should be discarded.

The data in the table is modified but these modifications are made permanent only when all the SQL statements are successfully executed. This is known as Atomicity according to DBMS. Batch updates are physically grouping the statements whereas Transaction Management is logically grouping the statements. Hence both are different. But, Transaction management can be implemented using Batch Updates i.e., physically grouping logically grouped statements.

If Transaction Management is not used and a simple Statement or Prepared Statement of JDBC is used to submit the SQL statements, then these statements are in auto-commit mode. This is because of "Connection" which opens JDBC Connection in Auto Commit Enabled state. Hence the SQL statements make changes on table data and these changes are made permanent at that instant itself.

This may result in loss of data in above scenario, even if one of the SQL statements fails to execute for whatever reason it may be. This can be avoided using transaction management. The transaction is like a group of

using `setAutoCommit()` method. We need to specify the disabled state as : `setAutoCommit(false)` at the start of the transaction , in an Application . Then we can make the changes permanent or discard them as shown below:

```
try  
{
```

```
    con.setAutoCommit(false);
```

1st SQL Statement; ← Assume Successful, hence go to next Statement

2nd SQL Statement; ← If Successful, go to next line and commit the changes otherwise, i.e, if failure implies Exception hence control goes to catch block.

```
}
```

```
    catch()
```

```
    {  
        con.rollback();  
    }
```

If control comes here, implies one of the SQL statement failed, hence all the changes previously made i.e prior to this failed SQL Statement, are discarded using `rollback()` method.

→ Q) Develop & Deploy a Web Application in which transaction management is implemented.

Directory Structure :

```
fundetransferapplication  
|  
|---- "transfeer.html"  
|  
|---- WEB-INF  
|  
|---- "web.xml"  
|  
|---- classes  
|  
|---- "MoneyTransferServlet.class"  
|  
|---- lib  
|  
|---- "...."
```

URL :-

http://localhost:8081/fundstransferapplication/transfer.html

transfer.html :-

<HTML>

<BODY BGCOLOR = CYAN>

<CENTER> <H2> On Line Funds Transfer Interface </H2>

<FORM ACTION = "./transfer">

Savings A/c No: <INPUT TYPE = "text" NAME = "t1">

Current A/c No: <INPUT TYPE = "text" NAME = "t2">

Transfer Amount : <INPUT TYPE = "text" NAME = "t3">

<INPUT TYPE = "submit" VALUE = "Transfer Fund">

</FORM>

</CENTER>

</BODY>

</HTML>

web.xml

<web-app>

<Servlet>

<Servlet-name> money-transfer </Servlet-name>

<Servlet-class> MoneyTransferServlet </Servlet-class>

</Servlet>

<Servlet-mapping>

<Servlet-name> moneytransfer </Servlet-name>

<url-pattern> / transfer </url-pattern>

</Servlet-mapping>

MoneyTransferServlet.java

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

public class MoneyTransferServlet extends GenericServlet
{
```

Connection con;

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

try
{

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

```
    String url = "jdbc:oracle:thin:@localhost:1521:server";
```

```
    con = DriverManager.getConnection(url, "scott", "tiger");
```

```
    System.out.println("Connection Established");
```

}

```
    catch(Exception e)
```

```
    { e.printStackTrace(); }
```

```
} // init()
```

Connection
is opened in
Auto Commit Enabled
mode.

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

```
    int sbaccno = Integer.parseInt(req.getParameter("t1"));
```

```
    int caccno = Integer.parseInt(req.getParameter("t2"));
```

```

res.setContentType ("text/html");
PrintWriter out = res.getWriter();
out.println ("<HTML>");
out.println ("<BODY BGCOLOR = Wheat>");

try
{
    Statement st = con.createStatement();
    // con.setAutoCommit(false);

    String sql1 = "UPDATE SAVINGS SET BALANCE = BALANCE-
                  + amount + WHERE ACCNO = " + sbacno;
    String sql2 = "UPDATE CURRENT SET BALANCE = BALANCE
                  + amount + WHERE ACNO = " + cacno;

    st.executeUpdate(sql1);
    st.executeUpdate(sql2);

    // con.commit();

    out.println ("<H2> Funds Transferred Successfully </H2>")
    // con.setAutoCommit(true);
    st.close();           I just to revert back to
}                                default value of connection.
It is not mandatory.

catch (Exception e)
{
    try
    {
        // con.rollback();
        out.println ("<H1> Problem In Transferring Funds </H1>")
    }
}

```

```
        catch( Exception e1 )
        {
            e1.printStackTrace();
        }
    }
```

```
    out.println("<BODY>");
    out.println("<HTML>");
    out.close();
```

```
} // Service()
```

```
public void destroy()
{
```

```
    try
    {
        ron.close();
    }
```

```
    catch( Exception e )
    {
        e.printStackTrace();
    }
}
```

```
} // destroy()
```

```
} // class.
```

21/09/09

Calling A Stored Procedure From A JDBC Application

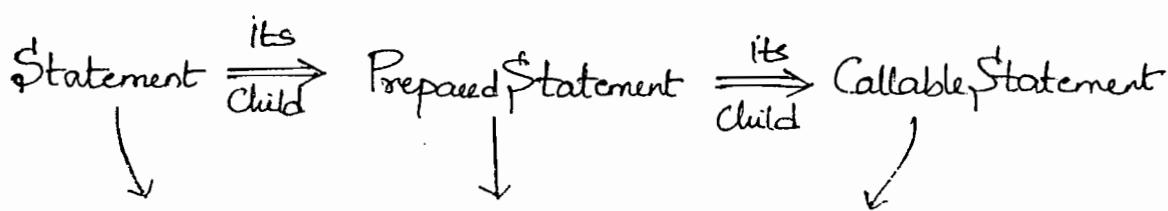
→ What is a Stored Procedure?

A stored Procedure is a Database Object that holds business logic within the DBMS. Stored Procedure is developed by PL/SQL programmers (not by the Java developers)

→ How to call a Stored Procedure from the JDBC Application?

Steps to Call a Stored Procedure from the JDBC Application:

- 1) Establishing the Connection
- 2) Creating the CallableStatement Object → does not hold SQL Stmt but holds only a call to stored procedure
- 3) Registering the out parameters if any.
- 4) Binding the in parameters → callee should supply these values in
- 5) Executing the call to the Stored Procedure a Stored Proc
- 6) Capturing the Results → caller should supply these values i.e. a CallableStatement
- 7) Closing the CallableStatement
- 8) closing the Connection.



All these are interfaces. Hence.

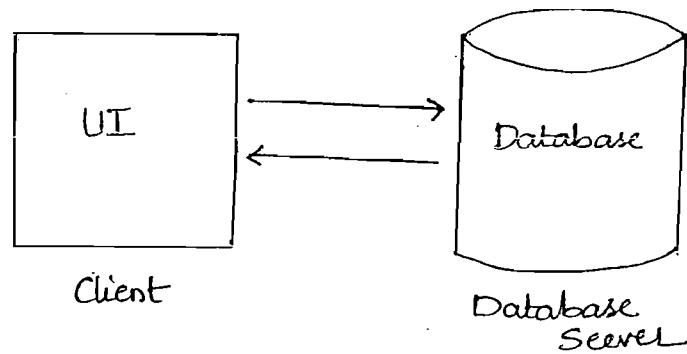
Client-Server Computing became famous from the early 80's till the late 90's. Client was developed using some technology and Server was only database server process.

These days, a General Business Application requires three layers (or three tiers) : User Interaction layer.

Processing Layer.

Database Layer.

But Client-Server Computing has two layers only



The processing layer is missing. It has to be placed either at UI side or at Database Server Side, since these are the only options available.

If placed along with UI layer i.e. at client, that client is called 'Fat Client' or 'Thick Client'. This results in a major setback if business rules change (which happens almost regularly). Then the modifications are to be done to every client. This is a laborious process and results in maintenance problems.

The best way is to dump the processing layer (i.e. business logic) on Server side. This can be achieved using Stored Procedures. Hence maintenance problems do not occur this way. But this indirectly increases the burden on the Server.

This drawback has been eliminated by using the Three tier Architecture i.e. Separating all the three layers. This has been used since early 2000's

Consider a Scenario:

for a certain client, a company developed A Business Application say 15 years back using 2-tier Architecture. The Business logic is stored at Server side as Stored Procedures. This Application could have been developed using some technologies. Now, say, at present the client wants to upgrade the application using latest technologies in Java.

But a Java Programmer needs to know about the database that is being maintained by the client, so as to develop the application. whatever may be the reason, most of the companies would never reveal the database details to outsiders because of the fact that total business logic would be exposed otherwise. They would only provide the documentation details of Stored Procedures.

The Java Programmer, after reading the documentation details for the Stored Procedures would be now able to develop the Application. Most of the Applications these days are developed as 3-tier Architecture, no problems would arise. Therefore, in the Industry, around 15-20% of projects which could be only migratory projects (migration from old to current generation projects) utilize this concept of stored Procedures.

→ Q) JDBC Application that Calls the Stored Procedure.

Creating A Table :-

```
SQL> CREATE TABLE EMP10 (ID NUMBER, NAME VARCHAR2(12));
```

```
SQL> INSERT INTO EMP10 (1001, 'Rama');
```

```
SQL> COMMIT;
```

Creating A Stored Procedure :-

create or replace procedure proc2 (n number, nm out varchar)
as
begin

Select name into nm from emp10 where id=n;
end proc2;

/

whenever command is submitted
to this stored procedure
it gives the emp name.

Stored Procedure .java :-

```
import java.sql.*;  

class StoredProcedure  

{  

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

    {
```

Step 1 { → Class.forName("oracle.jdbc.driver.OracleDriver");
 → String url = "jdbc:oracle:thin:@localhost:1521:orcl";
 → Connection con = DriverManager.getConnection(url, "scott", "tiger")

```
System.out.println("Connection is Established ....");
```

Step 2 → CallableStatement cst = con.prepareCall("{ call proc2(?,?) }");

Since two
parameters
i.e., in
our
" "

Step 3 → `rst.registerOutParameter(2, Types.VARCHAR);`
 ↓
 Data type corresponding
 to SQL

Step 4 → `rst.setInt(1, 1001);`
 Question Mark
 Number

Step 5 → `rst.execute();`

Step 6 → `String str = rst.getString(2);`
`System.out.println("Employee Name: " + str);`

Step 7 → `rst.close();`

Step 8 → `con.close();`

} // main()

} // class

→ Q) JDBC Application that calls A Stored Procedure.

Creating A Table :-

SQL> CREATE TABLE ACCOUNT (ID NUMBER(4), BALANCE NUMBER(7,2));

SQL> INSERT INTO ACCOUNT VALUES (1001, 5000);

SQL> COMMIT;

Creating A Stored Procedure :-

Create or replace procedure addinterest (id in number, bal out number)

as

begin

update account set balance = bal where id = id;

end;

/

when accno is
supplied to this
stored procedure, it
gives balance with interest
added to it.

Stored Procedure 2.java:

```
import java.sql.*;
```

```
class StoredProcedure2
```

```
{
```

```
public static void main(String[] args) throws Exception
```

```
{
```

```
int accno = Integer.parseInt(args[0]);
```

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

```
Connection con = DriverManager.getConnection("jdbc:odbc:java",
                                             "scott", "tiger");
```

```
System.out.println("Connection Established");
```

```
CallableStatement cst = con.prepareCall("{call addinterest(?,?)}");
```

```
cst.registerOutParameter(2, Types.FLOAT);
```

```
cst.setInt(1, accno);
```

(A class from java.sql package.)

```
cst.execute();
```

It has constants corresponding to
datatypes in SQL.)

```
System.out.println("Stored Procedure Executed ...");
```

```
System.out.println("New Balance of A/c "+accno+" is : Rs "+
                   cst.getFloat(2));
```

```
cst.close();
```

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

```
} //main()
```

```
} // class
```

Scrollability \Rightarrow

Normally in a ResultSet, the cursor is moved in forward direction using next() method and that too, one record at a time. This is known as a non-scrollable ResultSet.

If we can move the cursor in any direction and directly to any record, we need, then that ResultSet is said to be a scrollable ResultSet.

Consider :-

Statement st = con.createStatement();

Resultset rs = st.executeQuery("SELECT * FROM EMPLOYEE");
(OR)

PreparedStatement ps = con.prepareStatement("SELECT * FROM EMPLOYEE WHERE SALARY > ?");

Resultset rs = ps.executeQuery();

Just by looking at these statements we can never decide on what kind of ResultSet it is.

Only these statements decide the type of ResultSet. Here, we can say it is Non Scrollable & Non Updatable ResultSet.

We can also provide arguments to createStatement() method. Then this method gets overridden. There is a two argument method for Statement and three argument method for PreparedStatement. One of the argument is for specifying Scrollability and the other is for specifying updatability.

Statement st = con.createStatement(TYPE_FORWARD_ONLY, CONCUR_UPDATABLE);

ResultSet rs = st.executeQuery("SELECT * FROM EMPLOYEE");

↑
This is a non-scrollable
but updatable ResultSet

Similarly:

(TYPE_FORWARD_ONLY, CONCUR_READ_ONLY) ← non scrollable & non updatable

(TYPE_SCROLL_SENSITIVE, CONCUR_READ_ONLY) ← scrollable & non updatable

(TYPE_SCROLL_SENSITIVE, CONCUR_UPDATABLE)

↑ scrollable & updatable

(TYPE_SCROLL_INSENSITIVE, CONCUR_READ_ONLY) ← scrollable &
non updatable

(TYPE_SCROLL_INSENSITIVE, CONCUR_UPDATABLE)

↑ scrollable & updatable

(OR)

PreparedStatement ps = con.prepareStatement("SELECT * FROM EMPLOYEE
WHERE SALARY > ? ", TYPE_SCROLL_SENSITIVE,
CONCUR_READ_ONLY);

ResultSet rs = ps.executeQuery();

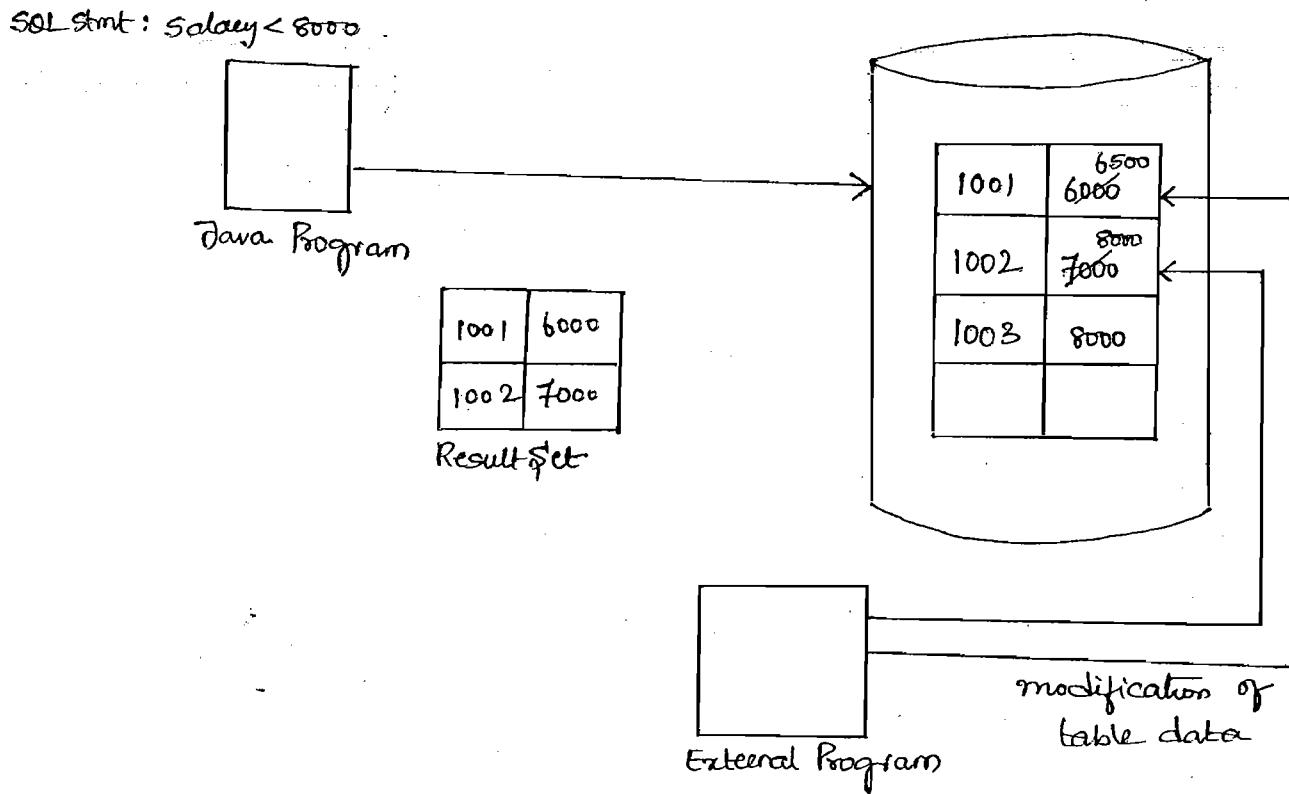
↑ This is a scrollable but
non updatable ResultSet.

→ what is SENSITIVE & INSENSITIVE?

Consider A Scenario:

A Java program submits the SQL Statement
to DBMS using Statement or Prepared Statement. The
ResultSet can be of any type of the 4 types of ResultSets.

At the same time, an external program



If SENSITIVE is used, the values in ResultSet are also affected by the modifications i.e. ResultSet is sensitive to external modifications done on database.

If INSENSITIVE is used, the values in ResultSet are not affected by the modifications i.e. ResultSet is insensitive to external modifications done on the database.

23 | 09 | 09

→ When Resultset is scrollable implies the cursor can be moved in any direction required.

→ When Result Set is Scrollable, the following methods can be called to deal with the cursor.

1) next()

2) $\text{frequency} = \pi \cdot \frac{1}{4} = \pi \cdot 0.25 = \pi/4$

It also performs two operations :-

- 1) moves cursor in backward direction
 - 2) returns true or false depending on availability of record
 - 3) first()
 - 4) last()
 - 5) afterLast() ^{← cursor moves to No Record Area}
 - 6) beforeFirst() ^{← cursor moves to Zero Record Area}
 - 7) absolute (int n) :- This method moves the cursor directly to the specified record. If the argument is a negative number, counting starts from the last record in the reverse direction
if there are 10 records
rs.absolute(5) ⇒ cursor to 5th record from first
rs.absolute(-2) ⇒ cursor to 2nd record from last i.e. 9th record.
 - 8) relative (int n) :- It moves the cursor to a specified number of records from the current record. Direction of movement depends upon the argument
if there are 10 records & currently the cursor is at 4th record
rs.relative(2) ⇒ cursor moves to 6th record
rs.relative(-2) ⇒ cursor moves to 2nd record.
 - 9) moveToInsertRow()
 - 10) moveToCurrentRow()
 - 11) boolean isFirst()
 - 12) isLast()
 - 13) isBeforeFirst()
 - 14) isAfterLast()
- } these are not cursor movement methods but they can be used to verify the location of cursor.

15) int getRow :- This method retrieves the current location of the cursor.

Combination of last() & getRow() can be used to determine no. of records present in the ResultSet.

Inserting a Record into the updatable ResultSet :-

Step 1 :- Move the cursor to a special row known as InsertRow/BufferedRow in order to compose a new Row

Step 2 :- By calling update methods, fill the column values

Step 3 :- Call insertRow() method on ResultSet object

Once the above method is successfully executed, a row is inserted into the ResultSet as well as into the database.

Steps to Update a Record :-

Step 1 :- Move the cursor to the specific row

Step 2 :- call update method to change the old value

With this, only the ResultSet content is modified. But not the table content

Step 3 :- call updateRow() method on ResultSet object so that database value also is modified.

Deleting a Record Permanently :- (programmatically)

Step 1 :- Move the cursor to the specified Record

Record is deleted from ResultSet as well as from
database table.

23/09/09

Connection Pooling :-

- It is the process of creating a collection (group or pool) database connections and keeping them in cache for us and reuse.
- Object oriented representation of connection pool is nothing but DataSource object.

fb6c: Slack: Qin:
② 192.168.100.1:1521:80

Tomcat 6.

↓
conf

↓
context.xml

before
</context>

resource name="jdbc/my"

auth="Container"

type="javax.sql.DataSource"

Servlet Config → ServletContext → RequestDispatcher
↓ forward
↑ to particular