
# Adv. Java By Venkatesh Mansani sir

**Author of this notes : TheCodingSam ( Click Here to see Portfolio )**

# JDBC

1) **JDBC:**

JDBC stands for java database connectivity.
JDBC is a specification for developing database applications by using Java programming language.

**Database Application:**
An application that communicates with a database is known as database application.

**Application:**
An application is a program in which we interact with on the desktop.

**Database:**
A database is a software and it is an organized collection of data.

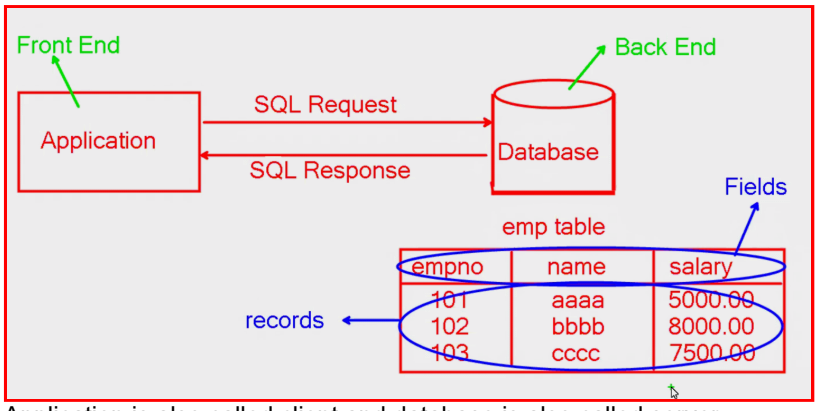
Data organized in a database in the form of tables.
Each table contains fields & records.

⇒Application is called frontend whereas database is called backend.

List of database software:
1. Oracle
2. MySQL
3. DB2
4. Derby
5. FoxPro
6. Visual FoxPro
7. MS SQL server
8. DBase
9. Sybase
10. MS-Access , …etc.

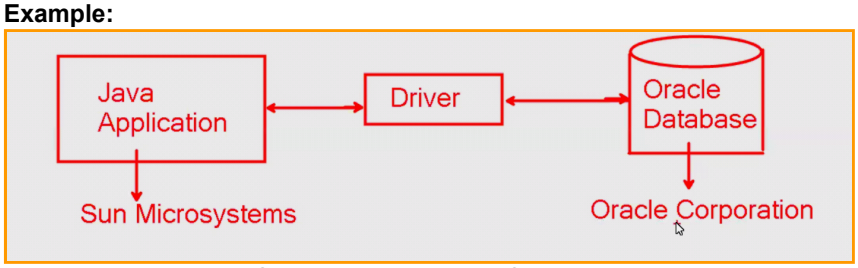
Database software contains two parts:
1. Database Application (example:SQL prompt)
2. Database (example: Oracle)

⇒ Applications are communicating with databases by using SQL.
⇒SQL stands for Structured Query Language and it is used by applications to communicate with databases.



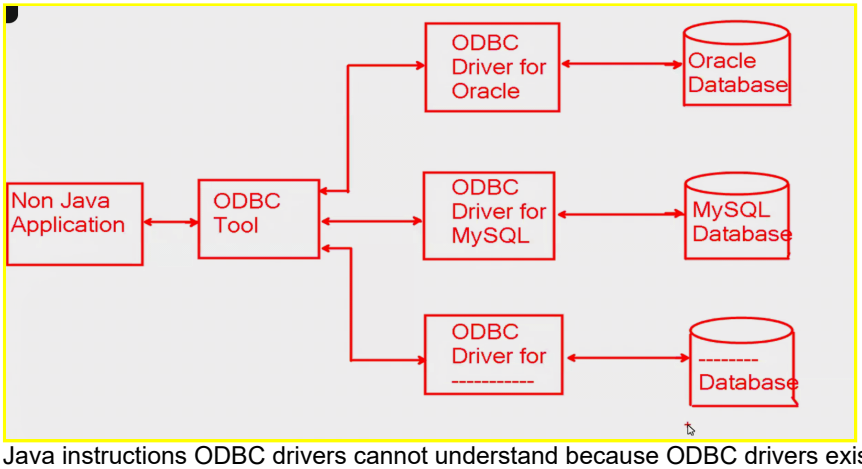

Application is also called client and database is also called server.
⇒ A client is a software that sends the request to a server to get the response.
⇒ A server is a software that receives a request from the client, processes the request, constructs the response & sends the response back to a client.

**Driver:**
A driver is a software and it is used to connect applications and databases.
⇒ Drivers are developed by first party vendors, second part vendors, & third party vendors.
**Example:**




In the above example, Sun Microsystems is a first party vendor, oracle corporation is a second party vendor, other than Sun Microsystems and oracle corporation is a third party vendor.
⇒ There are two categories of drivers.
1) ODBC drivers
2) JDBC drivers

ODBC drivers existed before java technology.
⇒ There are many ODBC drivers.
⇒ ODBC stands for Open DataBase Connectivity



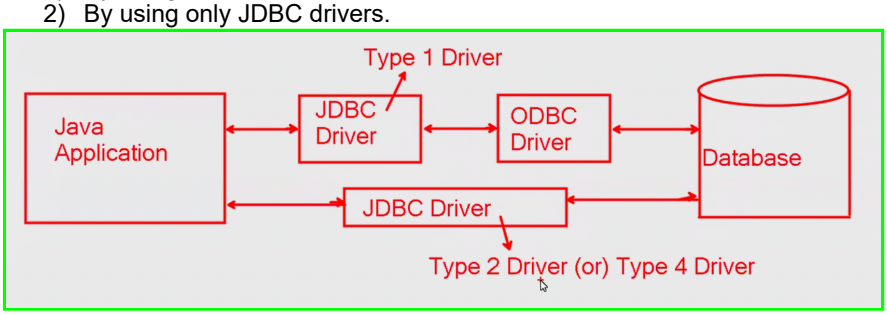

Java instructions ODBC drivers cannot understand because ODBC drivers existed before Java. (That compatibility was not there.)
⇒ To solve the above problem JDBC drivers were introduced.

**There are four types of JDBC Drivers :**

1) Type -1 Driver (JDBC ODBC Bridge Driver)
2) Type -2 Driver (JDBC Native API Driver)
3) Type -3 Driver (JDBC Network Protocol Driver)
4) Type -4 Driver (JDBC 100% Pure Java Driver)

**There are two ways to connect Java Application & Database.**

1) By using JDBC drivers & ODBC drivers.
2) By using only JDBC drivers.

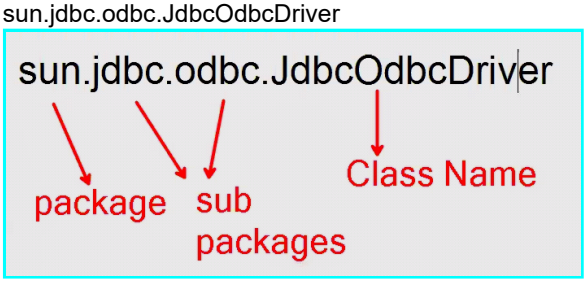



**Note:**
All JDBC drivers are classes in Java.

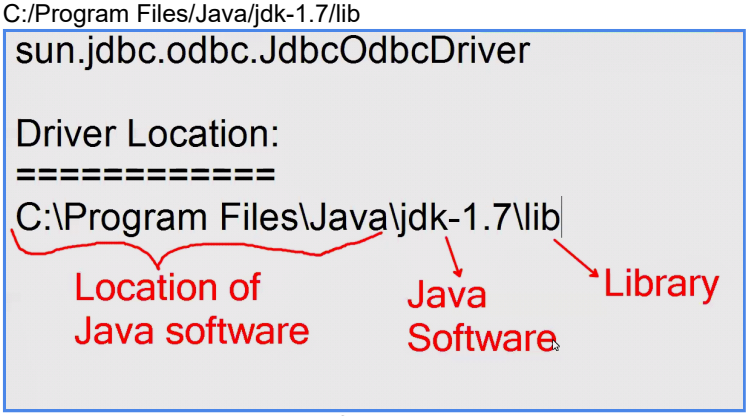
**Specification:**
It is a set of rules & guidelines that are used to develop applications & environments.

⇒ JDBC specification used by vendors to develop drivers.
JDBC specification used by Java programmers to develop database applications.

**Type -1 Driver (JDBC ODBC Bridge Driver)**
Driver class name :
sun.jdbc.odbc.JdbcOdbcDriver

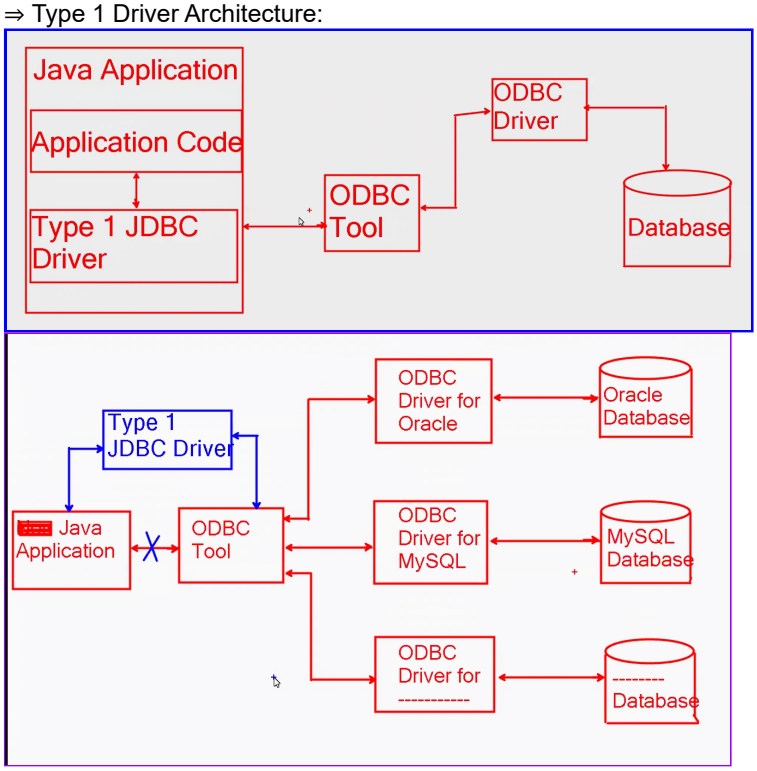



**Driver Location:**
C:/Program Files/Java/jdk-1.7/lib



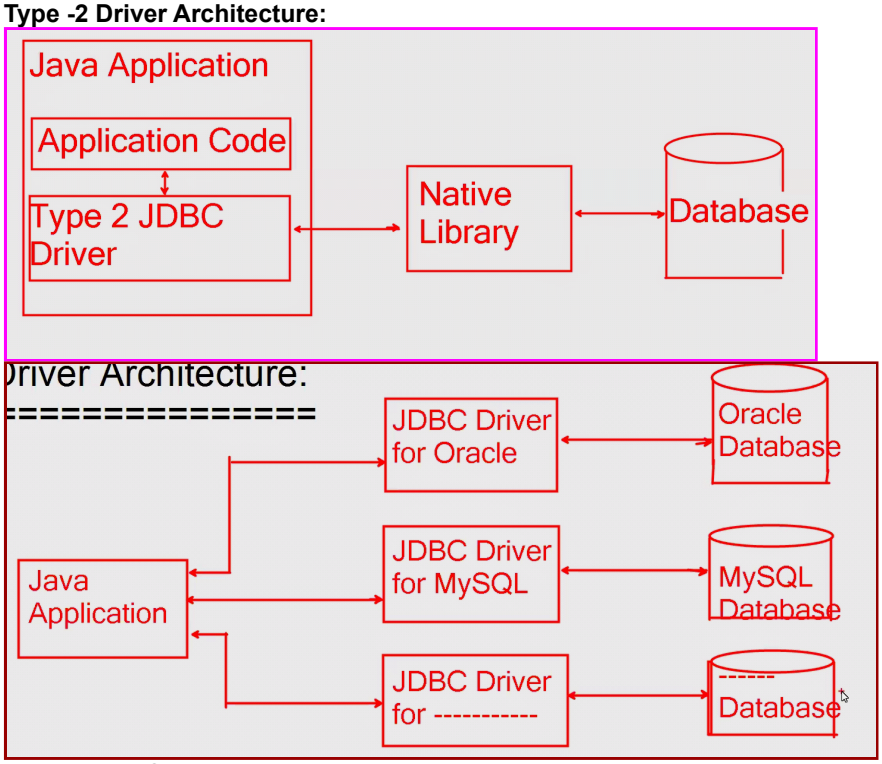

**Note :** Type 1 Driver removed from JDK 1.8 onwards.

⇒ Type 1 JDBC drivers developed by Sun Microsystems.
⇒ Type 1 JDBC drivers developed in "C" Language.
⇒ Type 1 Driver Architecture:

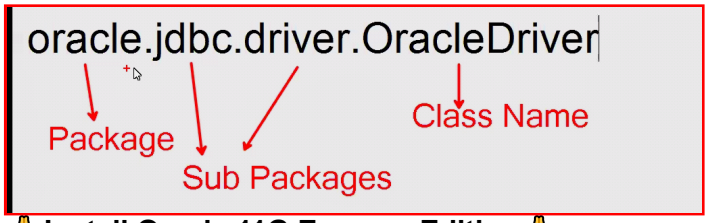



⇒ Type -1 Driver Functionality:
It converts Java instruction into ODBC understandable format.
Or it converts Java calls into odbc calls.
⇒ Type -1 Driver Advantages:
1) Only one driver that supports all odbc enabled databases.
⇒ Type -1 Driver Disadvantages:
1) Performance overhead since Java calls should go through via JDBC & ODBC drivers.
2) Database client software needs to be installed on every system.
3) Platform dependent.

**Type -2 Driver (JDBC Native API Driver):**
⇒ It is also called Partly Java Partly Native Driver or Partial Java Driver.
⇒ Type -2 Drivers developed in Java language and Native languages (c,c++).
**Type -2 Driver Architecture:**



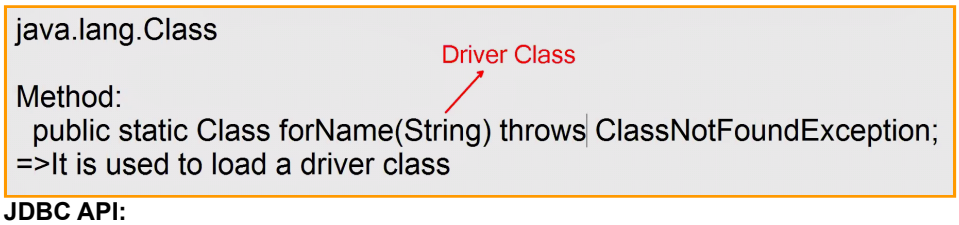

**Type -2 JDBC Driver Functionality:**
It converts Java calls into native calls.
**Type -2 JDBC Driver Class Name for Oracle Database:**
oracle.jdbc.driver.OracleDriver




## 🙏Install Oracle 11G Express Edition🙏

**Driver Location:**
ojdbc6_g.jar file in Oracle 11G Express Edition.
**JAR (Java Archive) file Location:**
C:/oraclexe/app/oracle/product/11.2.0/server/jdbc/lib
**Driver vendor:**
Oracle Corporation
**Steps To develop database application:**

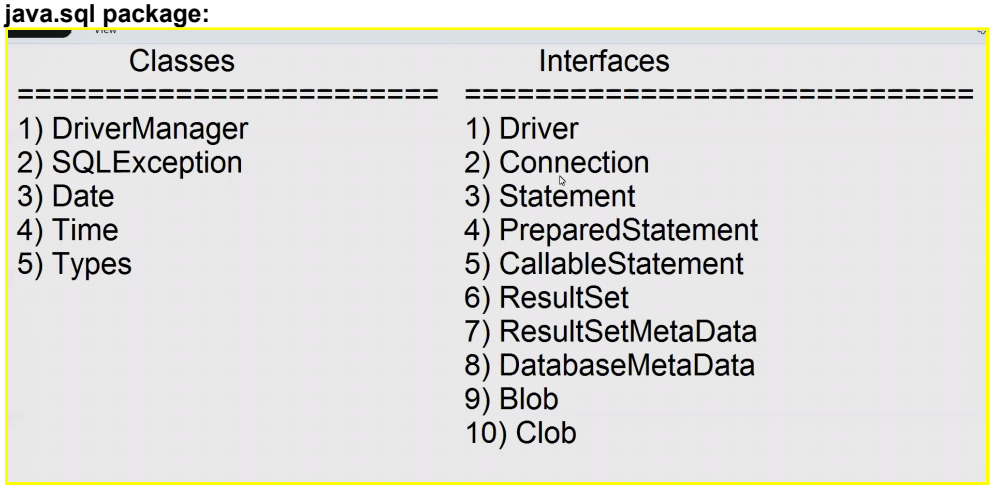
1) Loading a specific jdbc driver
2) Establishing a connection
3) Performing the task
4) Closing a connection

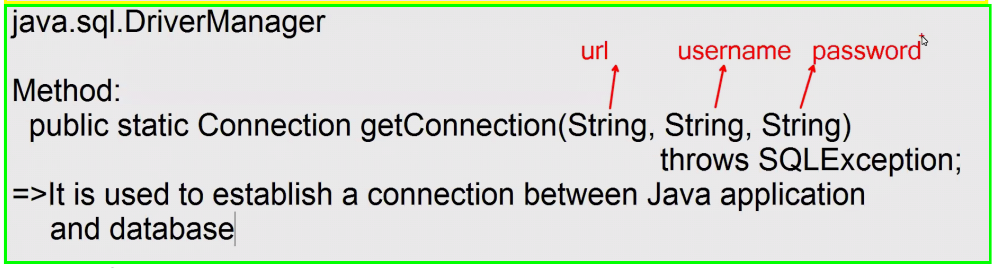



**JDBC API:**
JDBC API is a Java API that can access any kind of tabular data & data especially stored in RDBMS (Relational database Management system)
⇒ Java library is called Java API because it is an interface between application and programming language.
**JDBC API Contains the following packages:**

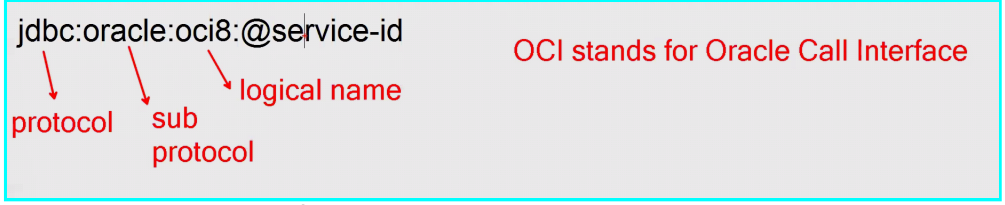
1) java.sql package
2) javax.sql package
3) javax.sql.rowset package

**java.sql package:**




**URL (uniform resource locator) to access database by using type 2 driver:**
jdbc:oracle:oci8:@service-id




To get service I'd , use the following sql query at sql prompt:

SQL> select*from global_name;

**Steps to load driver & get connection:**

```
/* Program to established a connection between java application
    and oracle database by using Type 2 JDBC driver...      */
import java.sql.*;
public class DriverConnectionDemo {

    public static void main(String[] args) {
        try {
            Class c = Class.forName("oracle.jdbc.driver.OracleDriver");
            System.out.println("Driver Loaded Successfully");
            Connection con =
DriverManager.getConnection("jdbc:oracle:oci8:@xe","sam","tiger");
            System.out.println("Connection Established Successfully");
        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }
    }
}
```

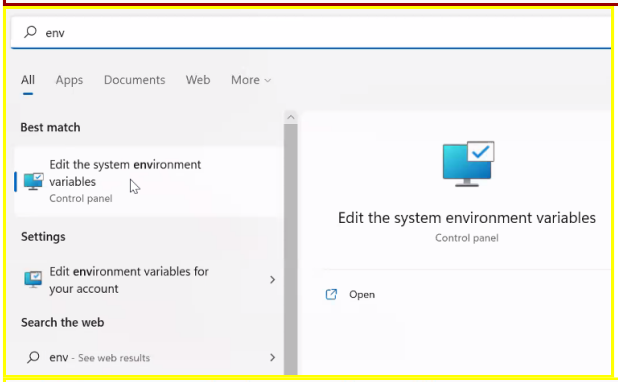
⇒**There are two ways to set the classpath to access jdbc driver class:**

1) **Temporary classpath**
2) **Permanent classpath**

1) Temporary classpath setting:
==========
C:\>set classpath="%classpath%".;C:\oraclexe\app\oracle\product\11.2.0\server\jdbc\lib\ojdbc6_g.jar;

append mode

It is a command and it is used to set the classpath




2) Permanent classpath setting:
==========
1) Click on Windows button
2) Type environment variables in search bar
3) Click on Advanced tab
4) Click on Environment Variables button
5) Click on New button under User variables section
6) Type variable name is classpath
7) Type variable value is C:\oraclexe\app\oracle\product\11.2.0\server\jdbc\lib\ojdbc6_g.jar;

**Note : if the classpath variable already present then select the classpath variable & click on Edit button , click on New button type the following ->**

C:\oraclexe\app\oracle\product\11.2.0\server\jdbc\lib\ojdbc6_g.jar

**Type 2 Driver Advantages;**

1) It is a little faster as compared to type 1 driver.

**Type 2 Driver Disadvantages:**

1) Separate driver required for every database.
2) All databases do not have type 2 drivers.
3) Database software needs to be installed on the same computer.

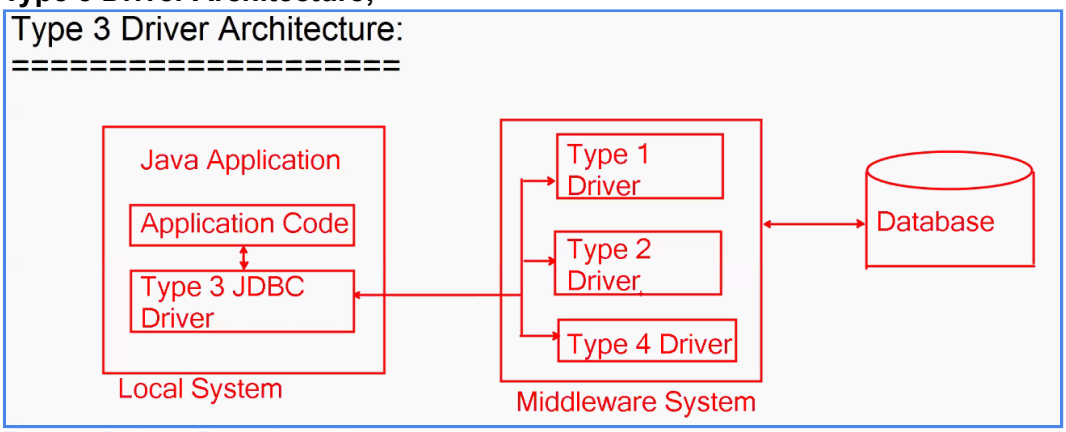
4) It is platform dependent.

Type 1 Driver Class Name:
sun.jdbc.odbc.JdbcOdbcDriver

Type 2 Driver Class Name for Oracle:
oracle.jdbc.driver.OracleDriver

**Type 3 Driver (JDBC NETWORK PROTOCOL DRIVER):**
It is also called JDBC Net Pure Java Driver (or) Middleware Driver.

**Type 3 Driver Architecture;**




**Type 3 Driver Functionality:**
It passes the java instructions from the local system to the middleware system.

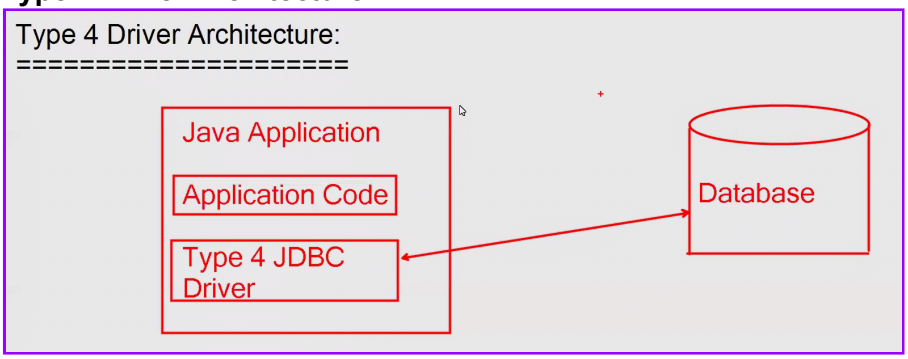
**Type 3 Driver Advantages:**
1) Java Calls are database independent from local system to middleware system.
2) Database not needed on the same computer.
3) It is platform independent.

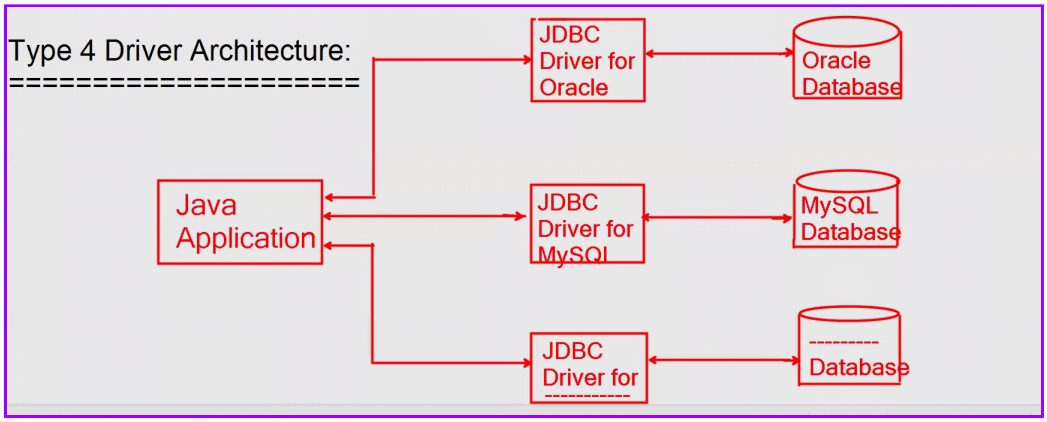
**Type 3 Driver Disadvantages:**
1) Extra layer added in this architecture(i.e. Middleware system)

**Note :** Type 3 drivers were developed in Java language only.

**Type 4 JDBC DRIVER (JDBC 100% Pure Java Driver)**
⇒ It is also called Thin Driver or JDBC Native Protocol Driver.

**Type 4 Driver Architecture:**






**Type 4 Driver Functionality:**
⇒ It passes the java instructions directly to a database.

**Type 4 Driver Advantages:**
1) It is the highest performance driver as compared to all other drivers.
2) Database not needed on the same system.
3) It is a platform independent.

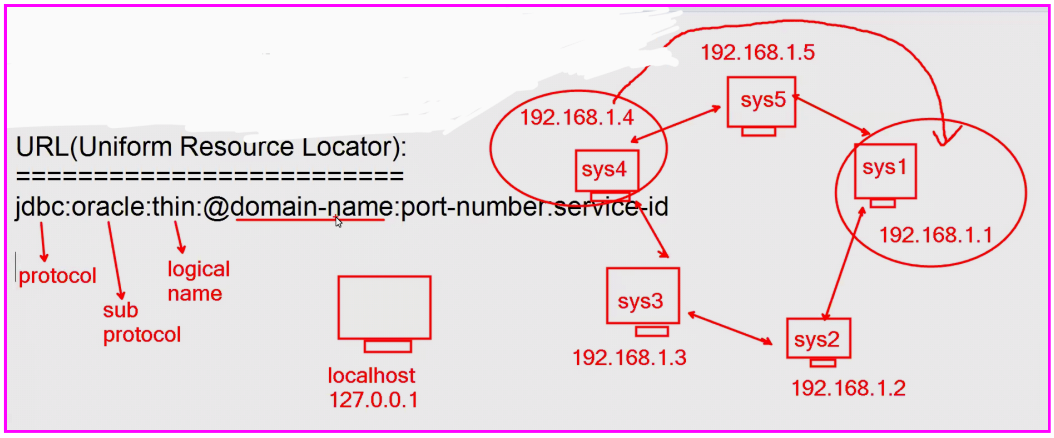
**Type 4 Driver Disadvantages:**
1) Separate driver needed for every database.

**Type 4 JDBC Driver Class Name for Oracle Database:**
oracle.jdbc.driver.OracleDriver

**URL (**uniform resource locator):
jdbc:oracle:thin:@domain-name:port-number:service-id

⇒**There are two types of addressing system:**
1) Letter Addressing System (Domain Naming System(DNS))
2) Number Addressing System (Internet Protocol Address (IP address))

⇒Standalone system domain name is localhost and **IP** address is 127.0.0.1
⇒Local area network IP addresses are 192.168.1.1 to 192.168.1.255




⇒ if the database software is installed on the same computer then use localhost as a domain name otherwise use computer name as a domain name.

**Port number :**
It is used to identify the service.
⇒Port numbers range from 0 to 65535.
⇒Reserved port numbers range from 0 to 1023.
⇒Free port numbers range from 1024 to 65535.

- HTTP Port Number is 80
- FTP Port Number is 21
- Telnet Port Number is 23

⇒To view port number open tnsnames.ora file from the following location:

To view port number open tnsnames.ora file from the following location:

C:\oraclexe\app\oracle\product\11.2.0\server\network\ADMIN

⇒To get service-id use the following sql query at the SQL prompt:

To get service-id use the following sql query at the sql prompt:
SQL>select * from global_name;

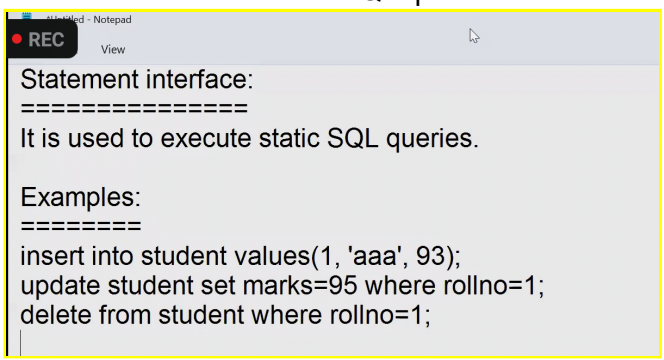
```
/* Program to established a connection between java application
    and oracle database by using Type 4 JDBC driver...      */
import java.sql.*;
public class DriverConnectionType4Demo {

    public static void main(String[] args) {
        try {
            Class c = Class.forName("oracle.jdbc.driver.OracleDriver");
            System.out.println("Driver Loaded Successfully");
            Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","sam","tiger");
            System.out.println("Connection Established Successfully");
        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }
    }
}
```

⇒ There are 3 statements in JDBC that are used to send SQL & PL/SQL statements to the database to get the data from the database.

1) Statement interface
2) PreparedStatement interface
3) CallableStatement interface

**Note :** all JDBC interfaces are implemented in driver software by the driver vendor.

**Statement interface :**
It is used to execute static SQL queries.




**PreparedStatement interface :**
It is used to execute dynamic SQL queries.

PreparedStatement interface:
==========
It is used to execute dynamic SQL queries.

Examples:
========
insert into student values(?, ?, ?);
update student set marks=? where rollno=?;
delete from student where rollno=?;

**CallableStatement interface:**

It is used to execute PL/SQL programs.

CallableStatement interface:
==========
It is used to execute PL/SQL programs.

Examples:
========
Procedures & Functions

java.sql.Connection

Methods:
public abstract Statement createStatement() throws SQLException;
public abstract PreparedStatement prepareStatement(String) throws SQLException;
public abstract CallableStatement prepareCall(String) throws SQLException;

Dynamic SQL Query

PL/SQL Procedure Name/Function Name

java.sql.Statement

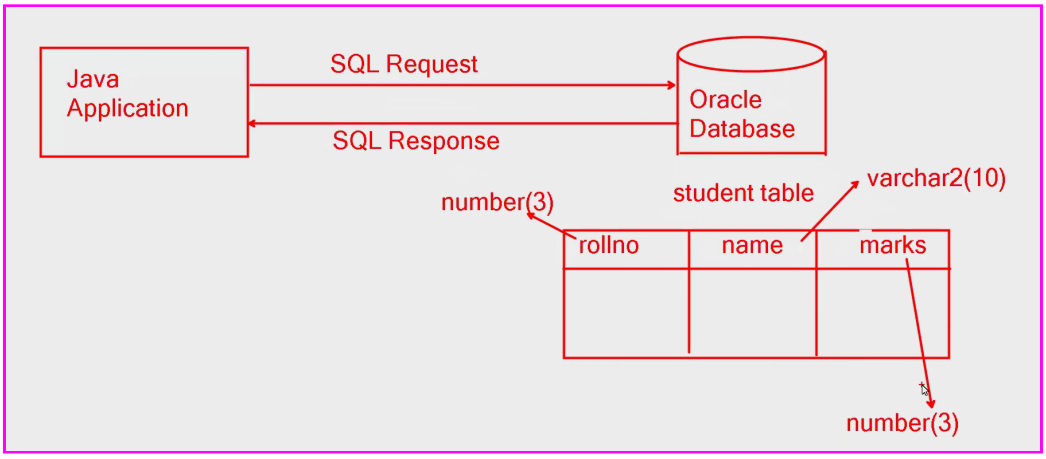
Static SQL Queries

Methods:
public abstract boolean execute(String) throws SQLException;
public abstract int executeUpdate(String) throws SQLException;
public abstract ResultSet executeQuery(String) throws SQLException;

**execute() method:**
⇒it is suitable to execute DDL queries.
⇒ DDL stands for data definition language.
Example : create , alter , drop, etc…

**executeUpdate() method;**
⇒ it is suitable to execute DML queries.
⇒ DML stands for data manipulation language
Example : insert, update, delete, …etc

**executeQuery() method;**
⇒ it is suitable to execute DQL queries.
⇒ DQL stands for data query language.
Example: select



```java
//Program to create a table
import java.sql.*;

class CreateDemo
{
   public static void main(String[] args)
   {
      try{
       Class.forName("oracle.jdbc.driver.OracleDriver");
      Connection con=DriverManager.getConnection(
                        "jdbc:oracle:thin:@localhost:1521:xe","system","manager");
      Statement stmt=con.createStatement();
      stmt.execute("create table student(rollno number(3), name varchar2(10),
                                                    marks number(3))");
      System.out.println("Table Created Successfully");
      }catch(Exception e)
      {
         System.err.println(e);
      }
   }
}
```

⇒ Java code executed under Java Runtime Environment (JRE) & SQL code executed under Database Environment

```java
//Program to insert a record
import java.sql.*;

class InsertDemo
{
   public static void main(String[] args)
   {
      try{
      Class.forName("oracle.jdbc.driver.OracleDriver");
      Connection con=DriverManager.getConnection(
                        "jdbc:oracle:thin:@localhost:1521:xe","system","manager");
      Statement stmt=con.createStatement();
      stmt.executeUpdate("insert into student values(1, 'aaa', 93)");
      System.out.println("One Record Inserted Successfully");
      }catch(Exception e)
      {
         System.err.println(e);
      }
   }
}
```

**ResultSet:**

A result set is an object that encapsulates a set of rows from a database.
ResultSet is generated based on sql query.
⇒ whenever ResultSet is generated then ResultSet pointer or cursor points before the first record.

```
java.sql.ResultSet

Methods:
  public abstract boolean next() throws SQLException;
=>It returns true & moves the cursor to next record if the record is
                                          present otherwise returns false.
  public abstract String getString(int) throws SQLException;
  public abstract int getInt(int) throws SQLException;
  public abstract float getFloat(int) throws SQLException;
=>The above 3 methods are used to get the data from ResultSet
  public abstract ResultSetMetaData getMetaData()
                                                  throws SQLException;
=>It returns ResultSetMetaData reference
```

**Metadata:**

It means data about data.

**ResultSetMetaData:**

It means data about ResultSet.

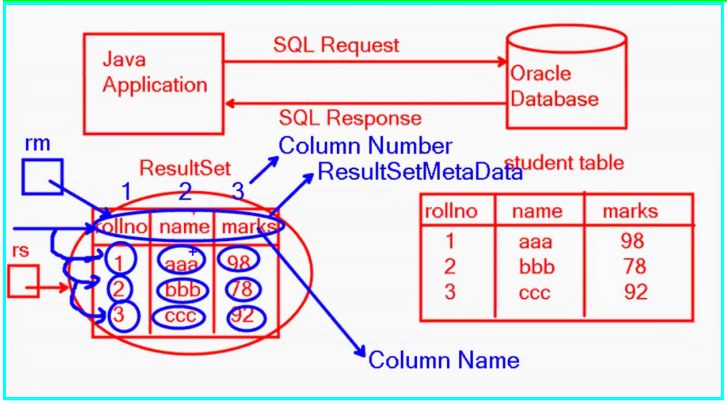
**DatabaseMetaData**:

It means data about databases.

java.sql.ResultSetMetaData

Methods:
public abstract int getColumnCount() throws SQLException;
=> It returns no. of columns

public abstract String getColumnName(int) throws SQLException;
=> It returns column name at specified column number




```java
//Program to retrieve the data from database.
import java.sql.*;

class SelectDemo
{
   public static void main(String[] args)
   {
      try{
      Class.forName("oracle.jdbc.driver.OracleDriver");
      Connection con=DriverManager.getConnection(
         "jdbc:oracle:thin:@localhost:1521:xe","system","manager");
      Statement stmt=con.createStatement();
      ResultSet rs=stmt.executeQuery("select * from student");
      ResultSetMetaData rm=rs.getMetaData();
      int n=rm.getColumnCount();
      for(int i=1;i<=n;i++)
      {
         System.out.print(rm.getColumnName(i)+"\t");
      }
       System.out.println();
       while(rs.next())
       {
          System.out.print(rs.getInt(1)+"\t");
          System.out.print(rs.getString(2)+"\t");
          System.out.println(rs.getInt(3));
       }
       }catch(Exception e)
       {
          System.err.println(e);
       }
   }
}
```

```java
while(rs.next()){
    System.out.print(rs.getInt("rollNo")+"\t");
    System.out.print(rs.getString("name")+"\t");
    System.out.println(rs.getInt("marks"));
}
```

```java
while(rs.next()){
    for(int i=1;i<=n;i++){
        System.out.println(rs.getString(i)+"\t");
    }
    System.out.println();
}
```

**ECLIPSE**:
Eclipse is an IDE (integrated development environment) for developing applications by using Java programming language & other's programming language like c,c++,perl,...etc
⇒IDE contains compiler, interpreter, debugger, editors, plugins,...etc.
⇒ A plugin is a software component that can be used to extend the functionality of an IDE.
⇒ Eclipse IDE software developed by Eclipse foundation and released in 2001.
⇒ Eclipse IDE software developed in Java Language.
⇒ There are two Eclipse software for Java.
1) Eclipse for Java
2) Eclipse for JavaEE
⇒ Eclipse for Java supports JavaSE applications only.
⇒ Eclipse for javaEE supports both javaSE & JavaEE applications.
⇒ Eclipse's official website is Here Eclipse .

⇒ To start Eclipse:
Double click on Eclipse icon from the following folder 🗁

```
Double click on eclipse icon from the following folder
C:\eclipse-jee-2021-12-R-win32-x86_64\eclipse
```

Select the workspace by click on browse button (Eclipse stores your projects in a folder is called workspace)
⇒Default workspace is 🗁

```
C:\Users\admin\eclipse-workspace
```

⇒click on launch button
**Perspective**:
A perspective is a layout of views & editors.
There are many perspectives:

1) Java perspective
2) JavaEE perspective
3) JPA (Java perspective API) perspective
4) XML (eXtensible Markup Language) perspective, …etc

To write core java programs , JDBC program & reflection API programs use java perspective.
To write Servlets, Servlet Listeners , Filters & JSP programs , use JavaEE perspective.
**To open perspective:**

1) Click on window menu
2) Click on perspective
3) Click on open perspective
4) Click other
5) Click select the perspective.
6) Click on open button

**To close perspective:**

1) Right click on perspective icon at top right corner
2) Click on close.

**To right JDBC program in Eclipse:**

1) Open Java perspective
2) Click on file menu
3) Click on new
4) Click on Java project
5) Type project name (example : jdbc)
6) Click on next button
7) Click on finish button
8) Click on the don't create button.
9) Right click on jdbc in a package explorer
10) Click on new button
11) Click on class
12) Type class name (example: ConnectionDemo)
13) Select main() method
14) Click on finish button
15) Write the program in the editor.

**To add JAR (java archive) files:**

1) Right click on jdbc in a package explorer
2) Click on build path
3) Click on configure build path
4) Select the classpath under libraries tab
5) Click on add external jars button
6) Select the jar file (example : ojdbc6_g.jar file.
7) Click on open button
8) Click on apply & close button

**To run the above applications:**

1) Right click on ConnectionDemo.java under jdbc project
2) Click on run as
3) Click on Java application

**Eclipse shortcut keys:**

1) Ctrl+w = close current editor(window)
2) Ctrl+shift+w = close all editors (windows)
3) Ctrl+s = save current program
4) Ctrl+shift+s = save all programs
5) Ctrl+shift+o = Organize import/delete statements.
6) Ctrl+shift+f = Format the text
7) Alt+shift+z (select the text) = to add try catch block
8) sysout (Ctrl+spaceBar) = to write System.out.println();
9) Ctrl+shift+L = to display shortcut keys list

**PreparedStatement interface:**

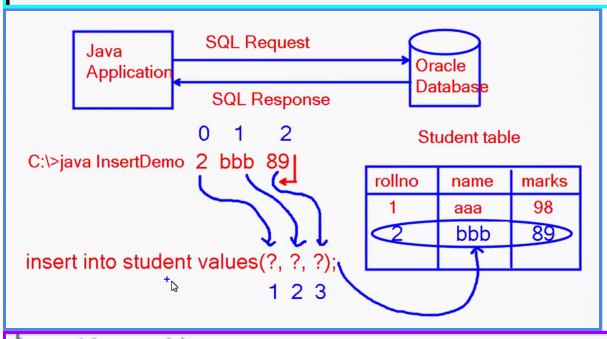
It is used to execute dynamic SQL queries.

Example:

1) insert into student values (?,?,?);
2) update student set marks=? Where rollno = ?;
3) delete from student where rollno =?;

If we use these types of queries with PreparedStatement interface only one time compiled and many times executed.

```
java.sql.PreparedStatement

Methods:
  public abstract boolean execute() throws SQLException;
  public abstract int executeUpdate() throws SQLException;
  public abstract ResultSet executeQuery() throws SQLException;
                                  question mark number
                                  value
  public abstract void setInt(int, int) throws SQLException;
  public abstract void setFloat(int, float) throws SQLException;
  public abstract void setString(int, String) throws SQLException;
=>The above 3 methods are used set the values.
```

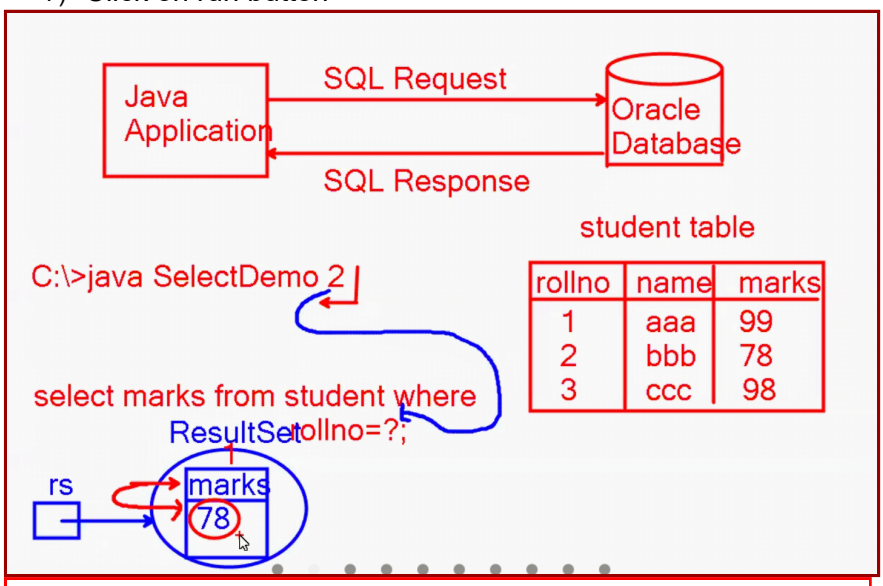



```
import java.sql.*;

class InsertDemo
{
   public static void main(String[] args)
   {
      try{
      Class.forName("oracle.jdbc.driver.OracleDriver");
      Connection con=DriverManager.getConnection(
         "jdbc:oracle:thin:@localhost:1521:xe","system","manager");
      PreparedStatement pstmt=con.prepareStatement(
                                    "insert into student values(?,?,?)");
      pstmt.setInt(1, Integer.parseInt(args[0]));
      pstmt.setString(2, args[1]);
      pstmt.setInt(3, Integer.parseInt(args[2]));
      pstmt.executeUpdate();
      System.out.println("One Record Inserted Successfully");
      }catch(Exception e)
      {
         System.err.println(e);
      }
   }
}
```

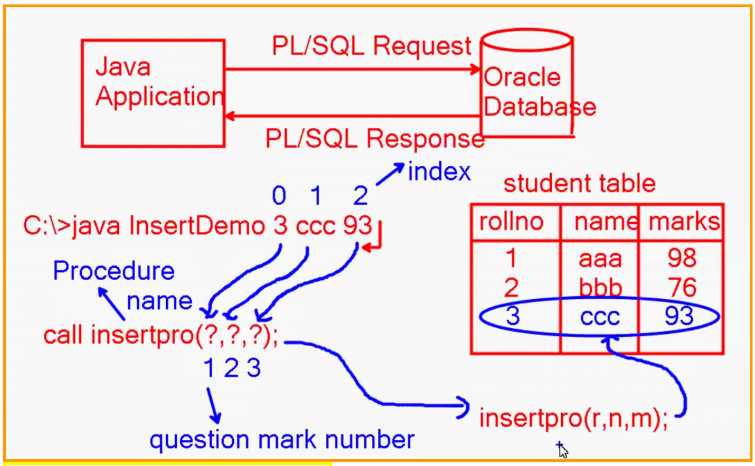
```
Statement interface:
===============
1) create table student(rollno number(3), name varchar2(10),
                                                marks number(3));
2) insert into student values(1,'aaa', 93);

3) select * from student;

PreparedStatement interface:
======================
4) insert into student values(?, ?, ?);

5) select marks from student where rollno=?;
```

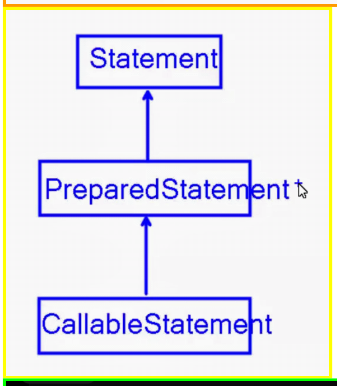
**To add command line arguments in Eclipse :**

1) Right click on insertDemo.java in package explorer.
2) Click on run as
3) Click on run configuration
4) Click on arguments tab
5) Type agreements in program Arguments box (example:9 iii 45)
6) Click on apply button
7) Click on run button




```
package jdbc;

import java.sql.*;

public class SelectDemo
{
    public static void main(String[] args)
    {
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection con=DriverManager.getConnection(
                "jdbc:oracle:thin:@localhost:1521:xe",
                                        "system", "manager");
            PreparedStatement pstmt=con.prepareStatement(
                        "select marks from student where rollno=?");
            pstmt.setInt(1, Integer.parseInt(args[0]));
            ResultSet rs=pstmt.executeQuery();
            rs.next();
            System.out.println(rs.getInt(1));
        } catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```




```
SQL> create or replace procedure insertpro(r student.r
ollno%type, n student.name%type, m student.marks%type)
as
  2  begin
  3  insert into student values(r,n,m);
  4  end;
  5  /

Procedure created.

SQL>
```

```
//Program to demonstrate CallableStatement
package jdbc;

import java.sql.*;

public class ProcedureDemo
{
    public static void main(String[] args)
    {
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1
            CallableStatement cstmt=con.prepareCall("{ call insertpro(?,?,?) }");
            cstmt.setInt(1, Integer.parseInt(args[0]));
            cstmt.setString(2, args[1]);
            cstmt.setString(2, args[1]);
            cstmt.setInt(3, Integer.parseInt(args[2]));
            cstmt.execute();
            System.out.println("One Record Inserted Successfully");
        } catch (Exception e)
        {
            e.printStackTrace();
        }
```

**ResultSet Enhancement:**

ResultSet Enhancements divided into two categories:

1) ResultSet Types
2) Concurrency Types

**ResultSet Types:**
There are three types of ResultSet.

1) public static final int TYPE_FORWARD_ONLY;
2) public static final int TYPE_SCROLL_INSENSITIVE;
3) public static final int TYPE_SCROLL_SENSITIVE;

The above 3 ResultSet Types are static members (variables) in the ResultSet interface.

**Concurrency Types:**
There are two types of concurrency control on ResultSet.

1) public static final int CONCUR_READ_ONLY;
2) public static final int CONCUR_UPDATABLE;

**ResultSet Types:**
A ResultSet is an object that encapsulates a set of rows from a database.
ResultSet is generated based on sql query.
Whenever ResultSet is generated then ResultSet pointer/cursor points before the first record.

1) **TYPE_FORWARD_ONLY:**

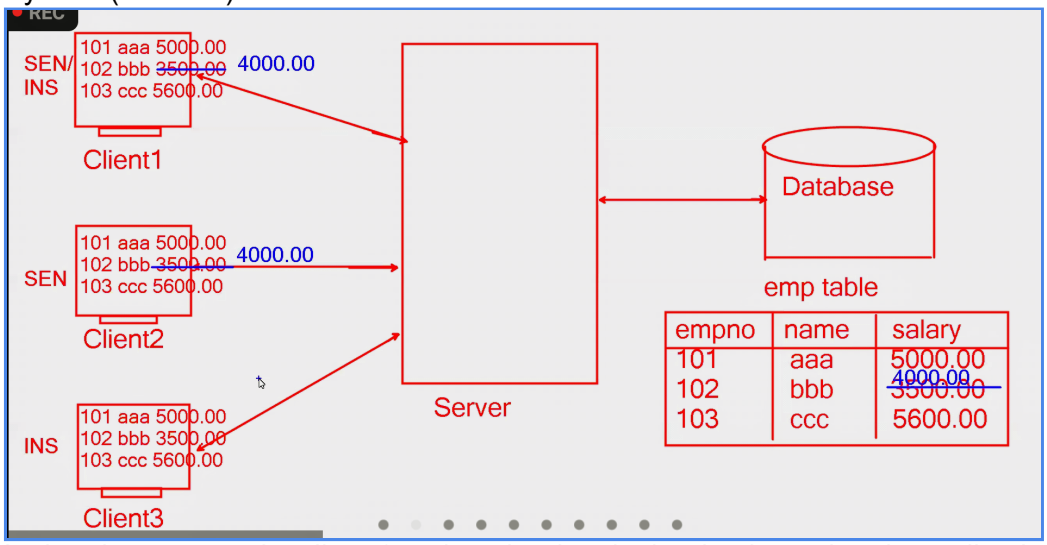
It is introduced in JDBC 1.0 version only
It supports only forward direction to iterate records in a ResultSet.
It does not support absolute & relative positions.
It is a default result set type.

2) **TYPE_SCROLL_INSENSITIVR**:

It is introduced in JDBC 2.0 version.
It supports both forward & backward
directions to iterate records in ResultSet.
It supports both absolute & relative position in a ResultSet.
It will not show changes made by others in a Distributed Database Management System (DDBMS).

3) **TYPE_SCROLL_SENSITIVE:**

It is introduced in JDBC 2.0 version.
It supports both forward & backward
directions to iterate records in ResultSet.
It supports both absolute & relative position in a ResultSet.
It will show changes made by others in a Distributed Database Management System (DDBMS).




In the above example, client 1 update the data in the database then immediately if it will be shown in client 2 because ResultSet type is SENSITIVE and it will not be shown in client 3 because ResultSet type is INSENSITIVE.

1) **CONCUR_READ_ONLY:**

It allows only read operation concurrently.

2) **CONCUR_UPDATABLE:**

It allows all operations concurrently.

By using ResultSet Enhancements feature we can do the following operations on ResultSet:

1. Moving a cursor in a scrollable ResultSet.
2. Updating records with methods.
3. Inserting records with methods.
4. Deleting records with methods.

### 1) Moving a cursor in a scrollable ResultSet:

```
java.sql.ResultSet

Methods:
  public abstract boolean isBeforeFirst() throws SQLException;
  public abstract boolean isAfterLast() throws SQLException;
  public abstract boolean isFirst() throws SQLException;
  public abstract boolean isLast() throws SQLException;
=>The above 4 methods are used to check the cursor position

  public abstract void beforeFirst() throws SQLException;
  public abstract void afterLast() throws SQLException;
  public abstract boolean first() throws SQLException;
  public abstract boolean last() throws SQLException;
=>The above 4 methods are used to move the cursor
```

```
  public abstract int getRow() throws SQLException;
=> It returns row number

  public abstract boolean previous() throws SQLException;
=> It returns true if the previous record is present, otherwise returns false

  public abstract boolean absolute(int) throws SQLException;
  public abstract boolean relative(int) throws SQLException;
```

```
rs.absolute(3); => It moves the cursor from starting to 3rd record in forward direction
rs.absolute(-3); => It moves the cursor from ending to 3rd record in backward direction
rs.relative(3); => It moves the cursor from current position to 3rd record in forward
                                                                                direction
rs.relative(-3); => It moves the cursor from current position to 3rd record in backward
                                                                                direction
```

```
java.sql.Connection
                                        ResultSet type      Concurrency type

Methods:
  public abstract Statement createStatement(int, int) throws SQLException;
  public abstract PreparedStatement prepareStatement(String, int, int)
                                                                    throws SQLException;
  public abstract CallableStatement prepareCall(String, int, int) throws SQLException;
```

```
package jdbc;

import java.sql.*;

public class MoveDemo
{
    public static void main(String[] args)
    {
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection con=DriverManager.getConnection(
                        "jdbc:oracle:thin:@localhost:1521:xe", "system", "manager");
            Statement stmt=con.createStatement(
                    ResultSet.TYPE_SCROLL_SENSITIVE,
                        ResultSet.CONCUR_UPDATABLE);
```

```
            ResultSet rs=stmt.executeQuery("select * from student");
            rs.absolute(5);
            System.out.print(rs.getInt("rollno")+"\t");
            System.out.print(rs.getString("name")+"\t");
            System.out.println(rs.getInt("marks"));
        } catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

Excep|

2) **Updating records with methods:**

java.sql.ResultSet

column number

value

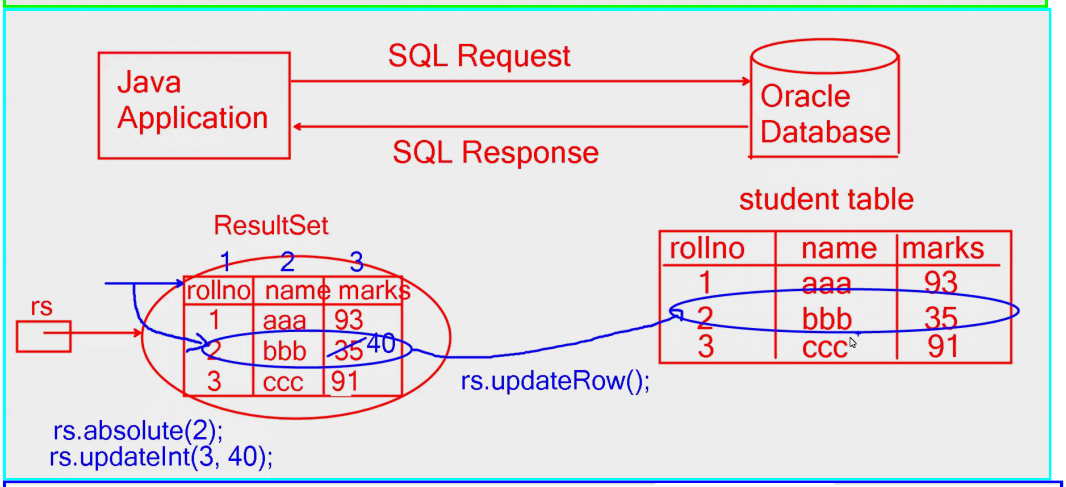
Methods:

```
public abstract void updateInt(int, int) throws SQLException;
public abstract void updateFloat(int, float) throws SQLException;
public abstract void updateString(int, String) throws SQLException;
```

=> The above 3 methods are used to update the ResultSet

```
public abstract void updateRow() throws SQLException;
```

=> It is used to update record in a database.




```java
import java.sql.Connection;

public class UpdateRowDataInTable {

    public static void main(String[] args) {
        try {
            Class c = Class.forName("oracle.jdbc.driver.OracleDriver");
            System.out.println("Driver Loaded Successfully");
            Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","sam","tiger");
            System.out.println("Connection Established Successfully");
            Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
            System.out.println("Statement Created");
            ResultSet rs = stmt.executeQuery("select rollno,name,marks from student");
            System.out.println("ResultSet Generated");
            rs.absolute(1);
            rs.updateInt(3, 40);
            rs.updateRow();
            System.out.println("One Recorded Updated Successfully :-)");
        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }
    }

}
```

3) **Inserting records with methods:**

Inserting records with methods:
==========
java.sql.ResultSet

Methods:
=======
public abstract void moveToInsertRow() throws SQLException;
=>It is used to move the cursor to insert a record.

public abstract void insertRow() throws SQLException;
=>It is used to insert a record in database.

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class InsertRowDataInTable {

    public static void main(String[] args) {
        try {
            Class c = Class.forName("oracle.jdbc.driver.OracleDriver");
            System.out.println("Driver Loaded Successfully");
            Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "sam", "tiger");
            System.out.println("Connection Established Successfully");
            Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
            System.out.println("Statement Created");
            ResultSet rs = stmt.executeQuery("select rollno,name,marks from student");
            System.out.println("ResultSet Generated");
            rs.moveToInsertRow();
            rs.updateInt(1, 11);
            rs.updateString(2, "kkk");
            rs.updateInt(3, 77);
            rs.insertRow();
            System.out.println("One Record Inserted Successfully :-)");
        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }
    }
}

```

4) **Deleting records with methods:**

Deleting records with methods:
==========
java.sql.ResultSet

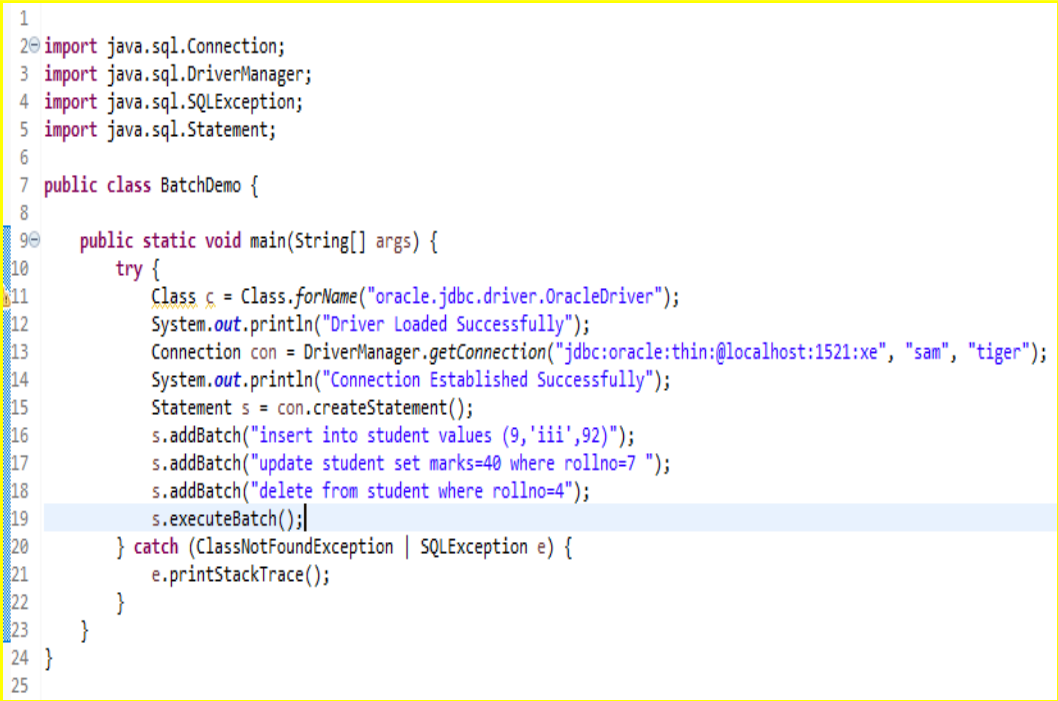
Method:
public abstract void deleteRow() throws SQLException;
=>It is used to delete a record in a database.

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class DeleteRowDataInTable {

    public static void main(String[] args) {
        try {
            Class c = Class.forName("oracle.jdbc.driver.OracleDriver");
            System.out.println("Driver Loaded Successfully");
            Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "sam", "tiger");
            System.out.println("Connection Established Successfully");
            Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
            System.out.println("Statement Created");
            ResultSet rs = stmt.executeQuery("select rollno,name,marks from student");
            System.out.println("ResultSet Generated");
            rs.absolute(3);
            rs.deleteRow();
            System.out.println("One Record Deleted Successfully :-( ");
        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }
    }
}

```

**Batch Updates:**
This features introduced in JDBC 2.0 version to execute more than one SQL query at a time.
It is also called as Batch Processing.
This feature used to reduce the network traffic.

java.sql.Statement

Methods:
public abstract void addBatch(String) throws SQLException;
=>It is used to add sql query.
public abstract int[] executeBatch() throws SQLException;
=>It is used to execute all sql queries at a time.




```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

public class BatchDemo {

    public static void main(String[] args) {
        try {
            Class c = Class.forName("oracle.jdbc.driver.OracleDriver");
            System.out.println("Driver Loaded Successfully");
            Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "sam", "tiger");
            System.out.println("Connection Established Successfully");
            Statement s = con.createStatement();
            s.addBatch("insert into student values (9,'iii',92)");
            s.addBatch("update student set marks=40 where rollno=7 ");
            s.addBatch("delete from student where rollno=4");
            s.executeBatch();
        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }
    }
}
```

**Advanced Data Types:**

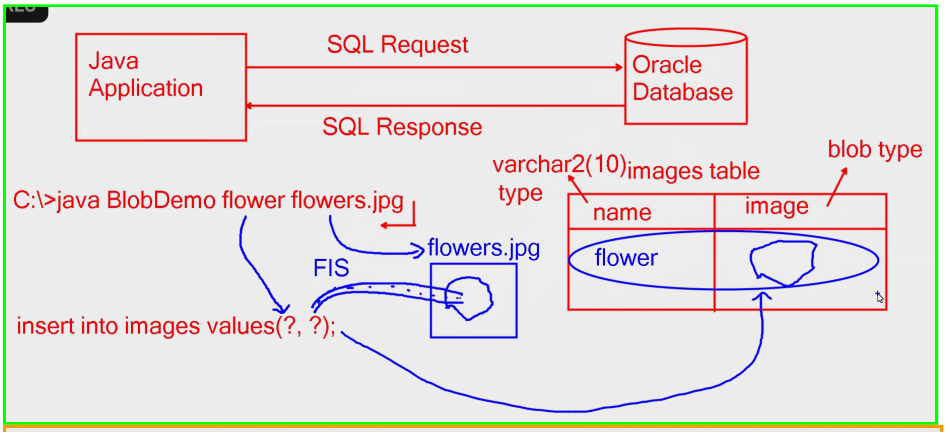
1) **BLOB**
2) **CLOB**

A. **BLOB :**

BLOB stands for Binary Large Object.
It can be used to store/retrieve large amount of binary data as a single entity in/from database.
It supports all types of data (text, image, graphics, animation, audio, video,... etc.)
It is mapped into java.sql.Blob interface in Java language.

B. **CLOB:**

CLOB stands for Character Large Object.
It can be used to store/retrieve large amount of Character data as a single entity in/from database.
It supports text only.
It is mapped into java.sql.Clob interface in Java language.




java.sql.PreparedStatement

Methods:

public abstract void setBinaryStream(int, InputStream, int) throws SQLException;
=>It is used to set the blob

public abstract void setCharacterStream(int, Reader, int) throws SQLException;
=>It is used to set the clob

question mark number
byte stream
stream length
question mark number
character stream
stream length

```java
import java.io.FileInputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class InsertImageInTable {

    public static void main(String[] args) {
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "sam", "tiger");
            PreparedStatement ps = con.prepareStatement("insert into images values(?,?)");
            ps.setString(1, "args[0]");
            FileInputStream fis = new FileInputStream(args[1]);//it opens a file for reading
            ps.setBinaryStream(2, fis, fis.available());//available() method returns length of file.
            ps.executeUpdate();
            System.out.println("One Image inserted successfully");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

**RowSets:**

A RowSets is an object that encapsulates set of rows from database.
RowSets is generated based on sql query.
Whenever RowSet in generated then RowSet pointer/cursor points to before first record.
There are five RowSets:

1) JdbcRowSet
2) CachedRowSet
3) WebRowSet
4) FilteredRowSet
5) JoinRowSet

The above all RowSets are interfaces in javax.sql.rowset package.
Differences between ResultSet & RowSets:

| ResultSet | RowSets |
|---|---|
| A ResultSet is non serializable object | All RowSets are serializable objects. |
| ResultSet is connected object | JdbcRowSet is a connected object and remaining RowSets are |

| | |
|---|---|
| | disconnected objects. |
| ResultSet is not a Java bean | All RowSets are Java beans. |
| In other to get ResultSet we need connection interface, DriverManager class & Statement interface. | In other to get RowSet we need RowSet implemention class only. |

**Serialization:**
It is a process of converting an object into a series of bits.
In Java, object must be serializable to do the following operations.

1) Writing an object into a file.
2) Reading an object from a file
3) Writing an object to a network.
4) Reading an object from a network.

Class must implements java.io.Serializable interface to make serializable object.

java.io.Serializable interfaces is called as marker interface, tag interface or empty interface because no members in this interface.

Example:
=======
```
import java.io.*;
class Emp implements Serializable
{
    int empNo;
    String name;
    float salary;
    =========
}
Emp e=new Emp();
```
Serializable object

**Connected object:**
It means the object always been connected to a database.

**Java Bean:**
A Java bean is a reusable software component.
A Java class is set to be Java bean if it is follows the following rules:

1) Class must be public.
2) Class must implements java.io.Serializable interface.
3) Class must be in a package.
4) Class must contain public default constructor.
5) Instance variables of a class must be private. (Instance variables are called property)
6) Each & every property must contain setter & getter Methods.
7) All setter & getter Methods must be public.

Java Bean Example:
==========
```
package demo;

import java.io.*;

public class MessageBean implements Serializable
{
    private String message;
    public void setMessage(String message)
    {
        this.message=message;
    }
    public String getMessage()
    {
        return message;
    }
}
```

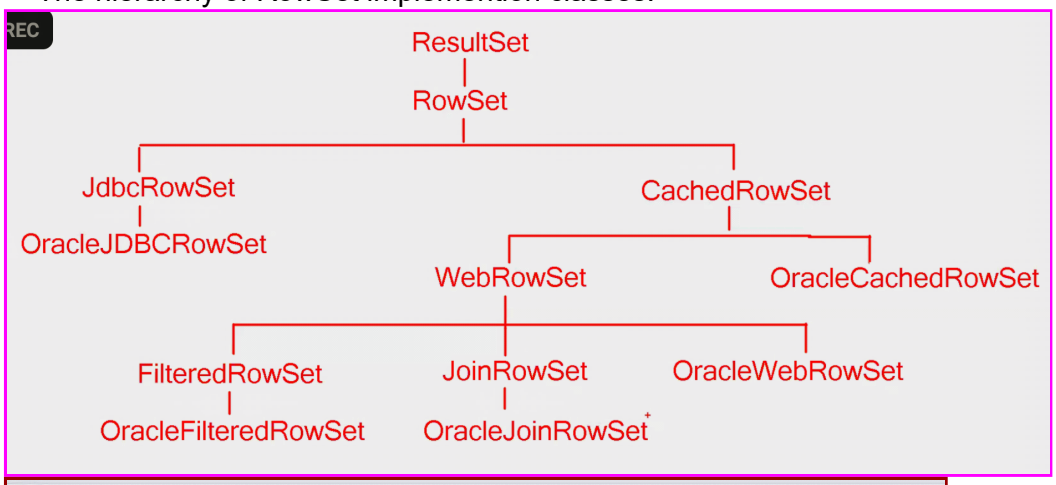
There are five RowSets:

- JdbcRowSet
- CachedRowSet
- WebRowSet
- FilteredRowSet
- JoinRowSet

The above all RowSet are interfaces see implemented by database vendors.
Oracle corporation implemented the above interfaces in the following classes:

1) OracleJDBCRowSet
2) OracleCachedRowSet
3) OracleWebRowSet
4) OracleFilteredRowSet
5) OracleJoinRowSet

The above RowSet implemention classes are the part of oracle.jdbc.rowset package
⇒ oracle.jdbc.rowset package is a part of ojdbc6_g.jar & ojdc8_g.jar file in Oracle software.
⇒ The hierarchy of RowSet implemention classes:




javax.sql.RowSet

Methods:

```
public abstract void setUrl(String) throws SQLException;
public abstract void setUsername(String) throws SQLException;
public abstract void setPassword(String) throws SQLException;
public abstract void setCommand(String) throws SQLException;
public abstract void execute() throws SQLException;
```

SQL Query

**JDBCRowSet:**
It is a serializable object.
It is a Connected object.
It is a Java bean.
In other to get JDBCRowSet, we require JDBCRowSet implemention class (OracleJDBCRowSet) only.
**Explainer Program here :**

```
//Program to demonstrate JdbcRowSet
package jdbc;

import java.sql.*;
import javax.sql.rowset.*;
import oracle.jdbc.rowset.*;

public class JRSDemo
{
    public static void main(String[] args)
    {
        try {
            JdbcRowSet jrs=new OracleJDBCRowSet();
            jrs.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
            jrs.setUsername("system");
            jrs.setPassword("manager");
            jrs.setCommand("select * from student");
            jrs.execute();
            while(jrs.next())
            {
                System.out.print(jrs.getInt("rollno")+"\t");
                System.out.print(jrs.getString("name")+"\t");
                System.out.println(jrs.getInt("marks"));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
```

**CachedRowSet:**
It is a serializable object.
It is a disconnected object.
It is a Java bean.
In other to get CachedRowSet, we require CachedRowSet implemention class (OracleCachedRowSet) only.
**Explainer Program here :**

```
package jdbc;

import java.sql.SQLException;
import javax.sql.rowset.CachedRowSet;
import oracle.jdbc.rowset.OracleCachedRowSet;

public class CRSDemo
{
    public static void main(String[] args)
    {
        try {
            CachedRowSet crs=new OracleCachedRowSet();
            crs.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
            crs.setUsername("system");
            crs.setPassword("manager");
            crs.setCommand("select * from student");
            crs.execute();
            while(crs.next())
            {
                System.out.print(crs.getInt(1)+"\t");
                System.out.print(crs.getString(2)+"\t");
                System.out.println(crs.getInt(3));
            }
        } catch (SQLException e)
        {
            e.printStackTrace();
        }
```

**WebRowSet:**
It is a serializable object.
It is a disconnected object.
It is a Java bean.
It allows to write WebRowSet data to xml file.
Generated XML file can be used in web applications.
(XML stands for eXtensible Markup Language).
In other to get WebRowSet, we require WebRowSet implementation class (OracleWebRowSet) only.

javax.sql.rowset.WebRowSet

Character Stream

Methods:
public abstract void writeXml(Writer) throws SQLException;
public abstract void writeXml(OutputStream) throws SQLException, IOException;
=>The above methods are used to write WebRowSet data to xml file

Byte Stream

Opens a file for writing

FileWriter fw=new FileWriter("student.xml");
FileOutputStream fos=new FileOutputStream("student.xml");

**Explainer Program here :**

```
package jdbc;

import java.io.*;
import java.sql.*;
import javax.sql.rowset.*;
import oracle.jdbc.rowset.*;

public class WRSDemo
{
    public static void main(String[] args)
    {
        try {
            WebRowSet wrs=new OracleWebRowSet();
            wrs.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
            wrs.setUsername("system");
            wrs.setPassword("manager");
            wrs.setCommand("select * from student");
            wrs.execute();
            FileOutputStream fos=new FileOutputStream("C:/student.xml");
            wrs.writeXml(fos);
        } catch (Exception e)
        {
            e.printStackTrace();
        }
    }
```

**FilteredRowSet :**
It is a serializable object.
It is a disconnected object.
It is a Java bean.
It allows filtering operations on RowSet data.
In other to get FilteredRowSet, we require FilteredRowSet implemention class (OracleFilteredRowSet) only.

**JoinRowSet :**
It is a serializable object.
It is a disconnected object.
It is a Java bean.
It allows to join two or more RowSets data.
In other to get JoinRowSet, we require JoinRowSet implemention class (OracleJoinRowSet) only.

java.util.Scanner

Constructor:
public Scanner(java.io.InputStream);

Methods:
public String next();
=>It is used read one word
public String nextLine();
=>It is used to read one line of text
public boolean nextBoolean();
public byte nextByte();
public short nextShort();
public int nextInt();
public long nextLong();
public float nextFloat();
public double nextDouble();
=> The above all methods are used to accept the data from keyboard.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.util.Scanner;
public class EnterDataToDBByScanner {
    public static void main(String[] args) throws Exception {
        Scanner s = new Scanner(System.in);
        System.out.println("Enter Roll Number: ");
        int rollno = s.nextInt();
        System.out.println("Enter Name: ");
        String name = s.next();
        System.out.println("Enter Marks: ");
        int marks = s.nextInt();
        Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "sam",
"tiger");
        con.setAutoCommit(false);
        PreparedStatement ps = con.prepareStatement("insert into student
values(?,?,?)");
        ps.setInt(1, rollno);
        ps.setString(2, name);
        ps.setInt(3, marks);
        ps.executeUpdate();
        System.out.println("Enter Save/Cancel to Commit : ");
        String option = s.next();
        if (option.equalsIgnoreCase("Save")) {
            con.commit();
        } else if (option.equalsIgnoreCase("Cancel")) {
            con.rollback();
        } else {
            System.out.println("Invalid options !!! ");
        }
        System.out.println("One record inserted successfully");
    }
}
```

**Transaction Management in JDBC:**
One transaction can have one or more operations.
To manage transaction manually , we use the following methods of java.sql.Connection interface.

java.sql.Connection

Methods:
public abstract void setAutoCommit(boolean) throws SQLException;
=> It is used to set auto commit mode
public abstract boolean getAutoCommit() throws SQLException;
=> It is used to get auto commit mode
public abstract void commit() throws SQLException;
=> It is used to save transaction
public abstract void rollback() throws SQLException;
=> It is used to cancel transaction

public abstract Savepoint setSavepoint() throws SQLException;
=> It is used to create a save point
public abstract Savepoint setSavepoint(String) throws SQLException;
=> It is used to create a save point with specified name
public abstract void rollback(Savepoint) throws SQLException;
=> It is used to cancel save point transactions
public abstract void releaseSavepoint(Savepoint) throws SQLException;
=> It is used to remove save point

Note: By default auto commit mode is true.

```
//Program to demonstrate Savepoint
import java.sql.*;
public class EnterDataToDB {
    public static void main(String[] args) throws Exception {
        Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "sam",
"tiger");
        Statement stmt = con.createStatement();
        con.setAutoCommit(false);
        Savepoint s = con.setSavepoint();
        stmt.executeUpdate("update student set marks = 50 where
rollno=7");
        con.rollback(s);
        stmt.executeUpdate("delete from student where rollno=8");
        con.commit();
    }
}
```

**Note:** A save point is a logical position in a transaction Management.

| Version | New Features |
|---|---|
| JDBC 1.x | Initial Specification |
| JDBC 2.x | 1) Result Enhancements<br>2) Batch Updates(Batch Processing)<br>3) Advanced Data Types(BLOB & CLOB) |
| JDBC 3.x | 1) Rowsets<br>2) Savepoint interface |
| JDBC 4.x | 1) Auto Loading Driver Facility |

# SERVLETS

**SERVLETS:**
Servlets is a specification for developing web applications with Java programming language.

**Web Application:**
A Web application is a Distributed application which runs on browser & server.

**Distributed Application :**
An application that is installed on one computer & runs on many computers is called as distributed application.

**There are two types of web application:**

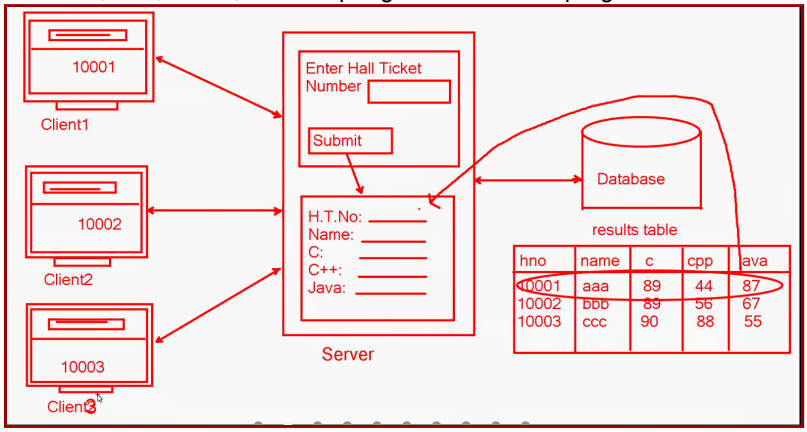
1) **Static web application**
2) **Dynamic web application**

1. **Static web application:**

A Web application that is already prepared and placed in server is known as static web application.
Static web applications are common to all users.
Static web application resides in server & runs in browser.
There server sends the program to browser whenever request comes to static web application
Static web applications can be developed by using html, css, javascript, VB script, Applets, … etc.
HTML stands for Hyper Text Markup Language.
CSS stands for Cascading Style Sheets.
VB Script stands for Visual Basic Script.
Applets are Java program.

2. **Dynamic Web Application:**

A Web application that is prepared dynamically whenever request comes to a server is known as dynamic web application.
Dynamic web application are specific to users.
Dynamic web application resides in server & runs in server only.
The server executes the program & sends the output to a browser whenever request comes to a dynamic web application.
Dynamic web applications can be developed by using Servlets, JSP, Struts , JSF, Spring MVC , ASP , ASP .Net ,....etc.
JSP stands for Java Server Pages.
JSF stands for Java Server Faces.
MVC stands for Model View Controller.
ASP stands for Active Server Pages.
Servlets , JSP, Struts, JSF & Spring MVC are Java programs.




**Browser:**

A browser is a software that executes web pages containing text, image, graphics, animation, audio and, video, …etc.
Browser is called as web client.


**List of Web Browsers:**

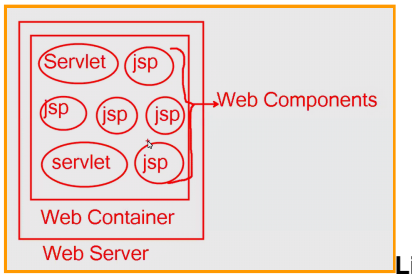
1) Internet Explorer
2) Google Chrome
3) Mozilla Firefox
4) Mosaic
5) New Planet
6) Netscape Navigator
7) Opera
8) Microsoft Edge
9) Hot Java
10) Cyber Dog
11) America Online
12) Win Web
13) Mac web
14) Lynx
15) Eye Browse, …etc.

**Server:**
A server is a software which recieves request from the client, process the request, constructs the response & sends the response back to a client.
**There are two types of servers:**

1) Web servers
2) Application servers

1. **Web servers:**

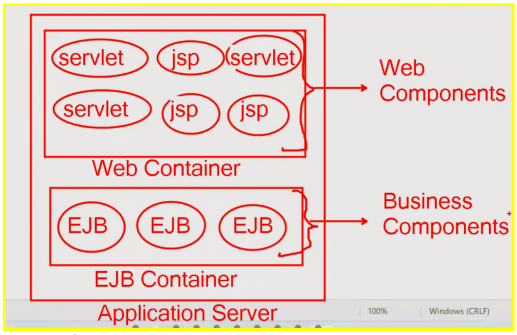
A server is a server which contains only web container.




**List of web servers :**

1) Tomcat web server
2) iPlanet web server
3) Java Web server
4) Resin web server
5) Pramathi web server
6) Personal web server
7) Jetty web server , ….etc.

**2. Application servers:**
An application server is a server which contains both web container & EJB container.
EJB stands for Enterprise Java Beans.

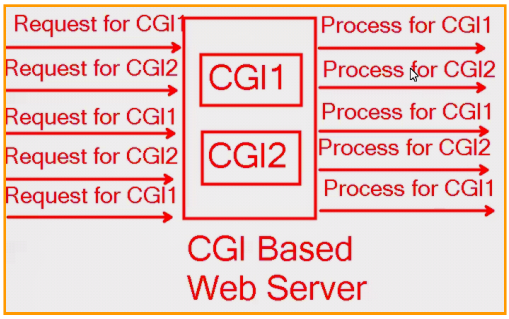
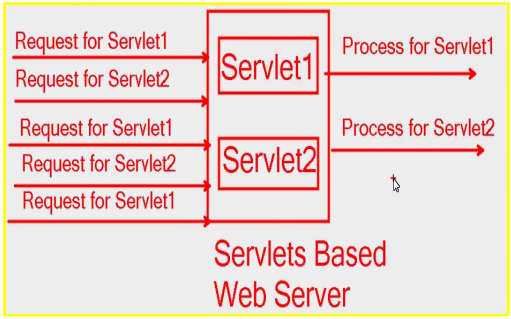



**List of application servers:**

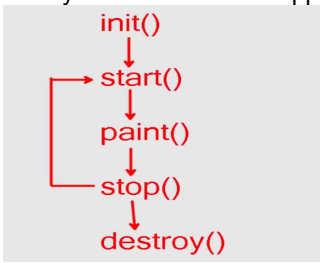
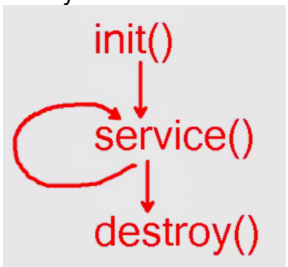
1) Weblogic application server
2) Web sphere application server
3) JBoss Application server(wildfly application server)
4) Sun one application server
5) Glass fish application server, …etc.

**Servlet** runs under web container.
Web container is a part of web server & application server.
Servlets specification used by vendors to develop web containers.
Servlets specification used by Java programmers to develop web applications.

**CGI (**Common Gateway Interface) **Vs Servlets:**

| **CGI** | **Servlets** |
|---|---|
| CGI is a specification for developing web applications with c, c++, Perl, …etc. | Servlets specification for developing web applications with Java programming language. |
| CGI based web server creates a new process for every request. | Servlets based web server creates a new process for very first request only. (Remaining requests are handled by child process). |
|   |   |

**Applets vs Servlets :**

| **Applets** | **Servlets** |
|---|---|
| An applet is a Java program that resides in server & runs in browser. | A servlets is a Java program that resides in server & runs in server only. |
| Applets are used to extend the functionality of browser. | Servlets are used to extend the functionality of server. |
| Applets do not have main() method because applet runs in browser. | Servlets do not have main() method because Servlet runs in server. |
| Applet has a life cycle methods to run in browser. | Servlets has a life cycle methods to run in server. |
| Life cycle methods of an Applets:   | Life cycle methods of a Servlet:   |

| | |
|---|---|
| init() method called by browser whenever an applet is opened. | init() method called by Web container whenever first request comes to a Servlet. |
| start() method called by browser whenever applet is opened and activated. | service() method called by Web container for every request. |
| paint() method called by browser whenever applet is opened and activated. | destroy() method called by Web container whenever servlets instance is removed from web container. |
| stop() method called by browser whenever applet is deactivated and closed. | Servlet instance is removed from web container Whenever web application is undeployed or server shuts down. |
| destroy() method called by browser whenever an applet is closed. | |
| The above life cycle methods are the part of java.applet.Applet class. | The above life cycle methods are the part of javax.servlet.Servlet interface |
| Every applet must extends java.applet.Applet class to derive life cycle methods. | Every servlets must implements javax.servlet.Servlet interface to derive life cycle methods. |
| Every Applet class must be public because it should be accessible to browser to create an object to call life cycle methods. | Every Servlet class must be public because it it should be accessible to web container to create an object to call life cycle methods. |

**Declaration rules to a source file (.java file) :**

1) A source file can have only one public class.
2) A source file can have any number of non public classes.
3) If the sources file contains public class then file name must be match with public class name.
4) If the source file does not contain any public class then no Naming restrictions to a file name.

**Servlets API:**

SERVLETS API is a Java API that can be used to develop web applications with Java programming language.

1) javax.servlet package (or) jakarta.servlet package
2) javax.servlet.http package (or) jakarta.servlet.http package

⇒ upto Tomcat 9 → javax
⇒ Tomcat 10 onwards → Jakarta

**1.javax.servlet package (or) jakarta.servlet package:**

| **Classes** | **Interfaces** |
|---|---|
| GenericServlet | Servlet |
| ServletInputStream | ServletRequest |
| ServletOutputStream | ServletResponse |
| ServletException | ServletConfig |
| UnavailableException | ServletContext |
| | RequestDispatcher |
| | SingleThreadModel |

**2. javax.servlet.http package (or) jakarta.servlet.http package:**

| **Classes** | **Interfaces** |
|---|---|
| HttpServlet | HttpServletRequest |
| Cookie | HttpServletResponse |
| | HttpSession |

Servlets API is a part of servlet-api.jar file in Tomcat server.

**Steps to download Tomcat Web Server :**

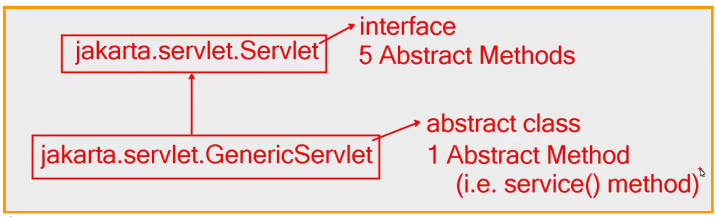
1) Open the browser
2) Open Google
3) Type Tomcat download in search bar
4) Click on the following Hyper link tomcat or direct download for windows Tomcat 10
5) Click on Tomcat 10 hyperlink under download section Tomcat 10

6) Click on 32/64-bit Windows Service Installer hyperlink

**Setting class path to access API :**

1) Open file explorer
2) Right click on this PC
3) Click on properties
4) Click on advanced system settings
5) Click on advanced tab
6) Click on Environment Variable button
7) Select the classpath variable under user variables
8) Click on edit button
9) Click on new button
10) Type the following
    C:\Program Files\Apache Software Foundation\Tomcat 10.0\lib\servlet-api.jar
11) Click on ok button. Done.

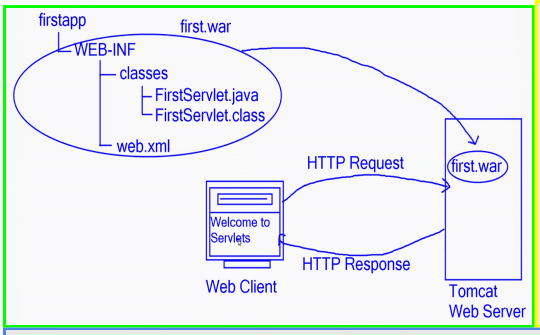
```
jakarta.servlet.Servlet
  public abstract void init(ServletConfig) throws ServletException;
  public abstract ServletConfig getServletConfig();
  public abstract void service(ServletRequest, ServletResponse) throws
ServletException, IOException;
  public abstract java.lang.String getServletInfo();
  public abstract void destroy();
```

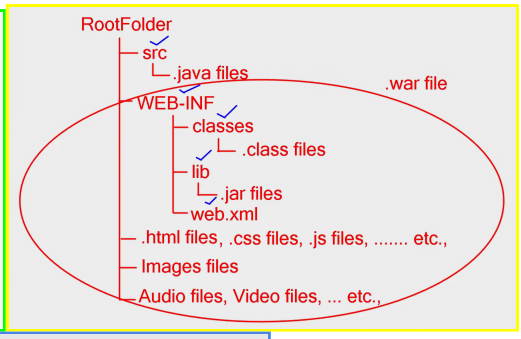



**Steps to develop web application:**

1) Create and compile web application source code(Servlet Program)
2) Write deployment descriptor(web.xml)
3) Create a WAR (Web ARchive) file.
4) Deploy a WAR file on server.

**Web application folder 🗀 structure:**




jakarta.servlet.ServletResponse

Methods:
byte stream
public abstract ServletOutputStream getOutputStream() throws IOException;
public abstract PrintWriter getWriter() throws IOException;
character stream

```
C:\>md firstapp

C:\>cd firstapp

C:\firstapp>md WEB-INF

C:\firstapp>cd WEB-INF

C:\firstapp\WEB-INF>md classes

C:\firstapp\WEB-INF>cd classes

C:\firstapp\WEB-INF\classes>start notepad FirstServlet.java
```

```
C:\firstapp\WEB-INF\classes>javac FirstServlet.java

C:\firstapp\WEB-INF\classes>cd..

C:\firstapp\WEB-INF>start notepad web.xml
```

```
<web-app>
<servlet>
<servlet-name>demo</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>demo</servlet-name>
<url-pattern>/test</url-pattern>
</servlet-mapping>
</web-app>
```

```
C:\firstapp\WEB-INF>cd..

C:\firstapp>jar cvf first.war .
```

c => Create
v => Verbose(Details of WAR file displayed)
f => File to be created
. => All files & folders of current directory

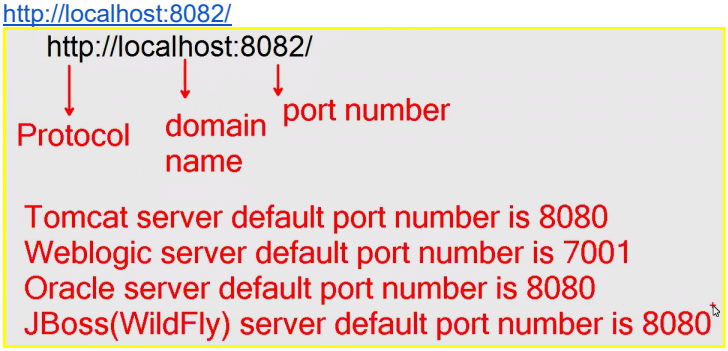
**jar:**
It is a jdk tool and it is used to create JAR (Java Archive) files, WAR(Web Archive) files , EAR(Enterprise Archive) files & RAR (Resource Archive) files.

**To start Tomcat Web Server:**

1) Open the following folder
   C:\Program Files\Apache Software Foundation\Tomcat 10.0\bin
2) Double click on Tomcat10.exe icon

**To open Tomcat Homepage:**

1) Open the browser
2) Type the following in address bar
   http://localhost:8082/



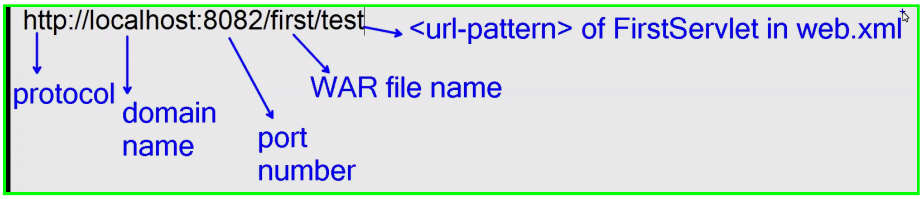

**To deploy a WAR file on server:**

1) Click on manager app button.
2) Type username (ex-admin)
3) Type password (ex-admin)
4) Click on sign in button
5) Scholl down
6) click on choose file button
7) Select the WAR file (first.war) from firstapp folder
8) Click on open button
9) Click on deploy button

**To run the above Tomcat application:**

1) Open the browser
2) Type the following in address bar
   http://localhost:8082/first/test

**Deployment location in a Tomcat Web Server:**
C:/Program Files/Apache Software Foundation/Tomcat 10.0/webapps




**http:**
HTTP stands for Hyper Text Transfer Protocol.
It transfer hyper text.
Hyper text means HTML text.
HTML stands for Hyper Text Markup Language.
This Protocol used by browser & server to communicate on the web.

**localhost:**
It is called domain name.
If the server is installed on same computer then use localhost as a domain name.
If the server is installed on other computer then use computer name as a domain name.

**8082:**
It is a port number and it is used to identify the service.
Tomcat server default port number is 8080.

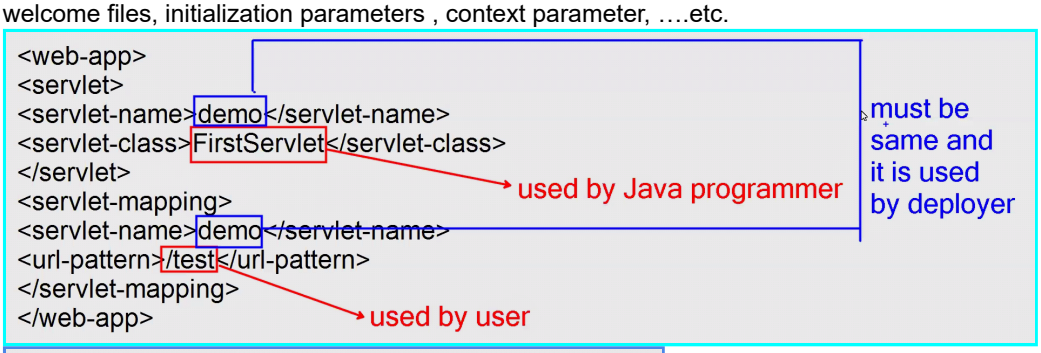
**To change the port number of Tomcat Web Server:**

1) Open server.xml file from the following location:

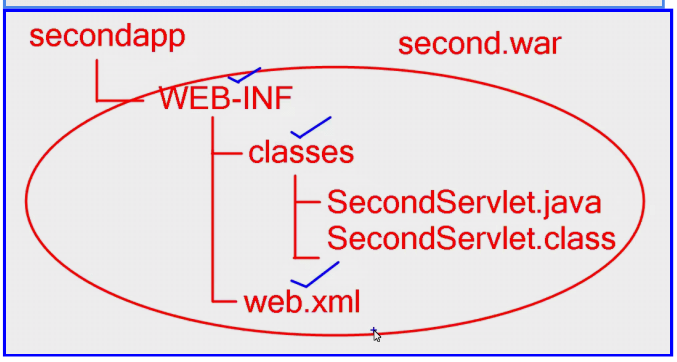
C:/Program Files/Apache Software Foundation/Tomcat 10.0/conf

2) Change the port number 8080 to 8082 in the following link where <Connector port = "8080" ……/>

**web.xml:**
It is called as configuration file and it is used to configure servlets, listeners, filters, JSPs, welcome files, initialization parameters , context parameter, ….etc.







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

public class SecondServlet extends GenericServlet
{
    public void service(ServletRequest req, ServletResponse res)
    {
        try{
        PrintWriter pw=res.getWriter();
        pw.println("<html><body bgcolor=yellow text=red><h1>");
        pw.println("Welcome to Naresh i Technologies");
        pw.println("</h1></body></html>");
        }catch(Exception e)
        {
```

```
<web-app>
<servlet>
<servlet-name>second</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>second</servlet-name>
<url-pattern>/second</url-pattern>
</servlet-mapping>
</web-app>
```

In the above application servlet code executed in server, html code transferred to browser & *html code executed in browser.

**To configure Tomcat Web Server in eclipse:**

1) Open JavaEE perspective.
2) Click on servers view
3) Right click in a servers view.
4) Click on new.
5) Click on server
6) Expand Apache
7) Select Tomcat v10.0 server
8) Click on next button
9) Select Tomcat installation directory by clicking on browse button.

Example: C:\Program Files\Apache Software Foundation\Tomcat 10.0

10) Click on select folder button.
11) Click on next button
12) Click on finish button.

**To check port numbers:**

1) Double click on Tomcat v10.0 server at localhost in a Servers view.
2) Tomcat admin port number must be 8005.
3) HTTP/1.1 pretty number be 1024 to 65535 (example :8082)

**Note : default port number is 8080**

**To start Tomcat server:**

1) Right click on Tomcat v10.0 servers at localhost in a server view.
2) Click on start.

**To stop Tomcat server:**

1) Right click on Tomcat v10.0 servers at localhost in a server view.
2) click on stop.

**To create a Web application folder structure :**

1) Click on file menu
2) Click on new
3) Click on Dynamic web project.
4) Type project name (ex - first)
5) Click on next button
6) Again click on next button.
7) Select generate web.xml deployment descriptor check box.
8) Click on finish button.

**To write servlet program:**

1) Write click on first in a project explorer.
2) Click on new
3) Click on servlet
4) Type package name (ex-first)
5) Type class name(ex-FirstServlet)

6) Type super class jakarta.servlet.GenericServlet
7) Click on next button
8) Select /FirstServlet in URL mappings box to change.
9) Click on edit button
10) Type /test in Pattern box
11) Click on ok button
12) Click on next button
13) Select jakarta.servlet.Servlet interface in interfaces box.
14) Click on remove button
15) Click on finish button.
16) Write the code in editor

**Example :**

```
PrintWriter pw = response.getWriter();
pw.println("welcome to servlets");
```

**To run the above application:**

1) Right click on first in a Project explorer
2) Click on run as
3) Click on run on server
4) Click on save button
5) Expand localhost
6) Then select Tomcat v10.0 server.
7) Click on next button
8) Click on finish button
9) Type /test (url-pattern) in address bar.

**To change the browser:**

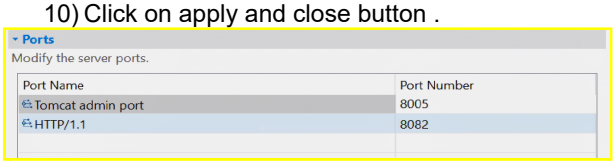
1) Click on window menu
2) Click on web browser
3) Click on internal web browser or choose your browser.

**To add servlet-api.jar in Eclipse:**

1) Right click on first in a Project Explorer
2) Click on Buildpath
3) Click on Configure Build Path
4) Click on Libraries tab
5) Click on Classpath
6) Click on add External JARs button
7) Select servlet-api.jar file from the following location:

C:\Program Files\Apache Software Foundation\Tomcat 10.0\lib

8) Click on open button
9) Click on apply button
10) Click on apply and close button .




The above classes are introduced in JDK 1.8 version in 2014.

New Date & Time API:
==========
java.time package
==========
Classes:
=======
1) LocalDate class
2) LocalTime class
3) LocalDateTime class

java.time.LocalTime

Methods:
public static LocalTime now();
public int getHour();
public int getMinute();
public int getSecond();
public int getNano();

```
package basics;
import java.io.*;
import java.time.LocalTime;
```

```
import jakarta.servlet.*;
public class TimeServlet extends GenericServlet {
    public void service(ServletRequest request,
ServletResponse response) throws ServletException, IOException {
        LocalTime lt = LocalTime.now();
        int h = lt.getHour();
        int m = lt.getMinute();
        PrintWriter pw = response.getWriter();
        pw.println("<html><body bgcolor=red text=yellow><h1>");
        pw.println("Present Time: " + h + ":" + m);
        pw.println("</h1></body></html>");
    }
}
```

```
package basics;
import java.io.*;
import jakarta.servlet.*;
public class CounterServlet extends GenericServlet {
    int count;
    public void service(ServletRequest request,
ServletResponse response) throws ServletException, IOException {
        count++;
        PrintWriter pw = response.getWriter();
        pw.println("<html><body bgcolor=yellow text=blue><h1>");
        pw.println("This page has been accessed "+count+" times");
        pw.println("</h1></body></html>");
    }
}
```

Program ⅘ copy 😵 from pc..

```
//program to demonstrate ServletOutputStream class
package basics;
import jakarta.servlet.*;
import java.io.*;
public class ImageServlet extends GenericServlet
{
    public void service(ServletRequest request ,
ServletResponse response)throws ServletException, IOException
    {
        ServletOutputStream sos = response.getOutputStream();
        FileInputStream fis = new FileInputStream("C:/image.jpg"); //it
opens a file for reading
        int n = fis.available(); //it returns file size
        byte[] b = new byte[n]; //it allocates the memory in a ram
        fis.read(b); //it will read image from file
        sos.write(b); //it passes image to a browser
    }
}
```

**MIME types:**
MIME stands for multipurpose internet mail extensions.
These types are used by browsers & servers to identify the content.
**List of MIME types:**

1) "text/html"
2) "text/xml"
3) "text/pdf"
4) "application/ms-word"
5) "application/vnd.ms-excel"
6) "image/jpg"
7) "image/bmp" ….etc.

⇒ Default MIME type is "text/html"
⇒ to change the MIME type use setContentType() method of jakarta.servlet.ServletResponse interface.

```
//program to demonstrate MIME type
package basics;
import jakarta.servlet.*;
import java.io.*;
public class MIMEServlet extends GenericServlet
{
    public void service(ServletRequest request ,
ServletResponse response)throws ServletException, IOException
    {
        response.setContentType("application/msword");
        PrintWriter pw = response.getWriter();
        pw.println("Welcome.......");
    }
}
```

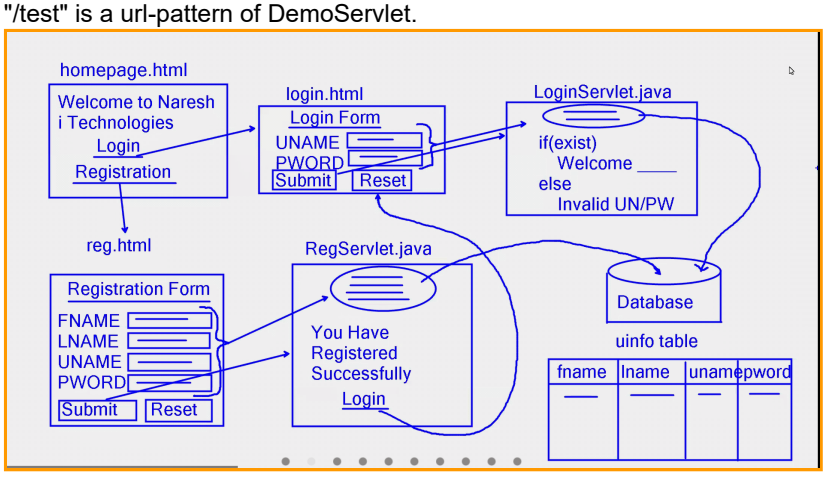
**Annotations :**
Annotations are meta tags that are used to pass some additional information to web container about servlets, listeners & filters.
All annotations begins with @symbols.
The following annotations are used in servlets:

1) @WebServlet
2) @WebListener
3) @WebFilter

The above all annotations are in jakarta.servlet.annotation package.

```
//program to demonstrate @WebServlet Annotation
package demo;
import jakarta.servlet.*;
import jakarta.servlet.annotation.*;
import java.io.*;
@WebServlet("/test")
public class DemoServlet extends GenericServlet
{
    public void service(ServletRequest request ,
ServletResponse response)throws ServletException, IOException
    {
        PrintWriter pw = response.getWriter();
        pw.println("<html><body bgcolor=blue text=yellow><h1>");
        pw.println("Welcome.......");
        pw.println("</h1></body></html>");
    }
}
```

In the above example @WebServlet annotation informs the following to web container.
"/test" is a url-pattern of DemoServlet.




**homepage.html file name of this given program**

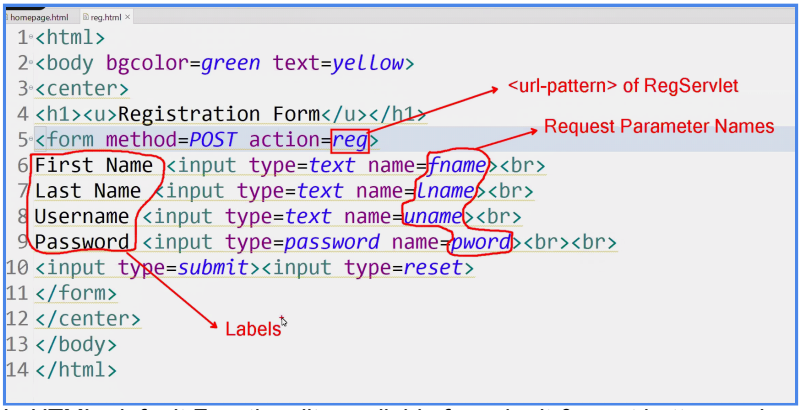
```
<!DOCTYPE html>
<html>
<head>
<title>Homepage</title>
```

```
</head>
<body>
<body bgcolor=yellow text=red>
    <center>
        <h1>Welcome to Naresh i Technologies</h1>
         <a href=login.html>Login</a><br>
        <a href=reg.html>Registration</a>
    </center>
</body>
</body>
</html>
```

<html> tag used to write html program
<body> tag used to write body part of the html
bgcolor attribute used to specify back ground color
text attribute used to specify text color
 tag for center alignment
<h1> tag for level 1 heading text
<a> tag for hyper link
href attribute used to specify hyper link reference
<br> tag for line break

**reg.html file name of this given program**

```
<html>
<head>
    <title>Registration</title>
</head>
<body bgcolor=green text=yellow>
    <center>
    <h1><u>Registration Form</u></h1>
    <form method=POST action=reg>
    First Name <input type=text name=fname><br>
    Last Name <input type=text name=Lname><br>
    Username <input type=text name=uname><br>
    Password <input type=password name=pword><br><br>
    <input type=submit><input type=reset>
    </form>
    </center>
</body>
</html>
```



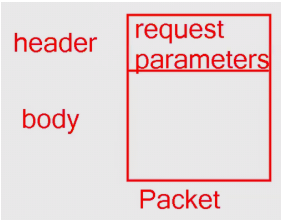
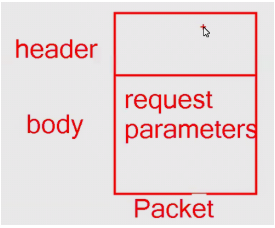

In HTML, default Functionality available for submit & reset buttons only.
The value of action structure is executed whenever submit button is clicked.
All fields data erased whenever reset button is clicked.

**login.html file name of this given program**

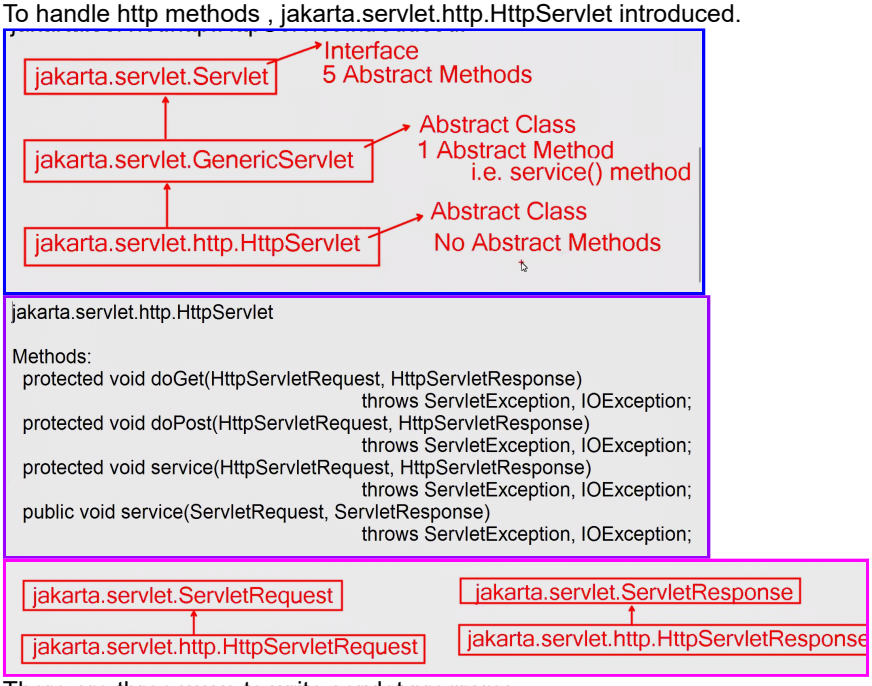
```
<html>
<head>
    <title>Login</title>
```

```
</head>
<body bgcolor=green text=yellow>
    <center>
    <h1><u>Login Form</u></h1>
    <form method=POST action=login>
    Username <input type=text name=uname><br>
    Password <input type=password name=pword><br><br>
    <input type=submit><input type=reset>
    </form>
    </center>
</body>
</html>
```

**GET vs POST:**

| GET | POST |
|---|---|
| It includes the request parameter in a request header in a packet. | It includes the request parameters in a request body in a packet. |
|   |   |
| In this approach request parameter are displayed in address bar. | In this approach request parameter are not displayed. |
| Here size of the data is limited | Here size of the data is not limited. |
| It is not suitable for uploading files | It is suitable for uploading files also. |
| It is little bit fast as compared to POST method. | It is little bit slow as compared to GET method. |
| Use this method if the data is not confidential | Use this method if the data is confidential |

To handle http methods , jakarta.servlet.http.HttpServlet introduced.




There are three ways to write servlet programs:

1) By implementing jakarta.servlet.Servlet interface.
2) By extending jakarta.servlet.GenericServlet class.
3) By extending jakarta.servlet.http.HttpServlet class

**doGet() method:**

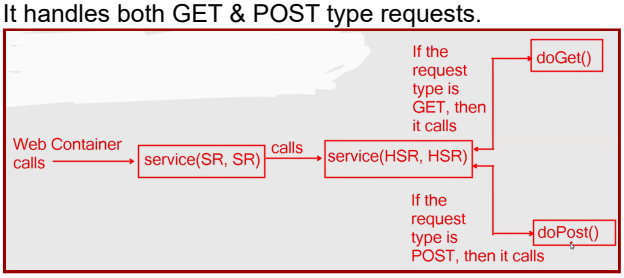
It handles GET type requests only.

**doPost() method:**

It handles POST type requests only.

**service() method:**

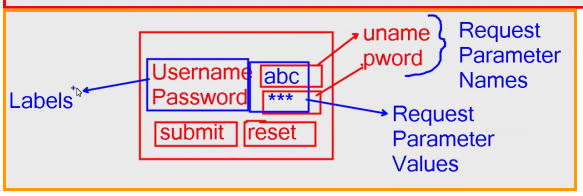
It handles both GET & POST type requests.




⇒ To handle only GET type requests override doGet() method.
⇒ To handle only POST type requests override doPost() method.
⇒ To handle both GET & POST requests and if the task is same then, override non life cycle service() method.
⇒ To handle both GET & POST requests and if the task is different then override both doGet() & doPost() methods:

jakarta.servlet.ServletRequest

Methods:
public abstract String getParameter(String);
=>It is used to get the request parameter value of specified request parameter name
public abstract Enumeration<String> getParameterNames();
=>It is used to get the request parameter names.
public abstract String[] getParameterValues(String);
=>It is used to get the request parameter values of specified request parameter name.
public abstract Map<String, String[]> getParameterMap();
=>It is used to get the request parameter names & values.




**Create table in oracle first to add values dynamically from html forms…**

```
create table uinfo (fname varchar2(12),lname varchar2(12),uname
varchar2(12),pword varchar2(12));
```

**RegServlet.java**

```
package login;

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

public class RegServlet extends HttpServlet {
    Connection con;

    public void init(ServletConfig config) throws ServletException {
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl",
"sam", "oracle");
        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }
    }

    public void destroy() {
        try {
            con.close();
        } catch (SQLException e) {
```

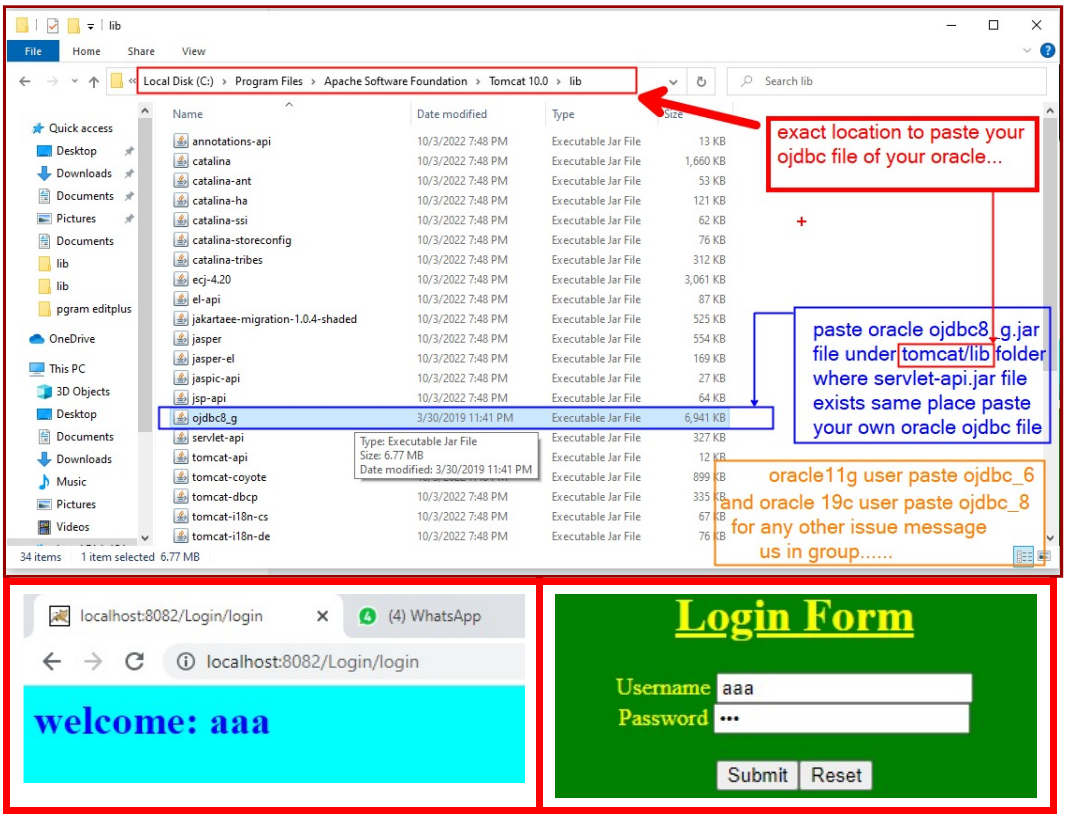
```
            e.printStackTrace();
        }
    }

    protected void doPost(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {
        try {
            String s1 = request.getParameter("fname");
            String s2 = request.getParameter("lname");
            String s3 = request.getParameter("uname");
            String s4 = request.getParameter("pword");
            PreparedStatement pstmt = con.prepareStatement("insert into
uinfo values(?,?,?,?)");
            pstmt.setString(1, s1);
            pstmt.setString(2, s2);
            pstmt.setString(3, s3);
            pstmt.setString(4, s4);
            pstmt.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        PrintWriter pw = response.getWriter();
        pw.println("<html><body bgcolor=green text=yellow>");
        pw.println("<h1>You Have Registered Successfully</h1>");
        pw.println("<a href=login.html>Login</a>");
        pw.println("</body></html>");
    }
}
```

**LoginServlet.java**

```
package login;

import java.io.*;
import java.sql.*;

import jakarta.servlet.*;
import jakarta.servlet.http.*;

public class LoginServlet extends HttpServlet {
    Connection con;

    public void init(ServletConfig config) throws ServletException {
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            this.con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl",
"sam", "oracle");
        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }
    }

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

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

```
        try {
            String s1 = request.getParameter("uname");
            String s2 = request.getParameter("pword");
            PreparedStatement pstmt = con.prepareStatement("select *
from uinfo where uname=? and pword=?");
            pstmt.setString(1, s1);
            pstmt.setString(2, s2);
            ResultSet rs = pstmt.executeQuery();
            PrintWriter pw = response.getWriter();
            pw.println("<html><body bgcolor=cyan text=blue><h1>");
            if (rs.next()) {
                pw.println("welcome : " + s1);
            } else {
                pw.println("invalid username or password");
            }
            pw.println("</h1></body></html>");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Before executing the above application copy ojdbc6_g.jar or ohdbc8_g.jar file into tomcat lib folder.




There object creation order by Web container:

1) ServletContext
2) User defined Servlet
3) ServletConfig
4) ServletRequest & ServletResponse

⇒ ServletContext is created by Web container whenever web application is deployed on server.
⇒ User defined servlet object is created by Web container for very first request only.
⇒ ServletConfig is created by Web container whenever init() method is called.
⇒ ServletRequest & ServletResponse created by Web container for every request.
⇒ ServletRequest & ServletResponse created by Web container whenever service() method is called.

User defined servlet object is created by Web container to call lifecycle methods.
ServletRequest is used to get the request parameters from html page.
ServletResponse is used to create byte stream, character stream & it is also used to set the MIME type.

**ServletConfig :**

It is used to get the initialisation parameters from web.xml

**initialisation parameters:**
initialisation parameters are specific to servlet.
initialisation parameters are used to initialise the servlet.
To configure initialisation parameters we use <init-param> , <param-name> & <param-value> tags in web.xml
**Example :**

```
<web-app>
<servlet>
<servlet-name>LoginServlet</servlet-name>
<servlet-class>login.LoginServlet</servlet-class>

<init-param>
<param-name>driver</param-name>
<param-value>oracle.jdbc.driver.OracleDriver</param-value>
</init-param>
<init-param>
<param-name>url</param-name>
<param-value>jdbc:oracle:thin:@localhost:1521:xe</param-value>
</init-param>
<init-param>
<param-name>username</param-name>
<param-value>system</param-value>
</init-param>
<init-param>
<param-name>password</param-name>
<param-value>manager</param-value>
</init-param>

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

To get the above initialisation parameters from web.xml , we use the following method of ServletConfig interface.

jakarta.servlet.ServletConfig

Methods:
value name
public abstract String getInitParameter(String);
=> It returns value of specified initialization parameter.
public abstract Enumeration<String> getInitParameterNames();
=> It returns all initialization parameter names.

**Example :**

```
    public void init(ServletConfig config) throws ServletException {
        try {
            String s1 = config.getInitParameter("driver");
            String s2 = config.getInitParameter("url");
            String s3 = config.getInitParameter("username");
            String s4 = config.getInitParameter("password");
            Class.forName(s1);
            this.con = DriverManager.getConnection(s2, s3, s4);
        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }
    }
```

**Context Parameters:**
Context Parameters are common to all servlets in a war file.
Context Parameters are used to initialise servlet.
To configure context parameters, we use <context-param>,<param-name> & <param-value> tags in web.xml

Web.xml file for applying context parameter for all servlet…

```
</welcome-file-list>
```

```
<context-param>
<param-name>driver</param-name>
<param-value>oracle.jdbc.driver.OracleDriver</param-value>
</context-param>
<context-param>
<param-name>url</param-name>
<param-value>jdbc:oracle:thin:@localhost:1521:orcl</param-value>
</context-param>
<context-param>
<param-name>username</param-name>
<param-value>sam</param-value>
</context-param>
<context-param>
<param-name>password</param-name>
<param-value>oracle</param-value>
</context-param>

  <servlet>
```

The above context parameters can be retrieved from web.xml by using getInitParameter() method of jakarta.servlet.ServletContext interface.

jakarta.servlet.ServletContext

context parameter value

context parameter name

Method:
public abstract String getInitParameter(String);
=> It is used to get the context parameter value of specified context parameter name

jakarta.servlet.ServletConfig

Method:
public abstract ServletContext getServletContext();
=> It is used to get the ServletContext reference

**Example :**

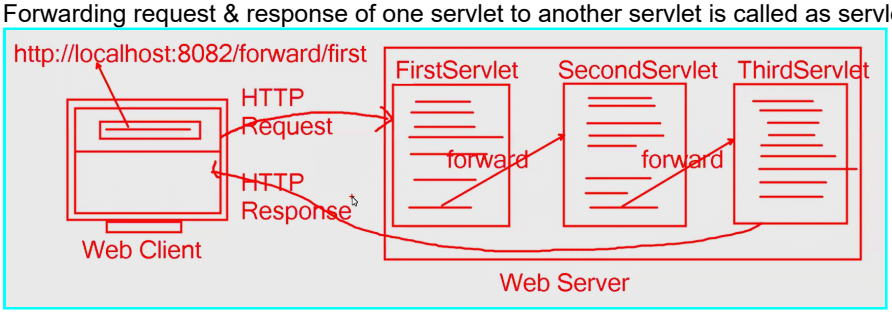
```
public void init(ServletConfig config) throws ServletException {
        try {
            ServletContext sc = config.getServletContext();
            String s1 = sc.getInitParameter("driver");
            String s2 = sc.getInitParameter("url");
            String s3 = sc.getInitParameter("username");
            String s4 = sc.getInitParameter("password");
            Class.forName(s1);
            con = DriverManager.getConnection(s2, s3, s4);
        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }
    }
```

| Initialization Parameters | Context Parameters |
|---|---|
| Initialization parameters are used to initialize servlet. | Initialization parameters are used to initialize servlet. |
| Initialization parameters are specific to servlet | Context parameters are common to all servlets in a war file. |
| To configure Initialization parameters, we use <init-param>, <param-name>,<param-value> tags in web.xml | To configure context parameters, we use <context-param>, <param-name>,<param-value> tags in web.xml |
| To retrieve initialization parameters from web.xml , we use getInitParameter() method of ServletConfig interface. | To retrieve context parameters from web.xml , we use getInitParameter() method of ServletContext interface. |

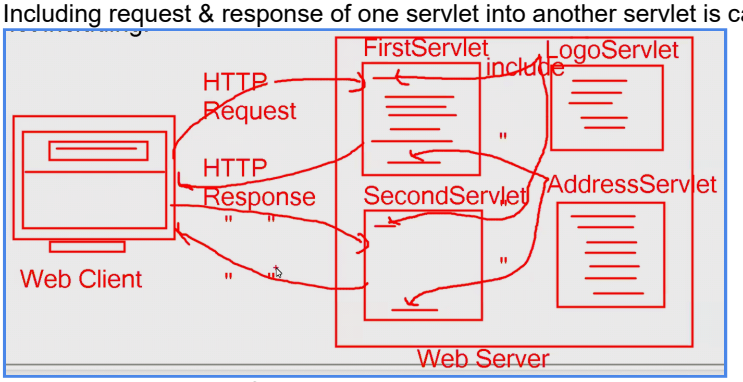
| ServletConfig | ServletContext |
|---|---|

| ServletConfig | ServletContext |
|---|---|
| It is an interface in jakarta.servlet package. | It is also an interface in jakarta.servlet package. |
| It is created by Web container whenever init() method is called. | It is created by Web container whenever web application is deployed on server. |
| Web container creates ServletConfig one per servlet | Web container creates ServletConfig one per web application (.war file) |
| It is used to retrieve initialization parameters from web.xml | It is used to retrieve context parameters from web.xml |

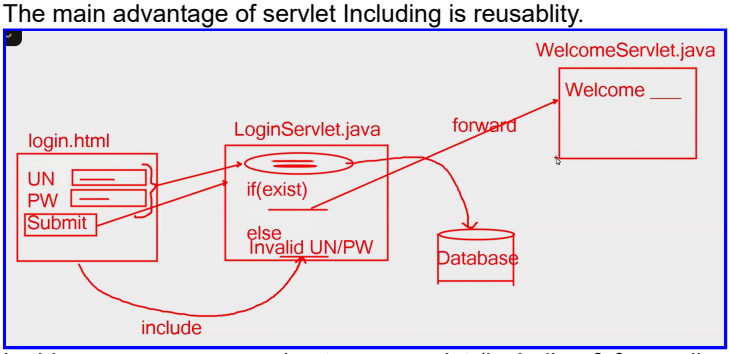
**Servlet Forwarding :**
Forwarding request & response of one servlet to another servlet is called as servlet forwarding.




The main advantage of servlet forwarding is modularity.
**Servlet Including :**
Including request & response of one servlet into another servlet is called as server including.




The main advantage of servlet Including is reusablity.




In this program we are going to use servlet (including & forwarding) methods…

```
jakarta.servlet.RequestDispatcher

Methods:
public abstract void forward(ServletRequest, ServletResponse)
                                        throws ServletException, IOException;
public abstract void include(ServletRequest, ServletResponse)
                                        throws ServletException, IOException;
```

```
jakarta.servlet.ServletRequest

Method:
  public abstract RequestDispatcher getRequestDispatcher(String);
=> It returns RequestDispatcher reference
```

<url-pattern> of Servlet/ HTML file name/|

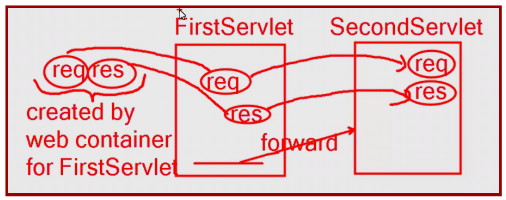
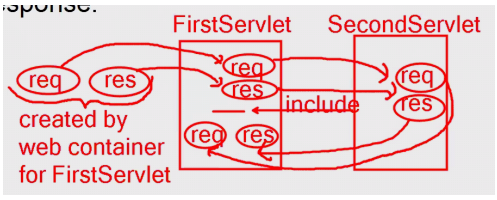
**LoginServlet.java file changes for including & forwarding servlet uses…**

```
if (rs.next()) {
                RequestDispatcher rd =
request.getRequestDispatcher("/welcome");
                rd.forward(request,response);
            } else {
```

```
                pw.println("<font color=yellow> Invalid
Username/Password </font>");
                RequestDispatcher rd =
request.getRequestDispatcher("/login.html");
                rd.include(request, response);
            }
```

**WelcomeServlet.java a new file**

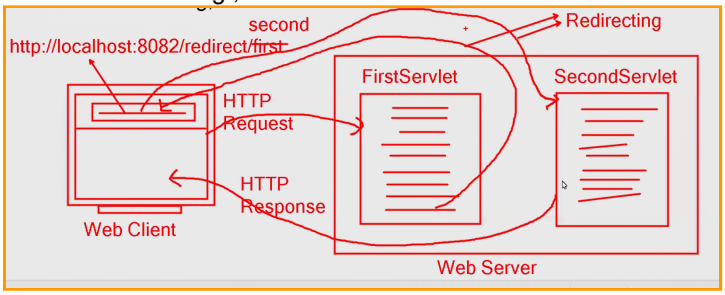
```
package login1;
import java.io.*;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.*;
public class WelcomeServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
        String s = request.getParameter("uname");
        PrintWriter pw = response.getWriter();
        pw.println("<html><body bgcolor=yellow text=red><center><h1>");
        pw.println("welcome: "+s);
        pw.println("</h1></center></body></html>");
    }
}
```

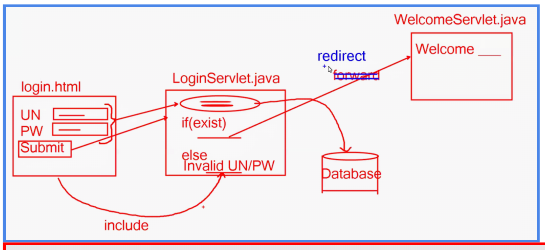
| Forwarding | Including |
|---|---|
| Forwarding request & response of one servlet to another servlet is called as servlet forwarding. | Including request & response of one servlet into another servlet is called as servlet including. |
| The main advantage of servlet forwarding is modularity. | The main advantage of servlet including is reusability. |
| In servlet forwarding web container creates only one pair of request & response. | In servlet including also web container creates only one pair of request & response. |
|  |  |
| In servlet forwarding <url-pattern> is not changed | In servlet including also <url-patter> is not changed. |
| Forward statement must be the last statement in a task code. | Include statements can be anywhere in a task code. |
| Forwarding supports to forward servlet to servlet , servlet to jsp & servlet to html. | Including supports to include servlet in a servlet, jsp in a servlet & html in a servlet. |
| Forwarding works within the server only. | Including also works within the server only. |

**Servlet Redirecting :**

Passing control from one servlet to another servlet is called as servlet redirecting.

In servlet redirecting , web container instructs the browser to execute next url.






jakarta.servlet.http.HttpServletResponse

Method:
public abstract void sendRedirect(String) throws IOException;
=> It is used to redirect the control from one servlet to another servlet.

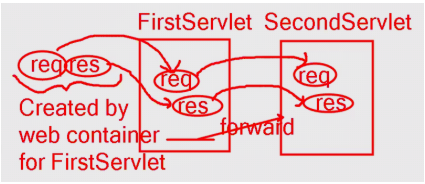
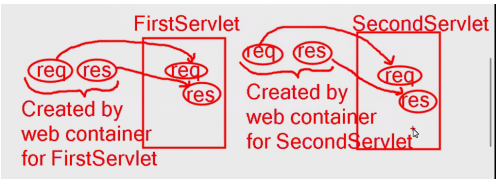
<url-pattern> of servlet

**LoginServlet.java file changes required for doing redirect program…**

```
if (rs.next()) {
                response.sendRedirect("welcome");
            } else {
                pw.println("<font color=yellow> Invalid
Username/Password </font>");
                RequestDispatcher rd =
request.getRequestDispatcher("/login.html");
                rd.include(request, response);
            }
```

**WelcomeServlet.java file changes required to run redirect method.**

```
protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException
{...same as above...}
```

| Forwarding | Redirecting |
|---|---|
| Forwarding one servlet to another servlet is called as servlet forwarding. | Passing control from one servlet to another servlet is called as servlet redirecting. |
| In servlet forwarding, server implicitly passes the request & response from one servlet to another servlet. | In servlet redirecting, server instructs the browser to execute next url. |
| In servlet forwarding both data & control passed to next servlet. | In servlet redirecting only control passed to next servlet. |
| In servlet forwarding only one pair of request & response created by Web container. | In servlet redirecting separate pair of request & response created by Web container for every servlets. |
|  |  |
| In servlet forwarding <url-pattern> is not changed. | In servlet redirecting <url-pattern> is changed. |
| Forwarding supports to forward servlet to servlet , html & JSP. | Redirecting supports to redirect servlet to servlet, html , JSP, asp , asp.net, php, …etc. |
| Forwarding works with in the server only. | Redirecting works with in the server & between two different servers also. |





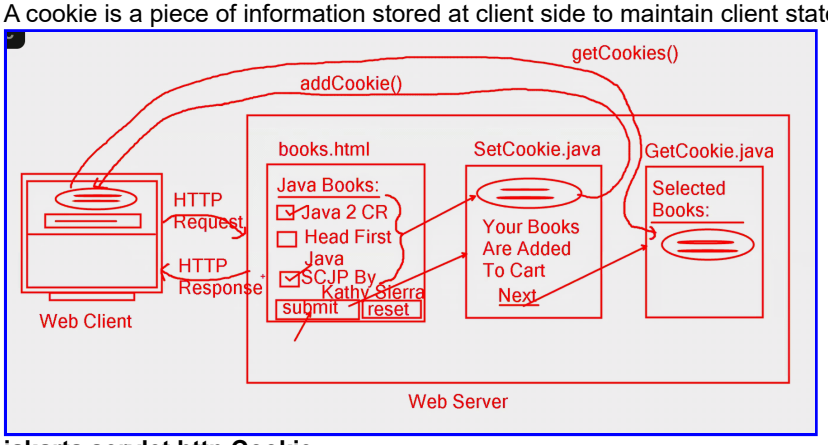
**Session Tracking :**
A session is a time period between login & logout.
A session Tracking is a mechanism that servlets use to maintain client state information about a series of requests from the same user across some time period.
Client state information can be username, password, examination I'd, account number, shopping items … etc.
Session Tracking mechanism is important in servlets to recognise client because http protocol is a stateless protocol and it does not maintain state information.
**There are four session Tracking methods:**

1) Cookies

2) Url Rewriting
3) Http session
4) Hidden form fields.

**Cookies :**

A cookie is a piece of information stored at client side to maintain client state information.




**jakarta.servlet.http.Cookie**

name value

Constructor:
public Cookie(String, String);
=> It is used to create a cookie.

Methods:
public void setMaxAge(int);
=> It is used to set the time interval
public int getMaxAge();
=> It is used to get the time interval
public String getName();
=> It returns name of the cookie
public void setValue(String);
=> It is used to change the value of cookie
public String getValue();
=> It is used to get the value of cookie

**Note :** By default cookies are vanished whenever browser window is closed.

jakarta.servlet.http.HttpServletRequest

Method:
public abstract Cookie[] getCookies();
=>It is used to retrieve all cookies from client.

jakarta.servlet.http.HttpServletResponse

Method:
public abstract void addCookie(Cookie);
=> It is used to add a cookie to client system.

**books.html file create for html page Book Project**

```
<!DOCTYPE html>
<html>
<head>
<title></title>
</head>
<body bgcolor=cyan text=blue>
<h1><u>Java Books:</u></h1>
<form action=set>
<input type=checkbox name=book1 value=Java2CompleteReference>Java2CompleteReference<br>
<input type=checkbox name=book2 value=HeadFirstJava>Head First Java<br>
<input type=checkbox name=book3 value=SCJPByKathySierra>SCJP By Kathy Sierra<br><br>
<input type=submit><input type=reset>
</form>
</body>
</html>
```

**SetCookie.java file create for java servlet Book Project**

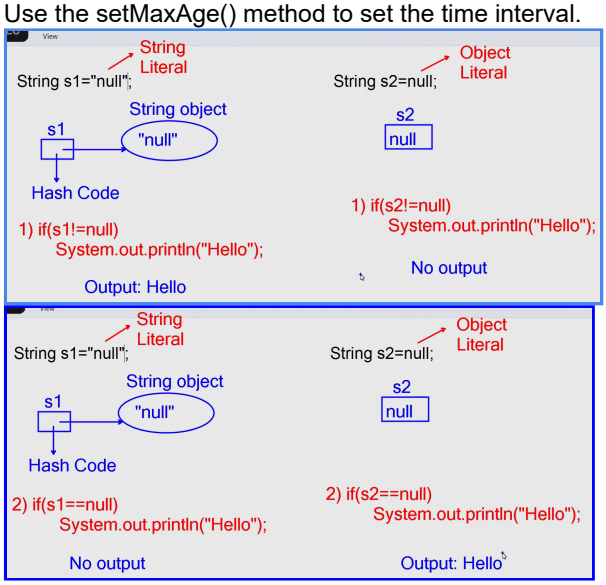
```
package cookie;

import java.io.*;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.*;
public class SetCookie extends HttpServlet {
    protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
```

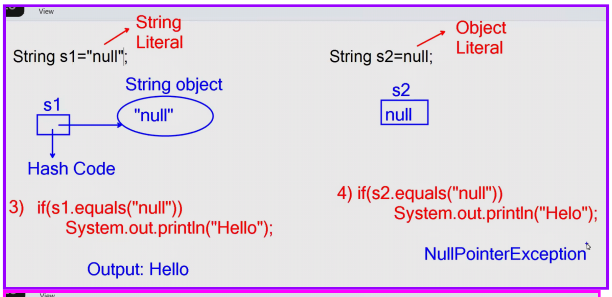
```
        String s1= request.getParameter("book1");
        String s2= request.getParameter("book2");
        String s3= request.getParameter("book3");
        if(s1!=null) {
            Cookie c1= new Cookie("book1",s1);
            response.addCookie(c1);
        }
        if(s2!=null) {
            Cookie c2= new Cookie("book2",s2);
            response.addCookie(c2);
        }
        if(s3!=null) {
            Cookie c3= new Cookie("book3",s3);
            response.addCookie(c3);
        }
        PrintWriter pw = response.getWriter();
        pw.println("<html><body bgcolor=yellow text=blue><center>");
        pw.println("<h1>Your Books are Added to Cart</h1>");
        pw.println("<a href=get>Next</a>");
        pw.println("</center></body></html>");
    }
}
```

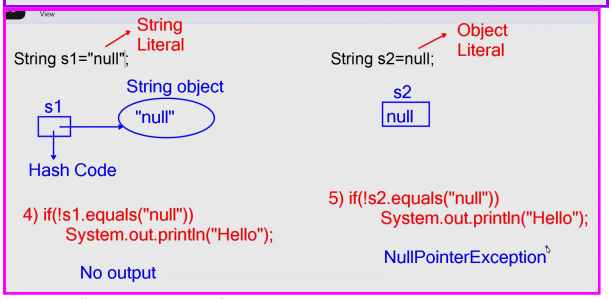
**GetCookie.java file create for java servlet Book Project**

```
package cookie;
import java.io.*;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.*;
public class GetCookie extends HttpServlet {
    protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
        PrintWriter pw = response.getWriter();
        pw.println("<html><body bgcolor=green text=yellow><center>");
        pw.println("<h1><u>Your Selected Books: </u></h1>");
        Cookie[] c1=request.getCookies();
        for (Cookie c2 : c1) {
            pw.println(c2.getValue()+"<br>");
        }
        pw.println("</center></body></html>");
    }
}
```

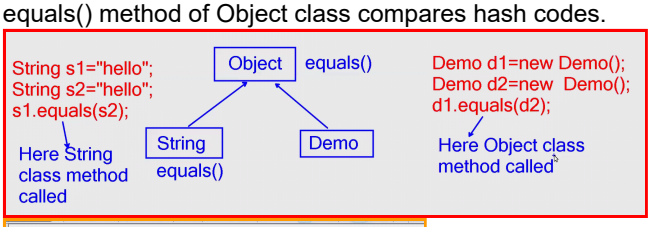
**Note:** In the above example cookies vanish whenever the browser window is closed. Use the setMaxAge() method to set the time interval.



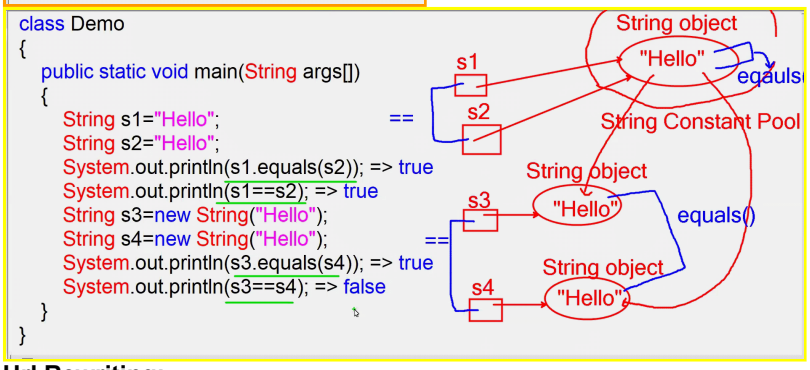



equals() method is String class compares the contents of String objects where as == operator compares hash codes.
equals() method of Object class compares hash codes.




```
class Demo
{
   public static void main(String args[])
   {
      String s1="Hello";
      String s2="Hello";
      System.out.println(s1.equals(s2)); => true
      System.out.println(s1==s2); => true
      String s3=new String("Hello");
      String s4=new String("Hello");
      System.out.println(s3.equals(s4)); => true
      System.out.println(s3==s4); => false
   }
}
```




**Url Rewriting:**
In this session tracking method , client state information(data) appended to URL.

**books.html file create for html page Book URL Project**

```
<!DOCTYPE html>
<html>
<head>
<title></title>
</head>
<body bgcolor=cyan text=blue>
<h1><u>Java Books:</u></h1>
<form action=set>
<input type=checkbox name=book1 value=Java2CompleteReference>Java2CompleteReference<br>
<input type=checkbox name=book2 value=HeadFirstJava>Head First Java<br>
<input type=checkbox name=book3 value=SCJPByKathySierra>SCJP By Kathy Sierra<br><br>
<input type=submit><input type=reset>
```

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

**SetUrl.java file create for java servlet Book URL Project**

```
package url;

import java.io.*;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.*;
public class SetUrl extends HttpServlet {
    protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
        String s1= request.getParameter("book1");
        String s2= request.getParameter("book2");
        String s3= request.getParameter("book3");
        PrintWriter pw = response.getWriter();
        pw.println("<html><body bgcolor=yellow text=blue><center>");
        pw.println("<h1>Your Books are Added to Cart</h1>");
        pw.println("<a href=get?
book1="+s1+"&book2="+s2+"&book3="+s3+">Next</a>");
        pw.println("</center></body></html>");
    }
}
```

**GetUrl.java file create for java servlet Book URL Project**

```
package url;

import java.io.*;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.*;
public class GetUrl extends HttpServlet {
    protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
        String s1= request.getParameter("book1");
        String s2= request.getParameter("book2");
        String s3= request.getParameter("book3");
        PrintWriter pw = response.getWriter();
        pw.println("<html><body bgcolor=green text=yellow><center>");
        pw.println("<h1><u>Your Selected Books: </u></h1>");
        if(!s1.equals("null"))
            pw.println(s1+"<br>");
        if(!s2.equals("null"))
            pw.println(s2+"<br>");
        if(!s3.equals("null"))
            pw.println(s3+"<br>");
        pw.println("</center></body></html>");
    }
}
```

| Cookies | Url Rewriting |
|---|---|
| In this session tracking method client state information stored in browser memory. | In this session tracking method client state information appended to URL. |
| here client state information not displayed. | Here client state information displayed in address bar. |
| It allows String type data only | It is allows String type data only. |
| Here size of the data is limited. | Here also size of the data is limited. |
| Cookies are not secure because cookies can be viewed by user through browser settings option. | Url Rewriting not secure because here client state information displayed in address bar. |

| | |
|---|---|
| Here it is possible to set the time interval by using setMaxAge() method of Cookie class. | Here it is not possible to set the time interval. |
| This session tracking method fails if the cookies are disabled. | This session tracking method always works. |

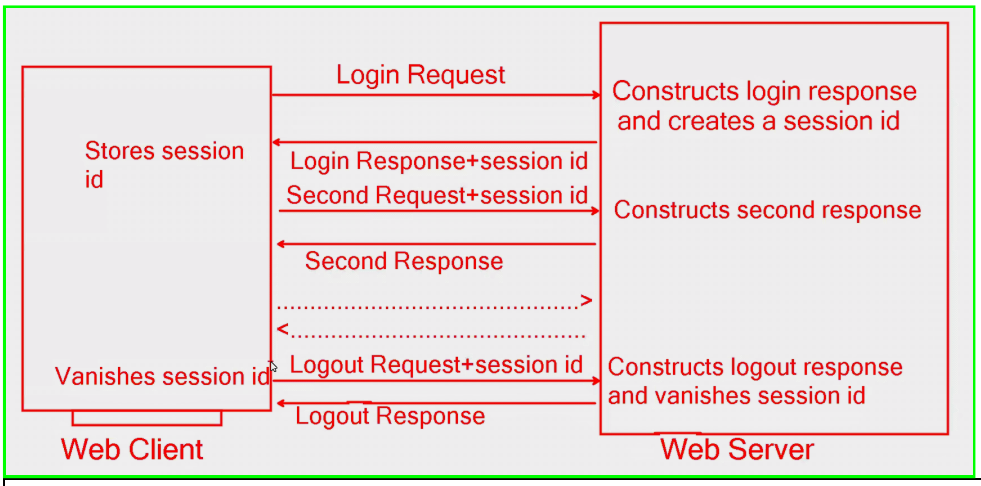
**Http sessions:**
In this session tracking method client state information stored at server side.
In this session tracking method session id created by Web container & passed to client system to identify the client.
To maintain session id there are two ways.

1) By using cookies.
2) By using url Rewriting.

This session tracking method can be implemented in two ways:

1) Http session with cookies
2) Http session with URL Rewriting

**Http session with cookies:**
In this session tracking method client state information stored at server side.
Here session id created by Web container, passed to client system & stored in cookie variable at client side.




```
jakarta.servlet.http.HttpSession
 Methods:
  public abstract long getCreationTime();
  public abstract String getId();
  public abstract long getLastAccessedTime();
  public abstract void setMaxInactiveInterval(int);
==> it is used to set the time interval in seconds.
  public abstract int getMaxInactiveInterval();
  public abstract Object getAttribute(String);
==> it is used to get the session variable value.
  public abstract void setAttribute(String, Object);
==> it is used to create session variable with name & value pair.
  public abstract void removeAttribute(String);
==> it is used to remove session variable
  public abstract void invalidate();
==> it is used to vanish the session id
```

**Note : By default sessions vanish after 30 minutes in a tomcat server.**

jakarta.servlet.http.HttpServletRequest

Method:
public abstract HttpSession getSession();
=> It is used to get the reference of HttpSession
(It creates a session id & returns)

**DemoServlet.java file for session1 project**

```
package session1;

import java.io.*;
import java.util.Date;
```

```
import jakarta.servlet.ServletException;
import jakarta.servlet.http.*;
public class DemoServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
        // TODO Auto-generated method stub
        PrintWriter pw = response.getWriter();
        HttpSession hs = request.getSession();
        pw.println("<html><body bgcolor=green text=white><h1>");
        pw.println("Session Id : "+hs.getId()+"<br>");
        pw.println("Creation Time: "+new Date(hs.getCreationTime())+"
<br>");
        pw.println("Last Access time:
"+new Date(hs.getLastAccessedTime())+"<br>");
        pw.println("Time Interval: "+hs.getMaxInactiveInterval()+"
Seconds <br>");
        pw.println("</h1></body></html>");
    }
}
```

**books.html file for session2 project**

```
<!DOCTYPE html>
<html>
<head>
<title></title>
</head>
<body bgcolor=cyan text=blue>
<h1><u>Java Books:</u></h1>
<form action=set>
<input type=checkbox name=book1 value=Java2CompleteReference>Java 2
Complete Reference<br>
<input type=checkbox name=book2 value=HeadFirstJava>Head First Java<br>
<input type=checkbox name=book3 value=SCJPByKathySierra>SCJP By Kathy
Sierra<br><br>
<input type=submit><input type=reset>
</form>
</body>
</html>
```

**SetSession.java file for session2 project**

```
package session2;

import java.io.*;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.*;
public class SetSession extends HttpServlet {
    protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
        String s1 = request.getParameter("book1");
        String s2 = request.getParameter("book2");
        String s3 = request.getParameter("book3");
        HttpSession hs = request.getSession();
        hs.setAttribute("book1", s1);
        hs.setAttribute("book2", s2);
        hs.setAttribute("book3", s3);
        PrintWriter pw = response.getWriter();
        pw.println("<html><body bgcolor=cyan text=blue><center>");
        pw.println("<h1>Your books are added to Cart: </h1>");
        pw.println("<a href=get>Next</a>");
        pw.println("</center></body></html>");
    }
}
```

**GetSession.java file for session2 project**

```
package session2;

import java.io.*;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.*;
public class GetSession extends HttpServlet {
    protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
        HttpSession hs = request.getSession();
        String s1= (String)hs.getAttribute("book1");
        String s2= (String)hs.getAttribute("book2");
        String s3= (String)hs.getAttribute("book3");
        PrintWriter pw = response.getWriter();
        pw.println("<html><body bgcolor=red text=yellow>");
        pw.println("<h1>Selected books: </h1>");
        if(s1!=null)
            pw.println(s1+"<br>");
        if(s2!=null)
            pw.println(s2+"<br>");
        if(s3!=null)
            pw.println(s3+"<br>");
        pw.println("</body></html>");
    }
}
```

**⇒ In the above example, session id created by web container, passed to client system and stored in cookie variable at client side.**

## Http Sessions with URL Rewriting:

In this session tracking method client state information is stored at server side.
In this session tracking method session id created by web container and appended to url.

**jakarta.servlet.http.HttpServletResponse**

Method:

```
    public abstract String encodeURL(String);
```

**⇒ it is used to append a session id to url.**

**SetSession.java same as above program only 2 lines change done.**

```
package session2;

import java.io.*;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.*;
public class SetSession extends HttpServlet {
    protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
        String s1 = request.getParameter("book1");
        String s2 = request.getParameter("book2");
        String s3 = request.getParameter("book3");
        HttpSession hs = request.getSession();
        hs.setAttribute("book1", s1);
        hs.setAttribute("book2", s2);
        hs.setAttribute("book3", s3);
        PrintWriter pw = response.getWriter();
        pw.println("<html><body bgcolor=cyan text=blue><center>");
        pw.println("<h1>Your books are added to Cart: </h1>");
        String s =response.encodeURL("get");
        pw.println("<a href="+s+">Next</a>");
        pw.println("</center></body></html>");
    }
}
```

| Http session with cookies | Http session with URL Rewriting |
|---|---|

| | |
|---|---|
| In this session tracking method, session id created by web container, passed to client system & stored in cookie variable at client side. | In this session tracking method, session id created by web container and appended to url. |
| This session tracking method fails if the cookies are disabled. | This session tracking method always works. |

| **Cookies** | **Sessions** |
|---|---|
| A cookies is a piece of information stored at client side to maintain client state information. | A session is a piece of information stored at server side to maintain client state information. |
| In this session tracking method, session id not created. | In this session tracking method , session id created by web container. |
| It supports string type data only. | It supports all types of data. |
| Here size of the data is limited | Here size of the data is not limited. |
| Cookies are not secure because stored at client side. | Sessions are secure because stored at server side. |
| By default cookies are vanished whenever browser window is closed | By default sessions are vanished after 30 minutes in a tomcat server |
| Here it is possible to set the time interval by using setMaxAge() method of Cookie class. | Here also it is possible to set the time interval by using setMaxInactiveInterval() method of HttpSession interface. |

**Hidden form fields:**
In this session tracking method client state information stored in hidden fields.
It is very easy to implement because it requires html code only.
It supports String type data only
Here also the size of the data is limited.
It is also not secure because hidden fields can be viewed by user through view page source code.
Here it is not possible to set the time interval.
Here always form submission is required.

**books.html file for hidden form fields in hff project**

```
<!DOCTYPE html>
<html>
<head>
<title></title>
</head>
<body bgcolor=cyan text=blue>
<h1><u>Java Books:</u></h1>
<form action=set>
<input type=checkbox name=book1 value=Java2CompleteReference>Java 2
Complete Reference<br>
<input type=checkbox name=book2 value=HeadFirstJava>Head First Java<br>
<input type=checkbox name=book3 value=SCJPByKathySierra>SCJP By Kathy
Sierra<br><br>
<input type=submit><input type=reset>
</form>
</body>
</html>
```

**SetFields.java file for hidden form field hff project**

```
package hff;

import java.io.*;
import jakarta.servlet.ServletException;
```

```
import jakarta.servlet.http.*;
public class SetFields extends HttpServlet {
   protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
      String s1= request.getParameter("book1");
      String s2= request.getParameter("book2");
      String s3= request.getParameter("book3");
      PrintWriter pw = response.getWriter();
      pw.println("<html><body bgcolor=yellow text=blue><center>");
      pw.println("<form action=get>");
      pw.println("<input type=hidden name=book1 value="+s1+">");
      pw.println("<input type=hidden name=book2 value="+s2+">");
      pw.println("<input type=hidden name=book3 value="+s3+">");
      pw.println("<h1>Your books are added to cart: </h1>");
      pw.println("<input type=submit value=next>");
      pw.println("</form></center></body></html>");
   }
}
```

**GetFields.java file for hidden form fields hff project**

```
package hff;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.*;
import java.io.*;
public class GetFields extends HttpServlet {
   protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
      String s1= request.getParameter("book1");
      String s2= request.getParameter("book2");
      String s3= request.getParameter("book3");
      PrintWriter pw = response.getWriter();
      pw.println("<html><body bgcolor=red text=white><center>");
      pw.println("<h1><u>Your Selected Books: </u></h1>");
      if(!s1.equals("null"))
         pw.println(s1+"<br>");
      if(!s2.equals("null"))
         pw.println(s2+"<br>");
      if(!s3.equals("null"))
         pw.println(s3+"<br>");
      pw.println("</center></body></html>");
   }
}
```

**jakarta.servlet.http.HttpServletRequest**

```
Methods:
public abstract HttpSession getSession(boolean);
public abstract HttpSession getSession();
```

```
HttpSession hs1 = request.getSession(true);
```

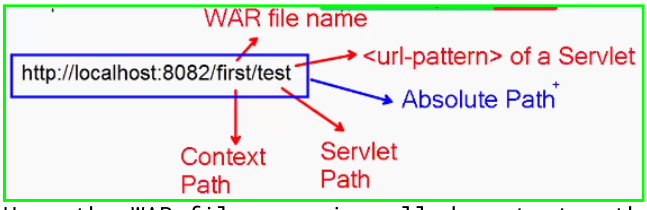
**or**

```
HttpSession hs2 = request.getSession();
```

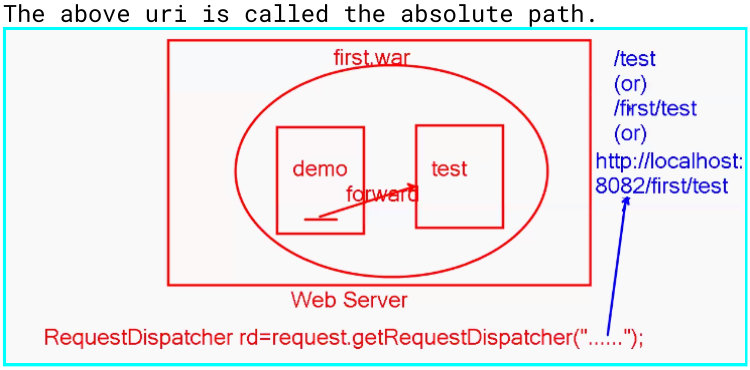
**==> The above true statements returns current session if the session already exist otherwise creates a new session & returns.**

```
HttpSession hs3 = request.getSession(false);
```

**==> The above statements returns current session if the session already exist otherwise it returns null**.




Here the WAR file name is called context path and <url-pattern> of a servlet is called servlet path.
The above uri is called the absolute path.




Here /test is a relative path to a context path.
Here /first/test is a relative path to a server path.
Here http://localhost:8082/first/test is a absolute path.

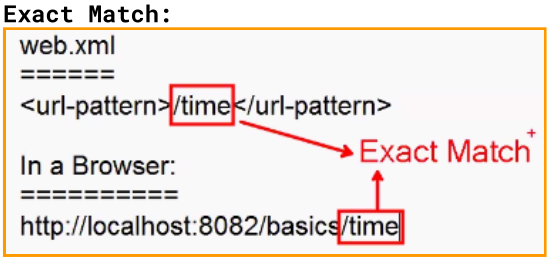
**web.xml:**

It is called as configuration file and it is used to configure servlets, listeners, filters, JSPs , initialization parameters, context parameters , welcome files, etc…

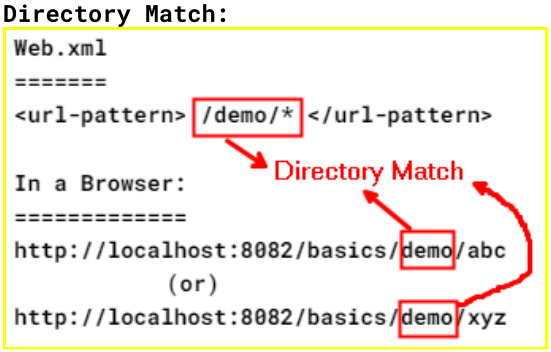
> there are 3 ways to configure servlet:

1. Exact Match
2. Directory Match
3. Extension Match

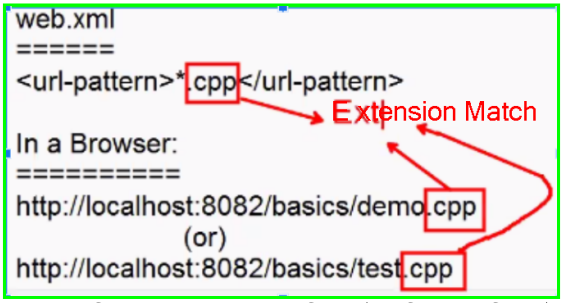
**Exact Match:**




**Directory Match:**




**Extension Match:**




**To configure welcome file(default file) in web.xml:**

```
<web-app>
<welcome-file-list>
<welcome-file>homepage.html</welcome-file>
</welcome-file-list>
=================
```

```
==================
</web-app>|
```

**To configure <load-on-startup> tag in web.xml :**

```
<web-app>
 <servlet>
<servlet-name>login</servlet-name>
<servlet-class>login.LoginServlet</servlet-class>
<load-on-startup>1</load-on-startup>
 </servlet>
 =============
 =============
 </web-app>
```

It indicates a web container to create an object of LoginServlet and call init() method whenever a web application is deployed on the server…

**Note: Here servlets are loaded in ascending order.**

**Note: negative numbers & zero are not considered.**

# Servlet Listeners

## Servlet Listeners:

⇒Servlet listeners are interfaces that contain event handlers(methods) to handle events (classes) which are generated by a web container.

### List of servlet listeners:

1) jakarta.servlet.ServletContextListener
2) jakarta.servlet.ServletContextAttributeListener
3) jakarta.servlet.http.HttpSessionListener
4) jakarta.servlet.http.HttpSessionAttributeListener ,,,,etc.

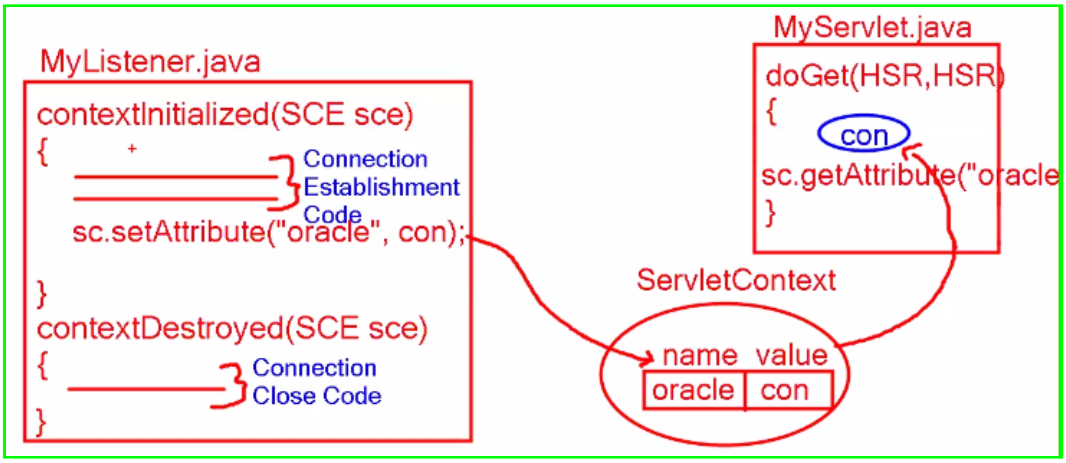
```
jakarta.servlet.ServletContextListener
Methods:
 public default void contextInitialized(ServletContextEvent);
 public default void contextDestroyed(ServletContextEvent);
```

The above event handlers (method) handle ServletContextEvent.

**ServletContextEvent** is generated by a web container whenever ServletContext is created & destroyed…

**ServletContext is created** by a web container whenever a web application is deployed on a server.

**ServletContext is destroyed** by a web container whenever a web application is undeployed from the server.

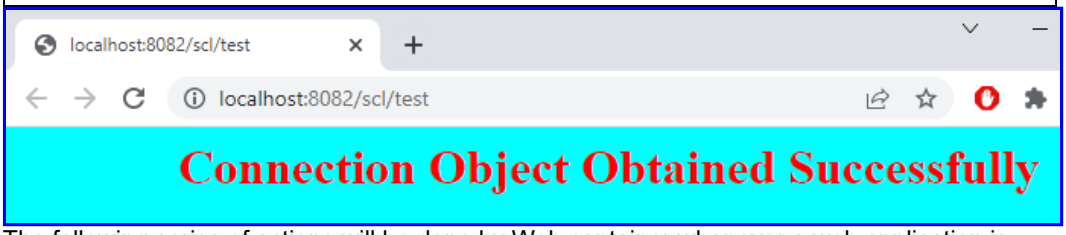



**MyListener.java in scl package (servlet context listener) for servlet listener 1st program**

```
package scl;

import java.sql.*;
import jakarta.servlet.*;
public class MyListener implements ServletContextListener {
    Connection con;
    public void contextInitialized(ServletContextEvent sce) {
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl",
"sam", "oracle");
            ServletContext sc = sce.getServletContext();
            sc.setAttribute("oracle", con);
        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }
    }
    public void contextDestroyed(ServletContextEvent sce) {
        try {
            con.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

**MyServlet.java in scl package (servlet context listener) for servlet listener 1st program**

```
package scl;

import java.io.*;
import java.sql.Connection;
import jakarta.servlet.*;
import jakarta.servlet.http.*;
public class MyServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
        ServletContext sc = request.getServletContext();
        Connection con =(Connection)sc.getAttribute("oracle");
        PrintWriter pw = response.getWriter();
        pw.println("<html><body bgcolor=cyan text=red><center> <h1>");
        pw.println("Connection Object Obtained Successfully");
        pw.println("</h1><center></body></html>");
    }
}
```




The following series of actions will be done by Web container whenever a web application is deployed on server.

1) ServletContext is created.
2) ServletContectEvent is generated.
3) ServletContectEvent is passed to contextInitialized() method.
4) contextInitialized() method body is executed (Connection Established & Connection object placed in ServletContext memory).

The following series of actions will be done by Web container whenever a web application is undeployed from server.

1) ServletContext is destroyed.
2) ServletContectEvent is generated.
3) ServletContectEvent is passed to contextDestroyed() method.
4) contextDestroyed() method body executed (Connection Closed).

## ServletContextAttributeListener:

It contains the following event handlers(method) to handle ServletContextAttributeEvent which is generated by a web container.

```
jakarta.servlet.ServletContextAttributeListener
 Methods:
  public default void attributeAdded(ServletContextAttributeEvent);
  public default void attributeRemoved(ServletContextAttributeEvent);
  public default void attributeReplaced(ServletContextAttributeEvent);
```

### ServletContextAttributeEvent:

It is generated by a web container whenever an attribute is added to ServletContext, attribute is removed from ServletContext & attribute is replaced in a ServletContext.

## HttpSessionListener:

It contains the following event handlers(methods) to handle HttpSessionEvent, which is generated by a web container.

```
  public default void sessionCreated(HttpSessionEvent);
  public default void sessionDestroyed(HttpSessionEvent);
```

### HttpSessionEvent:

it is generated by a web container whenever a session is created & destroyed.

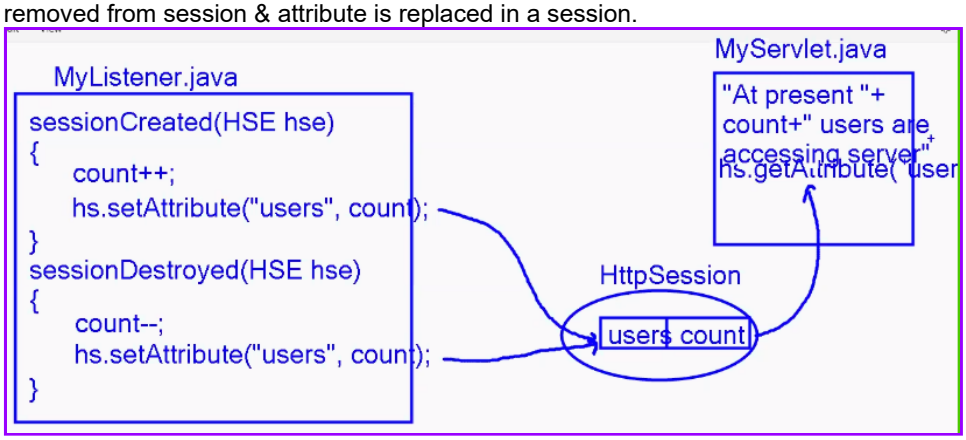
## HttpSessionAttributeListener:

It contains the following event handlers(method) to handle HttpSessionBindingEvent which is generated by a web container.

```
jakarta.servlet.ServletContextAttributeListener
 Methods:
  public default void attributeAdded(HttpSessionBindingEvent);
  public default void attributeRemoved(HttpSessionBindingEvent);
  public default void attributeReplaced(HttpSessionBindingEvent);
```

### HttpSessionBindingEvent:

It is generated by a web container whenever an attribute is added to session, attribute is removed from session & attribute is replaced in a session.




**MyListener.java in hsl package (Http Session Listener)**

```
package hsl;

import jakarta.servlet.http.*;
public class MyListener implements HttpSessionListener {
    int count = 0;
    public void sessionCreated(HttpSessionEvent hse) {
        count++;
        HttpSession hs = hse.getSession();
        hs.setAttribute("users", count);
    }
    public void sessionDestroyed(HttpSessionEvent hse) {
        count--;
        HttpSession hs = hse.getSession();
        hs.setAttribute("users", count);
    }
}
```

**MyServlet.java in hsl package (Http Session Listener)**

```
package hsl;

import java.io.*;
import java.sql.Connection;
import jakarta.servlet.*;
import jakarta.servlet.http.*;
```

```
public class MyServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {
        HttpSession hs = request.getSession();
        hs.setMaxInactiveInterval(30); //30sec
        int count =(Integer)hs.getAttribute("users");
        PrintWriter pw = response.getWriter();
        pw.println("<html><body bgcolor=cyan text=red><center> <h1>");
        pw.println("At present "+count+" users are accessing server");
        pw.println("</h1><center></body></html>");
    }
}
```

In the above example, a session is created by a web container whenever the getSession() method of HttpServletRequest is called and the session is destroyed after a time interval.(1 minute).

The following series of actions done by web container whenever session is created:

1) HttpSessionEvent is generated.
2) HttpSessionEvent is passed to sessionCreated() method.
3) sessionCreated() method is called.
4) sessionCreated() method body is executed.

The following series of action done by web container whenever session is destroyed:

1) HttpSessionEvent is generated.
2) HttpSessionEvent is passed to sessionDestroyed() method.
3) sessionDestroyed() method is called.
4) sessionDestroyed() method body is executed.

To configure listener class in web.xml :

```
 <web-app>
 <listener>
<listener-class>hsl.MyListener</listener-class>
 </listener>
====================
====================
 </web-app>
```

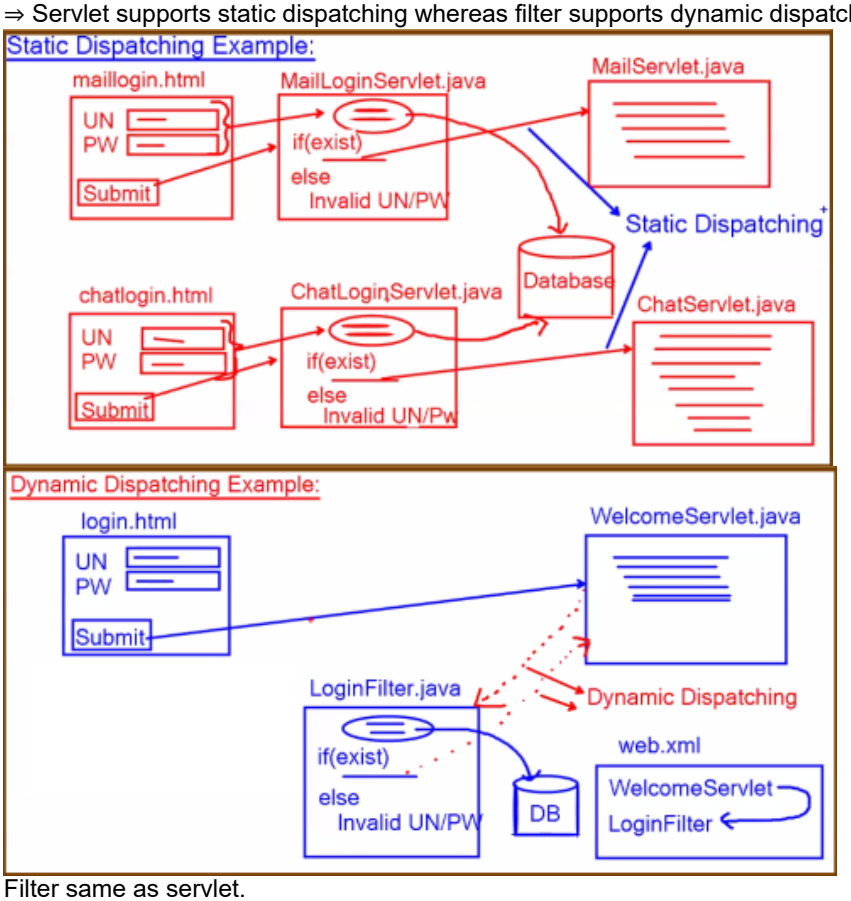
Instead of web.xml , we can use annotation as follows:

```
@WebListener
public class MyListener implements HttpSessionListener{
    //codes here
}
```

# Filters

# FILTERS

⇒ A filter is an object that can be declaratively inserted in a container process.
⇒ Filter provides the only mechanism by which we can plugin code between request & response.
⇒ Servlet supports static dispatching whereas filter supports dynamic dispatching.




Filter same as servlet.
Filters do not have a main() method because the filter runs on the server.
Filter contains life cycle methods to run on the server.

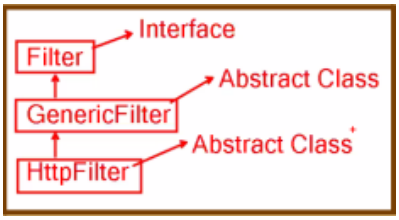
## Life cycle methods of a Filter:

1. init()
2. doFilter()
3. destroy()

init() method is called by the web container whenever the first request comes to a filter.
doFilter() method is called by the web container for every request.
destroy() method is called by the web container whenever the filter instance is removed from the web Page 1 of 2 container.
Filter instance is removed from web application, is undeployed or server shuts down.


The above life cycle methods are the part of jakarta.servlet.Filter interface.

```
jakarta.servlet.Filter
Methods:
public default void init(FilterConfig) throws ServletException;
public abstract void doFilter(ServletRequest, ServletResponse,
FilterChain) throws IOException ,ServletException;
public default void destroy();
```




```
jakarta.servlet.http.HttpFilter
Methods:
public void doFilter(ServletRequest, ServletResponse, FilterChain)
throws IOException ,ServletException;
protected void doFilter(HttpServletRequest, HttpServletResponse,
FilterChain) throws IOException ,ServletException;
```

Every filter must implement jakarta.servlet.Filter interface to derive life cycle methods.
Every filter class must be public because it should be accessible to web containers to create an object to call life cycle methods.
Use HttpFilter to utilize http specific services & common services.

**login.html file for filter project**

```
<!DOCTYPE html>
<html>
<head>
<title>Login Form</title>
</head>
<body bgcolor=green text=yellow>
<center>
<h1><u>Login Form</u></h1>
<form action=welcome method=POST>
Username <input type=text name=uname><br>
Password <input type=password name=pword><br><br>
<input type=submit><input type=reset>
</form>
</center>
</body>
</html>
```

**WelcomeServlet.java file for filter project.**

```
package filter;

import java.io.*;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.*;
public class WelcomeServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
        PrintWriter pw = response.getWriter();
        pw.println("<html><body bgcolor=yellow text=red><center>");
        pw.println("<h1>welcome...</h1>");
        pw.println("</center><body><html>");
    }
}
```

**LoginServlet.java file for filter project.**

```
package filter;

import java.io.*;
import java.sql.*;
import jakarta.servlet.*;
import jakarta.servlet.http.HttpFilter;
public class LoginFilter extends HttpFilter implements Filter {
    Connection con;
```

```
    public void destroy() {
        try {
            con.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public void doFilter(ServletRequest request,
ServletResponse response, FilterChain chain) {
        try {
            String s1 = request.getParameter("uname");
            String s2 = request.getParameter("pword");
            PreparedStatement pstmt = con.prepareStatement("select *
from uinfo where uname=? and pword=?");
            pstmt.setString(1, s1);
            pstmt.setString(2, s2);
            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) {
                chain.doFilter(request, response);
            } else {
                PrintWriter pw = response.getWriter();
                pw.println("<html><body bgcolor=red text=yellow><h1>");
                pw.println("invalid username/password");
                pw.println("</h1><body><html>");
            }
        } catch (SQLException | IOException | ServletException e) {
            e.printStackTrace();
        }

    }
    public void init(FilterConfig fConfig) throws ServletException {
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl",
"sam", "oracle");
        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }
    }
}
```

**web.xml file for filter project…**

**To configure filter in web.xml:**

```
 <web-app>
<servlet>
    <description></description>
    <display-name>WelcomeServlet</display-name>
    <servlet-name>WelcomeServlet</servlet-name>
    <servlet-class>filter.WelcomeServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>WelcomeServlet</servlet-name>
    <url-pattern>/welcome</url-pattern>
  </servlet-mapping>
  <filter>
    <display-name>LoginFilter</display-name>
    <filter-name>LoginFilter</filter-name>
    <filter-class>filter.LoginFilter</filter-class>
  </filter>
  <filter-mapping>
    <filter-name>LoginFilter</filter-name>
    <url-pattern>/welcome</url-pattern>
```

```
  </filter-mapping>
 </web-app>
```

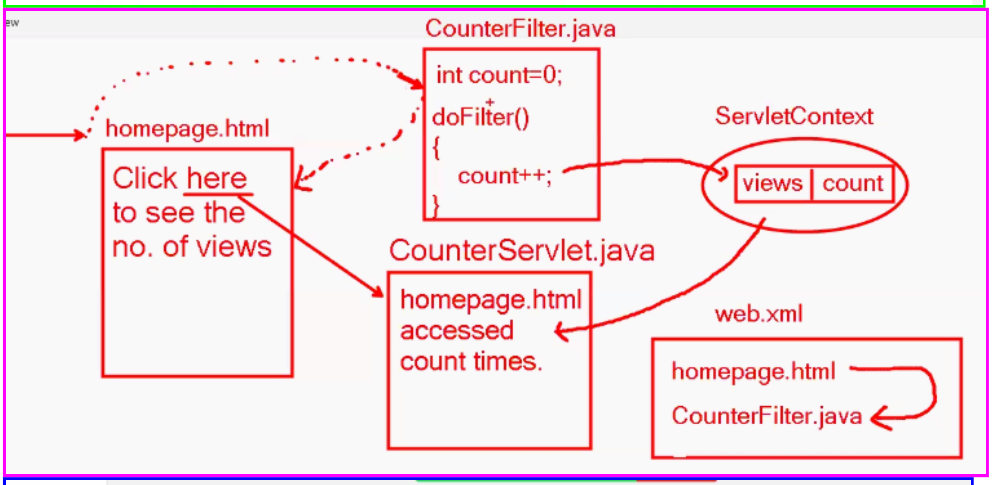
```
<filter-mapping>
  <filter-name>LoginFilter</filter-name>
  <url-pattern>/welcome</url-pattern>
 </filter-mapping>
</web-app>
```

It indicates web container to execute LoginFilter whenever request comes to WelcomeServlet

```
pstmt.setString(1, s1);
pstmt.setString(2, s2);
ResultSet rs=pstmt.executeQuery();
if(rs.next())
{
// pass the request along the filter chain
     chain.doFilter(request, response);
}
else
{
```

It indicates web container to check web.xml if any other filter configured or not?

If configured then control passed to next filter otherwise control passed to servlet.




```
1 <html>
2 <body bgcolor=green text=yellow>
3 <h1>Click <a href=count>here</a> to see the no. of views</h1>
4 </body>
5 </html>
```

<url-pattern> of CounterServlet

**homepage.html file for count project to count html views**

```
<html>
  <head>
    <title>Homepage</title>
  </head>
  <body bgcolor=green text=yellow>
    <h1>Click <a href=count>here</a> to see the number of views</h1>
  </body>
</html>
```

**CounterFilter.java file for count project for count homepage**

```
package count;

import java.io.IOException;
import jakarta.servlet.*;
import jakarta.servlet.http.*;
public class CounterFilter extends HttpFilter implements Filter {
    int count=0;
    public void doFilter(HttpServletRequest request,
HttpServletResponse response, FilterChain chain) throws IOException,
ServletException {
        count++;
```

```
        ServletContext sc = request.getServletContext();
        sc.setAttribute("views", count);
        chain.doFilter(request, response);
    }
}
```

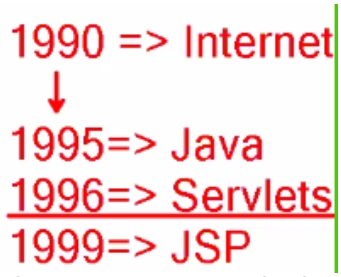
**CounterServlet.java file for count project for count homepage**

```
package count;

import java.io.*;
import jakarta.servlet.*;
import jakarta.servlet.http.*;
public class CounterServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
HttpServletResponse response) throws IOException, ServletException {
        ServletContext sc = request.getServletContext();
        int count =(Integer)sc.getAttribute("views");
        PrintWriter pw = response.getWriter();
        pw.println("<html><body bgcolor=yellow text=red><h1>");
        pw.println("The homepage has been accessed " + count + "
times.");
        pw.println("</h1></body></html>");
    }
}
```

# JSP

# JSP

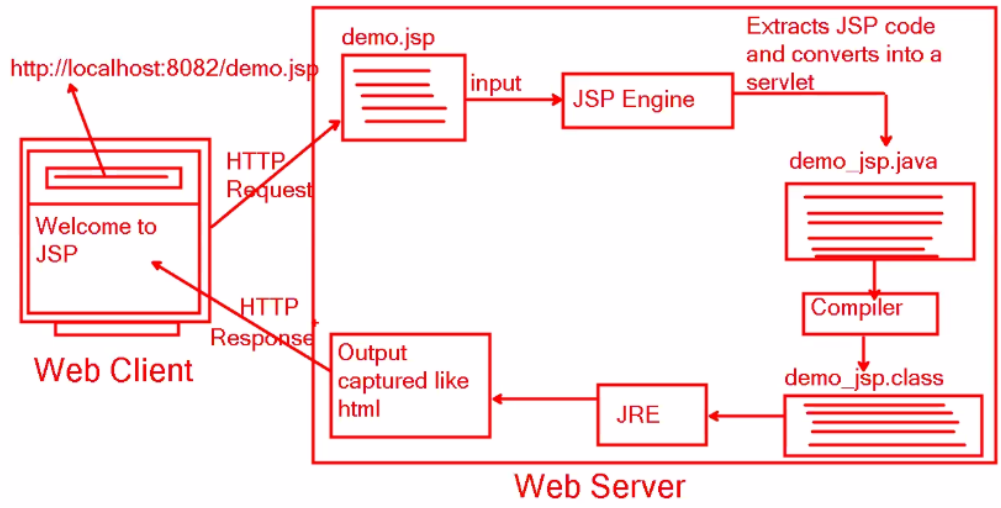
⇒JSP stands for Java Server Page.
⇒ JSP is a specification for developing web applications with the Java programming language.
⇒ Java Server Pages renamed as Jakarta Server Pages.




JSP Engine is a piece of software that converts JSP into Servlet.


JSP Engine is a part of the server only.

## JSP Architecture:




There are three phases in JSP architecture:

1) Conversion
2) Compilation
3) Execution

Conversion done by JSP engine
Compilation done by Java Compiler.
Execution done by Java Runtime Environment.
JSP Engine name is Jasper in a tomcat server.
Servlet Container(Web container) is Catalina in a Tomcat Server.

## JSP Elements:

JSP elements are used to write JSP Programs.

### JSP elements are divided into 3 categories.

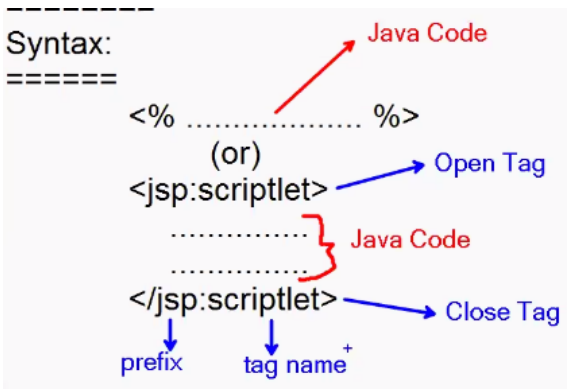
1. Scripting Elements.
2. Directives
3. Actions

## Scripting Elements:

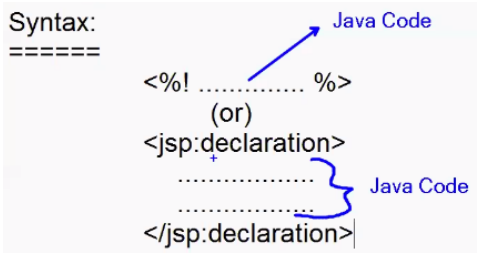
There are four Scripting Elements and used to write basic JSP Programs.

1. Scriptlets
2. Declarations
3. Expressions
4. Comments

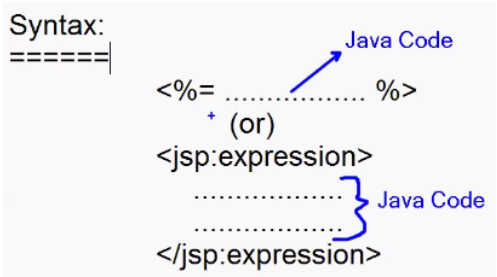
### Scriptlets:



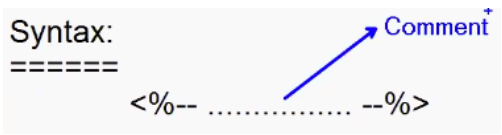

### Declarations:




### Expressions:




### Comments:




## JSP Implicit Object References.:

1. out ⇒ is an object reference of ⇒ JspWriter
2. request ⇒ is an object reference of ⇒ HttpServletRequest
3. response ⇒ is an object reference of ⇒ HttpServletResponse

4. config ⇒ is an object reference of ⇒ ServletConfig
5. application ⇒ is an object reference of ⇒ ServletContext
6. session ⇒ is an object reference of ⇒ HttpSession
7. page ⇒ is an object reference of ⇒ Object
8. pageContext ⇒ is an object reference of ⇒ PageContext
9. exception ⇒ is an object reference of ⇒ Throwable

**demo.jsp file**

```
<% out.println("Welcome to JSP"); %>
```

### JSP Deployment Location:

**C:\Program Files\Apache Software Foundation/Tomcat 10.0lwebappslROOT**

### Converted Servlet Location:

C:\Program Files\Apache Software Foundation\Tomcat 10.0/work/Catalina/localhost/ROOT/org/apache/jsp

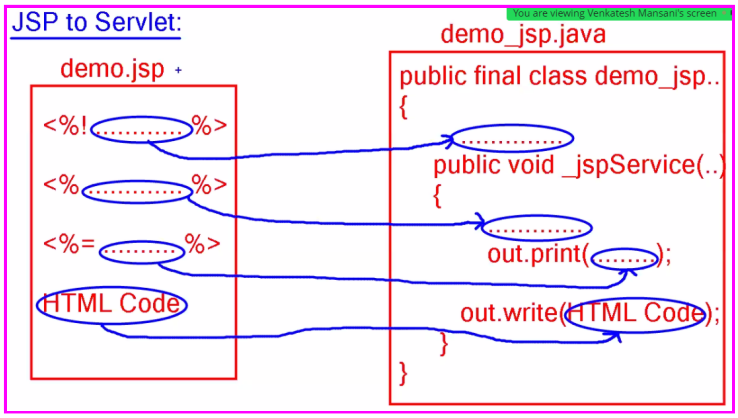
**test.jsp file**

```
<html>
<body bgcolor=yellow text=blue>
 <h1>
 <% out.println("Welcome to Naresh iTechnologies Ameerpet, Hyderabad");
%>
</h1>
</body>
 </html>
```

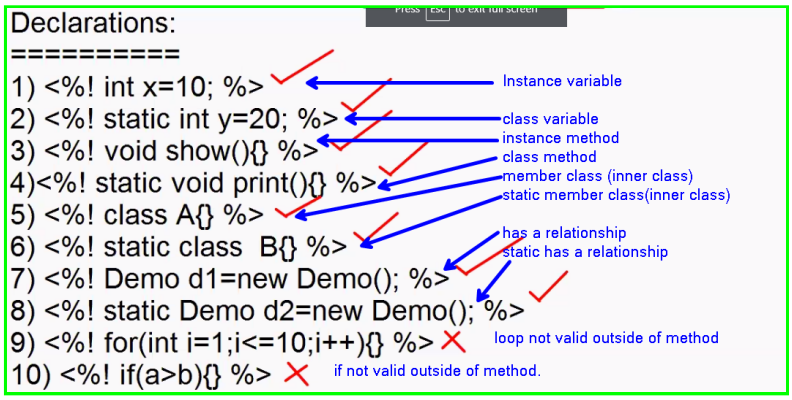
**second.jsp file in eclipse jsp project folder**

```
<html>
<body bgcolor=green text=white>
<h1>
<jsp:scriptlet>
out.println("Welcome to Naresh I technologies");
</jsp:scriptlet>
</h1>
</body>
</html>
```

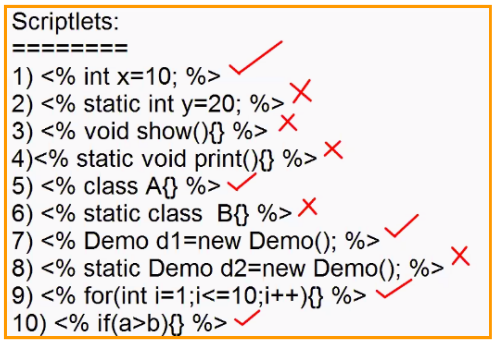
## JSP to Servlet:



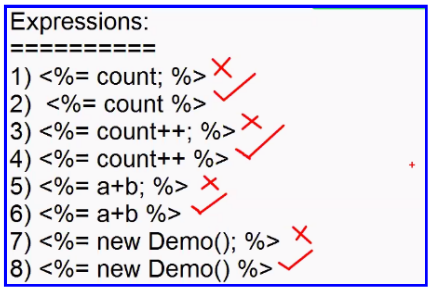

## Declarations:




## Scriptlets:




## Expressions:



java.time.LocalTime

Method:
public static LocalTime now();
=> It is used to get the present time.

**time.jsp file for checking time in eclipse jsp project**

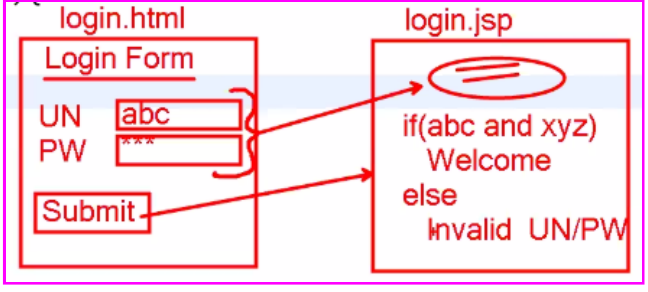
```
<html>
<body bgcolor=red text=yellow>
<% java.time.LocalTime lt = java.time.LocalTime.now(); %>
<%= lt %>
</body>
</html>
```

**Count.jsp file for checking views on website its in jsp project**

```
<html>
<body bgcolor=blue text=yellow>
<h1>
<%! int count=0; %>
This page has been accessed <%= ++count %> times
</h1>
</body>
</html>
```


**table.jsp file for printing table on website its in jsp project**

```
<html>
<body bgcolor=yellow text=red>
<table border=20>
<% for (int i=1;i<=100; i++){ %>
<tr>
<td>5</td>
<td>x</td>
<td><%= i %></td>
<td>=</td>
<td><%= 5*i %></td>
</tr>
<% } %>
</table>
</body>
</html>
```

**login.html file for login form in html its in jsp project**

```
<html>
<body bgcolor=yellow text=red>
<center>
<h1><u>Login Form</u></h1>
<form action="login.jsp">
Username <input type=text name=uname><br>
Password <input type=password name=pword><br><br>
<input type=submit><input type=reset>
</form>
</center>
</body>
</html>
```

**login.jsp file for login form in html its in jsp project**

```
<html>
<body bgcolor=red text=yellow>
<%
String s1=request.getParameter("uname");
String s2=request.getParameter("pword");
if(s1.equals("abc")&& s2.equals("xyz")){
    out.println("Welcome");
}else{
    out.println("Invalid username/password");
}
%>
</body>
</html>
```

# Directives:

**There are 3 directives in JSP:**

1. Include Directive
2. Page Directive
3. Taglib Directive

```
<%@ include ..... %>
<%@ page ..... %>
<%@ taglib ..... %>
```

## Include Directive:

```
Syntax:                          Attributes
======
<%@ include ..... %>
```

**Attributes:**

1) **file= "....."** **//html file or jsp file only.**

**include.jsp file for include directive its in jsp project**

```
<html>
<body bgcolor=green text=yellow>
<h1><u><% out.println("include directive example"); %></u></h1>
<%@ include file="count.jsp" %>
```

```
<%@ include file="time.jsp" %>
<%@ include file="table.jsp" %>
</body>
</html>
```


RequestDispatcher rd=request.getRequestDispatcher(".....");
rd.include(request, response);

<url-pattern> of Servlet/
HTML file name/
JSP file name

```
1 <html>
2 <body bgcolor=green text=yellow>
3 <h1><u><% out.println("Include Directive Example"); %></u></h1>
4 <%@ include file="count.jsp" %>
5 <%@ include file="time.jsp" %>
6 <%@ include file="table.jsp" %>
7 </body>
```

HTML file name /JSP file name

## Page Directive:

Syntax:
======
<%@ page ..... %>

Attributes

1. **language**

Attributes:
========
1) language="....."

Java → default

Example:
=======
<%@ page language="Java" %>

2. **import**

2) import="....."

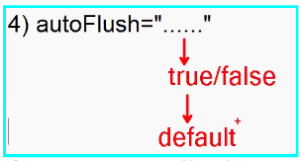
The following package are imported implicitly in JSP.
1) java.lang
2) javax.servlet (or) jakarta.servlet
3) javax.servlet.http (or) jakarta.servlet.http
4) javax.servlet.jsp (or) jakarta.servlet.jsp

Examples:
========
1) <%@ page import="java.util.Date" %>
2) <%@ page import="java.util.*" %>
3) <%@ page import="java.util.*, java.sql.*" %>

3. **buffer**

3) buffer="....."
It is used to specify the buffer size.

4. **autoFlush**



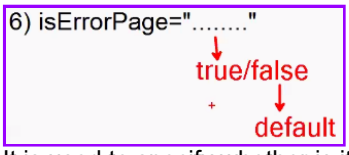

@The above buffer & autoFlush attributes are used to manage buffer memory efficiently.

```
Example:
=======
<%@  page buffer="16kb" autoFlush="true" %>
```

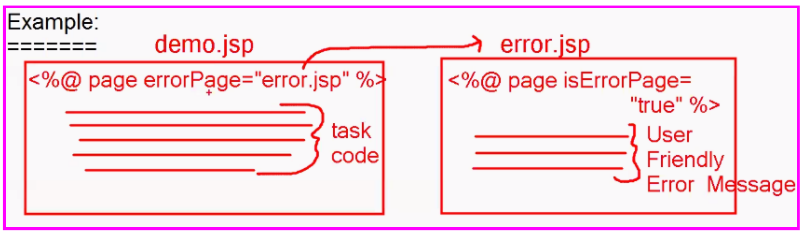
5. **errorPage**


It is used to specify an error page url.

6. **isErrorPage**




It is used to specify whether is it task page or error page.




In the above example error.jsp program executed whenever exception occurs in a demo.jsp.

**arithmetic.html file for arithmetic divide program in jsp project**

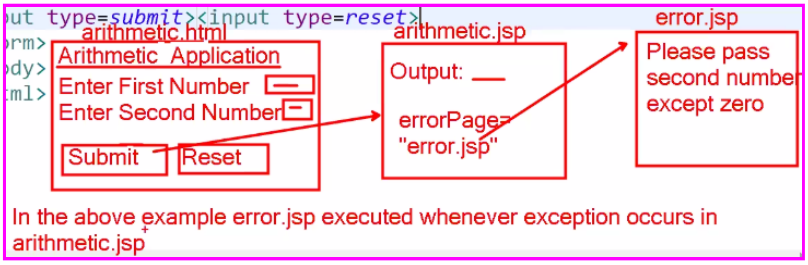
```
<!DOCTYPE html>
<html>
<body bgcolor=green text=yellow>
<h1><u>Arithmetic Application</u></h1>
<form action=arithmetic.jsp>
Enter First Number <input type=text name=first><br>
Enter Second Number <input type=text name=second><br><br>
<input type=submit><input type=reset>
</form>
</body>
</html>
```

**arithmetic.jsp file for arithmetic divide program in jsp project**

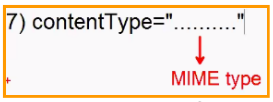
```
<html>
<body bgcolor=yellow text=blue>
<%@ page errorPage="error.jsp"%>
<%
String s1 = request.getParameter("first");
String s2 = request.getParameter("second");
int x= Integer.parseInt(s1);
int y= Integer.parseInt(s2);
int z=x/y;
out.println("output : "+ z);
%>
</body>
</html>
```

**error.jsp file for arithmetic divide program in jsp project**

```
<html>
<body bgcolor=red text=yellow>
<%@ page isErrorPage="true" %>
<% out.print("please pass second number except 0"); %>
</body>
</html>
```




### 7. contentType




It is used to specify the mime type.
Default mime type is “text/html”

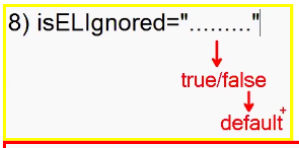
**demo2.jsp file for mime type its in jsp project**

```
<%@ page contentType="application/msword" %>
<% out.println("welcome to jsp"); %>
```

**demo3.jsp file for mime type its in jsp project**

```
<%@ page contentType="application/vnd.ms-excel" %>
<% out.println("welcome to jsp"); %>
```

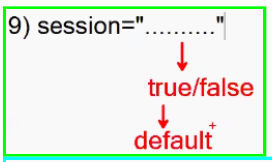
### 8. isELIgnored




Example:
=======
<%@ page isELIgnored="true" %>

This statement indicates a web container to ignore JSP Expression Language in a JSP.

### 9. session



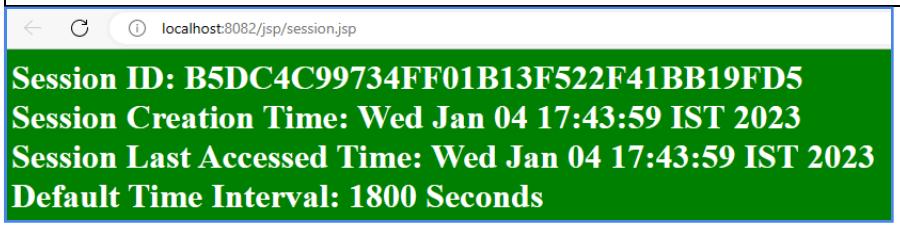

Example:
=======
<%@ page session="false" %>

This statement indicates web containers that do not create sessions for this application.
It does not allow access to implicit session objects in this application.

**session.jsp file for session its in jsp project**

```
<html>
 <body bgcolor=green text=white>
 <h1>
 <%@ page import= "java. util. *" %>
 <%= "Session ID: "+session. getId() %><br>
```

```
 <%= "Session Creation Time: "+new Date(session.getCreationTime()) %>
<br>
 <%= "Session Last Accessed Time: "+new Date(session.
getLastAccessedTime ()) %><br>
 <%= "Default Time Interval: " +session.getMaxInactiveInterval()+"
Seconds" %>
</h1>
</body>
</html>
```




## JSP Elements

```
JSP Elements:
============
1) Scripting Elements(Scriptlets, Declarations, Expressions  & Comments)
2) Directives(Include, Page & Taglib)
3) Actions
```

10. **isThreadSafe**

```
10) isThreadSafe="....."
                    ↓
                 true/false
                    ↓
                 default
```

⇒ By default all servlets & JSPs are multithreaded.
⇒ If we set false to this attribute then the converted servlet implements jakarta.servlet.SingleThreadModel interface.

⇒ If the servlet class implements jakarta.servlet.SingleThreadModel interface then the web container allows access to this servlet one thread at a time.

⇒ jakarta.servlet.SingleThreadModel interface is a marker interface , tag interface, or empty interface, because there are no members in this interface.

### TagLib Directive :

```
Syntax:                    Attributes+
======                   ↗
<%@ taglib ..... %>
```

```
Attributes:
========
1) uri="....."
2) prefix="....."
```

```
Example:
=======
<%@ taglib uri="....." prefix="....." %>
```

The above taglib directive required to include tag library while using JSTL
JSTL stands for JSP Standard Tag Library.

### Actions or JSP Actions :

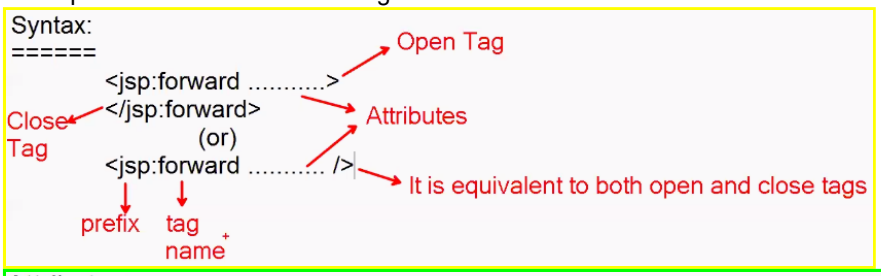
There are 9 JSP actions:

1) <jsp:forward>
2) <jsp:include>
3) <jsp:useBean>
4) <jsp:setProperty>
5) <jsp:getProperty>

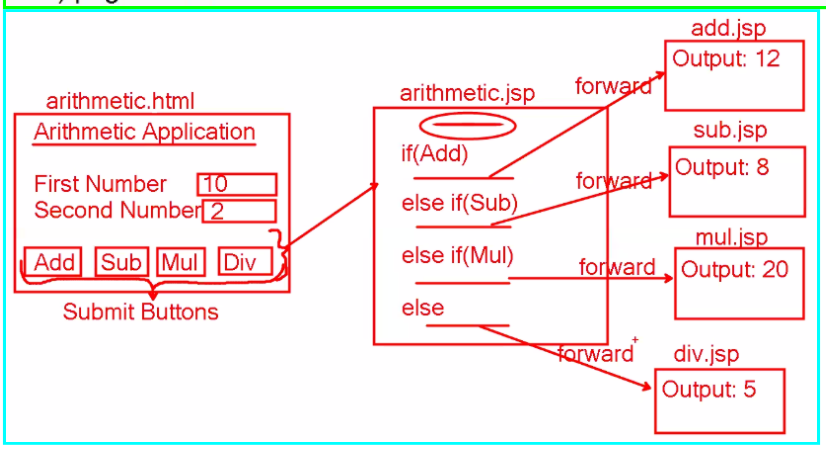
6) <jsp:plugin>
7) <jsp:fallback>
8) <jsp:params>
9) <jsp:param>

1. **<jsp:forward>**

It is used to forward the request & response of one JSP to another JSP.
It is equivalent to servlet forwarding.








**arithmetic.html file for calculator program in jsp**

```
<!DOCTYPE html>
<html>
<body bgcolor=green text=yellow><center>
<h1><u>Arithmetic Application</u></h1>
<form action=arithmetic.jsp>
First Number <input type=text name=first><br>
Second Number <input type=text name=second><br><br>
<input type=submit name=operation value=addition>
<input type=submit name=operation value=subtraction>
<input type=submit name=operation value=multiplication>
<input type=submit name=operation value=division>
</form>
</center>
</body>
</html>
```

**arithmetic.jsp file for calculator program in jsp**

```
<html>
<body bgcolor=yellow text=blue>
<%
	String s1 = request.getParameter("operation");
	if (s1.equals("addition")) {
	    %><jsp:forward page="add.jsp"/><%
	} else if (s1.equals("subtraction")) {
	    %><jsp:forward page="sub.jsp"/><%
	} else if (s1.equals("multiplication")) {
	    %><jsp:forward page="mul.jsp"/><%
	} else
```

```
          %><jsp:forward page="div.jsp"/><%
%>
</body>
</html>
```

**add.jsp file for calculator program in jsp**

```
<html>
<body bgcolor=red text=white>
    <h1>Addition Result</h1>
    <%
      String s1 = request.getParameter("first");
      String s2 = request.getParameter("second");
      int x = Integer.parseInt(s1);
      int y = Integer.parseInt(s2);
      out.println("Output: "+(x+y));
    %>
  </body>
</html>
```

**sub.jsp file for calculator program in jsp**

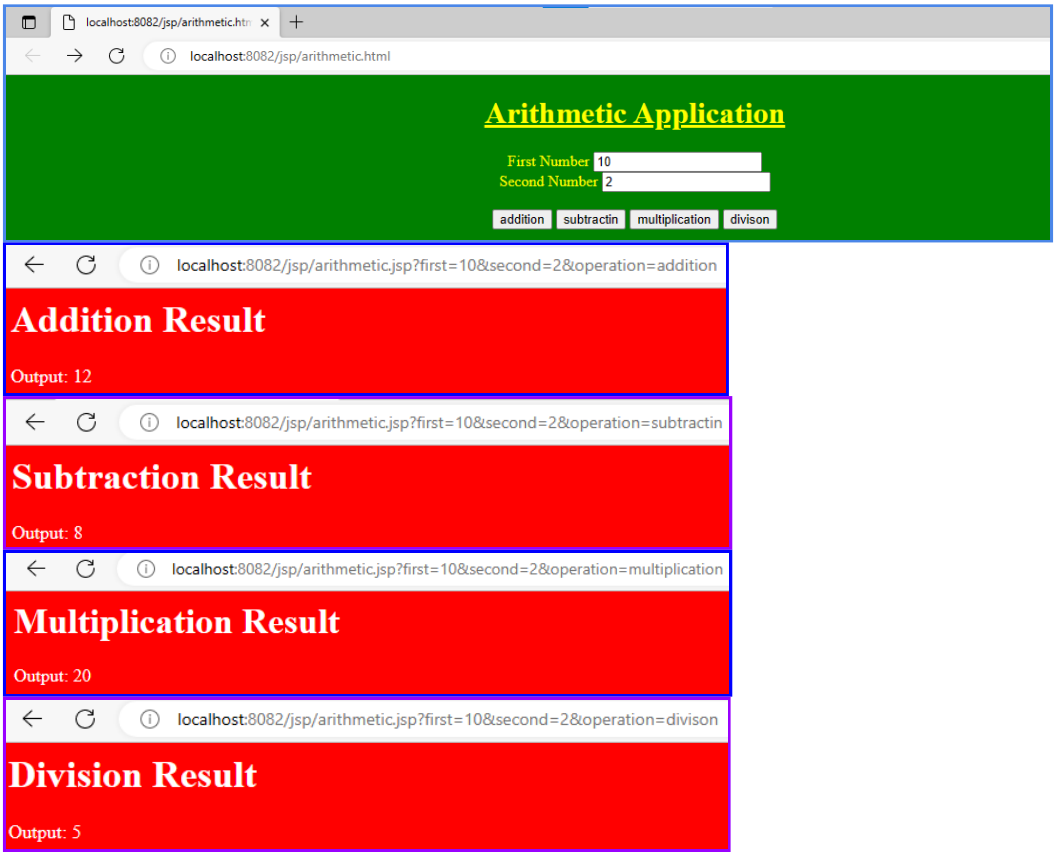
```
<html>
<body bgcolor=red text=white>
    <h1>Subtraction Result</h1>
    <%
      String s1 = request.getParameter("first");
      String s2 = request.getParameter("second");
      int x = Integer.parseInt(s1);
      int y = Integer.parseInt(s2);
      out.println("Output: "+(x-y));
    %>
  </body>
</html>
```

**mul.jsp file for calculator program in jsp**

```
<html>
<body bgcolor=red text=white>
    <h1>Multiplication Result</h1>
    <%
      String s1 = request.getParameter("first");
      String s2 = request.getParameter("second");
      int x = Integer.parseInt(s1);
      int y = Integer.parseInt(s2);
      out.println("Output: "+(x*y));
    %>
  </body>
</html>
```

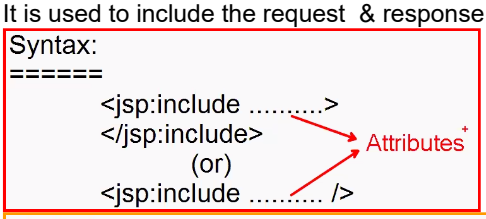
**div.jsp file for calculator program in jsp**

```
<html>
<body bgcolor=red text=white>
    <h1>Division Result</h1>
    <%
      String s1 = request.getParameter("first");
      String s2 = request.getParameter("second");
      int x = Integer.parseInt(s1);
      int y = Integer.parseInt(s2);
      out.println("Output: "+(x/y));
    %>
  </body>
</html>
```



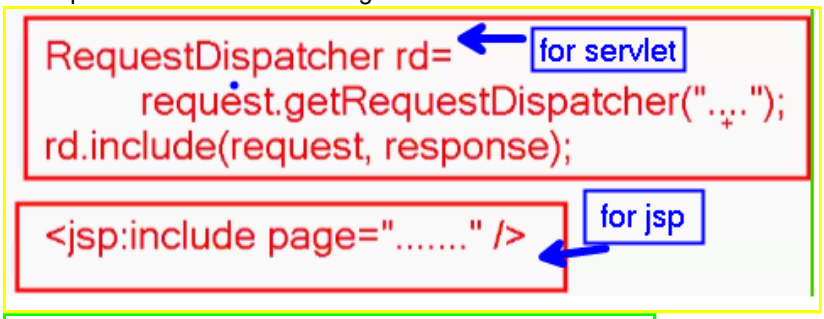

2. **<jsp:include>**

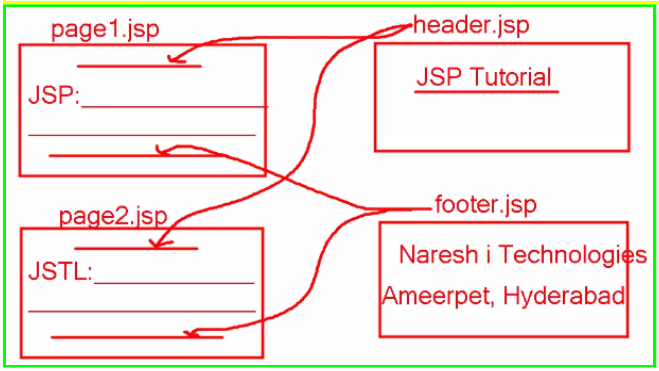
It is used to include the request & response of one jsp into another jsp





It is equivalent to servlet including.




| **header.jsp file for including jsp to another jsp it's in jsp project.** |
| --- |
| `<html>`<br>`<body>` |

```
<center><h1><u><font color=red>
<% out.println("JSP Tutorial"); %>
</font></u></h1></center>
</body>
</html>
```

**footer.jsp file for including jsp to another jsp it's in jsp project.**

```
<html>
<body>
<center><h2><font color=blue>
<% out.println("Naresh I technologies, Ameerpet, Hyderabad"); %>
</font></h2></center>
</body>
</html>
```

**page1.jsp file for including jsp to another jsp it's in jsp project.**

```
<html>
<body bgcolor=yellow text=green>
<jsp:include page="header.jsp"/>
<h1><u>JSP:</u></h1>
<% out.println("JSP stands for Java Server Pages. JSP is a specification
for developing web applications with Java programming language."); %>
<jsp:include page="footer.jsp"/>
</body>
</html>
```

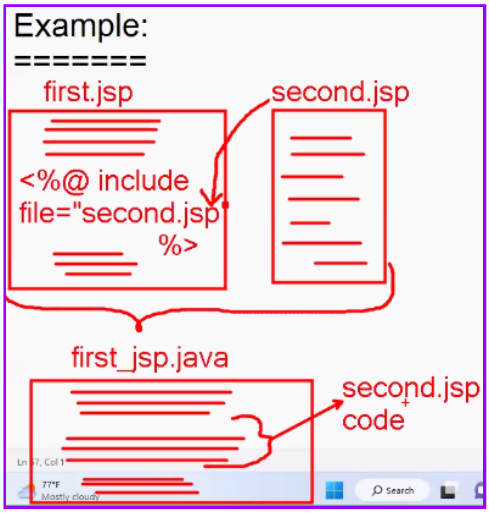
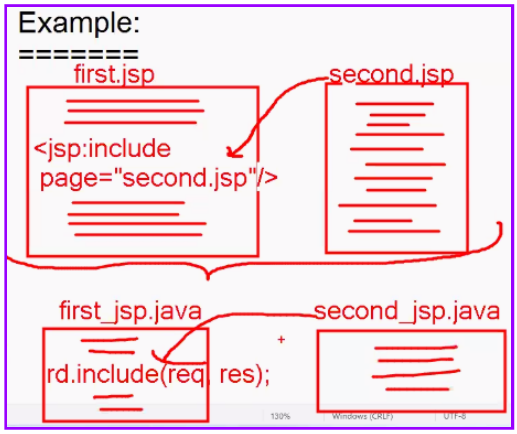
**page2.jsp file for including jsp to another jsp it's in jsp project.**

```
<html>
<body bgcolor=yellow text=green>
<jsp:include page="header.jsp"/>
<h1><u>JSTL:</u></h1>
<% out.println("JSTL stands for Java server pages Standard Tag Library.
It is introduced in JSP 2.0 version to simplify the JSP."); %>
<jsp:include page="footer.jsp"/>
</body>
</html>
```

**OUTPUT ::**


**Difference between Include Directive & Include Action:**

| Include Directive | Include Action |
|---|---|
| It includes the file at page translation time. | It includes the file at page request time. |
| It includes the file content. | It includes the request & response |

| | |
|---|---|
| It supports to include html in a jsp & jsp in a jsp. | It supports to include servlet in a jsp, jsp in a jsp & html in a jsp. |
| It is not equal to servlet including. | It is equal to servlet including. |
|  |  |

## Converted Servlet Life Cycle Methods:

1) jspInit() method
2) _jspService() Method
3) jspDestroy() method

### jspInit() method:

Is is called by web container whenever first request comes to a JSP.
This method belongs to jakarta.servlet.jsp.JspPage interface.
This method can be overridden by programmer because this method not overridden by web container.

```
Example:
=======
<%!
public void jspInit()
{
   =========
   =========
}
%>
```

### _jspService() Method:

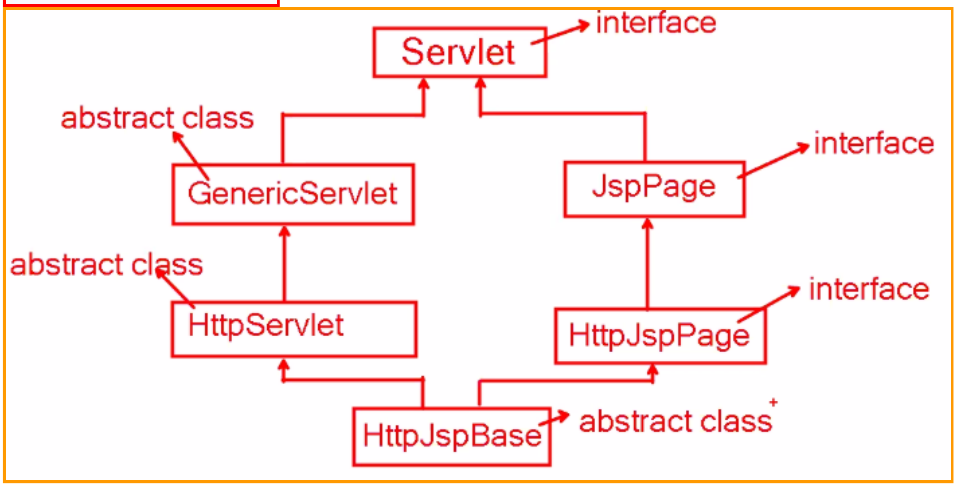
It is called by web container for every request.
This method belongs to jakarta.servlet.jsp.HttpJspPage interface.
This method cannot be overridden by programmer because it is already overridden by web container.
To write a code in this method, use scriptlet

```
Example:
=======
<%
========
========
%>
```

### jspDestroy() method:

It is called by the web container whenever a converted servlet instance is removed from the web container.
Converted servlet instances are removed whenever a web application is undeployed or server shuts down.
This method belongs to jakarta.servlet.jsp.JspPage interface.
This method can be overridden by a programmer because this method is not overridden by web container.

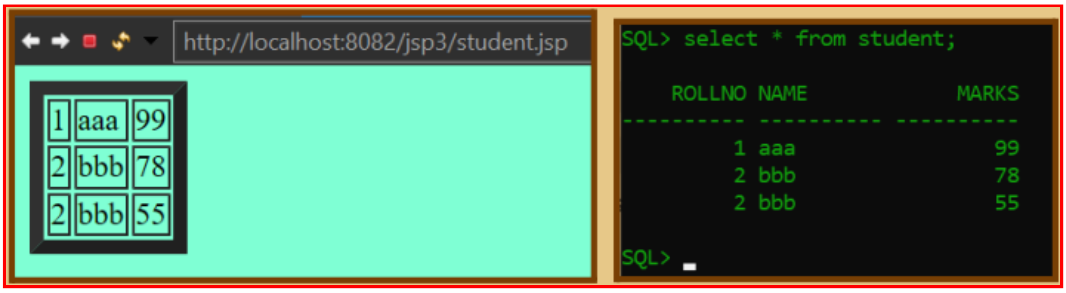
```
Example:
=======
<%!
public void jspDestroy()
{
   =========
   =========
}
```




**student.jsp file its in jsp project.**

```
<!-- JSP Program to demonstrate converted servlet life cycle method -->
<html>
<body bgcolor=green text=yellow>
    <%@ page import="java.sql.*"%>
    <%!Connection con;

    public void jspInit() {
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl",
"sam", "oracle");
        } catch (Exception e) {
            System.err.println(e);
        }
    }%>
    <table border=10>
        <%
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("select * from student");
        while (rs.next()) {
            out.println("<tr><td>" + rs.getInt("rollno") + "</td>");
            out.println("<td>" + rs.getString("name") + "</td>");
            out.println("<td>" + rs.getInt("marks") + "</td></tr>");
        }
        %>
    </table>
    <%!public void jspDestroy() {
        try {
            con.close();
        } catch (Exception e) {
```

```
            System.err.println(e);
        }
    }%>
</body>
</html>
```




## JSP using Java Beans:

A Java Bean is a reusable software component.
A Java class is said to be Java Bean if it is follows the following rules:

1) Class must be public
2) Class must implements java.io.Serializable interface
3) Class must be in a package
4) Class must contain public default constructor
5) All instance variables must be private. Instance variables are called properties.
6) Each and every property must contain setter & getter methods.
7) All setter & getter methods must be public.

**Java bean examples:**

```
Package demo;
Import java.io*;
public class MessageBean implements Serializable{
private String message;
void setMessage(String message){
        this.message=message;
    }
    public String getMessage(){
        return message;
    }
}
```

**The following actions are used to access a Java Bean in a JSP**

1) <jsp:useBean>
2) <jsp:setProperty>
3) <jsp:getProperty>

## <jsp:useBean> :

It is used to create an object to a Java Bean class.

Syntax:
======
<jsp:useBean ..... >
</jsp:useBean>
(or)
<jsp:useBean ..... />
Attributes

Attributes:
=======
1) id="....." (object reference)
2) class="....." (Java bean class name)
3) scope="....." (page/request/session/application)

Example:
=======
<jsp:useBean id="mb" class="demo.MessageBean" />
The above code is equivalent to the following code
<% demo.MessageBean mb=new demo.MessageBean(); %>

## <jsp:setProperty> :

It is used to call setter methods of a Java bean.

Syntax:
======
<jsp:setProperty .....>
</jsp:setProperty>
(or)
<jsp:setProperty ..... />
Attributes+

Attributes:
========
1) name="....." object reference
2) property="....." name of the property
3) value="....." value of the property+

Example:
=======
<jsp:setProperty name="mb" property="message" value="Welcome" />
The above code is equivalent to the following code
<% mb.setMessage("Welcome"); %>

## <jsp:getProperty> :

It is used to call getter methods of a Java Bean.

Syntax:
======
<jsp:getProperty ..... >
</jsp:getProperty>
(or)
<jsp:getProperty ..... />
Attributes+

Attributes:
========
1) name="...." object reference
2) property="....." name of the property+

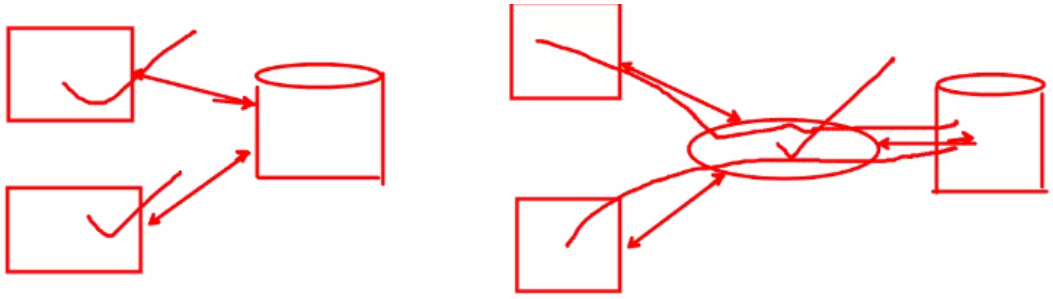
Example:
=======
<jsp:getProperty name="mb" property="message" />
The above code is equivalent to the following code.
<%= mb.getMessage() %>

**MessageBean.java file for message.jsp program its in jsp project**

```
package jsp;
import java.io.Serializable;
public class MessageBean implements Serializable {
    private String message;
    public String getMessage() {
        return message;
    }
    public void setMessage(String message) {
        this.message = message;
    }
}
```

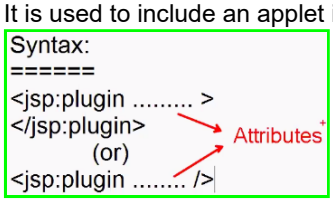
**message.jsp file for MessageBean program its in jsp project**

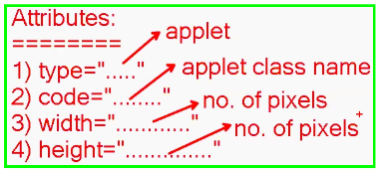
```
<html><body bgcolor=yellow text=red><h1>
<jsp:useBean id="mb" class="jsp.MessageBean" />
<jsp:setProperty name="mb" property="message" value="welcome" />
<jsp:getProperty name="mb" property="message"/>
</h1></body></html>
```



## <jsp:plugin> :

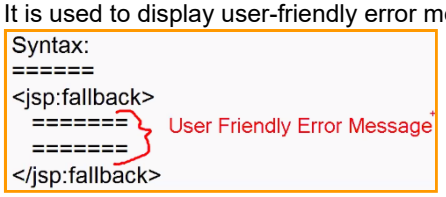
It is used to include an applet in a jsp.




## <jso:fallback> :

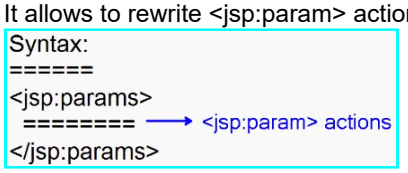
It is used to display user-friendly error messages.




This message is displayed whenever the browser is unable to load an applet.
This action must be used in <jsp:plugin> action only.

## <jsp:params> :

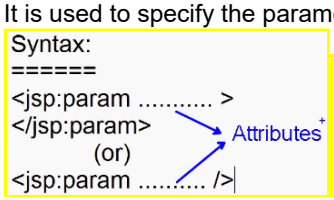
It allows to rewrite <jsp:param> actions.

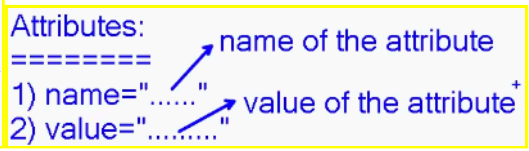



This action also must be used in <jsp:plugin> action only.

## <jsp:param> :

It is used to specify the parameter name with value.




This action must be used in <jsp:params> action only.

```
Example:
=======
<jsp:plugin type="applet" code="demo.MyApplet" width="400"
                                                height="400">
<jsp:params>
<jsp:param name="message" value="Welcome" />
</jsp:params>
<jsp:fallback>
Unable to load an Applet
</jsp:fallback>
</jsp:plugin>
```

The above parameter can be retrieved in applet by using getParameter() method of java.applet.Applet class.

# JSP EL:

- JSP EL stands for Java Server Pages Expression Language.
- JSP EL was introduced with JSTL in the JSP 2.0 version to simplify the JSP.
- JSTL stands for JSP Standard Tag Library.
- Both JSP EL & JSTL are used to simplify the JSP.
- JSP EL can be used independently and it can also be used with JSTL.
- By using JSP EL & JSTL in a JSP, JSP becomes a 100% tag based application.

- Expression can be replaced with JSP EL to make JSP as 100% tag based application.
- Scriptlets & Declarations can be replaced with JSTL to make JSP as a 100% tag based application.
- The pattern that identifies JSP Expression Language is ${...}
- By default JSP EL is enabled.
- To disable JSP EL in a JSP, use the following.
  **< % @ page isELIgnored=”true” % >**

| Operator | Meaning |
|---|---|
| + | Addition |
| - | Subtraction |
| * | Multiplication |
| / or div | Division |
| % or mod | Modulo Division |
| < or lt | Less than |
| > or gt | Greater than |
| <= or le | Less than or equal to |
| >= or ge | Greater than or equal to |
| == or eq | Equals to |
| != or ne | Not equals to |
| && or and | Logical AND |
| \|\| or or | Logical OR |
| ! or not | Logical NOT |

demo4.jsp file for JSP EL program its in jsp project.

```
<html><body bgcolor=red text=yellow><h1>
${ 5+2 }<br>
${ 5-2 }<br>
${ 5*2 }<br>
${ 5 div 2 }<br>
${ 5 mod 2 }<br>
${ 5 lt 2 }<br>
${ 5 ne 2 }<br>
</h1></body></html>
```


## JSP EL Implicit Object References:

1. pageScope
2. requestScope
3. sessionScope
4. applicationScope
5. param
6. initParam
7. Cookie

## pageScope:

It is use to get the values of page scope attributes:

## requestScope:

It is used to get the values of request scope attributes.

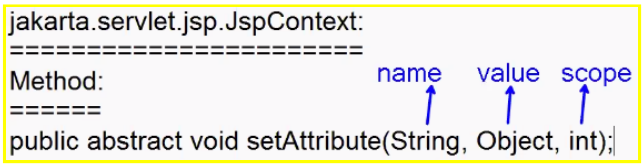
## sessionScope:

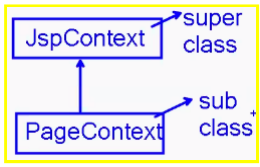
It is used to get the values of session scope attributes.

## applicationScope:

It is used to get the values of application scope attributes.

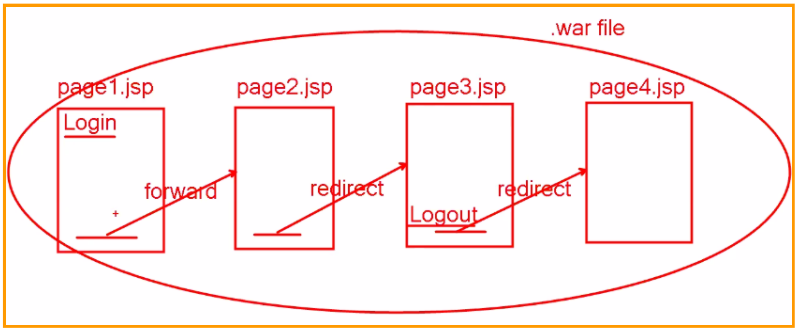
**jakarta.servlet.jsp.PageContext:**
**Fields(Variables):**
1) public static final int PAGE_SCOPE;
2) public static final int REQUEST_SCOPE;
3) public static final int SESSION_SCOPE;
4) public static final int APPLICATION_SCOPE;




**scope.jsp program in jsp project**

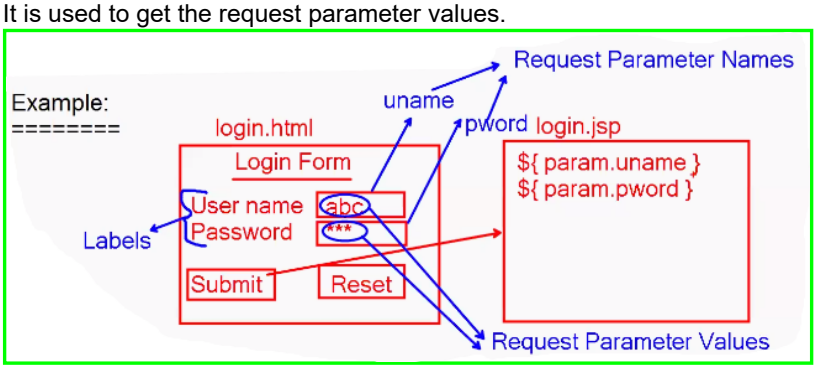
```
<html>
<body bgcolor=red text=yellow>
    <h1>
        <%
        pageContext.setAttribute("platform1", "JavaSE",
PageContext.PAGE_SCOPE);
        pageContext.setAttribute("platform2", "JavaEE",
PageContext.REQUEST_SCOPE);
        pageContext.setAttribute("platform3", "JavaME",
PageContext.SESSION_SCOPE);
        pageContext.setAttribute("platform4", "JavaFX",
PageContext.APPLICATION_SCOPE);
        %>
        ${ pageScope.platform1 }<br> ${ requestScope.platform2 }<br>
        ${ sessionScope.platform3 }<br> ${ applicationScope.platform4 }
    </h1>
</body>
</html>
```



In the above example, platform1 can be accessed in page1.jsp only because its scope is page scope , platform2 can be accessed in page1.jsp & page2.jsp because its scope is request scope, platform3 can be accessed in page1.jsp, page2.jsp, & page3.jsp because its scope is session scope and platform can be accessed in all pages in a .war file.

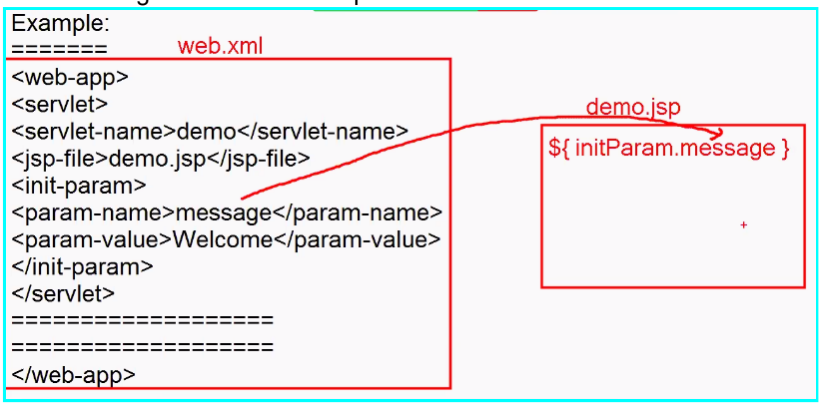
## param:

It is used to get the request parameter values.



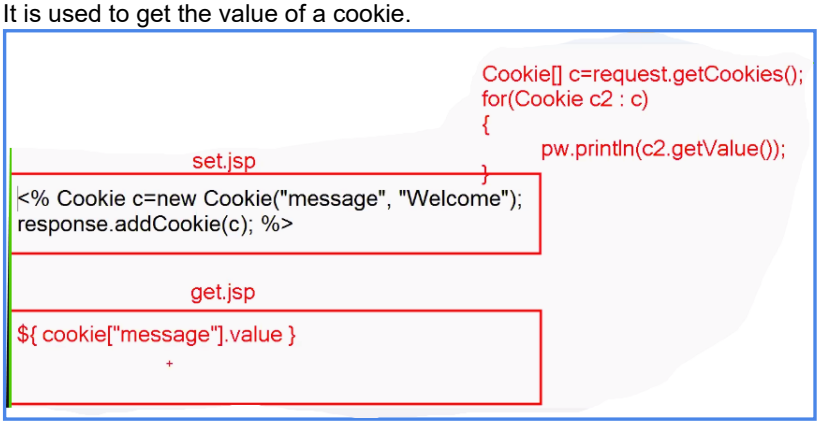

## initParam:

It is used to get the initialization parameter values.




## cookie:

It is used to get the value of a cookie.



# JSTL:

JSTL Tags are divided into 5 categories:
1) Core Tags
2) SQL Tags
3) Formatting Tags
4) Function Tags
5) XML Tags

first.jsp

```
<jsp:forward page="second.jsp" />
```

In order to use the above tags we must copy the following jar files into tomcat lib folder.
1) jstl.jar
2) standard.jar

## Core Tags:

1. <c:out>
2. <c:set>
3. <c:remove>
4. <c:if>
5. <c:choose>
6. <c:when>
7. <c:otherwise>
8. <c:forEach>
9. <c:forTokens>
10. <c:redirect>

In order to use the above tag, we must include the following taglib directive in a JSP.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

### <c:out>:

- It is used to display output messages.

**out.jsp file in jstl project**

```
<html>
<body bgcolor=green text=yellow>
    <h1>
        <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

        <c:out value="Welcome to JSTL" />
    </h1>
</body>
</html>
```


### <c:set>

- It is used to store the value in a variable

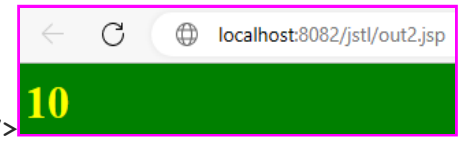
### <c:remove>

- It is used to remove the value from a variable.

**out2.jsp file in jstl project**

```
<html>
```

```
<body bgcolor=green text=yellow>
    <h1>
        <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
        <c:set var="a" value="10" />
        <c:out value="${a}" />
        <c:remove var="a" />
        <c:out value="${a}" />
    </h1>
</body>
</html>
```

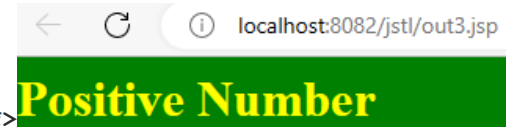



### <c:if>

- It is used to express the condition.

**out3.jsp file in jstl project**

```
<html>
<body bgcolor=green text=yellow>
    <h1>
        <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
        <c:set var="a" value="10" />
        <c:if test="${ a>0 }">
            <c:out value="Positive Number" />
        </c:if>
    </h1>
</body>
</html>
```




### <c:choose>

### <c:when>

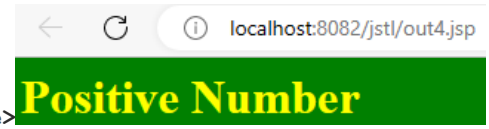
### <c:otherwise>

- The above 3 tags are equivalent to if else if… else statement & switch statement.

**out4.jsp file in jstl project**

```
<html>
<body bgcolor=green text=yellow>
    <h1>
        <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
        <c:set var="a" value="10" />
        <c:choose>
            <c:when test="${a>0}">
                <c:out value="Positive Number" />
            </c:when>
            <c:when test="${a<0}">
                <c:out value="Negative Number" />
            </c:when>
            <c:otherwise>
                <c:out value="Zero" />
```

```
            </c:otherwise>
        </c:choose>
    </h1>
</body>
</html>
```

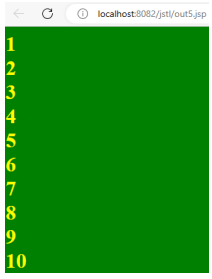



### <c:forEach> :

- It is equivalent to for loop & enhanced for loop.

**out5.jsp file in jstl project**

```
<html>
<body bgcolor=green text=yellow>
    <h1>
        <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
```




```
        <c:forEach var="i" begin="1" end="10" step="1">
            <c:out value="${i}" />
            <br>
        </c:forEach>
    </h1>
</body>
</html>
```

### <c:forTokens> :

- It is used to iterate token by token from a given string.
- It is equivalent to the StringTokenizer class.

**out6.jsp file in jstl project**

```
<html>
<body bgcolor=green text=yellow>
    <h1>
        <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
        <c:forTokens var="s" items="Welcome to JSTL" delims=" ">




            <c:out value="${s}" />
            <br>
        </c:forTokens>
    </h1>
</body>
</html>
```

### <c:redirect>

It is used to pass the control to the next jsp.
It is equivalent to servlet redirecting.

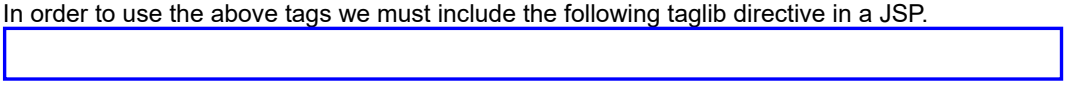
**test.jsp file in jstl project**

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<c:redirect url="out.jsp" />
```

## SQL Tags:

1. <sql:setDataSource>
2. <sql:update>
3. <sql:query>

In order to use the above tags we must include the following taglib directive in a JSP.



### <sql:setDataSource> :

It is used to establish the connection between JSP & database.

**test2.jsp file in jstl project**

```
<html>
<body bgcolor=green text=yellow>
    <h1>
        <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
        <%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
   <sql:setDataSource var="con" driver="oracle.jdbc.driver.OracleDriver"
            url="jdbc:oracle:thin:@localhost:1521:orcl" user="sam"
            password="oracle" />
        <c:out value="Connection Established Successfully" />
    </h1>
</body>
</html>
```

### <sql:update>

It is used to execute non select sql queries.

**insert.jsp file in jstl project**

```
<html>
<body bgcolor=yellow text=blue>
    <h1>
        <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
       <%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
```

```
<sql:setDataSource var="con" driver="oracle.jdbc.driver.OracleDriver"
            url="jdbc:oracle:thin:@localhost:1521:orcl" user="sam"
            password="oracle" />
        <c:out value="Connection Established Successfully" /><br>
        <sql:update dataSource="${con}" sql="insert into student
values(5,'eee',91)" />
        <c:out value="One Record inserted Successfully" />




    </h1>
</body>
</html>
```

### <sql:query>

It is used to execute select queries.

**insert2.jsp file in jstl project**

```
<html>
<body bgcolor=yellow text=blue>
    <table border=10>
        <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
       <%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>

<sql:setDataSource var="con" driver="oracle.jdbc.driver.OracleDriver"
            url="jdbc:oracle:thin:@localhost:1521:orcl" user="sam"
            password="oracle" />
        <c:out value="Connection Established Successfully" />
        <br>
        <sql:query var="rs" dataSource="${con}" sql="select * from
student" />
        <c:forEach var="record" items="${rs.rows}">
            <tr>
                <td><c:out value="${record.rollno}" /></td>
                <td><c:out value="${record.name}" /></td>
                <td><c:out value="${record.marks}" /></td>









            </tr>
        </c:forEach>
    </table>
</body>
</html>
```

# MVC

MVC stands for model view controller.
MVC architecture used to separate presentation logic, business logic, data access logic & data.
It is a convenient way to manage & update web application by using MVC architecture.
Here Model is a Java Bean, View is a JSP & Controller is a Servlet.

## MVC Architecture:

**result.html file in mvc project**

```
<html>
<body bgcolor=green text=yellow>
<form action=result>
<h1>
Enter Hall Ticket Number<br><input type=text name=hno><br><br>
<input type=submit><input type=reset>
</h1>
</form>
</body>
</html>
```

**ResultServlet.java file in mvc project**

```
package mvc;

import java.io.IOException;
import javax.servlet.*;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;
@WebServlet("/result")
public class ResultServlet extends HttpServlet {
```

```
    protected void doGet(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {
        String s = request.getParameter("hno");
        int h = Integer.parseInt(s);
        ResultDAO rdao = new ResultDAO();
        ResultBean rb = rdao.getResult(h);
        request.setAttribute("result", rb);
        RequestDispatcher rd =
request.getRequestDispatcher("/result.jsp");
        rd.forward(request, response);
    }
}
```

**ResultDAO.java file in mvc project**

```
package mvc;

import java.sql.*;
public class ResultDAO {
    ResultBean getResult(int hno) {
        ResultBean rb=null;
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection con=
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","sam","oracle");
            PreparedStatement pstmt = con.prepareStatement("select * from results
where hno=?");
            pstmt.setInt(1, hno);
            ResultSet rs= pstmt.executeQuery();
            rb= new ResultBean();
            if(rs.next()) {
                rb.setHno(rs.getInt("hno"));
                rb.setName(rs.getString("name"));
                rb.setC(rs.getInt("c"));
                rb.setCpp(rs.getInt("cpp"));
                rb.setJava(rs.getInt("java"));
            }
        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }
        return rb;
    }
}
```

**ResultBean.java file in mvc project**

```
package mvc;

public class ResultBean {
    int hno;
    String name;
    int c;
    int cpp;
    int java;
    public int getHno() {
        return hno;
    }
    public void setHno(int hno) {
        this.hno = hno;
    }
    public String getName() {
        return name;
    }
```

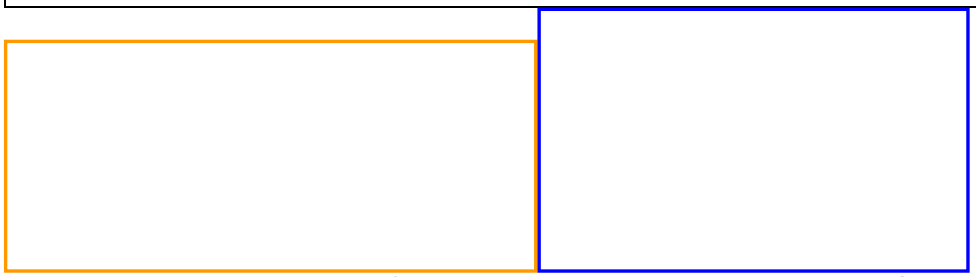
```
    public void setName(String name) {
        this.name = name;
    }
    public int getC() {
        return c;
    }
    public void setC(int c) {
        this.c = c;
    }
    public int getCpp() {
        return cpp;
    }
    public void setCpp(int cpp) {
        this.cpp = cpp;
    }
    public int getJava() {
        return java;
    }
    public void setJava(int java) {
        this.java = java;
    }
}
```

**result.jsp file in mvc project**

```
<html>
<body bgcolor=green text=yellow><h1>
<% mvc.ResultBean rb=(mvc.ResultBean)request.getAttribute("result"); %>
Hall Ticket Number <%= rb.getHno() %><br>
Name : <%= rb.getName() %><br>
C : <%= rb.getC() %><br>
C++ : <%= rb.getCpp() %><br>
Java : <%= rb.getJava() %>
</h1></body>
</html>
```

**Oracle table to show info for mvc project (table name is results)**

```
create table results(hno int,name varchar2(12),c int,cpp int,java int);
insert into results values(1001,'aaa',73,55,81);
insert into results values(1002,'bbb',93,46,88);
insert into results values(1003,'ccc',65,88,92);
```



In the above examples, result.html & result.jsp contains presentation logic, ResultServlet.java contains controlling logic, ResultDAO.java contains data access logic & ResultBean.java contains data whatever is retrieved from the database.

## Advantages of MVC:

1) Easy to maintain
2) Easy to update

3) Rapid application development
4) Parallel development

Jdbc servlet javabean jsp

# Reflection API

- It is used to analyze the structure of a class or interface.
- It is also used to get the variables (fields), methods & constructors of specified classes.
- It is also used to analyze methods, constructors & variables (fields).
- Reflection API used to develop Integrated Development Environments , debuggers & javap tools.

Java.lang.reflect package:
Classes:

1) Field
2) Method
3) Constructor

### java.lang.Class

Methods:

```
public static Class forName(String) throws ClassNotFoundException;
public Field[] getFields() throws SecurityException;
public Method[] getMethods() throws SecurityException;
public Constructor[] getConstructors() throws SecurityException
```

⇒ The above 3 methods are used to get the members of the current class & members of super classes.

```
public Field[] getDeclaredFields0 throws SecurityException;
public Method[] getDeclaredMethods() throws SecurityException;
public Constructor[] getDeclaredConstructors() throws SecurityException;
```

⇒ The above 3 methods are used to get the members of current class only.

**Demo.java file for reflection topic**

```
//Program to get list of methods of a specified class
 import java.lang.reflect.*;
 class Demo{
   public static void main(String args[]){
         try{
         Class c=Class.forName(args(0));
         Method[] m=c.getDeclaredMethods();
         for(Method m2 : m)
         {
             Svstem.out.println(m2);
         }
         }catch (Exception e){
             System.out.println(e);
         }
    }
}
```

*****Advanced Java Preparation/Reference**

1) Notes
2) Sun/Oracle JavaEE Tutorials
3) Core Servlets & JSPs
4) SCWCD/OCWCD Books

*****Core Java Preparation/Reference**

1) Notes
2) Java Complete Reference
3) Head First Java
4) SCJP By Kathy Sierra