# [JDBC](https://kdninfotech.com/advance-java-syllabus/#collapse-f0f54c165a900b6191ce)

JDBC Drivers Types In Java

Steps To Connect To The All Types Of Database.

Connectivity With Access Without And With DSN.

DriverManager Class In Jdbc.

Connection Interface In Jdbc.

Statement Interface In Jdbc.

ResultSet Interface Jdbc.

PreparedStatement In Jdbc.

ResultSetMetaData In Jdbc.

DatabaseMetaData In Jdbc

Storing Image In Jdbc.

Retrieving Image In Jdbc.

Storing File And Retrieving File Using Jdbc.

Stored Procedures And Functions In Plsql Using Java.

Transaction Management Tcl Commands.

Batch Processing Or Group Command Processing.

JDBC New Features In Java.

CRUD Mini Project.

# [Introduction Of Webdesign](https://kdninfotech.com/advance-java-syllabus/#collapse-14514dc65a900b6191ce)

HTML/CSS/JAVASCRIPT Introduction(Hw)

# [Basics Of Servlet](https://kdninfotech.com/advance-java-syllabus/#collapse-4b91dc465a900b6191ce)

Servlet In Java

Basics Of Web Pages

Servlet API Introduction

Servlet Interface In Servlet

GenericServlet In Servlet

HttpServlet In Servlet

Servlet Life Cycle In Java

Apache Tomcat Server

Steps To Create A Servlet In Tomcat And Connectivity

# [ServletRequest](https://kdninfotech.com/advance-java-syllabus/#collapse-35c8fe465a900b6191ce)

ServletRequest Methods In Servlet

Registration Example With DB Etc…

Servlet Collaboration

RequestDispatcher Interface

SendRedirect Method

ServletConfig

ServletConfig Methods

ServletConfig Example

ServletContext

# [Servlet Collaboration](https://kdninfotech.com/advance-java-syllabus/#collapse-c29457c65a900b6191ce)

RequestDispatcher Interface

SendRedirect Method

ServletConfig

ServletConfig Methods

ServletConfig Example

ServletContext

ServletContext Methods In Application Scope

ServletContext Example

Attribute

How To Set, Get And Remove Example With Get Set Methods

Session Tracking

Cookies Class N Their Methods

Hidden Form Field Techniques

URL Rewriting In Servlet

HttpSession Servlet

Event And Listener

Event And Listener Examples

Filter

Filter In Java

Authentication Filter With Example

FilterConfig With Example

ServletInputStream And ServletOutputStream Class Example.

Project Development In Servlet.

# [JSP](https://kdninfotech.com/advance-java-syllabus/#collapse-d9162c965a900b6191ce)

Basics Of JSP

Life Cycle Of JSP Pages

JSP API In Java

JSP In Netbean

# [Scripting Elements](https://kdninfotech.com/advance-java-syllabus/#collapse-741c4c365a900b6191ce)

Scriptlet Tag In Jsp

Expression Tag In Jsp

Declaration Tag In Jsp

# [9 Implicit Objects](https://kdninfotech.com/advance-java-syllabus/#collapse-74008fd65a900b6191ce)

Out Object

Request Object

Response Object

Config Object

Application Or Context Object

Session Object

PageContext Object

Page Object

Exception Object

# [Directive](https://kdninfotech.com/advance-java-syllabus/#collapse-4c12bcf65a900b6191ce)

Page Directive With Their Attribute

Include Directive

Taglib Directive With Their Attributes

# [Exception Handling](https://kdninfotech.com/advance-java-syllabus/#collapse-cf0d47e65a900b6191ce)

Action Elements.

Jsp:Forward Tag

Jsp:Include Tag

Bean Class Examples

Jsp:UseBean Tag

Jsp:SetProperty & Jsp:GetProperty Tags

# [Expression Language](https://kdninfotech.com/advance-java-syllabus/#collapse-98d68e365a900b6191ce)

Tables Of EL Operators.

# [MVC In JSP](https://kdninfotech.com/advance-java-syllabus/#collapse-d6f6ab065a900b6191ce)

Small MVC Project Using Jsp

# [JSTL](https://kdninfotech.com/advance-java-syllabus/#collapse-38db6ee65a900b6191ce)

Custom Tags

Custom Tag Or How To Create User Defined Tags.

Custom Tag API In Jsp.

Custom Tag Examples.

Attributes In Custom Tags.

Iteration Or Loops And If Else Statement Tag.

Custom URI In Jsp Tag.

# [Introduction Of An Hibernate.](https://kdninfotech.com/advance-java-syllabus/#collapse-63e93dd65a900b6191ce)

Examples Of Hibernate In Projects.

# [Introduction Of An Spring](https://kdninfotech.com/advance-java-syllabus/#collapse-ee3fac165a900b6191ce)

Examples Of Spring In Projects.

# **JDBC Driver**

|  |
| --- |
| JDBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:  JDBC-ODBC bridge driver  Native-API driver (partially java driver)  Network Protocol driver (fully java driver)  Thin driver (fully java driver) |

### **JDBC-ODBC bridge driver**

|  |
| --- |
| The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver. |



#### **In Java 8, the JDBC-ODBC Bridge has been removed.**

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

### **Advantages:**

ADVERTISEMENT

ADVERTISEMENT

easy to use.

can be easily connected to any database.

### **Disadvantages:**

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

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

### **2) Native-API driver**

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



### **Advantage:**

performance upgraded than JDBC-ODBC bridge driver.

### **Disadvantage:**

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

The Vendor client library needs to be installed on client machine.

### **3) Network Protocol driver**

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



### **Advantage:**

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

### **Disadvantages:**

Network support is required on client machine.

Requires database-specific coding to be done in the middle tier.

Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

### **4) Thin driver**

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



### **Advantage:**

Better performance than all other drivers.

No software is required at client side or server side.

### **Disadvantage:**

Drivers depend on the Database.

# Steps To Connect To The All Types Of Database.

# **Java Database Connectivity with 5 Steps**

[5 Steps to connect to the database in java](https://www.javatpoint.com/steps-to-connect-to-the-database-in-java)

[Register the driver class](https://www.javatpoint.com/steps-to-connect-to-the-database-in-java#step1)

[Create the connection object](https://www.javatpoint.com/steps-to-connect-to-the-database-in-java#step2)

[Create the Statement object](https://www.javatpoint.com/steps-to-connect-to-the-database-in-java#step3)

[Execute the query](https://www.javatpoint.com/steps-to-connect-to-the-database-in-java#step4)

[Close the connection object](https://www.javatpoint.com/steps-to-connect-to-the-database-in-java#step5)

|  |
| --- |
| There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:  Register the Driver class  Create connection  Create statement  Execute queries  Close connection |



### **Register the driver class**

|  |
| --- |
| The **forName()** method of Class class is used to register the driver class. This method is used to dynamically load the driver class. |

### **Syntax of forName() method**

**public** **static** **void** forName(String className)**throws** ClassNotFoundException

#### **Note: Since JDBC 4.0, explicitly registering the driver is optional. We just need to put vender's Jar in the classpath, and then JDBC driver manager can detect and load the driver automatically.**

### **Example to register the OracleDriver class**

Here, Java program is loading oracle driver to esteblish database connection.

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

### **2) Create the connection object**

|  |
| --- |
| The **getConnection()** method of DriverManager class is used to establish connection with the database. |

### **Syntax of getConnection() method**

1) **public** **static** Connection getConnection(String url)**throws** SQLException

2) **public** **static** Connection getConnection(String url,String name,String password)

**throws** SQLException

### **Example to establish connection with the Oracle database**

Connection con=DriverManager.getConnection(

"jdbc:oracle:thin:@localhost:1521:xe","system","password");

### **3) Create the Statement object**

|  |
| --- |
| The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database. |

### **Syntax of createStatement() method**

**public** Statement createStatement()**throws** SQLException

### **Example to create the statement object**

Statement stmt=con.createStatement();

### **4) Execute the query**

|  |
| --- |
| The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table. |

### **Syntax of executeQuery() method**

**public** ResultSet executeQuery(String sql)**throws** SQLException

### **Example to execute query**

ResultSet rs=stmt.executeQuery("select \* from emp");

**while**(rs.next()){

System.out.println(rs.getInt(1)+" "+rs.getString(2));

}

### **5) Close the connection object**

|  |
| --- |
| By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection. |

### **Syntax of close() method**

**public** **void** close()**throws** SQLException

### **Example to close connection**

con.close();

# Connectivity With Access Without And With DSN

There are two ways to connect java application with the access database.

Without DSN (Data Source Name)

With DSN

Java is mostly used with Oracle, mysql, or DB2 database. So you can learn this topic only for knowledge.

### **Example to Connect Java Application with access without DSN**

In this example, we are going to connect the java program with the access database. In such case, we have created the login table in the access database. There is only one column in the table named name. Let's get all the name of the login table.

**import** java.sql.\*;

**class** Test{

**public** **static** **void** main(String ar[]){

**try**{

   String database="student.mdb";//Here database exists in the current directory

   String url="jdbc:odbc:Driver={Microsoft Access Driver (\*.mdb)};

                    DBQ=" + database + ";DriverID=22;READONLY=**true**";

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

   Connection c=DriverManager.getConnection(url);

   Statement st=c.createStatement();

   ResultSet rs=st.executeQuery("select \* from login");

**while**(rs.next()){

    System.out.println(rs.getString(1));

   }

}**catch**(Exception ee){System.out.println(ee);}

}}

[download this example](https://static.javatpoint.com/src/jdbc/accesswithoutdsn.zip)

### **Example to Connect Java Application with access with DSN**

Connectivity with type1 driver is not considered good. To connect java application with type1 driver, create DSN first, here we are assuming your dsn name is mydsn.

**import** java.sql.\*;

**class** Test{

**public** **static** **void** main(String ar[]){

**try**{

   String url="jdbc:odbc:mydsn";

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

   Connection c=DriverManager.getConnection(url);

   Statement st=c.createStatement();

   ResultSet rs=st.executeQuery("select \* from login");

**while**(rs.next()){

    System.out.println(rs.getString(1));

   }

}**catch**(Exception ee){System.out.println(ee);}

}}

# DriverManager Class In Jdbc.

# **DriverManager class**

The DriverManager class is the component of JDBC API and also a member of the java.sql package. The DriverManager class acts as an interface between users and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver. It contains all the appropriate methods to register and deregister the database driver class and to create a connection between a Java application and the database. The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method DriverManager.registerDriver(). Note that before interacting with a Database, it is a mandatory process to register the driver; otherwise, an exception is thrown.

## **Methods of the DriverManager Class**

|  |  |
| --- | --- |
| **Method** | **Description** |
| **1) public static synchronized void registerDriver(Driver driver):** | is used to register the given driver with DriverManager. No action is performed by the method when the given driver is already registered. |
| **2) public static synchronized void deregisterDriver(Driver driver):** | is used to deregister the given driver (drop the driver from the list) with DriverManager. If the given driver has been removed from the list, then no action is performed by the method. |
| **3) public static Connection getConnection(String url) throws SQLException:** | is used to establish the connection with the specified url. The SQLException is thrown when the corresponding Driver class of the given database is not registered with the DriverManager. |
| **4) public static Connection getConnection(String url,String userName,String password) throws SQLException:** | is used to establish the connection with the specified url, username, and password. The SQLException is thrown when the corresponding Driver class of the given database is not registered with the DriverManager. |
| **5) public static Driver getDriver(String url)** | Those drivers that understand the mentioned URL (present in the parameter of the method) are returned by this method provided those drivers are mentioned in the list of registered drivers. |
| **6) pubic static int getLoginTimeout()** | The duration of time a driver is allowed to wait in order to establish a connection with the database is returned by this method. |
| **7) pubic static void setLoginTimeout(int sec)** | The method provides the time in seconds. sec mentioned in the parameter is the maximum time that a driver is allowed to wait in order to establish a connection with the database. If 0 is passed in the parameter of this method, the driver will have to wait infinitely while trying to establish the connection with the database. |
| **8) public static Connection getConnection(String URL, Properties prop) throws SQLException** | A connection object is returned by this method after creating a connection to the database present at the mentioned URL, which is the first parameter of this method. The second parameter, which is "prop", fetches the authentication details of the database (username and password.). Similar to the other variation of the getConnection() method, this method also throws the SQLException, when the corresponding Driver class of the given database is not registered with the DriverManager. |

# Connection Interface In Jdbc.

# **Connection interface**

A Connection is a session between a Java application and a database. It helps to establish a connection with the database.

The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData, i.e., an object of Connection can be used to get the object of Statement and DatabaseMetaData. The Connection interface provide many methods for transaction management like commit(), rollback(), setAutoCommit(), setTransactionIsolation(), etc.

#### **By default, connection commits the changes after executing queries.**

### **Commonly used methods of Connection interface:**

**public Statement createStatement():** creates a statement object that can be used to execute SQL queries.

**public Statement createStatement(int resultSetType,int resultSetConcurrency):** Creates a Statement object that will generate ResultSet objects with the given type and concurrency.

**public void setAutoCommit(boolean status):** is used to set the commit status. By default, it is true.

**public void commit():** saves the changes made since the previous commit/rollback is permanent.

**public void rollback():** Drops all changes made since the previous commit/rollback.

**public void close():** closes the connection and Releases a JDBC resources immediately.

## **Connection Interface Fields**

There are some common Connection interface constant fields that are present in the Connect interface. These fields specify the isolation level of a transaction.

**TRANSACTION\_NONE**: No transaction is supported, and it is indicated by this constant.

**TRANSACTION\_READ\_COMMITTED**: It is a constant which shows that the dirty reads are not allowed. However, phantom reads and non-repeatable reads can occur.

**TRANSACTION\_READ\_UNCOMMITTED**: It is a constant which shows that dirty reads, non-repeatable reads, and phantom reads can occur.

**TRANSACTION\_REPEATABLE\_READ**: It is a constant which shows that the non-repeatable reads and dirty reads are not allowed. However, phantom reads and can occur.

**TRANSACTION\_SERIALIZABLE**: It is a constant which shows that the non-repeatable reads, dirty reads as well as the phantom reads are not allowed.

# Statement Interface In Jdbc.

# **Statement interface**

The **Statement interface** provides methods to execute queries with the database. The statement interface is a factory of ResultSet i.e. it provides factory method to get the object of ResultSet.

### **Commonly used methods of Statement interface:**

The important methods of Statement interface are as follows:

|  |
| --- |
| **1) public ResultSet executeQuery(String sql):** is used to execute SELECT query. It returns the object of ResultSet. |
| **2) public int executeUpdate(String sql):** is used to execute specified query, it may be create, drop, insert, update, delete etc. |
| **3) public boolean execute(String sql):** is used to execute queries that may return multiple results. |
| **4) public int[] executeBatch():** is used to execute batch of commands. |

### **Example of Statement interface**

Let’s see the simple example of Statement interface to insert, update and delete the record.

**import** java.sql.\*;

**class** FetchRecord{

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

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

Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

Statement stmt=con.createStatement();

//stmt.executeUpdate("insert into emp765 values(33,'Irfan',50000)");

//int result=stmt.executeUpdate("update emp765 set name='Vimal',salary=10000 where id=33");

**int** result=stmt.executeUpdate("delete from emp765 where id=33");

System.out.println(result+" records affected");

con.close();

}}

# ResultSet Interface Jdbc.

The object of ResultSet maintains a cursor pointing to a row of a table. Initially, cursor points to before the first row.

#### **By default, ResultSet object can be moved forward only and it is not updatable.**

But we can make this object to move forward and backward direction by passing either TYPE\_SCROLL\_INSENSITIVE or TYPE\_SCROLL\_SENSITIVE in createStatement(int,int) method as well as we can make this object as updatable by:

Statement stmt = con.createStatement(ResultSet.TYPE\_SCROLL\_INSENSITIVE,

                     ResultSet.CONCUR\_UPDATABLE);

### **Commonly used methods of ResultSet interface**

|  |  |
| --- | --- |
| **1) public boolean next():** | is used to move the cursor to the one row next from the current position. |
| **2) public boolean previous():** | is used to move the cursor to the one row previous from the current position. |
| **3) public boolean first():** | is used to move the cursor to the first row in result set object. |
| **4) public boolean last():** | is used to move the cursor to the last row in result set object. |
| **5) public boolean absolute(int row):** | is used to move the cursor to the specified row number in the ResultSet object. |
| **6) public boolean relative(int row):** | is used to move the cursor to the relative row number in the ResultSet object, it may be positive or negative. |
| **7) public int getInt(int columnIndex):** | is used to return the data of specified column index of the current row as int. |
| **8) public int getInt(String columnName):** | is used to return the data of specified column name of the current row as int. |
| **9) public String getString(int columnIndex):** | is used to return the data of specified column index of the current row as String. |
| **10) public String getString(String columnName):** | is used to return the data of specified column name of the current row as String. |

### **Example of Scrollable ResultSet**

Let’s see the simple example of ResultSet interface to retrieve the data of 3rd row.

**import** java.sql.\*;

**class** FetchRecord{

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

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

Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

Statement stmt=con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE,ResultSet.CONCUR\_UPDATABLE);

ResultSet rs=stmt.executeQuery("select \* from emp765");

//getting the record of 3rd row

rs.absolute(3);

System.out.println(rs.getString(1)+" "+rs.getString(2)+" "+rs.getString(3));

con.close();

}}

# DatabaseMetaData In Jdbc

# **Java DatabaseMetaData interface**

DatabaseMetaData interface provides methods to get meta data of a database such as database product name, database product version, driver name, name of total number of tables, name of total number of views etc.

## **Commonly used methods of DatabaseMetaData interface**

**public String getDriverName()throws SQLException:**it returns the name of the JDBC driver.

**public String getDriverVersion()throws SQLException:**it returns the version number of the JDBC driver.

**public String getUserName()throws SQLException:**it returns the username of the database.

**public String getDatabaseProductName()throws SQLException:**it returns the product name of the database.

**public String getDatabaseProductVersion()throws SQLException:**it returns the product version of the database.

**public ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)throws SQLException:**it returns the description of the tables of the specified catalog. The table type can be TABLE, VIEW, ALIAS, SYSTEM TABLE, SYNONYM etc.

### **How to get the object of DatabaseMetaData:**

The getMetaData() method of Connection interface returns the object of DatabaseMetaData. Syntax:

**public** DatabaseMetaData getMetaData()**throws** SQLException

### **Simple Example of DatabaseMetaData interface :**

**import** java.sql.\*;

**class** Dbmd{

**public** **static** **void** main(String args[]){

**try**{

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

Connection con=DriverManager.getConnection(

"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

DatabaseMetaData dbmd=con.getMetaData();

System.out.println("Driver Name: "+dbmd.getDriverName());

System.out.println("Driver Version: "+dbmd.getDriverVersion());

System.out.println("UserName: "+dbmd.getUserName());

System.out.println("Database Product Name: "+dbmd.getDatabaseProductName());

System.out.println("Database Product Version: "+dbmd.getDatabaseProductVersion());

con.close();

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

}

}

Output:Driver Name: Oracle JDBC Driver

Driver Version: 10.2.0.1.0XE

Database Product Name: Oracle

Database Product Version: Oracle Database 10g Express Edition

Release 10.2.0.1.0 -Production

### **Example of DatabaseMetaData interface that prints total number of tables :**

**import** java.sql.\*;

**class** Dbmd2{

**public** **static** **void** main(String args[]){

**try**{

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

Connection con=DriverManager.getConnection(

"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

DatabaseMetaData dbmd=con.getMetaData();

String table[]={"TABLE"};

ResultSet rs=dbmd.getTables(**null**,**null**,**null**,table);

**while**(rs.next()){

System.out.println(rs.getString(3));

}

con.close();

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

}

}

### **Example of DatabaseMetaData interface that prints total number of views :**

**import** java.sql.\*;

**class** Dbmd3{

**public** **static** **void** main(String args[]){

**try**{

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

Connection con=DriverManager.getConnection(

"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

DatabaseMetaData dbmd=con.getMetaData();

String table[]={"VIEW"};

ResultSet rs=dbmd.getTables(**null**,**null**,**null**,table);

**while**(rs.next()){

System.out.println(rs.getString(3));

}

con.close();

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

}

}

# Storing Image In Jdbc.

# **Example to store image in Oracle database**

You can store images in the database in java by the help of **PreparedStatement** interface.

The **setBinaryStream()** method of PreparedStatement is used to set Binary information into the parameterIndex.

### **Signature of setBinaryStream method**

The syntax of setBinaryStream() method is given below:

1) **public** **void** setBinaryStream(**int** paramIndex,InputStream stream)

**throws** SQLException

2) **public** **void** setBinaryStream(**int** paramIndex,InputStream stream,**long** length)

**throws** SQLException

For storing image into the database, BLOB (Binary Large Object) datatype is used in the table. For example:

CREATE TABLE  "IMGTABLE"

   (    "NAME" VARCHAR2(4000),

    "PHOTO" BLOB

   )

/

Let's write the jdbc code to store the image in the database. Here we are using d:\\d.jpg for the location of image. You can change it according to the image location.

## **Java Example to store image in the database**

**import** java.sql.\*;

**import** java.io.\*;

**public** **class** InsertImage {

**public** **static** **void** main(String[] args) {

**try**{

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

Connection con=DriverManager.getConnection(

"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

PreparedStatement ps=con.prepareStatement("insert into imgtable values(?,?)");

ps.setString(1,"sonoo");

FileInputStream fin=**new** FileInputStream("d:\\g.jpg");

ps.setBinaryStream(2,fin,fin.available());

**int** i=ps.executeUpdate();

System.out.println(i+" records affected");

con.close();

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

}

}

If you see the table, record is stored in the database but image will not be shown. To do so, you need to retrieve the image from the database which we are covering in the next page.

# Retrieving Image In Jdbc

# **Example to retrieve image from Oracle database**

By the help of **PreparedStatement** we can retrieve and store the image in the database.

The **getBlob()** method of PreparedStatement is used to get Binary information, it returns the instance of Blob. After calling the **getBytes()** method on the blob object, we can get the array of binary information that can be written into the image file.

### **Signature of getBlob() method of PreparedStatement**

**public** Blob getBlob()**throws** SQLException

### **Signature of getBytes() method of Blob interface**

**public**  **byte**[] getBytes(**long** pos, **int** length)**throws** SQLException

We are assuming that image is stored in the imgtable.

CREATE TABLE  "IMGTABLE"

   (    "NAME" VARCHAR2(4000),

    "PHOTO" BLOB

   )

/

Now let's write the code to retrieve the image from the database and write it into the directory so that it can be displayed.

In AWT, it can be displayed by the Toolkit class. In servlet, jsp, or html it can be displayed by the img tag.

**import** java.sql.\*;

**import** java.io.\*;

**public** **class** RetrieveImage {

**public** **static** **void** main(String[] args) {

**try**{

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

Connection con=DriverManager.getConnection(

"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

PreparedStatement ps=con.prepareStatement("select \* from imgtable");

ResultSet rs=ps.executeQuery();

**if**(rs.next()){//now on 1st row

Blob b=rs.getBlob(2);//2 means 2nd column data

**byte** barr[]=b.getBytes(1,(**int**)b.length());//1 means first image

FileOutputStream fout=**new** FileOutputStream("d:\\sonoo.jpg");

fout.write(barr);

fout.close();

}//end of if

System.out.println("ok");

con.close();

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

}

}

Now if you see the d drive, sonoo.jpg image is created.

# Storing File And Retrieving File Using Jdbc.

The **ResultSet** interface provides the methods named **getClob()** and **getCharacterStream()** to retrieve **Clob** datatype, In which the contents of a file are typically stored.

These methods accept an integer representing the index of the column (or, a String value representing the name of the column) and retrieves the value at the specified column.

The difference is the getClob() method returns a Clob object and the getCgaracterStream() method returns a Reader object which holds the contents of the Clob datatype.

### **Example**

Assume we have created a table named Articles in the database with the following description.

+---------+--------------+------+-----+---------+-------+

| Field   | Type         | Null | Key | Default | Extra |

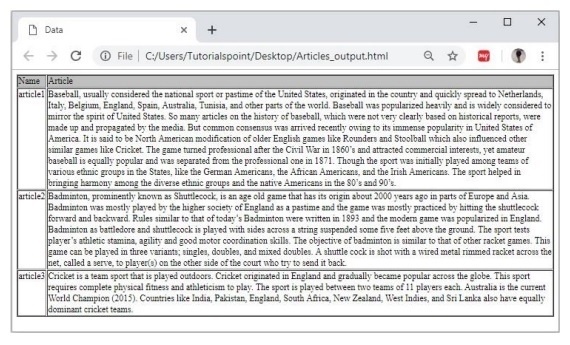
+---------+--------------+------+-----+---------+-------+

| Name    | varchar(255) | YES | | NULL | |

| Article | longtext | YES | | NULL | |

+---------+--------------+------+-----+---------+-------+

And, we have inserted three articles in it with names article 1, article 2 and, article 3 as shown below:



## **Example**

Following program retrieves the contents of the table Articles using the getString() and getClob() methods and saves it in the specified files.

import java.io.FileWriter;

import java.io.Reader;

import java.sql.Clob;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

public class RetrievingFileFromDatabase {

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

//Registering the Driver

DriverManager.registerDriver(new com.mysql.jdbc.Driver());

//Getting the connection

String mysqlUrl = "jdbc:mysql://localhost/sampleDB";

Connection con = DriverManager.getConnection(mysqlUrl, "root", "password");

System.out.println("Connection established......");

//Creating aStatement

Statement stmt = con.createStatement();

//Retrieving the data

ResultSet rs = stmt.executeQuery("select \* from Articles");

int j = 0;

System.out.println("Contents of the table are: ");

while(rs.next()) {

System.out.println(rs.getString("Name"));

Clob clob = rs.getClob("Article");

Reader reader = clob.getCharacterStream();

String filePath = "E:\Data\clob\_output"+j+".txt";

FileWriter writer = new FileWriter(filePath);

int i;

while ((i = reader.read())!=-1) {

writer.write(i);

}

writer.close();

System.out.println(filePath);

j++;

}

}

}

### **Output**

Connection established......

Contents of the table are:

article1

E:\Data\clob\_output0.txt

article2

E:\Data\clob\_output1.txt

article3

E:\Data\clob\_output2.txt

# **Store File In Database Using JDBC**

PreparedStatement provides the facility to store and retrieve the file in the database using JDBC.

## **PreparedStatement methods to store file:**

**1. public void setBinaryStream(int paramIndex,InputStream stream)   throws SQLException**

**2. public void setBinaryStream(int paramIndex,InputStream stream,long length)  throws SQLException**

## **Example:**

**JDBCTest.java**

|  |
| --- |
| **import** java.io.File;  **import** java.io.FileReader;  **import** java.sql.Connection;  **import** java.sql.PreparedStatement;  **import** com.w3spoint.util.JDBCUtil;  ***/\*\****  ***This class is used to store a file in DB.***  ***@author w3spoint***  ***\*/***  **public** **class** JDBCTest {  **public** **static** **void** main(String args[]){  Connection conn = **null**;  PreparedStatement preparedStatement = **null**;  String createTableQuery = "create table FILESTORE("  + "FILE\_ID NUMBER(5) NOT NULL, "  + "NAME CLOB NOT NULL, "  + "PRIMARY KEY (FILE\_ID) )";  **try**{  *//get connection*  conn = JDBCUtil.getConnection();  *//create preparedStatement*  preparedStatement =  conn.prepareStatement(createTableQuery);  *//execute query for create table*  preparedStatement.execute();  System.out.println("Table created successfully.");  String storeFileQuery = "insert into FILESTORE "  + "values (?,?)";  preparedStatement =  conn.prepareStatement(storeFileQuery);  *//Read source file*  File file = **new** File("F:**\\**test.txt");  FileReader fileReader = **new** FileReader(file);  preparedStatement.setInt(1,2);  preparedStatement.setCharacterStream(2,  fileReader,(**int**)file.length());  preparedStatement.executeUpdate();  System.out.println("File stored successfully.");  *//close connection*  preparedStatement.close();  conn.close();  }**catch**(Exception e){  e.printStackTrace();  }  }  } |

**JDBCUtil.java**

|  |
| --- |
| **import** java.sql.Connection;  **import** java.sql.DriverManager;  ***/\*\****  ***This is a utility class for JDBC connection.***  ***@author w3spoint***  ***\*/***  **public** **class** JDBCUtil {  *//JDBC and database properties.*  **private** **static** **final** String DB\_DRIVER =  "oracle.jdbc.driver.OracleDriver";  **private** **static** **final** String DB\_URL =  "jdbc:oracle:thin:@localhost:1521:XE";  **private** **static** **final** String DB\_USERNAME = "system";  **private** **static** **final** String DB\_PASSWORD = "oracle";  **public** **static** Connection getConnection(){  Connection conn = **null**;  **try**{  *//Register the JDBC driver*  **Class**.forName(DB\_DRIVER);  *//Open the connection*  conn = DriverManager.  getConnection(DB\_URL, DB\_USERNAME, DB\_PASSWORD);  **if**(conn != **null**){  System.out.println("Successfully connected.");  }**else**{  System.out.println("Failed to connect.");  }  }**catch**(Exception e){  e.printStackTrace();  }  **return** conn;  }  } |

## **Output:**

|  |
| --- |
| Successfully connected.  Table created successfully.  File stored successfully. |

# Stored Procedures And Functions In Plsql Using Java.

# **PL/SQL Procedure**

The PL/SQL stored procedure or simply a procedure is a PL/SQL block which performs one or more specific tasks. It is just like procedures in other programming languages.

The procedure contains a header and a body.

**Header:** The header contains the name of the procedure and the parameters or variables passed to the procedure.

**Body:** The body contains a declaration section, execution section and exception section similar to a general PL/SQL block.

## **How to pass parameters in procedure:**

When you want to create a procedure or function, you have to define parameters .There is three ways to pass parameters in procedure:

**IN parameters:**The IN parameter can be referenced by the procedure or function. The value of the parameter cannot be overwritten by the procedure or the function.

**OUT parameters:**The OUT parameter cannot be referenced by the procedure or function, but the value of the parameter can be overwritten by the procedure or function.

**INOUT parameters:**The INOUT parameter can be referenced by the procedure or function and the value of the parameter can be overwritten by the procedure or function.

#### **A procedure may or may not return any value.**

## **PL/SQL Create Procedure**

**Syntax for creating procedure:**

**CREATE** [OR REPLACE] **PROCEDURE** procedure\_name

    [ (parameter [,parameter]) ]

**IS**

    [declaration\_section]

**BEGIN**

    executable\_section

[EXCEPTION

    exception\_section]

**END** [procedure\_name];

## **Create procedure example**

In this example, we are going to insert record in user table. So you need to create user table first.

**Table creation:**

**create** **table** user(id number(10) **primary** **key**,**name** varchar2(100));

Now write the procedure code to insert record in user table.

**Procedure Code:**

**create** or replace **procedure** "INSERTUSER"

(id IN NUMBER,

**name** IN VARCHAR2)

**is**

**begin**

**insert** **into** user **values**(id,**name**);

**end**;

/

Output:

Procedure created.

## **PL/SQL program to call procedure**

Let's see the code to call above created procedure.

**BEGIN**

   insertuser(101,'Rahul');

   dbms\_output.put\_line('record inserted successfully');

**END**;

/

Now, see the "USER" table, you will see one record is inserted.

|  |  |
| --- | --- |
| **ID** | **Name** |
| 101 | Rahul |

## **PL/SQL Drop Procedure**

**Syntax for drop procedure**

**DROP** **PROCEDURE** procedure\_name;

## **Example of drop procedure**

**DROP** **PROCEDURE** pro1;

# Transaction Management Tcl Commands.

# **TCL Commands in SQL**

In SQL, TCL stands for **Transaction control language**.

A single unit of work in a database is formed after the consecutive execution of commands is known as a transaction.

There are certain commands present in SQL known as TCL commands that help the user manage the transactions that take place in a database.

**COMMIT. ROLLBACK** and **SAVEPOINT** are the most commonly used TCL commands in SQL.

Now let us take a deeper dive into the TCL commands of SQL with the help of examples. All the queries in the examples will be written using the MySQL database.

### **COMMIT**

COMMIT command in SQL is used to save all the transaction-related changes permanently to the disk. Whenever DDL commands such as INSERT, UPDATE and DELETE are used, the changes made by these commands are permanent only after closing the current session. So before closing the session, one can easily roll back the changes made by the DDL commands. Hence, if we want the changes to be saved permanently to the disk without closing the session, we will use the commit command.

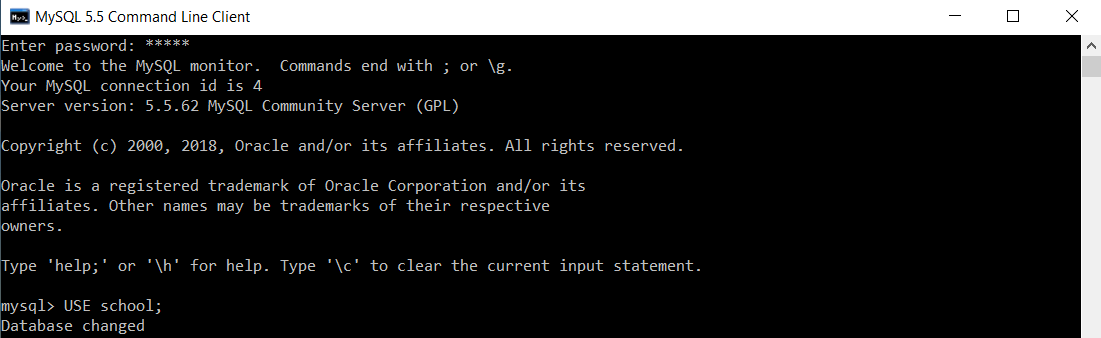
**Syntax:**

**COMMIT**;

**Example:**

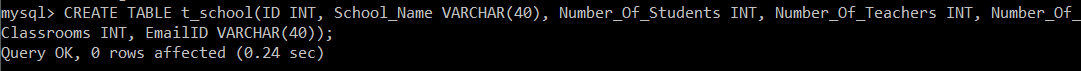
We will select an existing database, i.e., school.

mysql> USE school;



To create a table named t\_school, we will execute the following query:

mysql> **CREATE** **TABLE** t\_school(ID **INT**, School\_Name **VARCHAR**(40), Number\_Of\_Students **INT**, Number\_Of\_Teachers **INT**, Number\_Of\_Classrooms **INT**, EmailID **VARCHAR**(40));



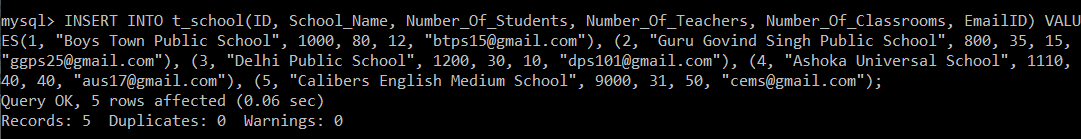
BEGIN / START TRANSACTION command is used to start the transaction.

mysql> START **TRANSACTION**;

TCL Commands in SQL

Now, we will execute the following query to insert multiple records at the same time in the t\_school table.

mysql> **INSERT** **INTO** t\_school(ID, School\_Name, Number\_Of\_Students, Number\_Of\_Teachers, Number\_Of\_Classrooms, EmailID) **VALUES**(1, "Boys Town Public School", 1000, 80, 12, "btps15@gmail.com"), (2, "Guru Govind Singh Public School", 800, 35, 15, "ggps25@gmail.com"), (3, "Delhi Public School", 1200, 30, 10, "dps101@gmail.com"), (4, "Ashoka Universal School", 1110, 40, 40, "aus17@gmail.com"), (5, "Calibers English Medium School", 9000, 31, 50, "cems@gmail.com");



We will now execute the SELECT query to verify the execution of the INSERT INTO query executed above.

mysql> **SELECT** \***FROM** t\_school;

After executing the SELECT query on the t\_school table, you will get the following output:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **School\_Name** | **Number\_Of\_Students** | **Number\_Of\_Teachers** | **Number\_Of\_Classrooms** | **EmailID** |
| 1 | Boys Town Public School | 1000 | 80 | 12 | btps15@gmail.com |
| 2 | Guru Govind Singh Public School | 800 | 35 | 15 | ggps25@gmail.com |
| 3 | Delhi Public School | 1200 | 30 | 10 | dps101@gmail.com |
| 4 | Ashoka Universal School | 1110 | 40 | 40 | aus17@gmail.com |
| 5 | Calibers English Medium School | 9000 | 31 | 50 | cems@gmail.com |

The output of the SELECT query shows that all the records are inserted successfully.

We will execute the COMMIT command to save the results of the operations carried on the t\_school table.

mysql> **COMMIT**;

TCL Commands in SQL

Autocommit is by default enabled in MySQL. To turn it off, we will set the value of autocommit as 0.

mysql> **SET** autocommit = 0;

TCL Commands in SQL

MySQL, by default, commits every query the user executes. But if the user wishes to commit only the specific queries instead of committing every query, then turning off the autocommit is useful.

### **2. SAVEPOINT**

We can divide the database operations into parts. For example, we can consider all the insert related queries that we will execute consecutively as one part of the transaction and the delete command as the other part of the transaction. Using the SAVEPOINT command in SQL, we can save these different parts of the same transaction using different names. **For example**, we can save all the insert related queries with the savepoint named INS. To save all the insert related queries in one savepoint, we have to execute the SAVEPOINT query followed by the savepoint name after finishing the insert command execution.

**Syntax:**

SAVEPOINT savepoint\_name;

### **3. ROLLBACK**

While carrying a transaction, we must create savepoints to save different parts of the transaction. According to the user's changing requirements, he/she can roll back the transaction to different savepoints. Consider a scenario: We have initiated a transaction followed by the table creation and record insertion into the table. After inserting records, we have created a savepoint INS. Then we executed a delete query, but later we thought that mistakenly we had removed the useful record. Therefore in such situations, we have an option of rolling back our transaction. In this case, we have to roll back our transaction using the ROLLBACK command to the savepoint INS, which we have created before executing the DELETE query.

**Syntax:**

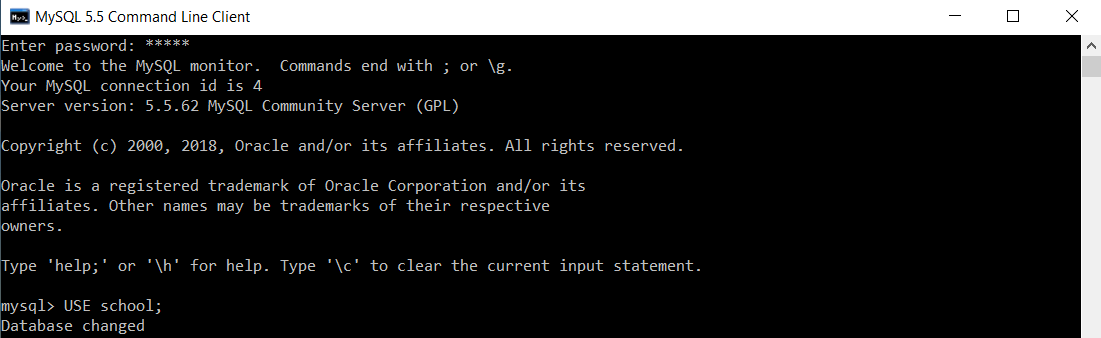
**ROLLBACK** **TO** savepoint\_name;

**Examples to understand the SAVEPOINT and ROLLBACK commands:**

**Example 1:**

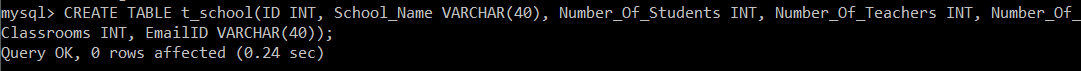
We will select an existing database, i.e., school.

mysql> USE school;



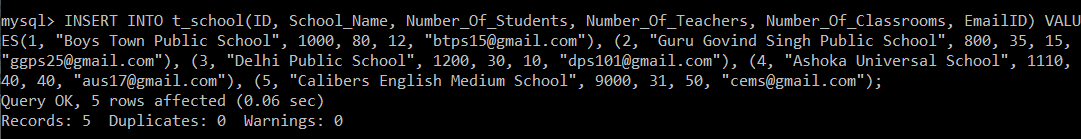
To create a table named t\_school, we will execute the following query:

mysql> **CREATE** **TABLE** t\_school(ID **INT**, School\_Name **VARCHAR**(40), Number\_Of\_Students **INT**, Number\_Of\_Teachers **INT**, Number\_Of\_Classrooms **INT**, EmailID **VARCHAR**(40));



Now, we will execute the following query to insert multiple records at the same time in the t\_school table.

mysql> **INSERT** **INTO** t\_school(ID, School\_Name, Number\_Of\_Students, Number\_Of\_Teachers, Number\_Of\_Classrooms, EmailID) **VALUES**(1, "Boys Town Public School", 1000, 80, 12, "btps15@gmail.com"), (2, "Guru Govind Singh Public School", 800, 35, 15, "ggps25@gmail.com"), (3, "Delhi Public School", 1200, 30, 10, "dps101@gmail.com"), (4, "Ashoka Universal School", 1110, 40, 40, "aus17@gmail.com"), (5, "Calibers English Medium School", 9000, 31, 50, "cems@gmail.com");



We will now execute the SELECT query to verify the execution of the INSERT INTO query executed above.

mysql> **SELECT** \***FROM** t\_school;

After executing the SELECT query on the t\_school table, you will get the following output:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **School\_Name** | **Number\_Of\_Students** | **Number\_Of\_Teachers** | **Number\_Of\_Classrooms** | **EmailID** |
| 1 | Boys Town Public School 1000 | 80 | 12 | btps15@gmail.com |  |
| 2 | Guru Govind Singh Public School | 800 | 35 | 15 | ggps25@gmail.com |
| 3 | Delhi Public School | 1200 | 30 | 10 | dps101@gmail.com |
| 4 | Ashoka Universal School | 1110 | 40 | 40 | aus17@gmail.com |
| 5 | Calibers English Medium School | 9000 | 31 | 50 | cems@gmail.com |

The output of the SELECT query shows that all the records are inserted successfully.

BEGIN / START TRANSACTION command is used to start the transaction.

mysql> START **TRANSACTION**;

TCL Commands in SQL

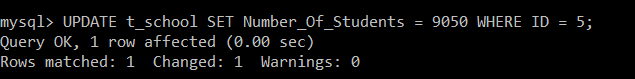
As we know, the SAVEPOINT command in SQL is used to save the different parts of the same transaction using different names. Consider till this point as one part of our transaction. We will save this part using a savepoint named Insertion.

mysql> SAVEPOINT Insertion;

TCL Commands in SQL

Now, we will execute the update command on the t\_school table to set the Number\_Of\_Students as 9050 for the record with ID 5.

mysql> **UPDATE** t\_school **SET** Number\_Of\_Students = 9050 **WHERE** ID = 5;



To verify that the record with ID 5 now has the Number\_Of\_Students as 9050, we will execute the SELECT query.

mysql> **SELECT** \***FROM** t\_school;

After executing the SELECT query on the t\_school table, you will get the following output:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **School\_Name** | **Number\_Of\_Students** | **Number\_Of\_Teachers** | **Number\_Of\_Classrooms** | **EmailID** |
| 1 | Boys Town Public School | 1000 | 80 | 12 | btps15@gmail.com |
| 2 | Guru Govind Singh Public School | 800 | 35 | 15 | ggps25@gmail.com |
| 3 | Delhi Public School | 1200 | 30 | 10 | dps101@gmail.com |
| 4 | Ashoka Universal School | 1110 | 40 | 40 | aus17@gmail.com |
| 5 | Calibers English Medium School | 9050 | 31 | 50 | cems@gmail.com |

The output of the SELECT query shows that the record with ID 5 is updated successfully.

Consider the update operation as one part of our transaction. We will save this part using a savepoint named Updation.

mysql> SAVEPOINT Updation;

TCL Commands in SQL

Suddenly, our requirement changed, and we realized that we had updated a record that was not supposed to be. In such a scenario, we need to roll back our transaction to the savepoint, which was created prior to the execution of the UPDATE command.

mysql> **ROLLBACK** **TO** Insertion;

TCL Commands in SQL

We didn't need the updation carried on the record. Hence, we have rolled back to the savepoint named Insertion.

For confirming that we have got the same t\_school table that we had before carrying out the updation operation, we will again execute the SELECT query.

mysql> **SELECT** \***FROM** t\_school;

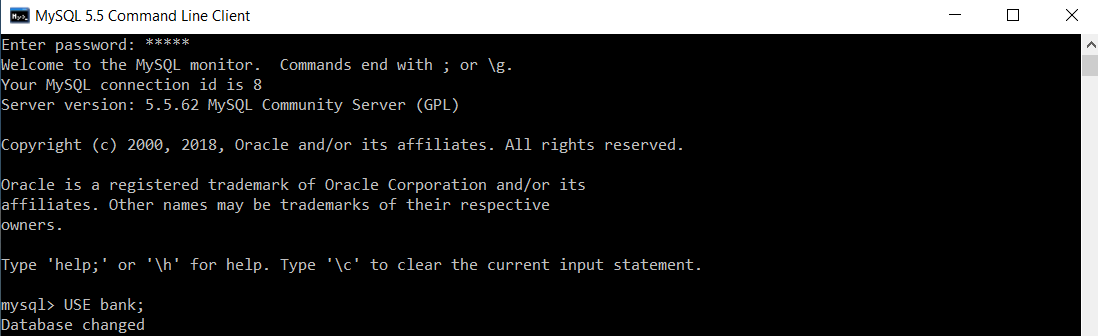
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **School\_Name** | **Number\_Of\_Students** | **Number\_Of\_Teachers** | **Number\_Of\_Classrooms** | **EmailID** |
| 1 | Boys Town Public School | 1000 | 80 | 12 | btps15@gmail.comm |
| 2 | Guru Govind Singh Public School | 800 | 35 | 15 | ggps25@gmail.comm |
| 3 | Delhi Public School | 1200 | 30 | 10 | dps101@gmail.comm |
| 4 | Ashoka Universal School | 1110 | 40 | 40 | aus17@gmail.comm |
| 5 | Calibers English Medium School | 9000 | 31 | 50 | cems@gmail.com |

The SELECT query output confirms that the transaction is now successfully rolled back to the savepoint 'Insertion'.

**Example 2:**

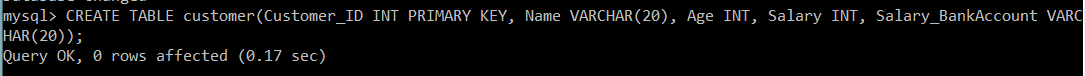
We will select an existing database, i.e., bank.

mysql> USE bank;



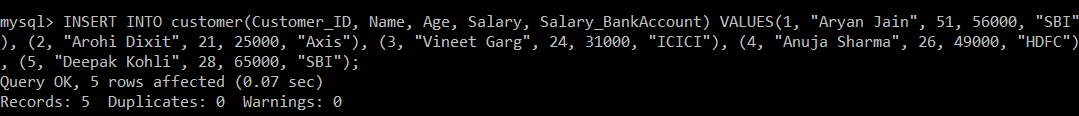
To create a table named customer, we will execute the following query:

mysql> **CREATE** **TABLE** customer(Customer\_ID **INT** **PRIMARY** **KEY**, **Name** **VARCHAR**(20), Age **INT**, Salary **INT**, Salary\_BankAccount **VARCHAR**(20));



Now, we will execute the following query to insert multiple records at the same time in the customer table.

mysql> **INSERT** **INTO** customer(Customer\_ID, **Name**, Age, Salary, Salary\_BankAccount) **VALUES**(1, "Aryan Jain", 51, 56000, "SBI"), (2, "Arohi Dixit", 21, 25000, "Axis"), (3, "Vineet Garg", 24, 31000, "ICICI"), (4, "Anuja Sharma", 26, 49000, "HDFC"), (5, "Deepak Kohli", 28, 65000, "SBI");



We will now execute the SELECT query to verify the execution of the INSERT INTO query executed above.

mysql> **SELECT** \***FROM** customer;

After executing the SELECT query on the t\_school table, you will get the following output:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Customer\_ID** | **Name** | **Age** | **Salary** | **Salary\_BankAccount** |
| 1 | Aryan Jain | 51 | 56000 | SBI |
| 2 | Arohi Dixit | 21 | 25000 | Axis |
| 3 | Vineet Garg | 24 | 31000 | ICICI |
| 4 | Anuja Sharma | 26 | 49000 | HDFC |
| 5 | Deepak Kohli | 28 | 65000 | SBI |

The output of the SELECT query shows that all the records are inserted successfully.

BEGIN / START TRANSACTION command is used to start the transaction.

mysql> START **TRANSACTION**;

TCL Commands in SQL

As we know, the SAVEPOINT command in SQL is used to save the different parts of the same transaction using different names. Consider till this point as one part of our transaction. We will save this part using a savepoint named Insertion.

mysql> SAVEPOINT Insertion;

TCL Commands in SQL

We will execute the delete command on the customer table to remove the record with ID 5.

mysql> **DELETE** **FROM** customer **WHERE** Customer\_ID = 5;

TCL Commands in SQL

We will execute the SELECT query to verify that the record with ID 5 has been removed.

mysql> **SELECT** \***FROM** customer;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Customer\_ID** | **Name** | **Age** | **Salary** | **Salary\_BankAccount** |
| 1 | Aryan Jain | 51 | 56000 | SBI |
| 2 | Arohi Dixit | 21 | 25000 | Axis |
| 3 | Vineet Garg | 24 | 31000 | ICICI |
| 4 | Anuja Sharma | 26 | 49000 | HDFC |

The output of the SELECT query shows that the record with ID 5 is removed successfully.

Consider the delete operation as one part of our transaction. We will save this part using a savepoint named Deletion.

mysql> SAVEPOINT Deletion;

TCL Commands in SQL

Suddenly, our requirement changed, and we realized that we had deleted a record that was not supposed to be. In such a scenario, we need to roll back our transaction to the savepoint, which was created prior to the execution of the DELETE command.

mysql> **ROLLBACK** **TO** Insertion;

We didn't need the deletion carried on the record. Hence, we have rolled back to the savepoint named Insertion.

For confirming that we have got the same customer table that we had before carrying out the deletion operation, we will again execute the SELECT query.

mysql> **SELECT** \***FROM** customer;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Customer\_ID** | **Name** | **Age** | **Salary** | **Salary\_BankAccount** |
| 1 | Aryan Jain | 51 | 56000 | SBI |
| 2 | Arohi Dixit | 21 | 25000 | Axis |
| 3 | Vineet Garg | 24 | 31000 | ICICI |
| 4 | Anuja Sharma | 26 | 49000 | HDFC |
| 5 | Deepak Kohli | 28 | 65000 | SBI |

The SELECT query output confirms that the transaction is now successfully rolled back to the savepoint 'Insertion'.

# JDBC New Features In Java.

# **Java 8 JDBC Improvements**

In Java 8, Java made two major changes in JDBC API.

#### **The JDBC-ODBC Bridge has been removed.**

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

#### **2) Added some new features in JDBC 4.2.**

Java JDBC 4.2 introduces the following features:

ADVERTISEMENT

ADVERTISEMENT

Addition of REF\_CURSOR support.

Addition of java.sql.DriverAction Interface

Addition of security check on deregisterDriver Method in DriverManager Class

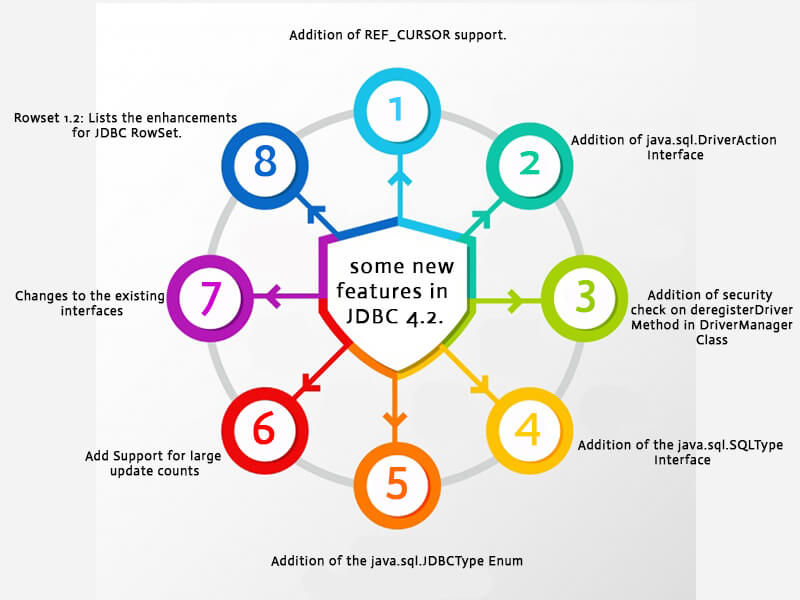
Addition of the java.sql.SQLType Interface

Addition of the java.sql.JDBCType Enum

Add Support for large update counts

Changes to the existing interfaces

Rowset 1.2: Lists the enhancements for JDBC RowSet.



## **Java JDBC DriverAction**

It is an interface that must be implemented when a Driver wants to be notified by DriverManager. It is added in java.sql package and contains only one abstract method.

Backward Skip 10sPlay VideoForward Skip 10s

### **DriverAction Method**

|  |  |
| --- | --- |
| **Method** | **Description** |
| void deregister() | This method called by DriverManager.deregisterDriver(Driver) to notify the JDBC driver that it was de-registered. |

The deregister method is intended only to be used by JDBC Drivers and not by applications.

JDBC drivers are recommended not to implement the DriverAction in a public class.

If there are active connections to the database at the time that the deregister method is called, it is implementation specific as to whether the connections are closed or allowed to continue. Once this method is called, it is implementation specific as to whether the driver may limit the ability to create new connections to the database, invoke other Driver methods or throw a SQLException.

## **Java JDBC4.2 DriverAction Example**

**import** java.sql.\*;

// implementing DriverAction interface

**class** JdbcExample **implements** DriverAction{

    // implementing deregister method of DriverAction interface

    @Override

**public** **void** deregister() {

        System.out.println("Driver deregistered");

    }

**public** **static** **void** main(String args[]){

**try**{

            // Creating driver instance

            Driver driver = **new** com.mysql.jdbc.Driver();

            // Creating Action Driver

            DriverAction da = **new** JdbcExample();

            // Registering driver by passing driver and driverAction

            DriverManager.registerDriver(driver, da);

            // Creating connection

            Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/student","root","mysql");

            //Here student is database name, root is username and password is mysql

            Statement stmt=con.createStatement();

            // Executing SQL query

            ResultSet rs=stmt.executeQuery("select \* from user");

**while**(rs.next()){

                System.out.println(rs.getInt(1)+""+rs.getString(2)+""+rs.getString(3));

            }

            // Closing connection

            con.close();

            // Calling deregisterDriver method

            DriverManager.deregisterDriver(driver);

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

    }

}

Output:

Arun 25

irfan 22

Neraj kumar 25

Driver deregistered

## **Java JDBC SQLType**

This interface is used to identify a generic SQL type, JDBC type or a vendor specific data type.

It provides following methods.

|  |  |
| --- | --- |
| **Method** | **Description** |
| String getName() | It returns the SQLType name that represents a SQL data type. |
| String getVendor() | It returns the name of the vendor that supports this data type. The value returned typically is the package name for this vendor. |
| Integer getVendorTypeNumber() | It returns the vendor specific type number for the data type. |

## **Java JDBCType**

It is an Enumeration which defines the constants that are used to identify generic SQL types, called JDBC types. It extends java.lang.Enum and implements java.sql.SQLType.

## **JDBCType Fields**

The following table contains constants defined in the JDBCType.

|  |  |
| --- | --- |
| **Enum constant** | **Description** |
| public static final JDBCType ARRAY | It identifies the generic SQL type ARRAY. |
| public static final JDBCType BIGINT | It identifies the generic SQL type BIGINT. |
| public static final JDBCType BIT | It identifies the generic SQL type BIT. |
| public static final JDBCType BLOB | It identifies the generic SQL type BLOB. |
| public static final JDBCType BOOLEAN | It identifies the generic SQL type BOOLEAN. |
| public static final JDBCType CHAR | It identifies the generic SQL type CHAR. |
| public static final JDBCType CLOB | It identifies the generic SQL type CLOB. |
| public static final JDBCType DATALINK | It identifies the generic SQL type DATALINK. |
| public static final JDBCType DATE | It identifies the generic SQL type DATE. |
| public static final JDBCType DECIMAL | It identifies the generic SQL type DECIMAL. |
| public static final JDBCType DISTINCT | It identifies the generic SQL type DISTINCT. |
| public static final JDBCType DOUBLE | It identifies the generic SQL type DOUBLE. |
| public static final JDBCType FLOAT | It identifies the generic SQL type FLOAT. |
| public static final JDBCType INTEGER | It identifies the generic SQL type INTEGER. |
| public static final JDBCType JAVA\_OBJECT | It indicates that the SQL type is database-specific and gets mapped to a Java object that can be accessed via the methods getObject and setObject. |
| Public static final JDBCType LONGNVARCHAR | It identifies the generic SQL type LONGNVARCHAR. |
| public static final JDBCType NCHAR | It identifies the generic SQL type NCHAR. |
| public static final JDBCType NCLOB | It identifies the generic SQL type NCLOB. |
| public static final JDBCType NULL | It identifies the generic SQL value NULL. |
| public static final JDBCType NUMERIC | It identifies the generic SQL type NUMERIC. |
| public static final JDBCType NVARCHAR | It identifies the generic SQL type NVARCHAR. |
| public static final JDBCType OTHER | It indicates that the SQL type is database-specific and gets mapped to a Java object that can be accessed via the methods getObject and setObject. |
| public static final JDBCType REAL | It identifies the generic SQL type REAL.Identifies the generic SQL type VARCHAR. |
| public static final JDBCType REF | It identifies the generic SQL type REF. |
| public static final JDBCType REF\_CURSOR | It identifies the generic SQL type REF\_CURSOR. |
| public static final JDBCType ROWID | It identifies the SQL type ROWID. |
| public static final JDBCType SMALLINT | It identifies the generic SQL type SMALLINT. |
| public static final JDBCType SQLXML | It identifies the generic SQL type SQLXML. |
| public static final JDBCType STRUCT | It identifies the generic SQL type STRUCT. |
| public static final JDBCType TIME | It identifies the generic SQL type TIME. |
| public static final JDBCType TIME\_WITH\_TIMEZONE | It identifies the generic SQL type TIME\_WITH\_TIMEZONE. |
| public static final JDBCType TIMESTAMP | It identifies the generic SQL type TIMESTAMP. |
| public static final JDBCType TIMESTAMP\_WITH\_TIMEZONE | It identifies the generic SQL type TIMESTAMP\_WITH\_TIMEZONE. |
| public static final JDBCType TINYINT | It identifies the generic SQL type TINYINT. |
| public static final JDBCType VARBINARY | It identifies the generic SQL type VARBINARY. |
| public static final JDBCType VARCHAR | It identifies the generic SQL type VARCHAR. |

## **JDBCType Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| public String getName() | It returns the SQLType name that represents a SQL data type. |
| public String getVendor() | It returns the name of the vendor that supports this data type. |
| public Integer getVendorTypeNumber() | It returns the vendor specific type number for the data type. |
| public static JDBCType valueOf(int type) | It returns the JDBCType that corresponds to the specified Types value. It throws IllegalArgumentException, if this enum type has no constant with the specified Types value. |
| public static JDBCType valueOf(String name) | It returns the enum constant of this type with the specified name. The string must match exactly an identifier used to declare an enum constant in this type. It throws IllegalArgumentException, if this enum type has no constant with the specified name. It throws NullPointerException, if the argument is null. |
| public static JDBCType[] values() | It returns an array containing the constants of this enum type, in the order they are declared. This method may be used to iterate over the constants. |

# Servlet In Java

# **Servlets | Servlet Tutorial**

**Servlet** technology is used to create a web application (resides at server side and generates a dynamic web page).

**Servlet** technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language. However, there were many disadvantages to this technology. We have discussed these disadvantages below.

There are many interfaces and classes in the Servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse, etc.

## **What is a Servlet?**

Servlet can be described in many ways, depending on the context.

Servlet is a technology which is used to create a web application.

Servlet is an API that provides many interfaces and classes including documentation.

Servlet is an interface that must be implemented for creating any Servlet.

Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.

Servlet is a web component that is deployed on the server to create a dynamic web page.



Do You Know?

What is the web application and what is the difference between Get and Post request?

What information is received by the web server if we request for a Servlet?

How to run servlet in Eclipse, MyEclipse and Netbeans IDE?

What are the ways for servlet collaboration and what is the difference between RequestDispatcher and sendRedirect() method?

What is the difference between ServletConfig and ServletContext interface?

How many ways can we maintain the state of a user? Which approach is mostly used in web development?

How to count the total number of visitors and whole response time for a request using Filter?

How to run servlet with annotation?

How to create registration form using Servlet and Oracle database?

How can we upload and download the file from the server?

### **What is a web application?**

A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter, etc. and other elements such as HTML, CSS, and JavaScript. The web components typically execute in Web Server and respond to the HTTP request.

### **CGI (Common Gateway Interface)**

CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process.



### **Disadvantages of CGI**

There are many problems in CGI technology:

If the number of clients increases, it takes more time for sending the response.

For each request, it starts a process, and the web server is limited to start processes.

It uses platform dependent language e.g. [C](https://www.javatpoint.com/c-programming-language-tutorial), [C++](https://www.javatpoint.com/cpp-tutorial), [perl](https://www.javatpoint.com/perl-tutorial).

### **Advantages of Servlet**



There are many advantages of Servlet over CGI. The web container creates threads for handling the multiple requests to the Servlet. Threads have many benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The advantages of Servlet are as follows:

**Better performance:** because it creates a thread for each request, not process.

**Portability:** because it uses Java language.

**Robust:** [JVM](https://www.javatpoint.com/jvm-java-virtual-machine) manages Servlets, so we don't need to worry about the memory leak, [garbage collection](https://www.javatpoint.com/Garbage-Collection), etc.

**Secure:** because it uses java language.

# Basics Of Web Pages

# **What is a Webpage**

A web page is a single hypertext document available on World Wide Web (WWW). It is composed of HTML elements and displayed on the user's browser such as [**Mozilla, Firefox**](https://www.javatpoint.com/mozilla-firefox)**,**[**Chrome**](https://www.javatpoint.com/google-chrome)**, etc.** It is also referred to as "**Page."**

In this topic, we are going to discuss various details of the webpage, including the following topics:

ADVERTISEMENT

ADVERTISEMENT

[What is a Webpage?](https://www.javatpoint.com/what-is-a-webpage#Webpage)

[Characteristics of a Webpage](https://www.javatpoint.com/what-is-a-webpage#Characteristics)

[Difference between a Webpage and a Website](https://www.javatpoint.com/what-is-a-webpage#Difference)

[How does a Web Page Work?](https://www.javatpoint.com/what-is-a-webpage#Work)

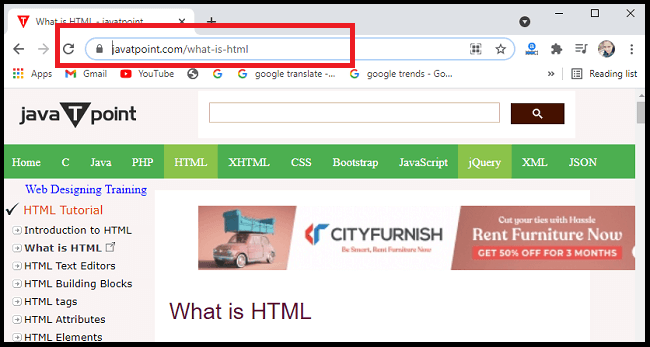
[Elements of a Webpage](https://www.javatpoint.com/what-is-a-webpage#Elements)

[Types of a Web page](https://www.javatpoint.com/what-is-a-webpage#Types)

[How to Create a Simple Webpage?](https://www.javatpoint.com/what-is-a-webpage#Simple)

## **What is a Webpage?**

A webpage is a document written in HTML and can be viewed on any web browser. It is contained within the web server, which can be accessed by entering the URL for that web page, and once it is loaded, it appears on the user's web browser. Each webpage is linked with a unique URL; hence two pages cannot have the same URL.



A webpage may contain **text, links for other pages, graphics, videos, etc**. Moreover, it is mainly used to provide information to the user in text, images, etc.

A webpage is a part of a website; it means a website contains different web pages. Such as **javaTpoint.com** is a website, and the page currently you are accessing is the webpage. It can be understood as an example of a book. So, a Website is like a complete book, and a webpage is like a page of that book.

The WWW or Internet contains millions of web pages, and daily, a lot is being added. Tim Berners-Lee developed **the first webpage.**

Let's understand some basic terms that are used with Webpage:

**WebSite**: A website is a collection of several web pages. These pages are linked together with hyperlinks. A website has a unique domain name, and we can access it by entering that domain name in the URL.

**Search Engine:** A search engine is an internet service that helps users find any information available on the internet. Some examples of search engines are **Google, Yahoo, Bing,** It is usually accessed with the help of Web browser.

**Web Browser:** A web browser or simply browser is application software used to access the internet. Some examples of Web browsers are **Google Chrome, Microsoft Internet Explorer, Safari, etc**. It does two things:

It connects to a web server on the internet and requests a page that the user wants to view; once it finds that page, it displays it on its device.

It can interpret the set of HTML tags within a page to display the page in the correct format.

**Webserver:** A web server can be understood as a computer that hosts or provide a website on the internet. It contains webserver software and component files of a website such as **HTML document, images, CSS stylesheet, and JS files.**

#### **Note: For practice, you can create web pages on your own without the need for a web server, and your browser will display those pages on your machine only.**

**HTML:** HTML is an abbreviation of **Hyper-Text Markup Language**. A markup language is a computer language that specifies how a page should be formatted. With the help of HTML, one can specify fonts, colors, images, headings, and everything that he wants to display on a page displayed by the browser.

#### **Note: A web browser can also display other documents such as a PDF document or images, but only an HTML document is referred to as Web page.**

## **Characteristics of a Web Page**

Following are some characteristics of a Web page:

A simple webpage can be created very quickly.

It takes very little time to create a webpage compared to a Website.

A web page and a website should be compatible with any device, such as Mobile, Desktop, Laptop, etc.

The search engine provides a web page through a link, and when a user clicks on that link, it is redirected to the webpage of a website.

A webpage can have any type of information including videos, and audios.

It can be made up of only HTML(Hypertext Markup Language), or CSS, or JavaScript for dynamic and attractive behavior.

## **Difference between a Webpage and a Website**

Since both Websites and Web pages are related to each other, some users may use them interchangeably, but they are much different from each other. The basic difference between them is that **webpage is a single web document, whereas a Website is a collection of different web pages.** Here are some more differences between both of them:

|  |  |
| --- | --- |
| **Website** | **Webpage** |
| A website is a collection of different web pages that are linked together with hyperlinks. | A webpage is a single hypertext document. |
| It consists of more than one webpage. | It is a single document that is displayed on the user's browser. |
| To develop a website, developers need more skills and more time compared to a webpage. | To develop a webpage, developers need basic HTML knowledge and less time. |
| A website is accessed through its domain name, and it does not include any extension in the URL. | A webpage is accessed through a unique URL with some extension. |
| It can contain information for different entities, such as Javatpoint.com, which contains information about different technologies. | It can contain information for a single entity, such as currently viewing a web page containing information about this page only. |
| It is a little challenging to create a perfect website and requires lots of programming. | It is very simple to create a webpage. |
| Some examples of the website are Javatpoint.com, Amazon.com, etc. | Some examples of Webpages are the currently viewing page, contact page, registration page, the home page, etc. |

#### **Note: The terms Webpage and Web page are the same, and both are technically correct. However, most style guides suggest using a Webpage instead of a Web page.**

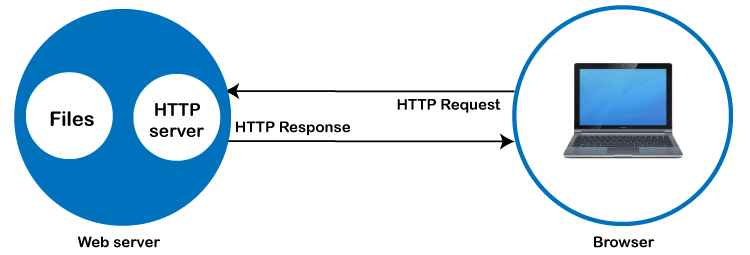
## **How does a Web Page Work?**

A simple web page is created using HTML, which is a markup language. However, we can also use CSS and JavaScript to add more functionalities and make it more attractive.

It is created using HTML, hence containing different markup tags that specify how the data should be formatted on screen.

The webpage is contained within the webserver. To load this webpage, a client sends the request to the server, and generally, the browser is known as the client, which can request the page on the internet.

The web browser requests the page on the internet. Once it is responded to by the server, the browser interprets the markup tags and displays them on the user's screen in the correct format.



The browser sends the request for a page or a file via **an HTTP request.** The HTTP is the **Hypertext Transfer Protocol**, a network protocol that allows transferring hypermedia documents over the internet between a browser and server.

Once the request reaches the server, the HTTP server accepts the request, finds the requested page, and sends it back to the browser through **the HTTP response**. If a server is unable to find the requested page, it returns a **404 response**.

## **Elements of a Webpage**

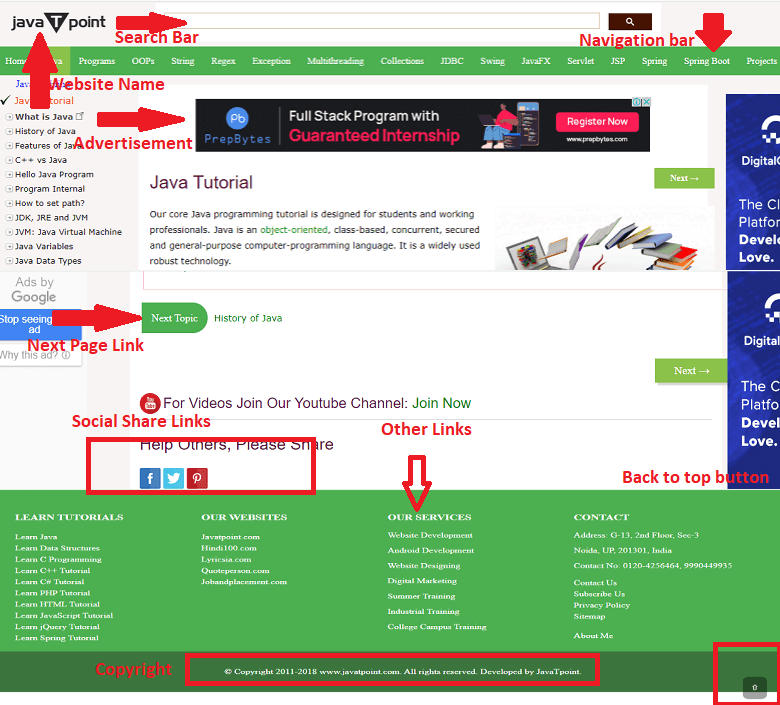
The main element of the webpage is a text file composed of [HTML](https://www.javatpoint.com/html-tutorial). Apart from this, a webpage can also have the following elements:

**CSS:** The CSS code is used to make the page more interactive and control its look and feel.

**Scripts:** The JavaScript code is included in a webpage to add interactivity to the page and add more functionalities to it.

**Media:** It is used to include media components such as audio, video, and images.

Although every web page is different from another web, some components are common to almost all the pages. Some of these components are given below; you can also relate these elements by the given image:



**Name of the Website:** Each webpage includes the name of the website or company, or blog to which it is attached. The name of the website and the logo are mostly situated at the top-left corner of the page. The logo may also contain a slogan of the site or a brief introduction to the site so that visitors can quickly identify what this site is about. It is one of the important components of the webpage.  
The website's name also has a link that can redirect to the home page of that site. **The name of the website usually includes at the header of the page.**

**Search bar:** A search bar is also an important component that should present on each page of a website or blog. The search blog allows the visitor to search related information on that website.

**Navigation Bar:** A navigation bar is a component of a webpage that contains links to some important sections of the website. It helps the visitors to easily traverse some major sections of the website. It is placed mainly on the top of the web page or the left side of the page. When the user clicks on any link given in the navigation bar, it redirects to the page.

**Heading of the page:** Heading of the page tells the main information about the page, i.e., what this is all about. **T**he heading is available on the top of the page, and it is included with the help of **the <h1> tag** of HTML.

**Content of the Page**: The content of the page means the information of the page. As you are visiting this page and reading this information, all the information contained within this page is known as the page's content. It may contain below sub-elements:

**Paragraphs:** A webpage can have different paragraphs as per their length. The opening paragraph is crucial on the whole page, as it draws the attention of the visitor. If the first paragraph is not interesting and not related to the topic, a user may leave the page immediately. To create a **paragraph <P> tag is used in HTML**.

**Subheadings:** A page may have different subheadings as per the topic, whether related to information about something or a website's web page. In HTML from <H2> to <H6> tags are used for including subheadings. Each page should be divided into different subheadings to make it easier for the users to read and understand.

**Images:** Each webpage contains images to make its content more attractive. To include an image, <img> tag is used in HTML.

**Feedback or Comment form: D**ifferent websites include a Feedback or Comment form on each webpage. It is used to know the visitor's views about the information of that page and any other feedback for the page or site. It lets to know the creator of the page that if the information is helpful or not.

**Social Share links:** Social share links allow the visitors to share that webpage with their friends on different social sites such as Facebook, Linked In, Twitter, etc.

**Copyright Info:** Each webpage must have Copyright information along with Privacy Policy. This information is available at the end of the page or on the footer of the page.

**Go to Top:** On most of the pages, a Back to Top button or link is available. This option helps the users to return to the upper section of the page.

**Advertisement Banner:** Whenever you visit any good website, you see different advertisement banners on each site's page. These banners are used to displays ads in different places on a page. These banners are used to provide monitory benefits to the owner of the website.

**Previous and Next buttons or links:** On most of the pages, previous and Next buttons are available that help the user easily navigate to the previous and next page related to the topic. A page can also have links to other pages.

**Company Info:** On the footer of the page, there is a brief introduction about the company or website of that page. By this, visitors get to know about the company and can build trust in it.

There can also be some additional information and tools such as a button to print the page that can also be helpful for users.

## **Types of a Web page**

There are mainly two types of a Web page based on functionality:

**Static Webpage**

**Dynamic Webpage**

### **Static Webpage**

Static webpages are those webpages that cannot be modified or altered by the client. These are also known as stationary or flat web pages. They are displayed on the client's browser in the same format and manner as they are saved in the webserver. Users can only load the page and read the information but cannot perform any change on the page.

A static webpage is generally made up of HTML and CSS only.

### **Dynamic Webpage**

As the name suggests, Dynamic webpages are dynamic, which means it shows different information at different point of time.

ADVERTISEMENT

ADVERTISEMENT

The dynamic webpage shows different content each time it is viewed. There are two types of Dynamic web pages, which are:

**Server-Side Dynamic Webpage:** These web pages are created using Server-side scripting. These pages are changed when they are visited or viewed. Some examples of server-side pages are l**ogin pages, submission forms, shopping carts, etc**. Various scripting languages such as **PHP, ASP.Net, JSP, etc**., can be used for server-side scripting.

**Client-side Dynamic Webpage:** These web pages are created using Client-side scripting. These pages get changed in response to actions that occur within that page, such as mouse or keyboard action. Scripting languages such **as JavaScript, Dart, etc**., can be used for client-Side scripting.

#### **Note: Scripting languages are programming languages that allow us to write programs in the form of scripts, and these are interpreted, not compiled.**

Apart from these two Webpages, there are some examples of common web pages that can be found on most of the websites, and these are as follow:

**Home Page:** The home page of any website is one of the most important pages. It is called a home page because this is like a starting point from where users can go anywhere on that website. This page usually contains links to the important zones of the site. It can also be known as the **index page**.

**Feed Page:** The feed page is usually found in those websites that update content. It is used to provide information to users for the latest information that has been updated.

**Menu Page:** The menu page is created to accomplish the navigation goal. The page contains a collection of different links that give access to different categories and zones.

**About-us Page:** This page contains brief information and details of the company, product, or website. It allows the visitors to know the details of the website that they are using.

**Registration Page:** The registration page allows users to create an account by signing up, and hence they can create a personalized account. It helps the company to know the visitor and provide personalized offers and deals to them.

**Contacts Page:** This is a simple page made for the visitors to contact the website owner. For any issue or any feedback, users can use this page.

**Landing Page:** This is one of the special types of a Web page used as a core part of a website or as an individual page**.** The main aim of creating this page is to convert the visitors into the lead. It represents clear and focused content on a specific goal.

## **How to Create a Simple Webpage?**

Creating a simple webpage is very easy; anyone with basic knowledge of computers and HTML can create it. But before creating a webpage, you should be aware of the below points:

A simple webpage can be created using HTML code only. Such pages are simple but not interactive and have very few functionalities.

To make your webpage interactive and add functionality, you need to learn and use scripting languages, such as **PHP, Python**, etc.

A web page can also be created using Notepad, but it is recommended to use IDEs for advanced uses such as **Atom, Sublime Text editor, PyCharm,**

**Follow the below steps to create your webpage:**

Open the Notepad application on your computer.

Write Below code on it.

<!DOCTYPE html**>**

**<html>**

**<head>**

**<title>**Your title goes here**</title>**

**</head>**

**<body** bgcolor="white" text="red"**>**

**<h1>**first Web page**</h1>**

**<p>**This is my first web page!**</p>**

**</body>**

**</html>**

In the above code, the following tags are used:

**<!DOCTYPE html>:** It is used for document type declaration, which means which version of HTML you are using. It indicates the browser that which language it is supposed to interpret.

**<html>:** it indicates the start and of the HTML code.

**<head>:** It can have different types of information such as title, meta tag, etc.; this information will not appear on the webpage.

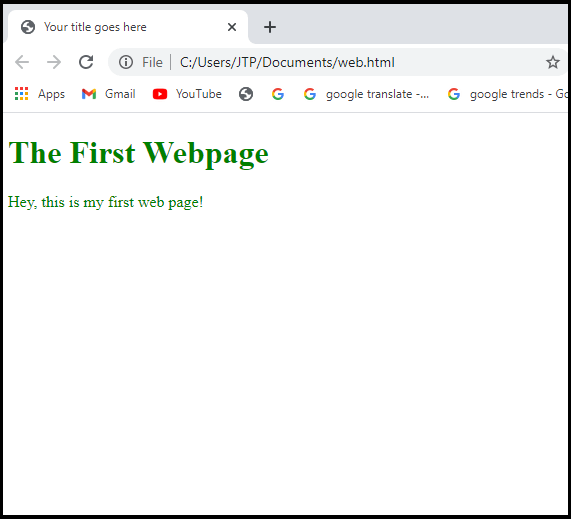
**<body>:** This tag contains other tags on the webpage, and users can see them. In above code, we have included <h1> heading tag and <p> paragraph tags.

#### **Note: It is important to end each tag in html, and put all the tags either in upper case or lower case. However, a lower case is recommended.**

Save the file with any name and **.html extension**. For example, save it with **html** name.

Go to the saved file (web.html), double click or right-click and run it.

It will open on your default browser and will display the below output:



You can also add more tags for different elements, such as add images, background images, border, table, table, etc., using HTML. You can learn all these from [here](https://www.javatpoint.com/html-tutorial).

#### **Note: This webpage is local to your machine only, and only you can see this on your browser. To view this on the internet, you need to first publish it.**

After creating the page, you can also make changes in your file through the editor. Just make the change, save the file again, and reload the page; those changes will appear on the screen.

# Servlet API Introduction

# **Servlet API**

[Servlet API](https://www.javatpoint.com/servlet-api)

[Interfaces in javax.servlet package](https://www.javatpoint.com/servlet-api#servletapi1)

[Classes in javax.servlet package](https://www.javatpoint.com/servlet-api#servletapi2)

[Interfaces in javax.servlet.http package](https://www.javatpoint.com/servlet-api#servletapi3)

[Classes in javax.servlet.http package](https://www.javatpoint.com/servlet-api#servletapi4)

The javax.servlet and javax.servlet.http packages represent interfaces and classes for servlet api.

The **javax.servlet** package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.

The **javax.servlet.http** package contains interfaces and classes that are responsible for http requests only.

Let's see what are the interfaces of javax.servlet package.

### **Interfaces in javax.servlet package**

There are many interfaces in javax.servlet package. They are as follows:

Servlet

ServletRequest

ServletResponse

RequestDispatcher

ServletConfig

ServletContext

SingleThreadModel

Filter

FilterConfig

FilterChain

ServletRequestListener

ServletRequestAttributeListener

ServletContextListener

ServletContextAttributeListener

### **Classes in javax.servlet package**

There are many classes in javax.servlet package. They are as follows:

GenericServlet

ServletInputStream

ServletOutputStream

ServletRequestWrapper

ServletResponseWrapper

ServletRequestEvent

ServletContextEvent

ServletRequestAttributeEvent

ServletContextAttributeEvent

ServletException

UnavailableException

### **Interfaces in javax.servlet.http package**

There are many interfaces in javax.servlet.http package. They are as follows:

HttpServletRequest

HttpServletResponse

HttpSession

HttpSessionListener

HttpSessionAttributeListener

HttpSessionBindingListener

HttpSessionActivationListener

HttpSessionContext (deprecated now)

### **Classes in javax.servlet.http package**

There are many classes in javax.servlet.http package. They are as follows:

HttpServlet

Cookie

HttpServletRequestWrapper

HttpServletResponseWrapper

HttpSessionEvent

HttpSessionBindingEvent

HttpUtils (deprecated now)

# Servlet Interface In Servlet

# **Servlet Interface**

[Servlet Interface](https://www.javatpoint.com/Servlet-interface)

[Methods of Servlet interface](https://www.javatpoint.com/Servlet-interface#servletmethods)

**Servlet interface provides** commonbehaviorto all the servlets.Servlet interface defines methods that all servlets must implement.

Servlet interface needs to be implemented for creating any servlet (either directly or indirectly). It provides 3 life cycle methods that are used to initialize the servlet, to service the requests, and to destroy the servlet and 2 non-life cycle methods.

### **Methods of Servlet interface**

There are 5 methods in Servlet interface. The init, service and destroy are the life cycle methods of servlet. These are invoked by the web container.

|  |  |
| --- | --- |
| **Method** | **Description** |
| **public void init(ServletConfig config)** | initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once. |
| **public void service(ServletRequest request,ServletResponse response)** | provides response for the incoming request. It is invoked at each request by the web container. |
| **public void destroy()** | is invoked only once and indicates that servlet is being destroyed. |
| **public ServletConfig getServletConfig()** | returns the object of ServletConfig. |
| **public String getServletInfo()** | returns information about servlet such as writer, copyright, version etc. |

### **Servlet Example by implementing Servlet interface**

Let's see the simple example of servlet by implementing the servlet interface.

### It will be better if you learn it after visiting steps to create a servlet.

*File: First.java*

**import** java.io.\*;

**import** javax.servlet.\*;

**public** **class** First **implements** Servlet{

ServletConfig config=**null**;

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

**this**.config=config;

System.out.println("servlet is initialized");

}

**public** **void** service(ServletRequest req,ServletResponse res)

**throws** IOException,ServletException{

res.setContentType("text/html");

PrintWriter out=res.getWriter();

out.print("<html><body>");

out.print("<b>hello simple servlet</b>");

out.print("</body></html>");

}

**public** **void** destroy(){System.out.println("servlet is destroyed");}

**public** ServletConfig getServletConfig(){**return** config;}

**public** String getServletInfo(){**return** "copyright 2007-1010";}

}

# GenericServlet In Servlet

# **GenericServlet class**

[GenericServlet class](https://www.javatpoint.com/GenericServlet-class)

[Methods of GenericServlet class](https://www.javatpoint.com/GenericServlet-class#genericmethods)

[Example of GenericServlet class](https://www.javatpoint.com/GenericServlet-class)

**GenericServlet** class implements **Servlet**, **ServletConfig** and **Serializable** interfaces. It provides the implementation of all the methods of these interfaces except the service method.

GenericServlet class can handle any type of request so it is protocol-independent.

You may create a generic servlet by inheriting the GenericServlet class and providing the implementation of the service method.

### **Methods of GenericServlet class**

There are many methods in GenericServlet class. They are as follows:

**public void init(ServletConfig config)** is used to initialize the servlet.

**public abstract void service(ServletRequest request, ServletResponse response)** provides service for the incoming request. It is invoked at each time when user requests for a servlet.

**public void destroy()** is invoked only once throughout the life cycle and indicates that servlet is being destroyed.

**public ServletConfig getServletConfig()** returns the object of ServletConfig.

**public String getServletInfo()** returns information about servlet such as writer, copyright, version etc.

**public void init()** it is a convenient method for the servlet programmers, now there is no need to call super.init(config)

**public ServletContext getServletContext()** returns the object of ServletContext.

**public String getInitParameter(String name)** returns the parameter value for the given parameter name.

**public Enumeration getInitParameterNames()** returns all the parameters defined in the web.xml file.

**public String getServletName()** returns the name of the servlet object.

**public void log(String msg)** writes the given message in the servlet log file.

**public void log(String msg,Throwable t)** writes the explanatory message in the servlet log file and a stack trace.

### **Servlet Example by inheriting the GenericServlet class**

Let's see the simple example of servlet by inheriting the GenericServlet class.

### It will be better if you learn it after visiting steps to create a servlet.

*File: First.java*

**import** java.io.\*;

**import** javax.servlet.\*;

**public** **class** First **extends** GenericServlet{

**public** **void** service(ServletRequest req,ServletResponse res)

**throws** IOException,ServletException{

res.setContentType("text/html");

PrintWriter out=res.getWriter();

out.print("<html><body>");

out.print("<b>hello generic servlet</b>");

out.print("</body></html>");

}

}

# HttpServlet In Servlet

# **HttpServlet class**

[HttpServlet class](https://www.javatpoint.com/HttpServlet-class)

[Methods of HttpServlet class](https://www.javatpoint.com/HttpServlet-class#httpservletmethods)

|  |
| --- |
| The HttpServlet class extends the GenericServlet class and implements Serializable interface. It provides http specific methods such as doGet, doPost, doHead, doTrace etc. |

### **Methods of HttpServlet class**

There are many methods in HttpServlet class. They are as follows:

**public void service(ServletRequest req,ServletResponse res)** dispatches the request to the protected service method by converting the request and response object into http type.

**protected void service(HttpServletRequest req, HttpServletResponse res)** receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.

**protected void doGet(HttpServletRequest req, HttpServletResponse res)** handles the GET request. It is invoked by the web container.

**protected void doPost(HttpServletRequest req, HttpServletResponse res)** handles the POST request. It is invoked by the web container.

**protected void doHead(HttpServletRequest req, HttpServletResponse res)** handles the HEAD request. It is invoked by the web container.

**protected void doOptions(HttpServletRequest req, HttpServletResponse res)** handles the OPTIONS request. It is invoked by the web container.

**protected void doPut(HttpServletRequest req, HttpServletResponse res)** handles the PUT request. It is invoked by the web container.

**protected void doTrace(HttpServletRequest req, HttpServletResponse res)** handles the TRACE request. It is invoked by the web container.

**protected void doDelete(HttpServletRequest req, HttpServletResponse res)** handles the DELETE request. It is invoked by the web container.

**protected long getLastModified(HttpServletRequest req)** returns the time when HttpServletRequest was last modified since midnight January 1, 1970 GMT.

# Servlet Life Cycle In Java

# **Life Cycle of a Servlet (Servlet Life Cycle)**

[Life Cycle of a Servlet](https://www.javatpoint.com/life-cycle-of-a-servlet)

[Servlet class is loaded](https://www.javatpoint.com/life-cycle-of-a-servlet#servletlifecycle1)

[Servlet instance is created](https://www.javatpoint.com/life-cycle-of-a-servlet#servletlifecycle2)

[init method is invoked](https://www.javatpoint.com/life-cycle-of-a-servlet#servletlifecycle3)

[service method is invoked](https://www.javatpoint.com/life-cycle-of-a-servlet#servletlifecycle4)

[destroy method is invoked](https://www.javatpoint.com/life-cycle-of-a-servlet#servletlifecycle5)

The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet:

Servlet class is loaded.

Servlet instance is created.

init method is invoked.

service method is invoked.

destroy method is invoked.



As displayed in the above diagram, there are three states of a servlet: new, ready and end. The servlet is in new state if servlet instance is created. After invoking the init() method, Servlet comes in the ready state. In the ready state, servlet performs all the tasks. When the web container invokes the destroy() method, it shifts to the end state.

### **Servlet class is loaded**

The classloader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.

### **2) Servlet instance is created**

The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.

### **3) init method is invoked**

|  |
| --- |
| The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface. Syntax of the init method is given below: |

**public** **void** init(ServletConfig config) **throws** ServletException

### **4) service method is invoked**

The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method. If servlet is initialized, it calls the service method. Notice that servlet is initialized only once. The syntax of the service method of the Servlet interface is given below:

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

**throws** ServletException, IOException

### **5) destroy method is invoked**

The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc. The syntax of the destroy method of the Servlet interface is given below:

**public** **void** destroy()

# Apache Tomcat Server

# **What is Tomcat?**

It is an open-source Java servlet container that implements many Java Enterprise Specs such as the Websites API, Java-Server Pages and last but not least, the Java Servlet. The complete name of Tomcat is "Apache Tomcat" it was developed in an open, participatory environment and released in 1998 for the very first time. It began as the reference implementation for the very first Java-Server Pages and the [Java Servlet](https://www.javatpoint.com/servlet-tutorial) API. However, it no longer works as the reference implementation for both of these technologies, but it is considered as the first choice among the users even after that. It is still one of the most widely used java-sever due to several capabilities such as good extensibility, proven core engine, and well-test and durable. Here we used the term "servlet" many times, so what is [java](https://www.javatpoint.com/java-tutorial) servlet; it is a kind of software that enables the webserver to handle the dynamic(java-based) content using the Http protocols.

## **So what exactly is Apache tomcat?**

If you are a little familiar with the websites or have some basic knowledge about the websites, you must have heard about the [HTTP protocol](https://www.javatpoint.com/http) or may also know what actually are they. If you want to provide any web-services such as you want to provide a simple static content possibly by using [HTML](https://www.javatpoint.com/html-tutorial) (or Hypertext Markup Language), or maybe you just want to send data from a server to point you, so you necessarily need a server and that server is [HTTP](https://www.javatpoint.com/http-full-form)(HyperText transfer protocol). So, as we all know that if anyone wants to make a simple, static website, he definitely requires an [HTTP](https://www.javatpoint.com/http-tutorial) server, but if he wants to make website dynamic, he has to use servlet. We use the HTTP server if we want to send simple data. If we want to send dynamic data or to make our website dynamic, we need to use the servlet. Hence, we need an HTTP server and what else we need is a container where we will run or servlet, so when we combine the [HTTP](https://www.javatpoint.com/computer-network-http) server and the servlet (or we can say servlet container), they both combine to become a single server know as tomcat server.

In simple words, we can say that The [Apache Tomcat](https://www.javatpoint.com/java-and-tomcat-setup-for-jenkins) is actually a server and a servlet container.

### **What kind of server is Tomcat?**

The Java ecosystem supports a wide variety of application servers, so let's have a little discussion on each of them and see where Tomcat fits in:

A servlet container is basically an implementation of the Java servlet specification, which is mainly used for the purpose of hosting Java servlets.

The Java enterprise application-server is an implementation of the Java specification.

A web- server is a kind of server designed to serve files using a local system such as Apache.

We can say that, at the center, the Tomcat is [JSP (Java Server Pages)](https://www.javatpoint.com/jsp-tutorial) and Servlet. The JSP is one of the server-side programming technologies that enables the developers to create platform-independent dynamic content and also known as the server-side view rendering technology. A servlet is a java-based software component that helps in extending the capabilities of a server. However, it can also respond to several kinds of requests and generally implemented web server containers to host the web-applications on the webservers. As the developer's point of view, we just have to write the java server pages (or JSP) or the servlet and not required to worry about routing; the Tomcat will handle the routing.

The Tomcat also consists of the webserver known as the Coyote engine due to which it's possible to extend the capability of Tomcat to include several java enterprise specs, and including the [Java Persistence API(JPA)](https://www.javatpoint.com/jpa-tutorial). The Tomcat also has an extended version known as the "TomEE" that contains more enterprise features.

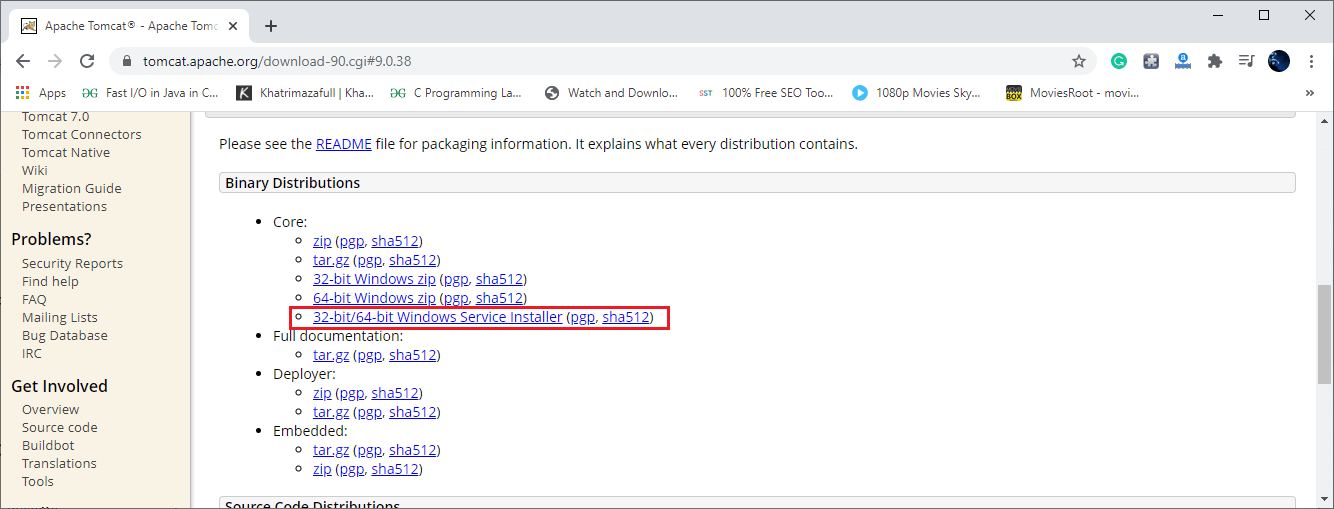
Let's see how to install Tomcat. But before doing that, we are required to download the Tomcat. If you are a window ten user, you can use the following given steps for downloading and installing the Tomcat on your system:

**Steps to download and install the Tomcat on**[**Windows 10 operating system**](https://www.javatpoint.com/what-is-windows)

There are many versions of the Tomcat available with different features on its website. You can choose the version according to your requirements, but we suggest you download the latest stable version.

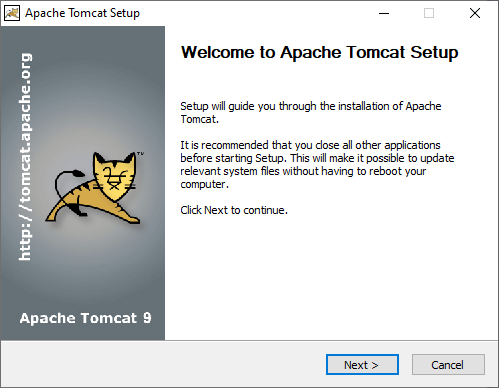
**Step 1.** Open the [Google Chrome](https://www.javatpoint.com/google-chrome) or any of your web browser and type "download Tomcat for windows" in the search box. You can also go directly on Tomcat's website by clicking on this <https://tomcat.apache.org/download-90.cgi#9.0.38>

Now download any version of Tomcat you want:

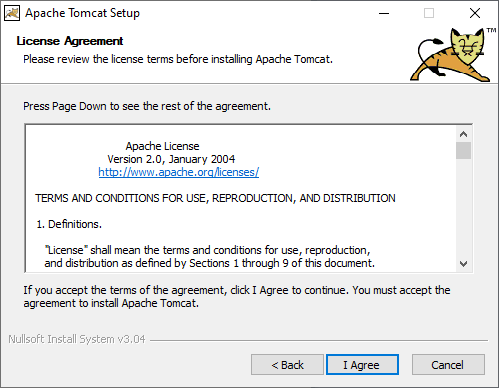


**Step 2.** Go to Download and click on the downloaded file and wait for little until the installation process starts.

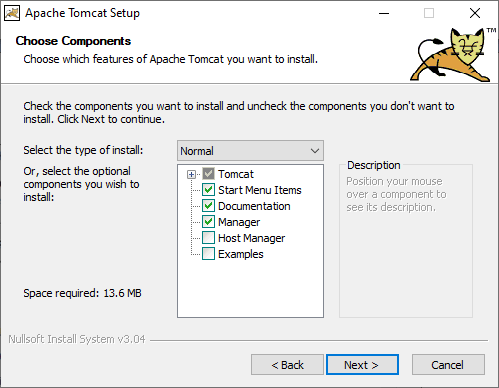
**Step 3.** Once the installation process gets started, click on the **"Next"** button, as shown below:



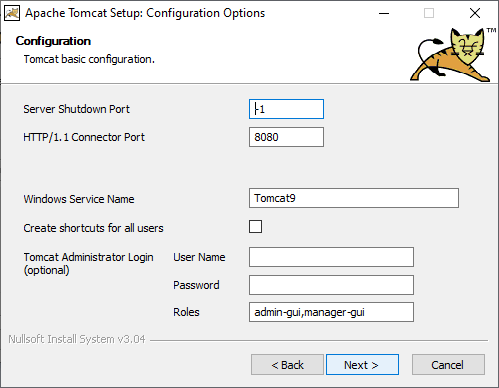
**Step 4.** Click on the button labeled as **"I Agree."**



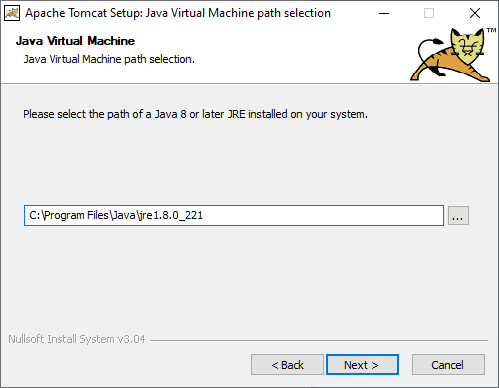
**Step 5.** Click on the **"Next"** button.



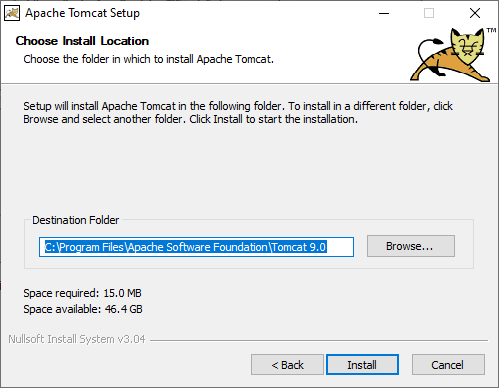
**Step 6.** Enter the user name and password and click on the **" Next"** button, as shown below:



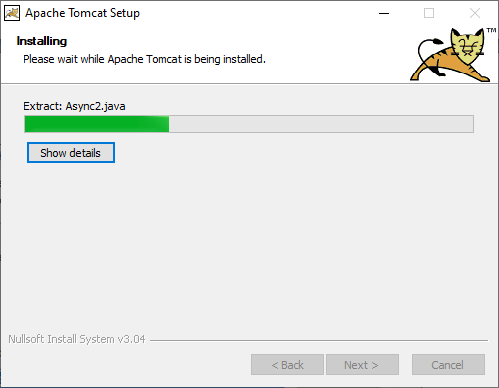
**Step 7.** Then click on the **"Next"** button again



**Step 8.** Now click on the **"Install"** button.

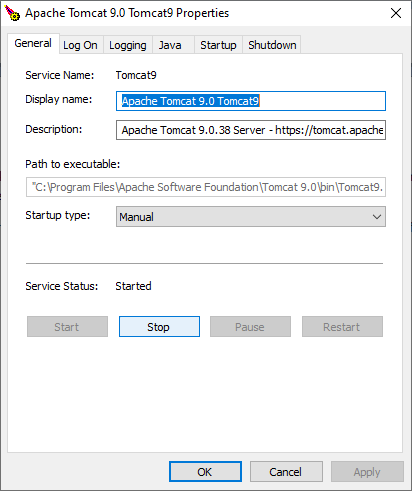


Wait for some time until the Tomcat gets installed.



**Step 9.** Now click on the **"Finish"** button, here the installation of Tomcat is completed. It may ask you to restart your system, so restart your system.

Now you can start the Tomcat by clicking on its icon, and you can start and stop the server.



### **Advantages of Tomcat:**

Some significant advantages of Tomcat are as follows:

**It is open-source**  
It means anyone from anywhere can download, install, and use it free of cost, which makes it the first choice among the new developers and new users.

**Incredibly Lightweight**  
It is actually a very light application, even with the JavaEE's certification. However, it provides all necessary and standard functionalities required to operate a server, which means it gives very fast load and redeploys as compared to its various alternatives.  
Yes, it is right that it does not offer so many features in case you want a number of features, it might be good for you, but if you want to have an easy and fast means in order to run your application, it is the best option for you.

**Highly flexible**  
Due to its built-in customization options, extensive and lightweight nature, it offers high flexibility, a user can run it in any fashion he wants, and it will still work as fine without any issues. Since it is open-source, anyone who has knowledge can tweak it according to his requirements.

**Stability**  
It is one of the most stable platforms available today to build on and using it to run our applications. It is incredibly stable because it runs independently of our Apache installation. In case if there is a big failure in Tomcat due to which it to stop working, the rest of our server would run just well.

**It provides us an extra level of security**  
As the several organizations usually like to position their Tomcat's installation behind the protection of an extra firewall which can be accessible only from the Apache installation.

**It is well documented**  
It has several excellent documentation available, including a vast range of freely available online tutorials that can be downloaded or viewed directly online by the user, which makes it one of the best choices to fill the requirement of an application server in mostly every java web-application.Whether a user is looking for the installation instructions, startup settings, server configuration notes, all kind of information about the Tomcat is already available on the internet.

**It is one of the most widely used application servers**  
According to an estimation, it holds almost 60 percent of the market share almost all java application server deployments, which makes it one of the most popular application servers used for java web-based applications. However, we cannot say that it implements all of the features required for a JavaEE application server; instead, it enables us to run Java EE application.  
Tomcat acts as a "webserver" or "servlet container." However, there is a plethora of terminology for anything.

**It's mature**  
We take a look back in the past; we will find that it has existed for almost 20 years, which is quite a significant time, in which it gets mature over time passage. Since the Tomcat is open-source software, it's updated, and new releases come out nearly on a regular basis, and the open-source community maintains it. The maturity makes it one of the most extremely stable application servers for the development of software, applications, and deploying java applications. Since now, it is extremely a stable option that becomes more powerful with excellent community support.

### **Let discuss some disadvantages of Tomcat**

It is not as fast as the Apache if we are working with the static pages

It has some issues like a memory leak

It's way to handle the logs.

Issues in the SSL installations

Its user interface is inferior and basic.

# Steps To Create A Servlet In Tomcat And Connectivity

# **Steps to create a servlet example**

[Steps to create the servlet using Tomcat server](https://www.javatpoint.com/steps-to-create-a-servlet-using-tomcat-server)

[Create a directory structure](https://www.javatpoint.com/steps-to-create-a-servlet-using-tomcat-server#servletstep1)

[Create a Servlet](https://www.javatpoint.com/steps-to-create-a-servlet-using-tomcat-server#servletstep2)

[Compile the Servlet](https://www.javatpoint.com/steps-to-create-a-servlet-using-tomcat-server#servletstep3)

[Create a deployment descriptor](https://www.javatpoint.com/steps-to-create-a-servlet-using-tomcat-server#servletstep4)

[Start the server and deploy the application](https://www.javatpoint.com/steps-to-create-a-servlet-using-tomcat-server#servletstep5)

There are given 6 steps to create a **servlet example**. These steps are required for all the servers.

The servlet example can be created by three ways:

By implementing Servlet interface,

By inheriting GenericServlet class, (or)

By inheriting HttpServlet class

The mostly used approach is by extending HttpServlet because it provides http request specific method such as doGet(), doPost(), doHead() etc.

Here, we are going to use **apache tomcat server** in this example. The steps are as follows:

Create a directory structure

Create a Servlet

Compile the Servlet

Create a deployment descriptor

Start the server and deploy the project

Access the servlet

[download this example of servlet](https://static.javatpoint.com/src/servlet/firstservlet.zip)  
[download example of servlet by extending GenericServlet](https://static.javatpoint.com/src/servlet/genericexample.zip)  
[download example of servlet by implementing Servlet interface](https://static.javatpoint.com/src/servlet/servletexample.zip)

### **1)Create a directory structures**

The **directory structure** defines that where to put the different types of files so that web container may get the information and respond to the client.

The Sun Microsystem defines a unique standard to be followed by all the server vendors. Let's see the directory structure that must be followed to create the servlet.



As you can see that the servlet class file must be in the classes folder. The web.xml file must be under the WEB-INF folder.

### **2)Create a Servlet**

|  |
| --- |
| There are three ways to create the servlet.  By implementing the Servlet interface  By inheriting the GenericServlet class  By inheriting the HttpServlet class  The HttpServlet class is widely used to create the servlet because it provides methods to handle http requests such as doGet(), doPost, doHead() etc. |
| In this example we are going to create a servlet that extends the HttpServlet class. In this example, we are inheriting the HttpServlet class and providing the implementation of the doGet() method. Notice that get request is the default request. |

**DemoServlet.java**

**import** javax.servlet.http.\*;

**import** javax.servlet.\*;

**import** java.io.\*;

**public** **class** DemoServlet **extends** HttpServlet{

**public** **void** doGet(HttpServletRequest req,HttpServletResponse res)

**throws** ServletException,IOException

{

res.setContentType("text/html");//setting the content type

PrintWriter pw=res.getWriter();//get the stream to write the data

//writing html in the stream

pw.println("<html><body>");

pw.println("Welcome to servlet");

pw.println("</body></html>");

pw.close();//closing the stream

}}

### **3)Compile the servlet**

For compiling the Servlet, jar file is required to be loaded. Different Servers provide different jar files:

|  |  |
| --- | --- |
| **Jar file** | **Server** |
| 1) servlet-api.jar | Apache Tomcat |
| 2) weblogic.jar | Weblogic |
| 3) javaee.jar | Glassfish |
| 4) javaee.jar | JBoss |

### **Two ways to load the jar file**

set classpath

paste the jar file in JRE/lib/ext folder

Put the java file in any folder. After compiling the java file, paste the class file of servlet in **WEB-INF/classes** directory.

### **4)Create the deployment descriptor (web.xml file)**

The **deployment descriptor** is an xml file, from which Web Container gets the information about the servet to be invoked.

ADVERTISEMENT

The web container uses the Parser to get the information from the web.xml file. There are many xml parsers such as SAX, DOM and Pull.

There are many elements in the web.xml file. Here is given some necessary elements to run the simple servlet program.

**web.xml file**

**<web-app>**

**<servlet>**

**<servlet-name>**sonoojaiswal**</servlet-name>**

**<servlet-class>**DemoServlet**</servlet-class>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>**sonoojaiswal**</servlet-name>**

**<url-pattern>**/welcome**</url-pattern>**

**</servlet-mapping>**

**</web-app>**

### **Description of the elements of web.xml file**

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

|  |
| --- |
| **<web-app>** represents the whole application. |
| **<servlet>** is sub element of <web-app> and represents the servlet. |
| **<servlet-name>** is sub element of <servlet> represents the name of the servlet. |
| **<servlet-class>** is sub element of <servlet> represents the class of the servlet. |
| **<servlet-mapping>** is sub element of <web-app>. It is used to map the servlet. |
| **<url-pattern>** is sub element of <servlet-mapping>. This pattern is used at client side to invoke the servlet. |

### **5)Start the Server and deploy the project**

To start Apache Tomcat server, double click on the startup.bat file under apache-tomcat/bin directory.

### **One Time Configuration for Apache Tomcat Server**

You need to perform 2 tasks:

set JAVA\_HOME or JRE\_HOME in environment variable (It is required to start server).

Change the port number of tomcat (optional). It is required if another server is running on same port (8080).

#### **How to set JAVA\_HOME in environment variable?**

To start Apache Tomcat server JAVA\_HOME and JRE\_HOME must be set in Environment variables.

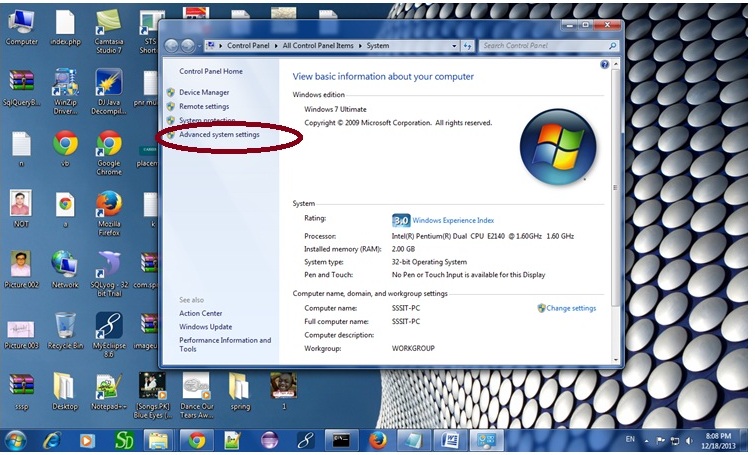
Go to My Computer properties -> Click on advanced tab then environment variables -> Click on the new tab of user variable -> Write JAVA\_HOME in variable name and paste the path of jdk folder in variable value -> ok -> ok -> ok.

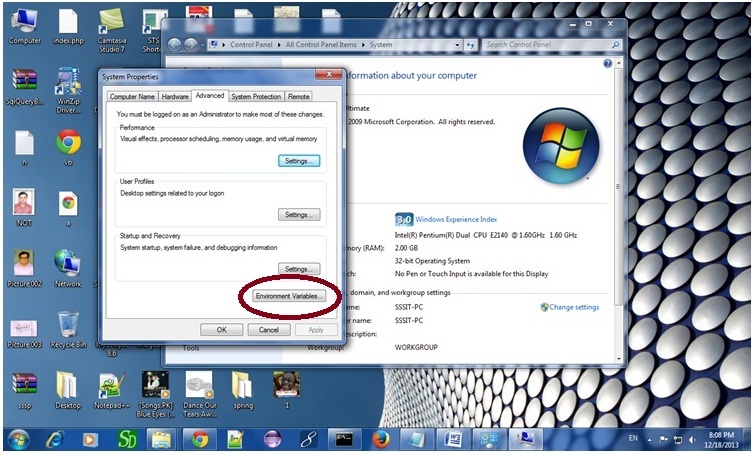
Go to My Computer properties:



Click on advanced system settings tab then environment variables:

ADVERTISEMENT

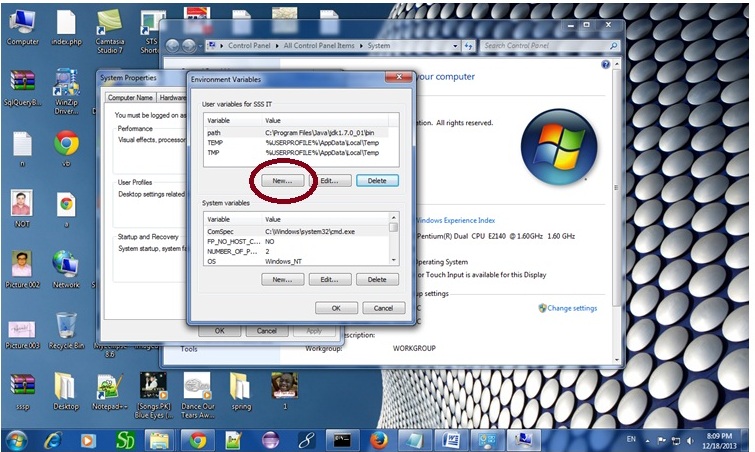




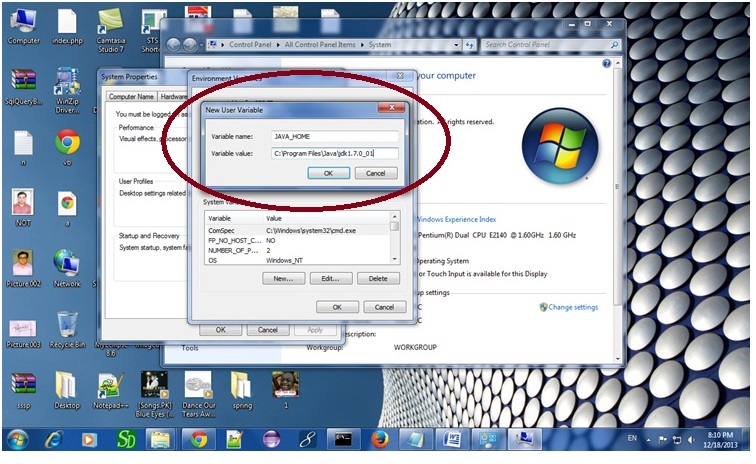
ADVERTISEMENT

ADVERTISEMENT

Click on the new tab of user variable or system variable:

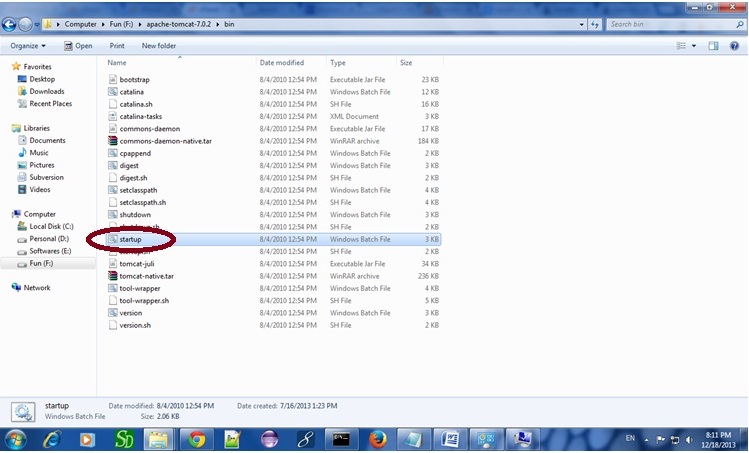


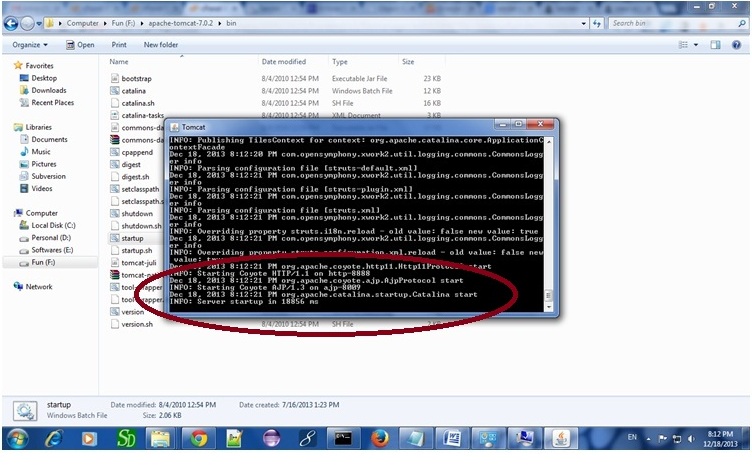
Write JAVA\_HOME in variable name and paste the path of jdk folder in variable value:



There must not be semicolon (;) at the end of the path.

|  |
| --- |
| After setting the JAVA\_HOME double click on the startup.bat file in apache tomcat/bin. |
| Note: There are two types of tomcat available:  Apache tomcat that needs to extract only (no need to install)  Apache tomcat that needs to install |
| It is the example of apache tomcat that needs to extract only. |





Now server is started successfully.

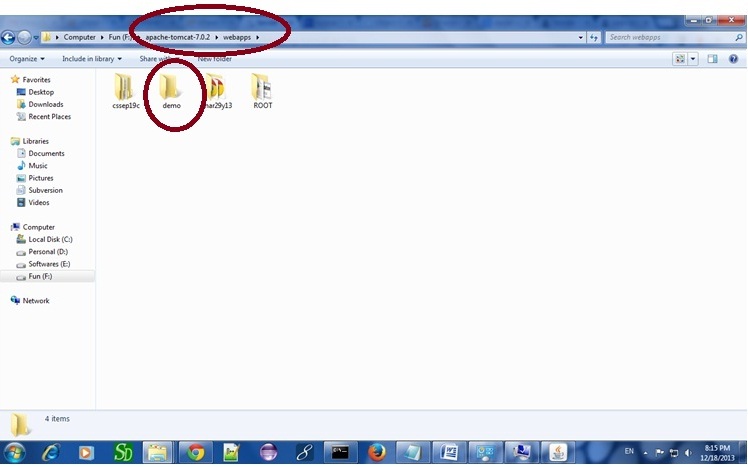
### **2) How to change port number of apache tomcat**

Changing the port number is required if there is another server running on the same system with same port number.Suppose you have installed oracle, you need to change the port number of apache tomcat because both have the default port number 8080.

Open **server.xml file** in notepad. It is located inside the **apache-tomcat/conf** directory . Change the Connector port = 8080 and replace 8080 by any four digit number instead of 8080. Let us replace it by 9999 and save this file.

### **5) How to deploy the servlet project**

Copy the project and paste it in the webapps folder under apache tomcat.



But there are several ways to deploy the project. They are as follows:

By copying the context(project) folder into the webapps directory

By copying the war folder into the webapps directory

By selecting the folder path from the server

By selecting the war file from the server

Here, we are using the first approach.

You can also create war file, and paste it inside the webapps directory. To do so, you need to use jar tool to create the war file. Go inside the project directory (before the WEB-INF), then write:

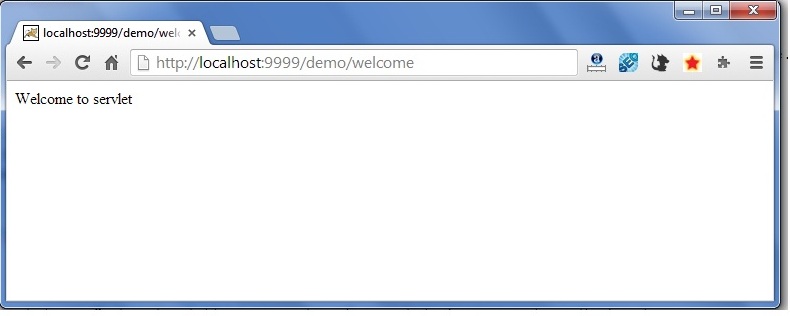
projectfolder> jar cvf myproject.war \*

Creating war file has an advantage that moving the project from one location to another takes less time.

### **6) How to access the servlet**

Open broser and write http://hostname:portno/contextroot/urlpatternofservlet. For example:

http://localhost:9999/demo/welcome



# ServletRequest Methods In Servlet

# **ServletRequest Interface**

[ServletRequest Interface](https://www.javatpoint.com/servletrequest)

[Methods of ServletRequest interface](https://www.javatpoint.com/servletrequest#methods)

[Example of ServletRequest interface](https://www.javatpoint.com/servletrequest#example)

[Displaying all the header information](https://www.javatpoint.com/displaying-all-the-header-information-in-the-servlet)

An object of ServletRequest is used to provide the client request information to a servlet such as content type, content length, parameter names and values, header informations, attributes etc.

### Methods of ServletRequest interface

There are many methods defined in the ServletRequest interface. Some of them are as follows:

|  |  |
| --- | --- |
| **Method** | **Description** |
| **public String getParameter(String name)** | is used to obtain the value of a parameter by name. |
| **public String[] getParameterValues(String name)** | returns an array of String containing all values of given parameter name. It is mainly used to obtain values of a Multi select list box. |
| **java.util.Enumeration getParameterNames()** | returns an enumeration of all of the request parameter names. |
| **public int getContentLength()** | Returns the size of the request entity data, or -1 if not known. |
| **public String getCharacterEncoding()** | Returns the character set encoding for the input of this request. |
| **public String getContentType()** | Returns the Internet Media Type of the request entity data, or null if not known. |
| **public ServletInputStream getInputStream() throws IOException** | Returns an input stream for reading binary data in the request body. |
| **public abstract String getServerName()** | Returns the host name of the server that received the request. |
| **public int getServerPort()** | Returns the port number on which this request was received. |

### **Example of ServletRequest to display the name of the user**

In this example, we are displaying the name of the user in the servlet. For this purpose, we have used the getParameter method that returns the value for the given request parameter name.

**index.html**

<form action="welcome" method="get">

Enter your name<input type="text" name="name"><br>

<input type="submit" value="login">

</form>

**DemoServ.java**

**import** javax.servlet.http.\*;

**import** javax.servlet.\*;

**import** java.io.\*;

**public** **class** DemoServ **extends** HttpServlet{

**public** **void** doGet(HttpServletRequest req,HttpServletResponse res)

**throws** ServletException,IOException

{

res.setContentType("text/html");

PrintWriter pw=res.getWriter();

String name=req.getParameter("name");//will return value

pw.println("Welcome "+name);

pw.close();

}}

### **Other examples of ServletRequest interface**

[**Example of ServletRequest to display all the header information**](https://www.javatpoint.com/displaying-all-the-header-information-in-the-servlet)

In this example, we are displaying the header information of the servlet such as content type, content length, user agent etc.

# Registration Example With DB Etc…

# **Example of Registration form in servlet**

Here, you will learn that how to create simple registration form in servlet. We are using oracle10g database. So you need to create a table first as given below:

CREATE TABLE  "REGISTERUSER"

   (    "NAME" VARCHAR2(4000),

    "PASS" VARCHAR2(4000),

    "EMAIL" VARCHAR2(4000),

    "COUNTRY" VARCHAR2(4000)

   )

/

To create the registration page in servlet, we can separate the database logic from the servlet. But here, we are mixing the database logic in the servlet only for simplicity of the program. We will develop this page in JSP following DAO, DTO and Singleton design pattern later.

### **Example of Registration form in servlet**

In this example, we have created the three pages.

register.html

Register.java

web.xml

**register.html**

In this page, we have getting input from the user using text fields and combobox. The information entered by the user is forwarded to Register servlet, which is responsible to store the data into the database.

<html>

<body>

<form action="servlet/Register" method="post">

Name:<input type="text" name="userName"/><br/><br/>

Password:<input type="password" name="userPass"/><br/><br/>

Email Id:<input type="text" name="userEmail"/><br/><br/>

Country:

<select name="userCountry">

<option>India</option>

<option>Pakistan</option>

<option>other</option>

</select>

<br/><br/>

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

</form>

</body>

</html>

**Register.java**

This servlet class receives all the data entered by user and stores it into the database. Here, we are performing the database logic. But you may separate it, which will be better for the web application.

**import** java.io.\*;

**import** java.sql.\*;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.\*;

**public** **class** Register **extends** HttpServlet {

**public** **void** doPost(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

String n=request.getParameter("userName");

String p=request.getParameter("userPass");

String e=request.getParameter("userEmail");

String c=request.getParameter("userCountry");

**try**{

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

Connection con=DriverManager.getConnection(

"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

PreparedStatement ps=con.prepareStatement(

"insert into registeruser values(?,?,?,?)");

ps.setString(1,n);

ps.setString(2,p);

ps.setString(3,e);

ps.setString(4,c);

**int** i=ps.executeUpdate();

**if**(i>0)

out.print("You are successfully registered...");

}**catch** (Exception e2) {System.out.println(e2);}

out.close();

}

}

**web.xml file**

The is the configuration file, providing information about the servlet.

<web-app>

<servlet>

<servlet-name>Register</servlet-name>

<servlet-**class**>Register</servlet-**class**>

</servlet>

<servlet-mapping>

<servlet-name>Register</servlet-name>

<url-pattern>/servlet/Register</url-pattern>

</servlet-mapping>

<welcome-file-list>

    <welcome-file>register.html</welcome-file>

</welcome-file-list>

</web-app>

To connect java application with the Oracle database ojdbc14.jar file is required to be loaded. Put this jar file in WEB-INF/lib folder.

# Servlet Collaboration

**What is Servlet Collaboration?**

The exchange of information among servlets of a particular Java web application is known as **Servlet Collaboration**. This enables passing/sharing information from one servlet to the other through method invocations.

**What are the principle ways provided by Java to achieve Servlet Collaboration?**  
The servlet api provides two interfaces namely:

javax.servlet.RequestDispatcher

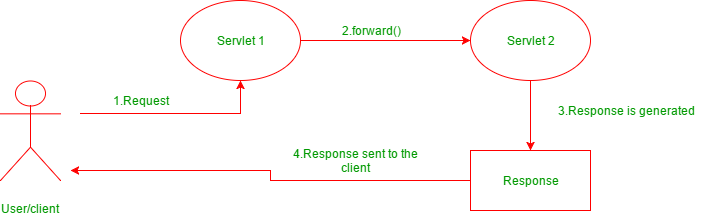
javax.servlet.http.HttpServletResponse

These two interfaces include the methods responsible for achieving the objective of sharing information between servlets.

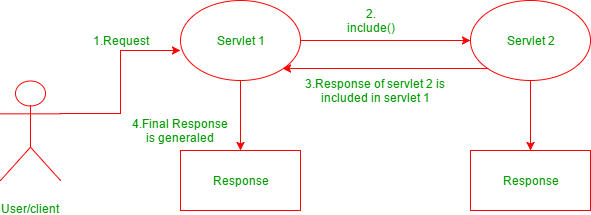
**Using RequestDispatcher Interface**

The RequestDispatcher interface provides the option of dispatching the client’s request to another web resource, which could be an HTML page, another servlet, JSP etc. It provides the following two methods:

**public void forward(ServletRequest request, ServletResponse response)throws ServletException, java.io.IOException:**  
The forward() method is used to transfer the client request to another resource (HTML file, servlet, jsp etc). When this method is called, the control is transferred to the next resource called. On the other hand, the include() method is used to include the content of the calling file into the called file. After calling this method, the control remains with the calling resource, but the processed output is included into the called resource.  
The following diagram explains the way it works:



**public void include(ServletRequest request, ServletResponse response)throws ServletException, java.io.IOException:**  
The include() method is used to include the contents of the calling resource into the called one. When this method is called, the control still remains with the calling resource. It simply includes the processed output of the calling resource into the called one.  
The following diagram explains how it works:



**Example of using RequestDispatcher for Servlet Collaboration**  
The following example explains how to use RequestDispatcher interface to achieve Servlet Collaboration:  
**index.html** 

html

|  |
| --- |
| <**html**>  <**head**>  <**body**>  <**form** action="login" method="post">  Name:<**input** type="text" name="userName"/><**br**/>  Password:<**input** type="password" name="userPass"/><**br**/>  <**input** type="submit" value="login"/>  </**form**>  </**body**>  </**html**> |

**Login.java**

Java

|  |
| --- |
| // First java servlet that calls another resource  **import** java.io.\*;  **import** javax.servlet.\*;  **import** javax.servlet.http.\*;  **public** **class** Login **extends** HttpServlet {  **public** **void** doPost(HttpServletRequest req,  HttpServletResponse res)  **throws** ServletException, IOException  {  // The method to receive client requests  // which are sent using 'post'  res.setContentType("text/html");  PrintWriter out = response.getWriter();  // fetches username  String n = request.getParameter("userName");  // fetches password  String p = request.getParameter("userPass");  **if**(p.equals("Thanos"){  RequestDispatcher rd = request.getRequestDispatcher("servlet2");  // Getting RequestDispatcher object  // for collaborating with servlet2  // forwarding the request to servlet2  rd.forward(request, response);  }  **else**{  out.print("Password mismatch");  RequestDispatcher rd = request.getRequestDispatcher("/index.html");  rd.include(request, response);  }  }  } |

**Welcome.java**

Java

|  |
| --- |
| // Called servlet in case password matches  **import** java.io.\*;  **import** javax.servlet.\*;  **import** javax.servlet.http.\*;  **public** **class** Welcome **extends** HttpServlet {  **public** **void** doPost(HttpServletRequest request,  HttpServletResponse response)  **throws** ServletException, IOException  {  response.setContentType("text/html");  PrintWriter out = response.getWriter();  // fetches username  String n = request.getParameter("userName");  // prints the message  out.print("Welcome " + n);  }  } |

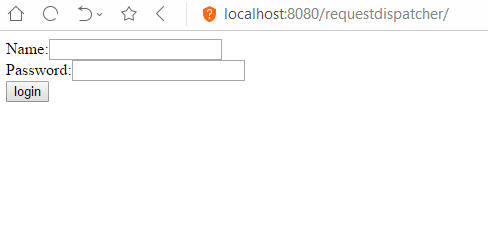
**web.xml**

html

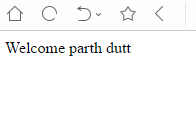
|  |
| --- |
| <**web-app**>  <**servlet**>  <**servlet-name**>Login</**servlet-name**>  <**servlet-class**>Login</**servlet-class**>  </**servlet**>  <**servlet**>  <**servlet-name**>WelcomeServlet</**servlet-name**>  <**servlet-class**>Welcome</**servlet-class**>  </**servlet**>  <**servlet-mapping**>  <**servlet-name**>Login</**servlet-name**>  <**url-pattern**>/servlet1</**url-pattern**>  </**servlet-mapping**>  <**servlet-mapping**>  <**servlet-name**>WelcomeServlet</**servlet-name**>  <**url-pattern**>/servlet2</**url-pattern**>  </**servlet-mapping**>  <**welcome-file-list**>  <**welcome-file**>index.html</**welcome-file**>  </**welcome-file-list**>  </**web-app**> |

**Output**:

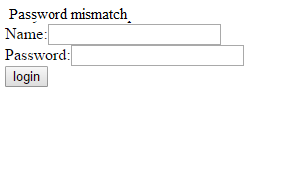
**index.html**



**If password matches:**



**If password doesn’t match:**



**Using HttpServletResponse Interface**

The HttpServletResponse interface is entrusted with managing Http responses. To achieve servlet collaboration, it uses the following method:

public void sendRedirect(String URL)throws IOException;

This method is used redirect response to another resource, which may be a servlet, jsp or an html file. The argument accepted by it, is a URL which can be both, absolute and relative. It works on the client side and uses the browser’s URL bar to make a request.

**Example of using sendRedirect() for redirection**

The following example of a web application created using servlet takes the text written in the text field in the webpage, and directs it to the servlet. The servlet then redirects it to google, which then produces search results based on the text written.  
**index.html**

html

|  |
| --- |
| <**html**>  <**head**>  <**body**>  <**form** action="search" method="GET">  <**input** type="text" name="name">  <**input** type="submit" value="search">  </**form**>  </**body**>  </**html**> |

Java

|  |
| --- |
| // Servlet class to redirect the text keyword  // in the 'name' field to google.com  // using sendRedirect()  **import** java.io.IOException;  **import** javax.servlet.ServletException;  **import** javax.servlet.http.HttpServlet;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **public** **class** MySearcher **extends** HttpServlet {  **protected** **void** doGet(HttpServletRequest  request,  HttpServletResponse response)  **throws** ServletException, IOException  {  String name = request.getParameter("name");  response.sendRedirect("<https://www.google.co.in/#q=>" + name);  // response redirected to google.com  }  } |

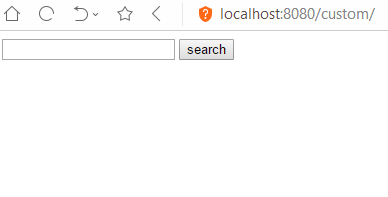
**web.xml**

html

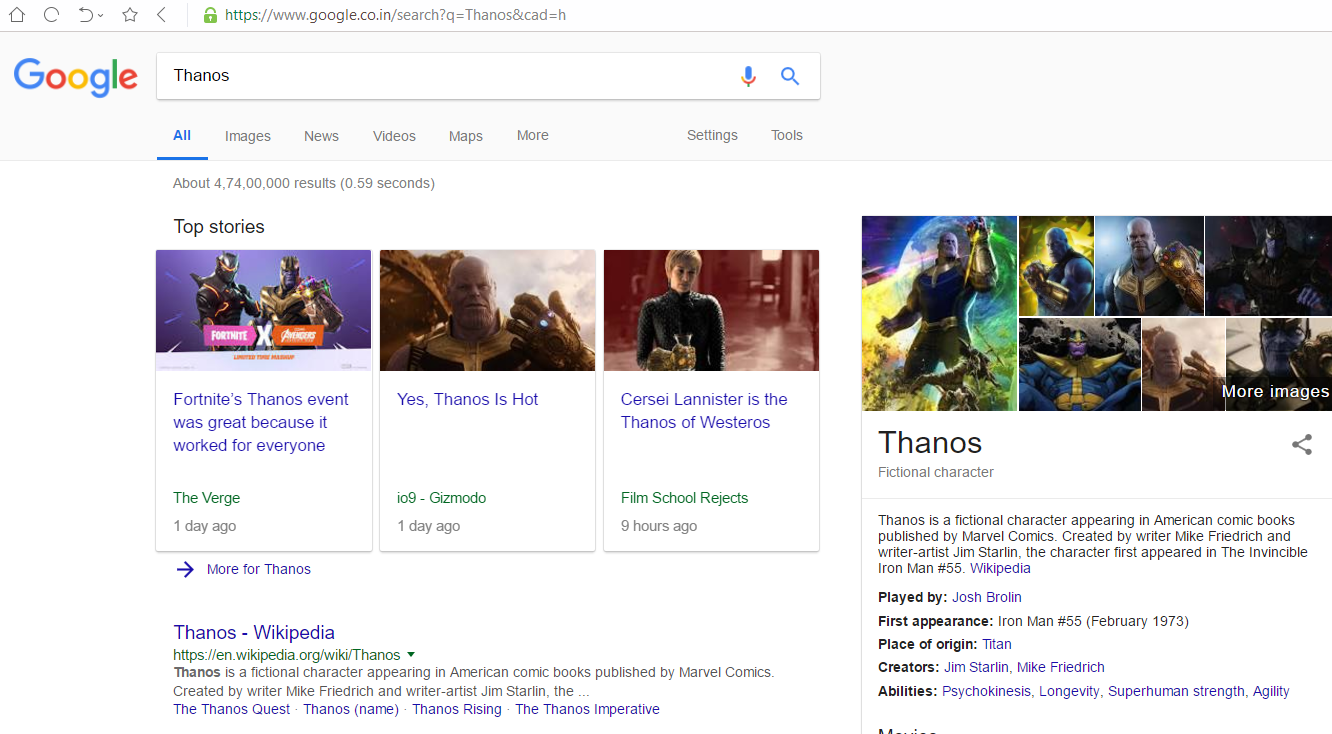
|  |
| --- |
| <**web-app**>  <**servlet**>  <**servlet-name**>MySearcher</**servlet-name**>  <**servlet-class**>MySearcher</**servlet-class**>  </**servlet**>  <**servlet-mapping**>  <**servlet-name**>MySearcher</**servlet-name**>  <**url-pattern**>/search</**url-pattern**>  </**servlet-mapping**>  <**welcome-file-list**>  <**welcome-file**>index.html</**welcome-file**>  </**welcome-file-list**>  </**web-app**> |

**Output:**

**index.html**



**Search result**



**What is the difference between forward() method of RequestDiispatcher and sendRedirect() of HttpServletResponse?**

Although the two methods appear to do the same thing, there are still differences between the two, which are as follows:

|  |  |
| --- | --- |
| **forward()** | **sendRedirect()** |
| It works on the server side | It works on the client side |
| It sends the same request and response objects to another resource. | It always send a new request |
| It works only within the server. | It can be used within and outside the server. |

# RequestDispatcher Interface

# **RequestDispatcher in Servlet**

[RequestDispatcher Interface](https://www.javatpoint.com/requestdispatcher-in-servlet)

[Methods of RequestDispatcher interface](https://www.javatpoint.com/requestdispatcher-in-servlet#rdmethod)

[forward method](https://www.javatpoint.com/requestdispatcher-in-servlet#rdforward)

[include method](https://www.javatpoint.com/requestdispatcher-in-servlet#rdinclude)

[How to get the object of RequestDispatcher](https://www.javatpoint.com/requestdispatcher-in-servlet#rdhow)

[Example of RequestDispatcher interface](https://www.javatpoint.com/requestdispatcher-in-servlet#rdex)

The RequestDispatcher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp. This interface can also be used to include the content of another resource also. It is one of the way of servlet collaboration.

There are two methods defined in the RequestDispatcher interface.

### **Methods of RequestDispatcher interface**

The RequestDispatcher interface provides two methods. They are:

**public void forward(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:**Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.

**public void include(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:**Includes the content of a resource (servlet, JSP page, or HTML file) in the response.



As you see in the above figure, response of second servlet is sent to the client. Response of the first servlet is not displayed to the user.



|  |
| --- |
| As you can see in the above figure, response of second servlet is included in the response of the first servlet that is being sent to the client. |

### **How to get the object of RequestDispatcher**

The getRequestDispatcher() method of ServletRequest interface returns the object of RequestDispatcher. Syntax:

#### **Syntax of getRequestDispatcher method**

**public** RequestDispatcher getRequestDispatcher(String resource);

#### **Example of using getRequestDispatcher method**

RequestDispatcher rd=request.getRequestDispatcher("servlet2");

//servlet2 is the url-pattern of the second servlet

rd.forward(request, response);//method may be include or forward

### **Example of RequestDispatcher interface**

In this example, we are validating the password entered by the user. If password is servlet, it will forward the request to the WelcomeServlet, otherwise will show an error message: sorry username or password error!. In this program, we are cheking for hardcoded information. But you can check it to the database also that we will see in the development chapter. In this example, we have created following files:

**index.html file:** for getting input from the user.

**Login.java file:** a servlet class for processing the response. If password is servet, it will forward the request to the welcome servlet.

**WelcomeServlet.java file:** a servlet class for displaying the welcome message.

**web.xml file:** a deployment descriptor file that contains the information about the servlet.



**index.html**

<form action="servlet1" method="post">

Name:<input type="text" name="userName"/><br/>

Password:<input type="password" name="userPass"/><br/>

<input type="submit" value="login"/>

</form>

**Login.java**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**public** **class** Login **extends** HttpServlet {

**public** **void** doPost(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

    response.setContentType("text/html");

    PrintWriter out = response.getWriter();

    String n=request.getParameter("userName");

    String p=request.getParameter("userPass");

**if**(p.equals("servlet"){

        RequestDispatcher rd=request.getRequestDispatcher("servlet2");

        rd.forward(request, response);

    }

**else**{

        out.print("Sorry UserName or Password Error!");

        RequestDispatcher rd=request.getRequestDispatcher("/index.html");

        rd.include(request, response);

        }

    }

}

**WelcomeServlet.java**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**public** **class** WelcomeServlet **extends** HttpServlet {

**public** **void** doPost(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

    response.setContentType("text/html");

    PrintWriter out = response.getWriter();

    String n=request.getParameter("userName");

    out.print("Welcome "+n);

    }

}

**web.xml**

<web-app>

 <servlet>

    <servlet-name>Login</servlet-name>

    <servlet-**class**>Login</servlet-**class**>

  </servlet>

  <servlet>

    <servlet-name>WelcomeServlet</servlet-name>

    <servlet-**class**>WelcomeServlet</servlet-**class**>

  </servlet>

  <servlet-mapping>

    <servlet-name>Login</servlet-name>

    <url-pattern>/servlet1</url-pattern>

  </servlet-mapping>

  <servlet-mapping>

    <servlet-name>WelcomeServlet</servlet-name>

    <url-pattern>/servlet2</url-pattern>

  </servlet-mapping>

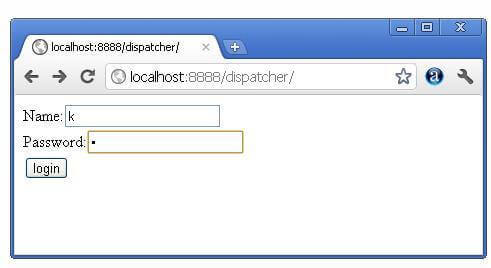
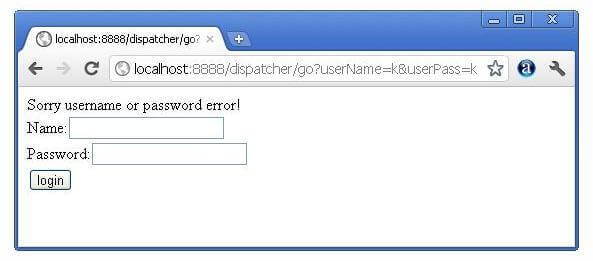
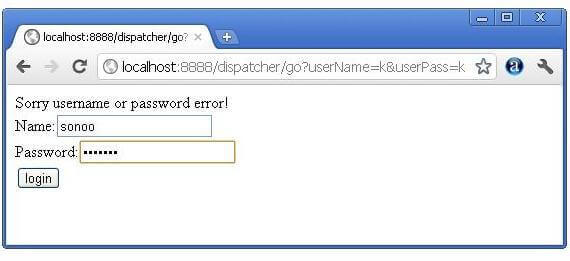
  <welcome-file-list>

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

  </welcome-file-list>

</web-app>

[download this example](https://static.javatpoint.com/src/servlet/requestdispatcher.zip)  
[download this example (developed in Myeclipse IDE)](https://static.javatpoint.com/src/servlet/requestdispatcherm.zip)  
[download this example (developed in eclipse IDE)](https://static.javatpoint.com/src/servlet/eclipse/requestdispatcher.zip)  
[download this example (developed in netbeans IDE)](https://static.javatpoint.com/src/servlet/netbeans/requestdispatcher.zip)

# SendRedirect Method

# **SendRedirect in servlet**

[sendRedirect method](https://www.javatpoint.com/sendRedirect()-method)

[Syntax of sendRedirect() method](https://www.javatpoint.com/sendRedirect()-method#redirectsyn)

[Example of RequestDispatcher interface](https://www.javatpoint.com/sendRedirect()-method#redirectex)

The **sendRedirect()** method of **HttpServletResponse** interface can be used to redirect response to another resource, it may be servlet, jsp or html file.

It accepts relative as well as absolute URL.

It works at client side because it uses the url bar of the browser to make another request. So, it can work inside and outside the server.

## **Difference between forward() and sendRedirect() method**

There are many differences between the forward() method of RequestDispatcher and sendRedirect() method of HttpServletResponse interface. They are given below:

|  |  |
| --- | --- |
| **forward() method** | **sendRedirect() method** |
| The forward() method works at server side. | The sendRedirect() method works at client side. |
| It sends the same request and response objects to another servlet. | It always sends a new request. |
| It can work within the server only. | It can be used within and outside the server. |
| Example: request.getRequestDispacher("servlet2").forward(request,response); | Example: response.sendRedirect("servlet2"); |

### **Syntax of sendRedirect() method**

**public** **void** sendRedirect(String URL)**throws** IOException;

### **Example of sendRedirect() method**

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

### **Full example of sendRedirect method in servlet**

|  |
| --- |
| In this example, we are redirecting the request to the google server. Notice that sendRedirect method works at client side, that is why we can our request to anywhere. We can send our request within and outside the server. |

*DemoServlet.java*

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**public** **class** DemoServlet **extends** HttpServlet{

**public** **void** doGet(HttpServletRequest req,HttpServletResponse res)

**throws** ServletException,IOException

{

res.setContentType("text/html");

PrintWriter pw=res.getWriter();

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

pw.close();

}}

### **Creating custom google search using sendRedirect**

In this example, we are using sendRedirect method to send request to google server with the request data.

*index.html*

<!DOCTYPE html**>**

**<html>**

**<head>**

**<meta** charset="ISO-8859-1"**>**

**<title>**sendRedirect example**</title>**

**</head>**

**<body>**

**<form** action="MySearcher"**>**

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

**<input** type="submit" value="Google Search"**>**

**</form>**

**</body>**

**</html>**

*MySearcher.java*

**import** java.io.IOException;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**public** **class** MySearcher **extends** HttpServlet {

**protected** **void** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

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

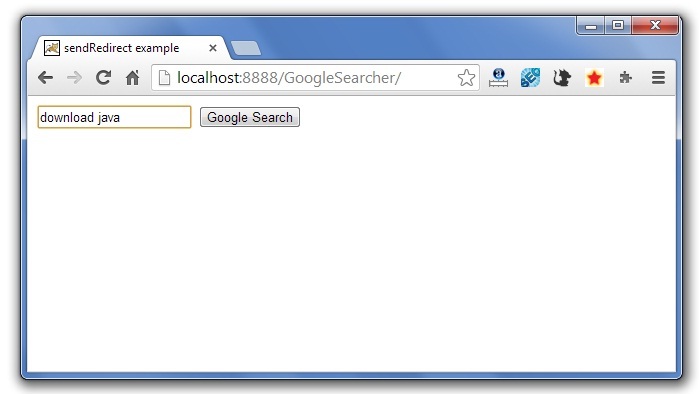
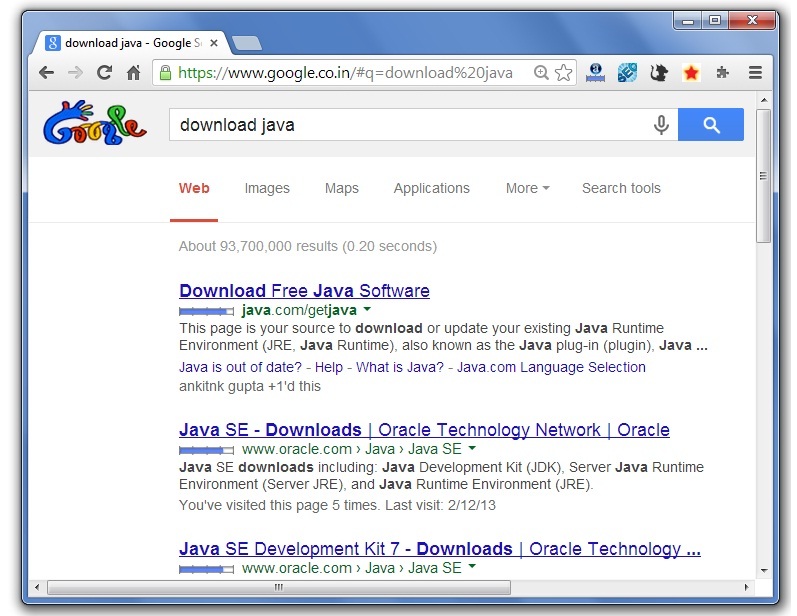
        response.sendRedirect("https://www.google.co.in/#q="+name);

    }

}

[download this example (developed in Eclipse)](https://static.javatpoint.com/src/servlet/eclipse/GoogleSearcher.zip)

#### **Output**

# **ServletConfig**

# **ServletConfig Interface**

[ServletConfig Interface](https://www.javatpoint.com/servletconfig)

[Methods of ServletConfig interface](https://www.javatpoint.com/servletconfig#configmethod)

[How to get the object of ServletConfig](https://www.javatpoint.com/servletconfig#configobject)

[Syntax to provide the initialization parameter for a servlet](https://www.javatpoint.com/servletconfig#configsyntax)

[Example of ServletConfig to get initialization parameter](https://www.javatpoint.com/servletconfig#configex1)

[Example of ServletConfig to get all the initialization parameter](https://www.javatpoint.com/servletconfig#configex2)

An object of ServletConfig is created by the web container for each servlet. This object can be used to get configuration information from web.xml file.

If the configuration information is modified from the web.xml file, we don't need to change the servlet. So it is easier to manage the web application if any specific content is modified from time to time.

### **Advantage of ServletConfig**

The core advantage of ServletConfig is that you don't need to edit the servlet file if information is modified from the web.xml file.

### **Methods of ServletConfig interface**

**public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.

**public Enumeration getInitParameterNames():**Returns an enumeration of all the initialization parameter names.

**public String getServletName():**Returns the name of the servlet.

**public ServletContext getServletContext():**Returns an object of ServletContext.

### **How to get the object of ServletConfig**

**getServletConfig() method** of Servlet interface returns the object of ServletConfig.

#### **Syntax of getServletConfig() method**

**public** ServletConfig getServletConfig();

### **Example of getServletConfig() method**

ServletConfig config=getServletConfig();

//Now we can call the methods of ServletConfig interface

### **Syntax to provide the initialization parameter for a servlet**

The init-param sub-element of servlet is used to specify the initialization parameter for a servlet.

<web-app>

  <servlet>

    ......

    <init-param>

      <param-name>parametername</param-name>

      <param-value>parametervalue</param-value>

    </init-param>

    ......

  </servlet>

</web-app>

### **Example of ServletConfig to get initialization parameter**

In this example, we are getting the one initialization parameter from the web.xml file and printing this information in the servlet.

**DemoServlet.java**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**public** **class** DemoServlet **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

    response.setContentType("text/html");

    PrintWriter out = response.getWriter();

    ServletConfig config=getServletConfig();

    String driver=config.getInitParameter("driver");

    out.print("Driver is: "+driver);

    out.close();

    }

}

**web.xml**

<web-app>

<servlet>

<servlet-name>DemoServlet</servlet-name>

<servlet-**class**>DemoServlet</servlet-**class**>

<init-param>

<param-name>driver</param-name>

<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>DemoServlet</servlet-name>

<url-pattern>/servlet1</url-pattern>

</servlet-mapping>

</web-app>

[download this example (developed in Myeclipse IDE)](https://static.javatpoint.com/src/servlet/config1.zip)  
[download this example(developed in Eclipse IDE)](https://static.javatpoint.com/src/servlet/eclipse/config1.zip)  
[download this example(developed in Netbeans IDE)](https://static.javatpoint.com/src/servlet/netbeans/config4.zip)

### **Example of ServletConfig to get all the initialization parameters**

In this example, we are getting all the initialization parameter from the web.xml file and printing this information in the servlet.

**DemoServlet.java**

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** java.util.Enumeration;

**import** javax.servlet.ServletConfig;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**public** **class** DemoServlet **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

    response.setContentType("text/html");

    PrintWriter out = response.getWriter();

    ServletConfig config=getServletConfig();

    Enumeration<String> e=config.getInitParameterNames();

    String str="";

**while**(e.hasMoreElements()){

    str=e.nextElement();

    out.print("<br>Name: "+str);

    out.print(" value: "+config.getInitParameter(str));

    }

    out.close();

}

}

**web.xml**

<web-app>

<servlet>

<servlet-name>DemoServlet</servlet-name>

<servlet-**class**>DemoServlet</servlet-**class**>

<init-param>

<param-name>username</param-name>

<param-value>system</param-value>

</init-param>

<init-param>

<param-name>password</param-name>

<param-value>oracle</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>DemoServlet</servlet-name>

<url-pattern>/servlet1</url-pattern>

</servlet-mapping>

</web-app>

# ServletConfig Methods

# **ServletConfig Interface**

[ServletConfig Interface](https://www.javatpoint.com/servletconfig)

[Methods of ServletConfig interface](https://www.javatpoint.com/servletconfig#configmethod)

[How to get the object of ServletConfig](https://www.javatpoint.com/servletconfig#configobject)

[Syntax to provide the initialization parameter for a servlet](https://www.javatpoint.com/servletconfig#configsyntax)

[Example of ServletConfig to get initialization parameter](https://www.javatpoint.com/servletconfig#configex1)

[Example of ServletConfig to get all the initialization parameter](https://www.javatpoint.com/servletconfig#configex2)

An object of ServletConfig is created by the web container for each servlet. This object can be used to get configuration information from web.xml file.

If the configuration information is modified from the web.xml file, we don't need to change the servlet. So it is easier to manage the web application if any specific content is modified from time to time.

### **Advantage of ServletConfig**

The core advantage of ServletConfig is that you don't need to edit the servlet file if information is modified from the web.xml file.

### **Methods of ServletConfig interface**

**public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.

**public Enumeration getInitParameterNames():**Returns an enumeration of all the initialization parameter names.

**public String getServletName():**Returns the name of the servlet.

**public ServletContext getServletContext():**Returns an object of ServletContext.

### **How to get the object of ServletConfig**

**getServletConfig() method** of Servlet interface returns the object of ServletConfig.

#### **Syntax of getServletConfig() method**

**public** ServletConfig getServletConfig();

### **Example of getServletConfig() method**

ServletConfig config=getServletConfig();

//Now we can call the methods of ServletConfig interface

### **Syntax to provide the initialization parameter for a servlet**

The init-param sub-element of servlet is used to specify the initialization parameter for a servlet.

<web-app>

  <servlet>

    ......

    <init-param>

      <param-name>parametername</param-name>

      <param-value>parametervalue</param-value>

    </init-param>

    ......

  </servlet>

</web-app>

### **Example of ServletConfig to get initialization parameter**

In this example, we are getting the one initialization parameter from the web.xml file and printing this information in the servlet.

**DemoServlet.java**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**public** **class** DemoServlet **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

    response.setContentType("text/html");

    PrintWriter out = response.getWriter();

    ServletConfig config=getServletConfig();

    String driver=config.getInitParameter("driver");

    out.print("Driver is: "+driver);

    out.close();

    }

}

**web.xml**

<web-app>

<servlet>

<servlet-name>DemoServlet</servlet-name>

<servlet-**class**>DemoServlet</servlet-**class**>

<init-param>

<param-name>driver</param-name>

<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>DemoServlet</servlet-name>

<url-pattern>/servlet1</url-pattern>

</servlet-mapping>

</web-app>

### **Example of ServletConfig to get all the initialization parameters**

In this example, we are getting all the initialization parameter from the web.xml file and printing this information in the servlet.

**DemoServlet.java**

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** java.util.Enumeration;

**import** javax.servlet.ServletConfig;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**public** **class** DemoServlet **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

    response.setContentType("text/html");

    PrintWriter out = response.getWriter();

    ServletConfig config=getServletConfig();

    Enumeration<String> e=config.getInitParameterNames();

    String str="";

**while**(e.hasMoreElements()){

    str=e.nextElement();

    out.print("<br>Name: "+str);

    out.print(" value: "+config.getInitParameter(str));

    }

    out.close();

}

}

**web.xml**

<web-app>

<servlet>

<servlet-name>DemoServlet</servlet-name>

<servlet-**class**>DemoServlet</servlet-**class**>

<init-param>

<param-name>username</param-name>

<param-value>system</param-value>

</init-param>

<init-param>

<param-name>password</param-name>

<param-value>oracle</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>DemoServlet</servlet-name>

<url-pattern>/servlet1</url-pattern>

</servlet-mapping>

</web-app>

# ServletConfig Example

# **ServletConfig Interface**

[ServletConfig Interface](https://www.javatpoint.com/servletconfig)

[Methods of ServletConfig interface](https://www.javatpoint.com/servletconfig#configmethod)

[How to get the object of ServletConfig](https://www.javatpoint.com/servletconfig#configobject)

[Syntax to provide the initialization parameter for a servlet](https://www.javatpoint.com/servletconfig#configsyntax)

[Example of ServletConfig to get initialization parameter](https://www.javatpoint.com/servletconfig#configex1)

[Example of ServletConfig to get all the initialization parameter](https://www.javatpoint.com/servletconfig#configex2)

An object of ServletConfig is created by the web container for each servlet. This object can be used to get configuration information from web.xml file.

If the configuration information is modified from the web.xml file, we don't need to change the servlet. So it is easier to manage the web application if any specific content is modified from time to time.

### **Advantage of ServletConfig**

The core advantage of ServletConfig is that you don't need to edit the servlet file if information is modified from the web.xml file.

### **Methods of ServletConfig interface**

**public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.

**public Enumeration getInitParameterNames():**Returns an enumeration of all the initialization parameter names.

**public String getServletName():**Returns the name of the servlet.

**public ServletContext getServletContext():**Returns an object of ServletContext.

### **How to get the object of ServletConfig**

**getServletConfig() method** of Servlet interface returns the object of ServletConfig.

#### **Syntax of getServletConfig() method**

**public** ServletConfig getServletConfig();

### **Example of getServletConfig() method**

ServletConfig config=getServletConfig();

//Now we can call the methods of ServletConfig interface

### **Syntax to provide the initialization parameter for a servlet**

The init-param sub-element of servlet is used to specify the initialization parameter for a servlet.

<web-app>

  <servlet>

    ......

    <init-param>

      <param-name>parametername</param-name>

      <param-value>parametervalue</param-value>

    </init-param>

    ......

  </servlet>

</web-app>

### **Example of ServletConfig to get initialization parameter**

In this example, we are getting the one initialization parameter from the web.xml file and printing this information in the servlet.

**DemoServlet.java**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**public** **class** DemoServlet **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

    response.setContentType("text/html");

    PrintWriter out = response.getWriter();

    ServletConfig config=getServletConfig();

    String driver=config.getInitParameter("driver");

    out.print("Driver is: "+driver);

    out.close();

    }

}

**web.xml**

<web-app>

<servlet>

<servlet-name>DemoServlet</servlet-name>

<servlet-**class**>DemoServlet</servlet-**class**>

<init-param>

<param-name>driver</param-name>

<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>DemoServlet</servlet-name>

<url-pattern>/servlet1</url-pattern>

</servlet-mapping>

</web-app>

### **Example of ServletConfig to get all the initialization parameters**

In this example, we are getting all the initialization parameter from the web.xml file and printing this information in the servlet.

**DemoServlet.java**

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** java.util.Enumeration;

**import** javax.servlet.ServletConfig;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**public** **class** DemoServlet **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

    response.setContentType("text/html");

    PrintWriter out = response.getWriter();

    ServletConfig config=getServletConfig();

    Enumeration<String> e=config.getInitParameterNames();

    String str="";

**while**(e.hasMoreElements()){

    str=e.nextElement();

    out.print("<br>Name: "+str);

    out.print(" value: "+config.getInitParameter(str));

    }

    out.close();

}

}

**web.xml**

<web-app>

<servlet>

<servlet-name>DemoServlet</servlet-name>

<servlet-**class**>DemoServlet</servlet-**class**>

<init-param>

<param-name>username</param-name>

<param-value>system</param-value>

</init-param>

<init-param>

<param-name>password</param-name>

<param-value>oracle</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>DemoServlet</servlet-name>

<url-pattern>/servlet1</url-pattern>

</servlet-mapping>

</web-app>

# ServletContext

# **ServletContext Interface**

[ServletContext Interface](https://www.javatpoint.com/servletcontext)

[Usage of ServletContext Interface](https://www.javatpoint.com/servletcontext#contextusage)

[Methods of ServletContext interface](https://www.javatpoint.com/servletcontext#contextmethods)

[How to get the object of ServletContext](https://www.javatpoint.com/servletcontext#contextobject)

[Syntax to provide the initialization parameter in Context scope](https://www.javatpoint.com/servletcontext#contextsyn)

[Example of ServletContext to get initialization parameter](https://www.javatpoint.com/servletcontext#contextex1)

[Example of ServletContext to get all the initialization parameter](https://www.javatpoint.com/servletcontext#contextex2)

An object of ServletContext is created by the web container at time of deploying the project. This object can be used to get configuration information from web.xml file. There is only one ServletContext object per web application.

If any information is shared to many servlet, it is better to provide it from the web.xml file using the **<context-param>** element.

### **Advantage of ServletContext**

**Easy to maintain** if any information is shared to all the servlet, it is better to make it available for all the servlet. We provide this information from the web.xml file, so if the information is changed, we don't need to modify the servlet. Thus it removes maintenance problem.

### **Usage of ServletContext Interface**

There can be a lot of usage of ServletContext object. Some of them are as follows:

The object of ServletContext provides an interface between the container and servlet.

The ServletContext object can be used to get configuration information from the web.xml file.

The ServletContext object can be used to set, get or remove attribute from the web.xml file.

The ServletContext object can be used to provide inter-application communication.



### **Commonly used methods of ServletContext interface**

|  |
| --- |
| There is given some commonly used methods of ServletContext interface.  **public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.  **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters.  **public void setAttribute(String name,Object object):**sets the given object in the application scope.  **public Object getAttribute(String name):**Returns the attribute for the specified name.  **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters as an Enumeration of String objects.  **public void removeAttribute(String name):**Removes the attribute with the given name from the servlet context. |

### **How to get the object of ServletContext interface**

**getServletContext() method** of ServletConfig interface returns the object of ServletContext.

**getServletContext() method** of GenericServlet class returns the object of ServletContext.

#### **Syntax of getServletContext() method**

**public** ServletContext getServletContext()

#### **Example of getServletContext() method**

//We can get the ServletContext object from ServletConfig object

ServletContext application=getServletConfig().getServletContext();

//Another convenient way to get the ServletContext object

ServletContext application=getServletContext();

### **Syntax to provide the initialization parameter in Context scope**

|  |
| --- |
| The **context-param** element, subelement of web-app, is used to define the initialization parameter in the application scope. The param-name and param-value are the sub-elements of the context-param. The param-name element defines parameter name and and param-value defines its value. |

<web-app>

 ......

  <context-param>

    <param-name>parametername</param-name>

    <param-value>parametervalue</param-value>

  </context-param>

 ......

</web-app>

### **Example of ServletContext to get the initialization parameter**

|  |
| --- |
| In this example, we are getting the initialization parameter from the web.xml file and printing the value of the initialization parameter. Notice that the object of ServletContext represents the application scope. So if we change the value of the parameter from the web.xml file, all the servlet classes will get the changed value. So we don't need to modify the servlet. So it is better to have the common information for most of the servlets in the web.xml file by context-param element. Let's see the simple example: |

**DemoServlet.java**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**public** **class** DemoServlet **extends** HttpServlet{

**public** **void** doGet(HttpServletRequest req,HttpServletResponse res)

**throws** ServletException,IOException

{

res.setContentType("text/html");

PrintWriter pw=res.getWriter();

//creating ServletContext object

ServletContext context=getServletContext();

//Getting the value of the initialization parameter and printing it

String driverName=context.getInitParameter("dname");

pw.println("driver name is="+driverName);

pw.close();

}}

**web.xml**

<web-app>

<servlet>

<servlet-name>sonoojaiswal</servlet-name>

<servlet-**class**>DemoServlet</servlet-**class**>

</servlet>

<context-param>

<param-name>dname</param-name>

<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>

</context-param>

<servlet-mapping>

<servlet-name>sonoojaiswal</servlet-name>

<url-pattern>/context</url-pattern>

</servlet-mapping>

</web-app>

### **Example of ServletContext to get all the initialization parameters**

|  |
| --- |
| In this example, we are getting all the initialization parameter from the web.xml file. For getting all the parameters, we have used the getInitParameterNames() method in the servlet class. |

**DemoServlet.java**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**public** **class** DemoServlet **extends** HttpServlet{

**public** **void** doGet(HttpServletRequest req,HttpServletResponse res)

**throws** ServletException,IOException

{

res.setContentType("text/html");

PrintWriter out=res.getWriter();

ServletContext context=getServletContext();

Enumeration<String> e=context.getInitParameterNames();

String str="";

**while**(e.hasMoreElements()){

    str=e.nextElement();

    out.print("<br> "+context.getInitParameter(str));

}

}}

**web.xml**

<web-app>

<servlet>

<servlet-name>sonoojaiswal</servlet-name>

<servlet-**class**>DemoServlet</servlet-**class**>

</servlet>

<context-param>

<param-name>dname</param-name>

<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>

</context-param>

<context-param>

<param-name>username</param-name>

<param-value>system</param-value>

</context-param>

<context-param>

<param-name>password</param-name>

<param-value>oracle</param-value>

</context-param>

<servlet-mapping>

<servlet-name>sonoojaiswal</servlet-name>

<url-pattern>/context</url-pattern>

</servlet-mapping>

</web-app>

# RequestDispatcher Interface

# **RequestDispatcher in Servlet**

[RequestDispatcher Interface](https://www.javatpoint.com/requestdispatcher-in-servlet)

[Methods of RequestDispatcher interface](https://www.javatpoint.com/requestdispatcher-in-servlet#rdmethod)

[forward method](https://www.javatpoint.com/requestdispatcher-in-servlet#rdforward)

[include method](https://www.javatpoint.com/requestdispatcher-in-servlet#rdinclude)

[How to get the object of RequestDispatcher](https://www.javatpoint.com/requestdispatcher-in-servlet#rdhow)

[Example of RequestDispatcher interface](https://www.javatpoint.com/requestdispatcher-in-servlet#rdex)

The RequestDispatcher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp. This interface can also be used to include the content of another resource also. It is one of the way of servlet collaboration.

There are two methods defined in the RequestDispatcher interface.

### **Methods of RequestDispatcher interface**

The RequestDispatcher interface provides two methods. They are:

**public void forward(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:**Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.

**public void include(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:**Includes the content of a resource (servlet, JSP page, or HTML file) in the response.



As you see in the above figure, response of second servlet is sent to the client. Response of the first servlet is not displayed to the user.



|  |
| --- |
| As you can see in the above figure, response of second servlet is included in the response of the first servlet that is being sent to the client. |

### **How to get the object of RequestDispatcher**

The getRequestDispatcher() method of ServletRequest interface returns the object of RequestDispatcher. Syntax:

#### **Syntax of getRequestDispatcher method**

**public** RequestDispatcher getRequestDispatcher(String resource);

#### **Example of using getRequestDispatcher method**

RequestDispatcher rd=request.getRequestDispatcher("servlet2");

//servlet2 is the url-pattern of the second servlet

rd.forward(request, response);//method may be include or forward

### **Example of RequestDispatcher interface**

In this example, we are validating the password entered by the user. If password is servlet, it will forward the request to the WelcomeServlet, otherwise will show an error message: sorry username or password error!. In this program, we are cheking for hardcoded information. But you can check it to the database also that we will see in the development chapter. In this example, we have created following files:

**index.html file:** for getting input from the user.

**Login.java file:** a servlet class for processing the response. If password is servet, it will forward the request to the welcome servlet.

**WelcomeServlet.java file:** a servlet class for displaying the welcome message.

**web.xml file:** a deployment descriptor file that contains the information about the servlet.



**index.html**

<form action="servlet1" method="post">

Name:<input type="text" name="userName"/><br/>

Password:<input type="password" name="userPass"/><br/>

<input type="submit" value="login"/>

</form>

**Login.java**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**public** **class** Login **extends** HttpServlet {

**public** **void** doPost(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

    response.setContentType("text/html");

    PrintWriter out = response.getWriter();

    String n=request.getParameter("userName");

    String p=request.getParameter("userPass");

**if**(p.equals("servlet"){

        RequestDispatcher rd=request.getRequestDispatcher("servlet2");

        rd.forward(request, response);

    }

**else**{

        out.print("Sorry UserName or Password Error!");

        RequestDispatcher rd=request.getRequestDispatcher("/index.html");

        rd.include(request, response);

        }

    }

}

**WelcomeServlet.java**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**public** **class** WelcomeServlet **extends** HttpServlet {

**public** **void** doPost(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

    response.setContentType("text/html");

    PrintWriter out = response.getWriter();

    String n=request.getParameter("userName");

    out.print("Welcome "+n);

    }

}

**web.xml**

<web-app>

 <servlet>

    <servlet-name>Login</servlet-name>

    <servlet-**class**>Login</servlet-**class**>

  </servlet>

  <servlet>

    <servlet-name>WelcomeServlet</servlet-name>

    <servlet-**class**>WelcomeServlet</servlet-**class**>

  </servlet>

  <servlet-mapping>

    <servlet-name>Login</servlet-name>

    <url-pattern>/servlet1</url-pattern>

  </servlet-mapping>

  <servlet-mapping>

    <servlet-name>WelcomeServlet</servlet-name>

    <url-pattern>/servlet2</url-pattern>

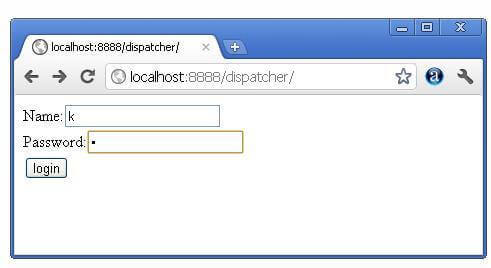
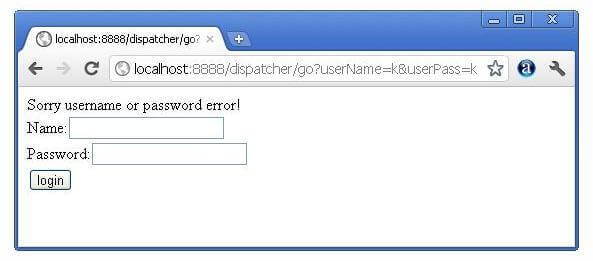
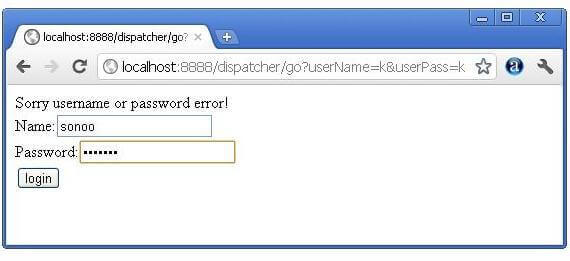
  </servlet-mapping>

  <welcome-file-list>

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

  </welcome-file-list>

</web-app>

# SendRedirect Method

# **SendRedirect in servlet**

[sendRedirect method](https://www.javatpoint.com/sendRedirect()-method)

[Syntax of sendRedirect() method](https://www.javatpoint.com/sendRedirect()-method#redirectsyn)

[Example of RequestDispatcher interface](https://www.javatpoint.com/sendRedirect()-method#redirectex)

The **sendRedirect()** method of **HttpServletResponse** interface can be used to redirect response to another resource, it may be servlet, jsp or html file.

It accepts relative as well as absolute URL.

It works at client side because it uses the url bar of the browser to make another request. So, it can work inside and outside the server.

## **Difference between forward() and sendRedirect() method**

There are many differences between the forward() method of RequestDispatcher and sendRedirect() method of HttpServletResponse interface. They are given below:

|  |  |
| --- | --- |
| **forward() method** | **sendRedirect() method** |
| The forward() method works at server side. | The sendRedirect() method works at client side. |
| It sends the same request and response objects to another servlet. | It always sends a new request. |
| It can work within the server only. | It can be used within and outside the server. |
| Example: request.getRequestDispacher("servlet2").forward(request,response); | Example: response.sendRedirect("servlet2"); |

### **Syntax of sendRedirect() method**

**public** **void** sendRedirect(String URL)**throws** IOException;

### **Example of sendRedirect() method**

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

### **Full example of sendRedirect method in servlet**

|  |
| --- |
| In this example, we are redirecting the request to the google server. Notice that sendRedirect method works at client side, that is why we can our request to anywhere. We can send our request within and outside the server. |

*DemoServlet.java*

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**public** **class** DemoServlet **extends** HttpServlet{

**public** **void** doGet(HttpServletRequest req,HttpServletResponse res)

**throws** ServletException,IOException

{

res.setContentType("text/html");

PrintWriter pw=res.getWriter();

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

pw.close();

}}

### **Creating custom google search using sendRedirect**

In this example, we are using sendRedirect method to send request to google server with the request data.

*index.html*

<!DOCTYPE html**>**

**<html>**

**<head>**

**<meta** charset="ISO-8859-1"**>**

**<title>**sendRedirect example**</title>**

**</head>**

**<body>**

**<form** action="MySearcher"**>**

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

**<input** type="submit" value="Google Search"**>**

**</form>**

**</body>**

**</html>**

*MySearcher.java*

**import** java.io.IOException;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**public** **class** MySearcher **extends** HttpServlet {

**protected** **void** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

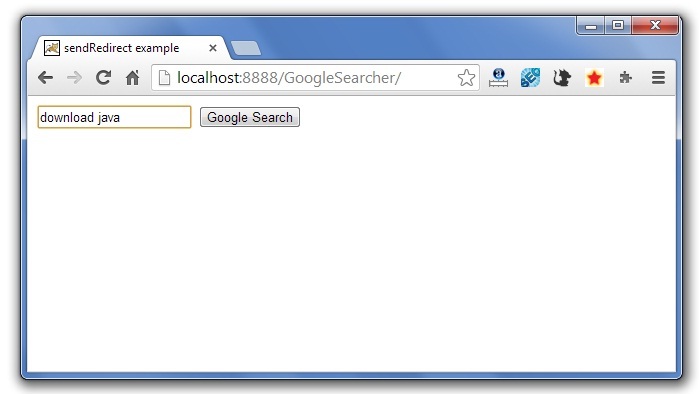
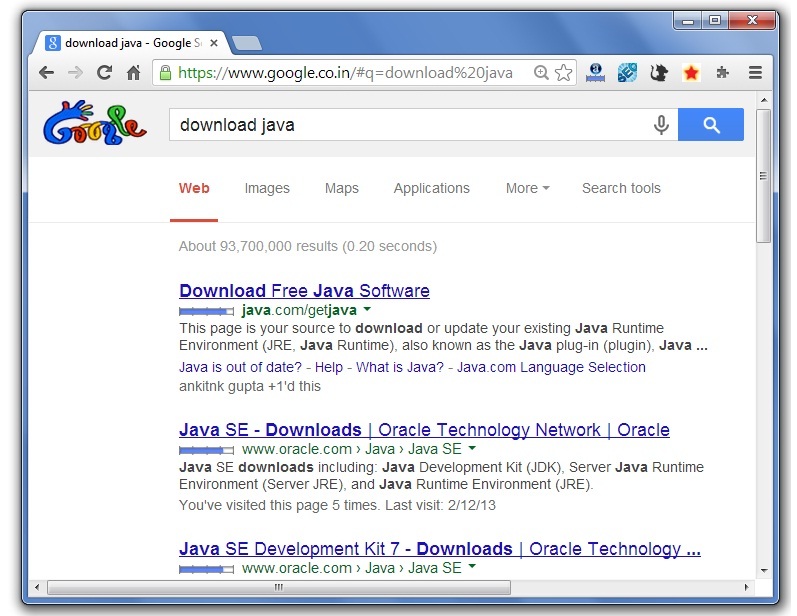
        String name=request.getParameter("name");

        response.sendRedirect("https://www.google.co.in/#q="+name);

    }

}

#### **Output**

# ServletConfig

# **ServletConfig Interface**

[ServletConfig Interface](https://www.javatpoint.com/servletconfig)

[Methods of ServletConfig interface](https://www.javatpoint.com/servletconfig#configmethod)

[How to get the object of ServletConfig](https://www.javatpoint.com/servletconfig#configobject)

[Syntax to provide the initialization parameter for a servlet](https://www.javatpoint.com/servletconfig#configsyntax)

[Example of ServletConfig to get initialization parameter](https://www.javatpoint.com/servletconfig#configex1)

[Example of ServletConfig to get all the initialization parameter](https://www.javatpoint.com/servletconfig#configex2)

An object of ServletConfig is created by the web container for each servlet. This object can be used to get configuration information from web.xml file.

If the configuration information is modified from the web.xml file, we don't need to change the servlet. So it is easier to manage the web application if any specific content is modified from time to time.

### **Advantage of ServletConfig**

The core advantage of ServletConfig is that you don't need to edit the servlet file if information is modified from the web.xml file.

### **Methods of ServletConfig interface**

**public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.

**public Enumeration getInitParameterNames():**Returns an enumeration of all the initialization parameter names.

**public String getServletName():**Returns the name of the servlet.

**public ServletContext getServletContext():**Returns an object of ServletContext.

### **How to get the object of ServletConfig**

**getServletConfig() method** of Servlet interface returns the object of ServletConfig.

#### **Syntax of getServletConfig() method**

**public** ServletConfig getServletConfig();

### **Example of getServletConfig() method**

ServletConfig config=getServletConfig();

//Now we can call the methods of ServletConfig interface

### **Syntax to provide the initialization parameter for a servlet**

The init-param sub-element of servlet is used to specify the initialization parameter for a servlet.

<web-app>

  <servlet>

    ......

    <init-param>

      <param-name>parametername</param-name>

      <param-value>parametervalue</param-value>

    </init-param>

    ......

  </servlet>

</web-app>

### **Example of ServletConfig to get initialization parameter**

In this example, we are getting the one initialization parameter from the web.xml file and printing this information in the servlet.

**DemoServlet.java**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**public** **class** DemoServlet **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

    response.setContentType("text/html");

    PrintWriter out = response.getWriter();

    ServletConfig config=getServletConfig();

    String driver=config.getInitParameter("driver");

    out.print("Driver is: "+driver);

    out.close();

    }

}

**web.xml**

<web-app>

<servlet>

<servlet-name>DemoServlet</servlet-name>

<servlet-**class**>DemoServlet</servlet-**class**>

<init-param>

<param-name>driver</param-name>

<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>DemoServlet</servlet-name>

<url-pattern>/servlet1</url-pattern>

</servlet-mapping>

</web-app>

### **Example of ServletConfig to get all the initialization parameters**

In this example, we are getting all the initialization parameter from the web.xml file and printing this information in the servlet.

**DemoServlet.java**

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** java.util.Enumeration;

**import** javax.servlet.ServletConfig;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**public** **class** DemoServlet **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

    response.setContentType("text/html");

    PrintWriter out = response.getWriter();

    ServletConfig config=getServletConfig();

    Enumeration<String> e=config.getInitParameterNames();

    String str="";

**while**(e.hasMoreElements()){

    str=e.nextElement();

    out.print("<br>Name: "+str);

    out.print(" value: "+config.getInitParameter(str));

    }

    out.close();

}

}

**web.xml**

<web-app>

<servlet>

<servlet-name>DemoServlet</servlet-name>

<servlet-**class**>DemoServlet</servlet-**class**>

<init-param>

<param-name>username</param-name>

<param-value>system</param-value>

</init-param>

<init-param>

<param-name>password</param-name>

<param-value>oracle</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>DemoServlet</servlet-name>

<url-pattern>/servlet1</url-pattern>

</servlet-mapping>

</web-app>

# ServletConfig Methods

# **ServletConfig Interface**

[ServletConfig Interface](https://www.javatpoint.com/servletconfig)

[Methods of ServletConfig interface](https://www.javatpoint.com/servletconfig#configmethod)

[How to get the object of ServletConfig](https://www.javatpoint.com/servletconfig#configobject)

[Syntax to provide the initialization parameter for a servlet](https://www.javatpoint.com/servletconfig#configsyntax)

[Example of ServletConfig to get initialization parameter](https://www.javatpoint.com/servletconfig#configex1)

[Example of ServletConfig to get all the initialization parameter](https://www.javatpoint.com/servletconfig#configex2)

An object of ServletConfig is created by the web container for each servlet. This object can be used to get configuration information from web.xml file.

If the configuration information is modified from the web.xml file, we don't need to change the servlet. So it is easier to manage the web application if any specific content is modified from time to time.

### **Advantage of ServletConfig**

The core advantage of ServletConfig is that you don't need to edit the servlet file if information is modified from the web.xml file.

### **Methods of ServletConfig interface**

**public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.

**public Enumeration getInitParameterNames():**Returns an enumeration of all the initialization parameter names.

**public String getServletName():**Returns the name of the servlet.

**public ServletContext getServletContext():**Returns an object of ServletContext.

### **How to get the object of ServletConfig**

**getServletConfig() method** of Servlet interface returns the object of ServletConfig.

#### **Syntax of getServletConfig() method**

**public** ServletConfig getServletConfig();

### **Example of getServletConfig() method**

ServletConfig config=getServletConfig();

//Now we can call the methods of ServletConfig interface

### **Syntax to provide the initialization parameter for a servlet**

The init-param sub-element of servlet is used to specify the initialization parameter for a servlet.

<web-app>

  <servlet>

    ......

    <init-param>

      <param-name>parametername</param-name>

      <param-value>parametervalue</param-value>

    </init-param>

    ......

  </servlet>

</web-app>

### **Example of ServletConfig to get initialization parameter**

In this example, we are getting the one initialization parameter from the web.xml file and printing this information in the servlet.

**DemoServlet.java**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**public** **class** DemoServlet **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

    response.setContentType("text/html");

    PrintWriter out = response.getWriter();

    ServletConfig config=getServletConfig();

    String driver=config.getInitParameter("driver");

    out.print("Driver is: "+driver);

    out.close();

    }

}

**web.xml**

<web-app>

<servlet>

<servlet-name>DemoServlet</servlet-name>

<servlet-**class**>DemoServlet</servlet-**class**>

<init-param>

<param-name>driver</param-name>

<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>DemoServlet</servlet-name>

<url-pattern>/servlet1</url-pattern>

</servlet-mapping>

</web-app>

### **Example of ServletConfig to get all the initialization parameters**

In this example, we are getting all the initialization parameter from the web.xml file and printing this information in the servlet.

**DemoServlet.java**

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** java.util.Enumeration;

**import** javax.servlet.ServletConfig;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**public** **class** DemoServlet **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

    response.setContentType("text/html");

    PrintWriter out = response.getWriter();

    ServletConfig config=getServletConfig();

    Enumeration<String> e=config.getInitParameterNames();

    String str="";

**while**(e.hasMoreElements()){

    str=e.nextElement();

    out.print("<br>Name: "+str);

    out.print(" value: "+config.getInitParameter(str));

    }

    out.close();

}

}

**web.xml**

<web-app>

<servlet>

<servlet-name>DemoServlet</servlet-name>

<servlet-**class**>DemoServlet</servlet-**class**>

<init-param>

<param-name>username</param-name>

<param-value>system</param-value>

</init-param>

<init-param>

<param-name>password</param-name>

<param-value>oracle</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>DemoServlet</servlet-name>

<url-pattern>/servlet1</url-pattern>

</servlet-mapping>

</web-app>

# ServletConfig Example

# **ServletConfig Interface**

[ServletConfig Interface](https://www.javatpoint.com/servletconfig)

[Methods of ServletConfig interface](https://www.javatpoint.com/servletconfig#configmethod)

[How to get the object of ServletConfig](https://www.javatpoint.com/servletconfig#configobject)

[Syntax to provide the initialization parameter for a servlet](https://www.javatpoint.com/servletconfig#configsyntax)

[Example of ServletConfig to get initialization parameter](https://www.javatpoint.com/servletconfig#configex1)

[Example of ServletConfig to get all the initialization parameter](https://www.javatpoint.com/servletconfig#configex2)

An object of ServletConfig is created by the web container for each servlet. This object can be used to get configuration information from web.xml file.

If the configuration information is modified from the web.xml file, we don't need to change the servlet. So it is easier to manage the web application if any specific content is modified from time to time.

### **Advantage of ServletConfig**

The core advantage of ServletConfig is that you don't need to edit the servlet file if information is modified from the web.xml file.

### **Methods of ServletConfig interface**

**public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.

**public Enumeration getInitParameterNames():**Returns an enumeration of all the initialization parameter names.

**public String getServletName():**Returns the name of the servlet.

**public ServletContext getServletContext():**Returns an object of ServletContext.

### **How to get the object of ServletConfig**

**getServletConfig() method** of Servlet interface returns the object of ServletConfig.

#### **Syntax of getServletConfig() method**

**public** ServletConfig getServletConfig();

### **Example of getServletConfig() method**

ServletConfig config=getServletConfig();

//Now we can call the methods of ServletConfig interface

### **Syntax to provide the initialization parameter for a servlet**

The init-param sub-element of servlet is used to specify the initialization parameter for a servlet.

<web-app>

  <servlet>

    ......

    <init-param>

      <param-name>parametername</param-name>

      <param-value>parametervalue</param-value>

    </init-param>

    ......

  </servlet>

</web-app>

### **Example of ServletConfig to get initialization parameter**

In this example, we are getting the one initialization parameter from the web.xml file and printing this information in the servlet.

**DemoServlet.java**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**public** **class** DemoServlet **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

    response.setContentType("text/html");

    PrintWriter out = response.getWriter();

    ServletConfig config=getServletConfig();

    String driver=config.getInitParameter("driver");

    out.print("Driver is: "+driver);

    out.close();

    }

}

**web.xml**

<web-app>

<servlet>

<servlet-name>DemoServlet</servlet-name>

<servlet-**class**>DemoServlet</servlet-**class**>

<init-param>

<param-name>driver</param-name>

<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>DemoServlet</servlet-name>

<url-pattern>/servlet1</url-pattern>

</servlet-mapping>

</web-app>

### **Example of ServletConfig to get all the initialization parameters**

In this example, we are getting all the initialization parameter from the web.xml file and printing this information in the servlet.

**DemoServlet.java**

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** java.util.Enumeration;

**import** javax.servlet.ServletConfig;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**public** **class** DemoServlet **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

    response.setContentType("text/html");

    PrintWriter out = response.getWriter();

    ServletConfig config=getServletConfig();

    Enumeration<String> e=config.getInitParameterNames();

    String str="";

**while**(e.hasMoreElements()){

    str=e.nextElement();

    out.print("<br>Name: "+str);

    out.print(" value: "+config.getInitParameter(str));

    }

    out.close();

}

}

**web.xml**

<web-app>

<servlet>

<servlet-name>DemoServlet</servlet-name>

<servlet-**class**>DemoServlet</servlet-**class**>

<init-param>

<param-name>username</param-name>

<param-value>system</param-value>

</init-param>

<init-param>

<param-name>password</param-name>

<param-value>oracle</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>DemoServlet</servlet-name>

<url-pattern>/servlet1</url-pattern>

</servlet-mapping>

</web-app>

# ServletContext

# **ServletContext Interface**

[ServletContext Interface](https://www.javatpoint.com/servletcontext)

[Usage of ServletContext Interface](https://www.javatpoint.com/servletcontext#contextusage)

[Methods of ServletContext interface](https://www.javatpoint.com/servletcontext#contextmethods)

[How to get the object of ServletContext](https://www.javatpoint.com/servletcontext#contextobject)

[Syntax to provide the initialization parameter in Context scope](https://www.javatpoint.com/servletcontext#contextsyn)

[Example of ServletContext to get initialization parameter](https://www.javatpoint.com/servletcontext#contextex1)

[Example of ServletContext to get all the initialization parameter](https://www.javatpoint.com/servletcontext#contextex2)

An object of ServletContext is created by the web container at time of deploying the project. This object can be used to get configuration information from web.xml file. There is only one ServletContext object per web application.

If any information is shared to many servlet, it is better to provide it from the web.xml file using the **<context-param>** element.

### **Advantage of ServletContext**

**Easy to maintain** if any information is shared to all the servlet, it is better to make it available for all the servlet. We provide this information from the web.xml file, so if the information is changed, we don't need to modify the servlet. Thus it removes maintenance problem.

### **Usage of ServletContext Interface**

There can be a lot of usage of ServletContext object. Some of them are as follows:

The object of ServletContext provides an interface between the container and servlet.

The ServletContext object can be used to get configuration information from the web.xml file.

The ServletContext object can be used to set, get or remove attribute from the web.xml file.

The ServletContext object can be used to provide inter-application communication.



### **Commonly used methods of ServletContext interface**

|  |
| --- |
| There is given some commonly used methods of ServletContext interface.  **public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.  **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters.  **public void setAttribute(String name,Object object):**sets the given object in the application scope.  **public Object getAttribute(String name):**Returns the attribute for the specified name.  **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters as an Enumeration of String objects.  **public void removeAttribute(String name):**Removes the attribute with the given name from the servlet context. |

### **How to get the object of ServletContext interface**

**getServletContext() method** of ServletConfig interface returns the object of ServletContext.

**getServletContext() method** of GenericServlet class returns the object of ServletContext.

#### **Syntax of getServletContext() method**

**public** ServletContext getServletContext()

#### **Example of getServletContext() method**

//We can get the ServletContext object from ServletConfig object

ServletContext application=getServletConfig().getServletContext();

//Another convenient way to get the ServletContext object

ServletContext application=getServletContext();

### **Syntax to provide the initialization parameter in Context scope**

|  |
| --- |
| The **context-param** element, subelement of web-app, is used to define the initialization parameter in the application scope. The param-name and param-value are the sub-elements of the context-param. The param-name element defines parameter name and and param-value defines its value. |

<web-app>

 ......

  <context-param>

    <param-name>parametername</param-name>

    <param-value>parametervalue</param-value>

  </context-param>

 ......

</web-app>

### **Example of ServletContext to get the initialization parameter**

|  |
| --- |
| In this example, we are getting the initialization parameter from the web.xml file and printing the value of the initialization parameter. Notice that the object of ServletContext represents the application scope. So if we change the value of the parameter from the web.xml file, all the servlet classes will get the changed value. So we don't need to modify the servlet. So it is better to have the common information for most of the servlets in the web.xml file by context-param element. Let's see the simple example: |

**DemoServlet.java**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**public** **class** DemoServlet **extends** HttpServlet{

**public** **void** doGet(HttpServletRequest req,HttpServletResponse res)

**throws** ServletException,IOException

{

res.setContentType("text/html");

PrintWriter pw=res.getWriter();

//creating ServletContext object

ServletContext context=getServletContext();

//Getting the value of the initialization parameter and printing it

String driverName=context.getInitParameter("dname");

pw.println("driver name is="+driverName);

pw.close();

}}

**web.xml**

<web-app>

<servlet>

<servlet-name>sonoojaiswal</servlet-name>

<servlet-**class**>DemoServlet</servlet-**class**>

</servlet>

<context-param>

<param-name>dname</param-name>

<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>

</context-param>

<servlet-mapping>

<servlet-name>sonoojaiswal</servlet-name>

<url-pattern>/context</url-pattern>

</servlet-mapping>

</web-app>

### **Example of ServletContext to get all the initialization parameters**

|  |
| --- |
| In this example, we are getting all the initialization parameter from the web.xml file. For getting all the parameters, we have used the getInitParameterNames() method in the servlet class. |

**DemoServlet.java**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**public** **class** DemoServlet **extends** HttpServlet{

**public** **void** doGet(HttpServletRequest req,HttpServletResponse res)

**throws** ServletException,IOException

{

res.setContentType("text/html");

PrintWriter out=res.getWriter();

ServletContext context=getServletContext();

Enumeration<String> e=context.getInitParameterNames();

String str="";

**while**(e.hasMoreElements()){

    str=e.nextElement();

    out.print("<br> "+context.getInitParameter(str));

}

}}

**web.xml**

<web-app>

<servlet>

<servlet-name>sonoojaiswal</servlet-name>

<servlet-**class**>DemoServlet</servlet-**class**>

</servlet>

<context-param>

<param-name>dname</param-name>

<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>

</context-param>

<context-param>

<param-name>username</param-name>

<param-value>system</param-value>

</context-param>

<context-param>

<param-name>password</param-name>

<param-value>oracle</param-value>

</context-param>

<servlet-mapping>

<servlet-name>sonoojaiswal</servlet-name>

<url-pattern>/context</url-pattern>

</servlet-mapping>

</web-app>

# ServletContext Methods In Application Scope

# **ServletContext Interface**

[ServletContext Interface](https://www.javatpoint.com/servletcontext)

[Usage of ServletContext Interface](https://www.javatpoint.com/servletcontext#contextusage)

[Methods of ServletContext interface](https://www.javatpoint.com/servletcontext#contextmethods)

[How to get the object of ServletContext](https://www.javatpoint.com/servletcontext#contextobject)

[Syntax to provide the initialization parameter in Context scope](https://www.javatpoint.com/servletcontext#contextsyn)

[Example of ServletContext to get initialization parameter](https://www.javatpoint.com/servletcontext#contextex1)

[Example of ServletContext to get all the initialization parameter](https://www.javatpoint.com/servletcontext#contextex2)

An object of ServletContext is created by the web container at time of deploying the project. This object can be used to get configuration information from web.xml file. There is only one ServletContext object per web application.

If any information is shared to many servlet, it is better to provide it from the web.xml file using the **<context-param>** element.

### **Advantage of ServletContext**

**Easy to maintain** if any information is shared to all the servlet, it is better to make it available for all the servlet. We provide this information from the web.xml file, so if the information is changed, we don't need to modify the servlet. Thus it removes maintenance problem.

### **Usage of ServletContext Interface**

There can be a lot of usage of ServletContext object. Some of them are as follows:

The object of ServletContext provides an interface between the container and servlet.

The ServletContext object can be used to get configuration information from the web.xml file.

The ServletContext object can be used to set, get or remove attribute from the web.xml file.

The ServletContext object can be used to provide inter-application communication.



### **Commonly used methods of ServletContext interface**

|  |
| --- |
| There is given some commonly used methods of ServletContext interface.  **public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.  **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters.  **public void setAttribute(String name,Object object):**sets the given object in the application scope.  **public Object getAttribute(String name):**Returns the attribute for the specified name.  **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters as an Enumeration of String objects.  **public void removeAttribute(String name):**Removes the attribute with the given name from the servlet context. |

### **How to get the object of ServletContext interface**

**getServletContext() method** of ServletConfig interface returns the object of ServletContext.

**getServletContext() method** of GenericServlet class returns the object of ServletContext.

#### **Syntax of getServletContext() method**

**public** ServletContext getServletContext()

#### **Example of getServletContext() method**

//We can get the ServletContext object from ServletConfig object

ServletContext application=getServletConfig().getServletContext();

//Another convenient way to get the ServletContext object

ServletContext application=getServletContext();

### **Syntax to provide the initialization parameter in Context scope**

|  |
| --- |
| The **context-param** element, subelement of web-app, is used to define the initialization parameter in the application scope. The param-name and param-value are the sub-elements of the context-param. The param-name element defines parameter name and and param-value defines its value. |

<web-app>

 ......

  <context-param>

    <param-name>parametername</param-name>

    <param-value>parametervalue</param-value>

  </context-param>

 ......

</web-app>

### **Example of ServletContext to get the initialization parameter**

|  |
| --- |
| In this example, we are getting the initialization parameter from the web.xml file and printing the value of the initialization parameter. Notice that the object of ServletContext represents the application scope. So if we change the value of the parameter from the web.xml file, all the servlet classes will get the changed value. So we don't need to modify the servlet. So it is better to have the common information for most of the servlets in the web.xml file by context-param element. Let's see the simple example: |

**DemoServlet.java**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**public** **class** DemoServlet **extends** HttpServlet{

**public** **void** doGet(HttpServletRequest req,HttpServletResponse res)

**throws** ServletException,IOException

{

res.setContentType("text/html");

PrintWriter pw=res.getWriter();

//creating ServletContext object

ServletContext context=getServletContext();

//Getting the value of the initialization parameter and printing it

String driverName=context.getInitParameter("dname");

pw.println("driver name is="+driverName);

pw.close();

}}

**web.xml**

<web-app>

<servlet>

<servlet-name>sonoojaiswal</servlet-name>

<servlet-**class**>DemoServlet</servlet-**class**>

</servlet>

<context-param>

<param-name>dname</param-name>

<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>

</context-param>

<servlet-mapping>

<servlet-name>sonoojaiswal</servlet-name>

<url-pattern>/context</url-pattern>

</servlet-mapping>

</web-app>

### **Example of ServletContext to get all the initialization parameters**

|  |
| --- |
| In this example, we are getting all the initialization parameter from the web.xml file. For getting all the parameters, we have used the getInitParameterNames() method in the servlet class. |

**DemoServlet.java**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**public** **class** DemoServlet **extends** HttpServlet{

**public** **void** doGet(HttpServletRequest req,HttpServletResponse res)

**throws** ServletException,IOException

{

res.setContentType("text/html");

PrintWriter out=res.getWriter();

ServletContext context=getServletContext();

Enumeration<String> e=context.getInitParameterNames();

String str="";

**while**(e.hasMoreElements()){

    str=e.nextElement();

    out.print("<br> "+context.getInitParameter(str));

}

}}

**web.xml**

<web-app>

<servlet>

<servlet-name>sonoojaiswal</servlet-name>

<servlet-**class**>DemoServlet</servlet-**class**>

</servlet>

<context-param>

<param-name>dname</param-name>

<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>

</context-param>

<context-param>

<param-name>username</param-name>

<param-value>system</param-value>

</context-param>

<context-param>

<param-name>password</param-name>

<param-value>oracle</param-value>

</context-param>

<servlet-mapping>

<servlet-name>sonoojaiswal</servlet-name>

<url-pattern>/context</url-pattern>

</servlet-mapping>

</web-app>

# ServletContext Example

# **ServletContext Interface**

[ServletContext Interface](https://www.javatpoint.com/servletcontext)

[Usage of ServletContext Interface](https://www.javatpoint.com/servletcontext#contextusage)

[Methods of ServletContext interface](https://www.javatpoint.com/servletcontext#contextmethods)

[How to get the object of ServletContext](https://www.javatpoint.com/servletcontext#contextobject)

[Syntax to provide the initialization parameter in Context scope](https://www.javatpoint.com/servletcontext#contextsyn)

[Example of ServletContext to get initialization parameter](https://www.javatpoint.com/servletcontext#contextex1)

[Example of ServletContext to get all the initialization parameter](https://www.javatpoint.com/servletcontext#contextex2)

An object of ServletContext is created by the web container at time of deploying the project. This object can be used to get configuration information from web.xml file. There is only one ServletContext object per web application.

If any information is shared to many servlet, it is better to provide it from the web.xml file using the **<context-param>** element.

### **Advantage of ServletContext**

**Easy to maintain** if any information is shared to all the servlet, it is better to make it available for all the servlet. We provide this information from the web.xml file, so if the information is changed, we don't need to modify the servlet. Thus it removes maintenance problem.

### **Usage of ServletContext Interface**

There can be a lot of usage of ServletContext object. Some of them are as follows:

The object of ServletContext provides an interface between the container and servlet.

The ServletContext object can be used to get configuration information from the web.xml file.

The ServletContext object can be used to set, get or remove attribute from the web.xml file.

The ServletContext object can be used to provide inter-application communication.



### **Commonly used methods of ServletContext interface**

|  |
| --- |
| There is given some commonly used methods of ServletContext interface.  **public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.  **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters.  **public void setAttribute(String name,Object object):**sets the given object in the application scope.  **public Object getAttribute(String name):**Returns the attribute for the specified name.  **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters as an Enumeration of String objects.  **public void removeAttribute(String name):**Removes the attribute with the given name from the servlet context. |

### **How to get the object of ServletContext interface**

**getServletContext() method** of ServletConfig interface returns the object of ServletContext.

**getServletContext() method** of GenericServlet class returns the object of ServletContext.

#### **Syntax of getServletContext() method**

**public** ServletContext getServletContext()

#### **Example of getServletContext() method**

//We can get the ServletContext object from ServletConfig object

ServletContext application=getServletConfig().getServletContext();

//Another convenient way to get the ServletContext object

ServletContext application=getServletContext();

### **Syntax to provide the initialization parameter in Context scope**

|  |
| --- |
| The **context-param** element, subelement of web-app, is used to define the initialization parameter in the application scope. The param-name and param-value are the sub-elements of the context-param. The param-name element defines parameter name and and param-value defines its value. |

<web-app>

 ......

  <context-param>

    <param-name>parametername</param-name>

    <param-value>parametervalue</param-value>

  </context-param>

 ......

</web-app>

### **Example of ServletContext to get the initialization parameter**

|  |
| --- |
| In this example, we are getting the initialization parameter from the web.xml file and printing the value of the initialization parameter. Notice that the object of ServletContext represents the application scope. So if we change the value of the parameter from the web.xml file, all the servlet classes will get the changed value. So we don't need to modify the servlet. So it is better to have the common information for most of the servlets in the web.xml file by context-param element. Let's see the simple example: |

**DemoServlet.java**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**public** **class** DemoServlet **extends** HttpServlet{

**public** **void** doGet(HttpServletRequest req,HttpServletResponse res)

**throws** ServletException,IOException

{

res.setContentType("text/html");

PrintWriter pw=res.getWriter();

//creating ServletContext object

ServletContext context=getServletContext();

//Getting the value of the initialization parameter and printing it

String driverName=context.getInitParameter("dname");

pw.println("driver name is="+driverName);

pw.close();

}}

**web.xml**

<web-app>

<servlet>

<servlet-name>sonoojaiswal</servlet-name>

<servlet-**class**>DemoServlet</servlet-**class**>

</servlet>

<context-param>

<param-name>dname</param-name>

<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>

</context-param>

<servlet-mapping>

<servlet-name>sonoojaiswal</servlet-name>

<url-pattern>/context</url-pattern>

</servlet-mapping>

</web-app>

### **Example of ServletContext to get all the initialization parameters**

|  |
| --- |
| In this example, we are getting all the initialization parameter from the web.xml file. For getting all the parameters, we have used the getInitParameterNames() method in the servlet class. |

**DemoServlet.java**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**public** **class** DemoServlet **extends** HttpServlet{

**public** **void** doGet(HttpServletRequest req,HttpServletResponse res)

**throws** ServletException,IOException

{

res.setContentType("text/html");

PrintWriter out=res.getWriter();

ServletContext context=getServletContext();

Enumeration<String> e=context.getInitParameterNames();

String str="";

**while**(e.hasMoreElements()){

    str=e.nextElement();

    out.print("<br> "+context.getInitParameter(str));

}

}}

**web.xml**

<web-app>

<servlet>

<servlet-name>sonoojaiswal</servlet-name>

<servlet-**class**>DemoServlet</servlet-**class**>

</servlet>

<context-param>

<param-name>dname</param-name>

<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>

</context-param>

<context-param>

<param-name>username</param-name>

<param-value>system</param-value>

</context-param>

<context-param>

<param-name>password</param-name>

<param-value>oracle</param-value>

</context-param>

<servlet-mapping>

<servlet-name>sonoojaiswal</servlet-name>

<url-pattern>/context</url-pattern>

</servlet-mapping>

</web-app>

# How To Set, Get And Remove Example With Get Set Methods

# **Getter and Setter Method in Java Example**

Getter and setter methods are frequently used in Java programming. **Getter and setter methods in Java** are widely used to access and manipulate the values of class fields. Usually, class fields are decorated with a private access specifier. Thus, to access them, public access specifiers are used with the getter and setter methods.

## **The Need of Getter and Setter Method**

One may argue that declare the class fields as public and remove the getter and setter methods. However, such a coding style is bad, and one may put some absurd value on the class fields. Let's understand it with the help of an example.

**public** **class** GetterSetterExample

{

**public** salary;

**public** storeSalaryDB(**int** salary)

    {

        // code for storing the salary in the database

    }

    // main method

**public** **static** **void** main(String argvs[])

    {

      GetterSetterExample obj = **new** GetterSetterExample();

      obj.salary = -50000;

      // storing salary in database

      obj.storeSalaryDB(salary);

    }

}

Observe that the code is storing a negative salary in the database that is wrong. An organization never credits a negative salary to the account of an employee. Assigning an absurd amount to the salary variable happened because it is declared with a public access specifier. The correct way to write the above code is:

**public** **class** GetterSetterExample

{

**private** salary;

    // a setter method that assign a

    // value to the salary variable

**void** setSalary(**int** s)

    {

**if**(s < 0 )

        {

            s = -s;

        }

**this**.salary = s;

    }

    // a getter mehtod to retrieve

    // the salary

**int** getSalary()

    {

**return** **this**.salary;

    }

**public** storeSalaryDB(**int** salary)

    {

        // code for storing the salary in the database

         System.out.println("The ")

    }

    // main method

**public** **static** **void** main(String argvs[])

    {

        // creating an object of the class GetterSetterExample

        GetterSetterExample obj = **new** GetterSetterExample();

      obj.setSalary(-50000);

**int** salary = obj.getSalary();

      // storing salary in database

      obj.storeSalaryDB(salary);

    }

}

Now, we can see better control over what we send to the database to store. Whenever the salary is negative, we are converting the salary into a positive value, and then we are sending it to the database to store. Thus, no matter what value we send to the setter method, the if-block of the setter method takes care of the absurd value and thus gives better control on the salary value.

## **Getter Setter Java Program**

**FileName:** GetterSetterExample1.java

**class** Employee

{

    // class member variable

**private** **int** eId;

**private** String eName;

**private** String eDesignation;

**private** String eCompany;

**public** **int** getEmpId()

    {

**return** eId;

    }

**public** **void** setEmpId(**final** **int** eId)

    {

**this**.eId = eId;

    }

**public** String getEmpName()

    {

**return** eName;

    }

**public** **void** setEmpName(**final** String eName)

    {

        // Validating the employee's name and

        // throwing an exception if the name is null or its length is less than or equal to 0.

**if**(eName == **null** ||  eName.length() <= 0)

        {

**throw** **new** IllegalArgumentException();

        }

**this**.eName = eName;

    }

**public** String getEmpDesignation()

    {

**return** eDesignation;

    }

**public** **void** setEmpDesignation(**final** String eDesignation)

    {

**this**.eDesignation = eDesignation;

    }

**public** String getEmpCompany()

    {

**return** eCompany;

    }

**public** **void** setEmpCompany(**final** String eCompany)

    {

**this**.eCompany = eCompany;

    }

    // for printing the values

    @Override

**public** String toString()

    {

        String str = "Employee: [id = " + getEmpId() + ", name = " + getEmpName() + ", designation = " + getEmpDesignation() + ", company = " + getEmpCompany() + "]";

**return** str;

    }

}

// Main class.

**public** **class** GetterSetterExample1

{

    // main method

**public** **static** **void** main(String argvs[])

    {

        // Creating an object of the Employee class

**final** Employee emp = **new** Employee();

        // the employee details are getting set using the setter methods.

        emp.setEmpId(107);

        emp.setEmpName("Kathy");

        emp.setEmpDesignation("Software Tester");

        emp.setEmpCompany("XYZ Corporation");

        // Displaying the details of the employee details using the

        // 'toString()' method, which uses the getter methods

        System.out.println(emp.toString());

    }

}

**Output:**

Employee: [id = 107, name = Kathy, designation = Software Tester, company = XYZ Corporation]

## **Bad Practices in Getter and Setter Methods**

There are some common bad practices that people usually do when they deal with the getter and setter methods.

### **Bad Practice 1:**

Using getter and setter for the variable that is declared with low restricted scope.

**public** salary;

**void** setSalary(**int** s)

    {

        salary = s;

    }

**int** getSalary()

    {

**return** salary;

    }

It is evident that from the main method, one can directly access the variable salary, which is not only bad but also makes the presence of the getter and setter methods irrelevant.

### **Bad Practice 2:**

Using an object reference in the setter method. Consider the following program.

**FileName:** GetterSetterExample2.java

**class** ABC

{

**private** **int**[] val;

**void** setVal(**int**[] arr)

{

**this**.val = arr; // line 7

}

// for displaying the value

// present in the val array

**void** display()

{

**int** size = (**this**.val).length;

**for**(**int** i = 0; i < size; i++)

{

System.out.print(**this**.val[i] + " ");

}

}

}

// Main class

**public** **class** GetterSetterExample2

{

// main method

**public** **static** **void** main(String argvs[])

{

// instantiating the class ABC

ABC obj = **new** ABC();

**int** mainArr[] = {3, 4, 6, 8, 78, 9};

// invoking the setter method

obj.setVal(mainArr);

// invoking the display method

obj.display();

// updating the value at the 0th index

mainArr[0] = -1;

System.out.println();

obj.display();

}

}

**Output:**

3 4 6 8 78 9

-1 4 6 8 78 9

**Explanation:**

References are a bit tricky to deal with! In the above code, at line 43, the value got updated at the 0th index for array mainArr[]. However, it also got reflected in the array val[]. It should not happen as val[] array is declared private; hence, it is expected that any code outside of the class ABC should not modify it. However, because of the references, everything is messed up. The setter method setVal() expecting a reference of an int array, and at line 7, the reference of the int arr[] is getting copied to val[]. Note that the reference variable arr[] is storing the reference of the array mainArr[]. Thus, we can say val[] is storing the reference of the mainArr[].

Therefore, whatever we change in the mainArr[] also gets reflected in the val[] array, which violates the purpose of the setter method. Also, there is no meaning in adding the private access specifier to the val[] array; because one can change the value of the val[] array in the main method, which is evident by looking at the output.

A better way of writing the above code is:

**FileName:** GetterSetterExample3.java

**class** ABC

{

**private** **int**[] val;

**void** setVal(**int**[] arr)

{

**int** size = arr.length;

// allocating the memory as

// per the array arr size

val = **new** **int**[size]; // line 11

**for**(**int** i = 0; i < size; i++)

{

// copying the value one by one

// into the val array

**this**.val[i] = arr[i]; // line 17

}

}

// for displaying the value

// present in the val array

**void** display()

{

**int** size = (**this**.val).length;

**for**(**int** i = 0; i < size; i++)

{

System.out.print(**this**.val[i] + " ");

}

}

}

// Main class.

**public** **class** GetterSetterExample3

{

// main method

**public** **static** **void** main(String argvs[])

{

// instantiating the class ABC

ABC obj = **new** ABC();

**int** mainArr[] = {3, 4, 6, 8, 78, 9};

// invoking the setter method

obj.setVal(mainArr);

// invoking the display method

obj.display();

// updating the value at the 0th index

mainArr[0] = -1; // line 53

System.out.println();

// invoking the display method again

obj.display();

}

}

**Output:**

3 4 6 8 78 9

3 4 6 8 78 9

**Explanation:**

In the above code, we are doing the deep copy of elements of the array arr[]. In line 11, we are creating an entirely new array. Thus, the val[] is not referring to the arr[]. Also, in line 17, only values of the element are getting copied. Therefore, when we change the value of the 0th element at line 53, the change is not reflected in the val[]. Thus, the above code respects the encapsulation of the private member variable val[].

### **Bad Practice 3:**

Returning an object reference in the getter method. Observe the following program.

**FileName:** GetterSetterExample4.java

**class** ABC

{

**private** **int**[] val = {67, 43, 68, 112, 70, 12};

// the getter method

**public** **int**[] getVal()

{

    // returning the reference

**return** val; // line 9

}

// for displaying the value

// present in the val array

**void** display()

{

**int** size = (**this**.val).length;

**for**(**int** i = 0; i < size; i++)

{

System.out.print(**this**.val[i] + " ");

}

}

}

// Main class.

**public** **class** GetterSetterExample4

{

// main method

**public** **static** **void** main(String argvs[])

{

// instantiating the class ABC

ABC obj = **new** ABC();

// invoking the getter method

// and storing the result

**int** arr[] = obj.getVal();

// invoking the display method

obj.display();

// updating the value at the 0th index

arr[0] = -1; // line 42

System.out.println();

// invoking the display method again

obj.display();

}

}

**Output:**

67 43 68 112 70 12

-1 43 68 112 70 12

**Explanation:**

The above code is not handling the references properly. The getter method is returning the reference of the array. The arr[] is storing the reference of the array val[], which is declared private in the class ABC. Because of exposing the reference to the outer world, arr[] can manipulate the val[], and thus, the encapsulation of the class ABC is breached. The proper way to handle the above is:

**FileName:** GetterSetterExample5.java

**class** ABC

{

**private** **int**[] val = {67, 43, 68, 112, 70, 12};

// the getter method

**public** **int**[] getVal()

{

**int** size = val.length;

// creating a new array

**int** temp[] = **new** **int**[size];

// copying the content of the array to temp array

**for**(**int** i = 0; i < size; i++)

{

temp[i] = val[i];

}

**return** temp;

}

// for displaying the value

// present in the val array

**void** display()

{

**int** size = (**this**.val).length;

**for**(**int** i = 0; i < size; i++)

{

System.out.print(**this**.val[i] + " ");

}

}

}

// Main class.

**public** **class** GetterSetterExample5

{

// main method

**public** **static** **void** main(String argvs[])

{

// instantiating the class ABC

ABC obj = **new** ABC();

// invoking the getter method

// and storing the result

**int** arr[] = obj.getVal();

// invoking the display method

obj.display();

// updating the value at the 0th index

arr[0] = -1; // line 54

System.out.println();

obj.display();

}

}

**Output:**

67 43 68 112 70 12

67 43 68 112 70 12

**Explanation:** In the above code, the reference of the private array is not sent to the outside world. In the getter method, a new array is created whose reference is sent to the main method. Therefore, when the value at the 0th index gets changed at line 54, that change impacts the temp[] array, not the private array val[]. Thus, the encapsulation of the class ABC is maintained, as the reference of the array val[] is not exposed to the outside world.

**FileName:** GetterSetterExample6.java

**class** ABC

{

**private** String str = **null**;

// a setter method

**void** setVal(String s)

{

// reference is getting copied

**this**.str = s;

}

// for displaying the string

**void** display()

{

System.out.println( "The String is: " + **this**.str);

}

}

// Main class.

**public** **class** GetterSetterExample6

{

// main method

**public** **static** **void** main(String argvs[])

{

// creating an object of the class ABC

ABC obj = **new** ABC();

// input string

String inputStr = "Hello India!";

// invoking the setter method

obj.setVal(inputStr);

obj.display();

// manipulation is not allowed!

// it leads to the creation of the new string

inputStr = "Hello World!";

obj.display();

}

}

**Output:**

The String is: Hello India!

The String is: Hello India!

# Session Tracking

# **Session Tracking in Servlets**

[Session Tracking](https://www.javatpoint.com/session-tracking-in-servlets#session1)

[Session Tracking Techniques](https://www.javatpoint.com/session-tracking-in-servlets#session1tech)

**Session** simply means a particular interval of time.

**Session Tracking** is a way to maintain state (data) of an user. It is also known as **session management** in servlet.

Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.

HTTP is stateless that means each request is considered as the new request. It is shown in the figure given below:



### **Why use Session Tracking?**

**To recognize the user** It is used to recognize the particular user.

### **Session Tracking Techniques**

There are four techniques used in Session tracking:

**Cookies**

**Hidden Form Field**

**URL Rewriting**

**HttpSession**

# Cookies Class N Their Methods

# **Cookies in Servlet**

A **cookie** is a small piece of information that is persisted between the multiple client requests.

A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

### **How Cookie works**

By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.



### **Types of Cookie**

There are 2 types of cookies in servlets.

Non-persistent cookie

Persistent cookie

### **Non-persistent cookie**

It is **valid for single session** only. It is removed each time when user closes the browser.

### **Persistent cookie**

It is **valid for multiple session** . It is not removed each time when user closes the browser. It is removed only if user logout or signout.

### **Advantage of Cookies**

Simplest technique of maintaining the state.

Cookies are maintained at client side.

### **Disadvantage of Cookies**

It will not work if cookie is disabled from the browser.

Only textual information can be set in Cookie object.

#### **Note: Gmail uses cookie technique for login. If you disable the cookie, gmail won't work.**

### **Cookie class**

**javax.servlet.http.Cookie** class provides the functionality of using cookies. It provides a lot of useful methods for cookies.

### **Constructor of Cookie class**

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| Cookie() | constructs a cookie. |
| Cookie(String name, String value) | constructs a cookie with a specified name and value. |

### **Useful Methods of Cookie class**

There are given some commonly used methods of the Cookie class.

|  |  |
| --- | --- |
| **Method** | **Description** |
| public void setMaxAge(int expiry) | Sets the maximum age of the cookie in seconds. |
| public String getName() | Returns the name of the cookie. The name cannot be changed after creation. |
| public String getValue() | Returns the value of the cookie. |
| public void setName(String name) | changes the name of the cookie. |
| public void setValue(String value) | changes the value of the cookie. |

### **Other methods required for using Cookies**

|  |
| --- |
| For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces. They are:  **public void addCookie(Cookie ck):**method of HttpServletResponse interface is used to add cookie in response object.  **public Cookie[] getCookies():**method of HttpServletRequest interface is used to return all the cookies from the browser. |
|  |
|  |

### **How to create Cookie?**

Let's see the simple code to create cookie.

Cookie ck=**new** Cookie("user","sonoo jaiswal");//creating cookie object

response.addCookie(ck);//adding cookie in the response

### **How to delete Cookie?**

Let's see the simple code to delete cookie. It is mainly used to logout or signout the user.

Cookie ck=**new** Cookie("user","");//deleting value of cookie

ck.setMaxAge(0);//changing the maximum age to 0 seconds

response.addCookie(ck);//adding cookie in the response

### **How to get Cookies?**

Let's see the simple code to get all the cookies.

Cookie ck[]=request.getCookies();

**for**(**int** i=0;i<ck.length;i++){

 out.print("<br>"+ck[i].getName()+" "+ck[i].getValue());//printing name and value of cookie

}

### **Simple example of Servlet Cookies**

In this example, we are storing the name of the user in the cookie object and accessing it in another servlet. As we know well that session corresponds to the particular user. So if you access it from too many browsers with different values, you will get the different value.



### **index.html**

<form action="servlet1" method="post">

Name:<input type="text" name="userName"/><br/>

<input type="submit" value="go"/>

</form>

### **FirstServlet.java**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**public** **class** FirstServlet **extends** HttpServlet {

**public** **void** doPost(HttpServletRequest request, HttpServletResponse response){

**try**{

    response.setContentType("text/html");

    PrintWriter out = response.getWriter();

    String n=request.getParameter("userName");

    out.print("Welcome "+n);

    Cookie ck=**new** Cookie("uname",n);//creating cookie object

    response.addCookie(ck);//adding cookie in the response

    //creating submit button

    out.print("<form action='servlet2'>");

    out.print("<input type='submit' value='go'>");

    out.print("</form>");

    out.close();

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

  }

}

### **SecondServlet.java**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**public** **class** SecondServlet **extends** HttpServlet {

**public** **void** doPost(HttpServletRequest request, HttpServletResponse response){

**try**{

    response.setContentType("text/html");

    PrintWriter out = response.getWriter();

    Cookie ck[]=request.getCookies();

    out.print("Hello "+ck[0].getValue());

    out.close();

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

    }

}

### **web.xml**

<web-app>

<servlet>

<servlet-name>s1</servlet-name>

<servlet-**class**>FirstServlet</servlet-**class**>

</servlet>

<servlet-mapping>

<servlet-name>s1</servlet-name>

<url-pattern>/servlet1</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>s2</servlet-name>

<servlet-**class**>SecondServlet</servlet-**class**>

</servlet>

<servlet-mapping>

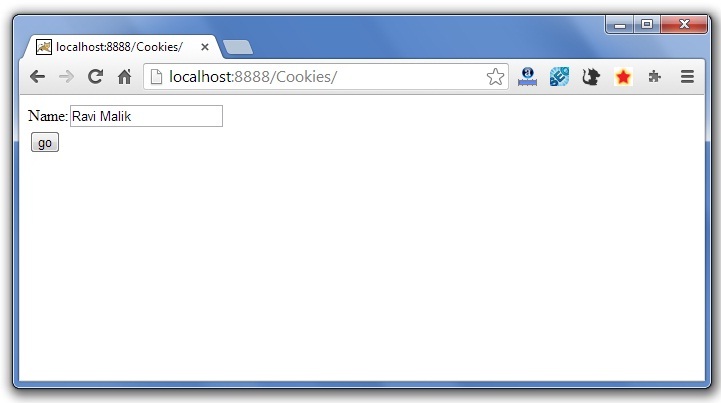
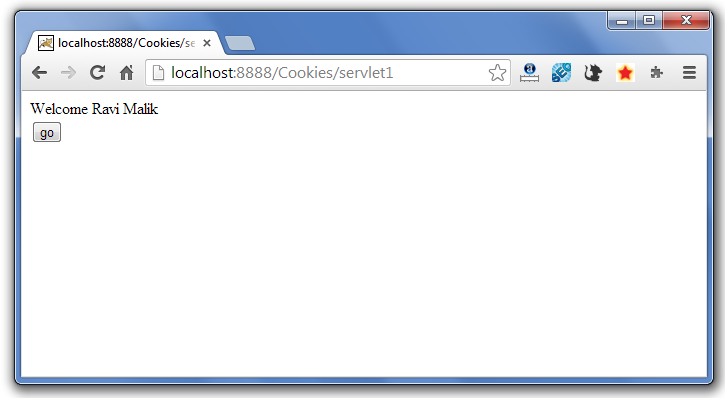
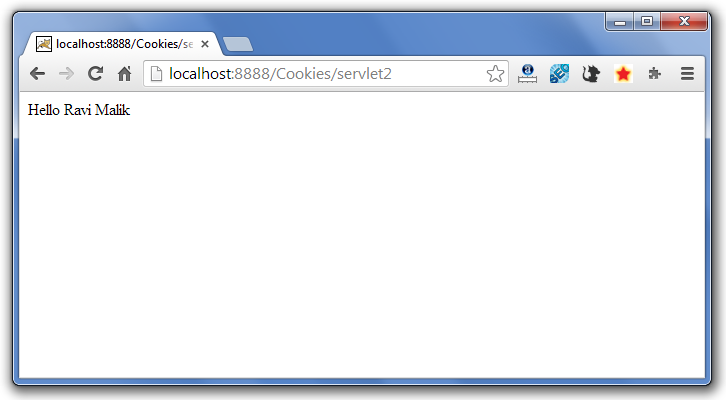
<servlet-name>s2</servlet-name>

<url-pattern>/servlet2</url-pattern>

</servlet-mapping>

</web-app>

#### **Output**

# Hidden Form Field Techniques

# **2) Hidden Form Field**

[Hidden Form Field](https://www.javatpoint.com/hidden-form-field-in-session-tracking)

[Example of Hidden Form Field](https://www.javatpoint.com/hidden-form-field-in-session-tracking#session2ex)

In case of Hidden Form Field **a hidden (invisible) textfield** is used for maintaining the state of an user.

In such case, we store the information in the hidden field and get it from another servlet. This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.

Let's see the code to store value in hidden field.

<input type="hidden" name="uname" value="Vimal Jaiswal">

Here, uname is the hidden field name and Vimal Jaiswal is the hidden field value.

### **Real application of hidden form field**

It is widely used in comment form of a website. In such case, we store page id or page name in the hidden field so that each page can be uniquely identified.

### **Advantage of Hidden Form Field**

It will always work whether cookie is disabled or not.

### **Disadvantage of Hidden Form Field:**

It is maintained at server side.

Extra form submission is required on each pages.

Only textual information can be used.

### **Example of using Hidden Form Field**

In this example, we are storing the name of the user in a hidden textfield and getting that value from another servlet.



### **index.html**

<form action="servlet1">

Name:<input type="text" name="userName"/><br/>

<input type="submit" value="go"/>

</form>

### **FirstServlet.java**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**public** **class** FirstServlet **extends** HttpServlet {

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

**try**{

        response.setContentType("text/html");

        PrintWriter out = response.getWriter();

        String n=request.getParameter("userName");

        out.print("Welcome "+n);

        //creating form that have invisible textfield

        out.print("<form action='servlet2'>");

        out.print("<input type='hidden' name='uname' value='"+n+"'>");

        out.print("<input type='submit' value='go'>");

        out.print("</form>");

        out.close();

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

    }

}

### **SecondServlet.java**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**public** **class** SecondServlet **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response)

**try**{

        response.setContentType("text/html");

        PrintWriter out = response.getWriter();

        //Getting the value from the hidden field

        String n=request.getParameter("uname");

        out.print("Hello "+n);

        out.close();

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

    }

}

### **web.xml**

<web-app>

<servlet>

<servlet-name>s1</servlet-name>

<servlet-**class**>FirstServlet</servlet-**class**>

</servlet>

<servlet-mapping>

<servlet-name>s1</servlet-name>

<url-pattern>/servlet1</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>s2</servlet-name>

<servlet-**class**>SecondServlet</servlet-**class**>

</servlet>

<servlet-mapping>

<servlet-name>s2</servlet-name>

<url-pattern>/servlet2</url-pattern>

</servlet-mapping>

</web-app>

# URL Rewriting In Servlet

# **3)URL Rewriting**

[URL Rewriting](https://www.javatpoint.com/url-rewriting-in-session-tracking)

[Advantage of URL Rewriting](https://www.javatpoint.com/url-rewriting-in-session-tracking#urladv)

[Disadvantage of URL Rewriting](https://www.javatpoint.com/url-rewriting-in-session-tracking#urldisadv)

[Example of URL Rewriting](https://www.javatpoint.com/url-rewriting-in-session-tracking#urlex)

In URL rewriting, we append a token or identifier to the URL of the next Servlet or the next resource. We can send parameter name/value pairs using the following format:

url?name1=value1&name2=value2&??

A name and a value is separated using an equal = sign, a parameter name/value pair is separated from another parameter using the ampersand(&). When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server. From a Servlet, we can use getParameter() method to obtain a parameter value.



### **Advantage of URL Rewriting**

It will always work whether cookie is disabled or not (browser independent).

Extra form submission is not required on each pages.

### **Disadvantage of URL Rewriting**

It will work only with links.

It can send Only textual information.

### **Example of using URL Rewriting**

In this example, we are maintaning the state of the user using link. For this purpose, we are appending the name of the user in the query string and getting the value from the query string in another page.

### **index.html**

<form action="servlet1">

Name:<input type="text" name="userName"/><br/>

<input type="submit" value="go"/>

</form>

### **FirstServlet.java**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**public** **class** FirstServlet **extends** HttpServlet {

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

**try**{

        response.setContentType("text/html");

        PrintWriter out = response.getWriter();

        String n=request.getParameter("userName");

        out.print("Welcome "+n);

        //appending the username in the query string

        out.print("<a href='servlet2?uname="+n+"'>visit</a>");

        out.close();

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

    }

}

### **SecondServlet.java**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**public** **class** SecondServlet **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response)

**try**{

        response.setContentType("text/html");

        PrintWriter out = response.getWriter();

        //getting value from the query string

        String n=request.getParameter("uname");

        out.print("Hello "+n);

        out.close();

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

    }

}

### **web.xml**

<web-app>

<servlet>

<servlet-name>s1</servlet-name>

<servlet-**class**>FirstServlet</servlet-**class**>

</servlet>

<servlet-mapping>

<servlet-name>s1</servlet-name>

<url-pattern>/servlet1</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>s2</servlet-name>

<servlet-**class**>SecondServlet</servlet-**class**>

</servlet>

<servlet-mapping>

<servlet-name>s2</servlet-name>

<url-pattern>/servlet2</url-pattern>

</servlet-mapping>

</web-app>

# HttpSession Servlet

# **4) HttpSession interface**

[HttpSession interface](https://www.javatpoint.com/http-session-in-session-tracking)

[How to get the HttpSession object](https://www.javatpoint.com/http-session-in-session-tracking#httpsessionhow)

[Commonly used methods of HttpSession interface](https://www.javatpoint.com/http-session-in-session-tracking#httpsessionmethod)

[Example of using HttpSession](https://www.javatpoint.com/http-session-in-session-tracking#httpsessionex)

In such case, container creates a session id for each user.The container uses this id to identify the particular user.An object of HttpSession can be used to perform two tasks:

bind objects

view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.



### **How to get the HttpSession object ?**

The HttpServletRequest interface provides two methods to get the object of HttpSession:

**public HttpSession getSession():**Returns the current session associated with this request, or if the request does not have a session, creates one.

**public HttpSession getSession(boolean create):**Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.

### **Commonly used methods of HttpSession interface**

**public String getId():**Returns a string containing the unique identifier value.

**public long getCreationTime():**Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.

**public long getLastAccessedTime():**Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.

**public void invalidate():**Invalidates this session then unbinds any objects bound to it.

### **Example of using HttpSession**

In this example, we are setting the attribute in the session scope in one servlet and getting that value from the session scope in another servlet. To set the attribute in the session scope, we have used the setAttribute() method of HttpSession interface and to get the attribute, we have used the getAttribute method.

### **index.html**

<form action="servlet1">

Name:<input type="text" name="userName"/><br/>

<input type="submit" value="go"/>

</form>

### **FirstServlet.java**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**public** **class** FirstServlet **extends** HttpServlet {

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

**try**{

        response.setContentType("text/html");

        PrintWriter out = response.getWriter();

        String n=request.getParameter("userName");

        out.print("Welcome "+n);

        HttpSession session=request.getSession();

        session.setAttribute("uname",n);

        out.print("<a href='servlet2'>visit</a>");

        out.close();

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

    }

}

### **SecondServlet.java**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**public** **class** SecondServlet **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response)

**try**{

        response.setContentType("text/html");

        PrintWriter out = response.getWriter();

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

        String n=(String)session.getAttribute("uname");

        out.print("Hello "+n);

        out.close();

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

    }

}

### **web.xml**

<web-app>

<servlet>

<servlet-name>s1</servlet-name>

<servlet-**class**>FirstServlet</servlet-**class**>

</servlet>

<servlet-mapping>

<servlet-name>s1</servlet-name>

<url-pattern>/servlet1</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>s2</servlet-name>

<servlet-**class**>SecondServlet</servlet-**class**>

</servlet>

<servlet-mapping>

<servlet-name>s2</servlet-name>

<url-pattern>/servlet2</url-pattern>

</servlet-mapping>

</web-app>

# Event And Listener

# **JavaScript addEventListener()**

The **addEventListener()** method is used to attach an event handler to a particular element. It does not override the existing event handlers. Events are said to be an essential part of the JavaScript. A web page responds according to the event that occurred. Events can be user-generated or generated by API's. An event listener is a JavaScript's procedure that waits for the occurrence of an event.

The addEventListener() method is an inbuilt function of [JavaScript](https://www.javatpoint.com/javascript-tutorial). We can add multiple event handlers to a particular element without overwriting the existing event handlers.

### **Syntax**

element.addEventListener(event, function, useCapture);

Although it has three parameters, the parameters **event** and **function** are widely used. The third parameter is optional to define. The values of this function are defined as follows.

### **Parameter Values**

**event:** It is a required parameter. It can be defined as a string that specifies the event's name.

#### **Note: Do not use any prefix such as "on" with the parameter value. For example, Use "click" instead of using "onclick".**

**function:** It is also a required parameter. It is a [JavaScript function](https://www.javatpoint.com/javascript-function) which responds to the event occur.

**useCapture:** It is an optional parameter. It is a Boolean type value that specifies whether the event is executed in the bubbling or capturing phase. Its possible values are **true** and **false**. When it is set to true, the event handler executes in the capturing phase. When it is set to false, the handler executes in the bubbling phase. Its default value is **false**.

Let's see some of the illustrations of using the addEventListener() method.

### **Example**

It is a simple example of using the addEventListener() method. We have to click the given [HTML button](https://www.javatpoint.com/html-button-tag) to see the effect.

<!DOCTYPE html**>**

**<html>**

**<body>**

**<p>** Example of the addEventListener() method. **</p>**

**<p>** Click the following button to see the effect. **</p>**

**<button** id = "btn"**>** Click me **</button>**

**<p** id = "para"**></p>**

**<script>**

document.getElementById("btn").addEventListener("click", fun);

function fun() {

document.getElementById("para").innerHTML = "Hello World" + "**<br>**" + "Welcome to the  javaTpoint.com";

}

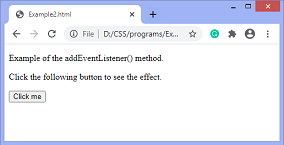
**</script>**

**</body>**

**</html>**

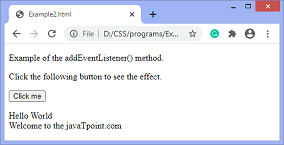
[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=javascript-addeventlistener1)

**Output**



After clicking the given [HTML](https://www.javatpoint.com/html-tutorial) button, the output will be -

ADVERTISEMENT



Now, in the next example we will see how to add many events to the same element without overwriting the existing events.

### **Example**

In this example, we are adding multiple events to the same element.

<!DOCTYPE html**>**

**<html>**

**<body>**

**<p>** This is an example of adding multiple events to the same element. **</p>**

**<p>** Click the following button to see the effect. **</p>**

**<button** id = "btn"**>** Click me **</button>**

**<p** id = "para"**></p>**

**<p** id = "para1"**></p>**

**<script>**

function fun() {

    alert("Welcome to the javaTpoint.com");

}

function fun1() {

   document.getElementById("para").innerHTML =  "This is second function";

}

function fun2() {

   document.getElementById("para1").innerHTML =  "This is third function";

}

var mybtn = document.getElementById("btn");

mybtn.addEventListener("click", fun);

mybtn.addEventListener("click", fun1);

mybtn.addEventListener("click", fun2);

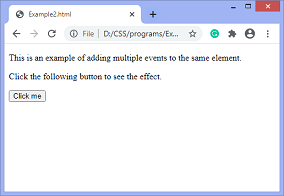
**</script>**

**</body>**

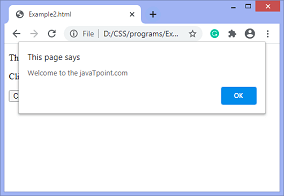
**</html>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=javascript-addeventlistener2)

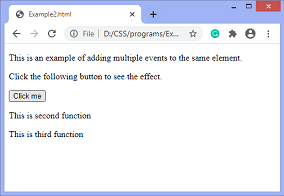
**Output**



Now, when we click the button, an alert will be displayed. After clicking the given HTML button, the output will be -



When we exit the alert, the output is -



### **Example**

In this example, we are adding multiple events of a different type to the same element.

<!DOCTYPE html**>**

**<html>**

**<body>**

**<p>** This is an example of adding multiple events of different type to the same element. **</p>**

**<p>** Click the following button to see the effect. **</p>**

**<button** id = "btn"**>** Click me **</button>**

**<p** id = "para"**></p>**

**<script>**

function fun() {

    btn.style.width = "50px";

    btn.style.height = "50px";

    btn.style.background = "yellow";

    btn.style.color = "blue";

}

function fun1() {

   document.getElementById("para").innerHTML =  "This is second function";

}

function fun2() {

    btn.style.width = "";

    btn.style.height = "";

    btn.style.background = "";

    btn.style.color = "";

}

var mybtn = document.getElementById("btn");

mybtn.addEventListener("mouseover", fun);

mybtn.addEventListener("click", fun1);

mybtn.addEventListener("mouseout", fun2);

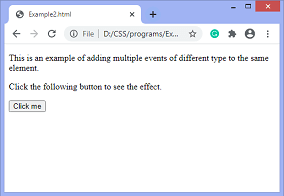
**</script>**

**</body>**

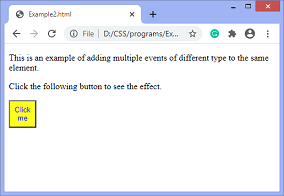
**</html>**

**Output**

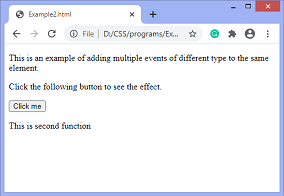
ADVERTISEMENT



When we move the cursor over the button, the output will be -



After clicking the button and leave the cursor, the output will be -



## **Event Bubbling or Event Capturing**

Now, we understand the use of the third parameter of JavaScript's addEventListener(), i.e., **useCapture.**

In HTML DOM, **Bubbling** and **Capturing** are the two ways of event propagation. We can understand these ways by taking an example.

Suppose we have a div element and a paragraph element inside it, and we are applying the **"click"** event to both of them using the **addEventListener()** method. Now the question is on clicking the paragraph element, which element's click event is handled first.

So, in **Bubbling,** the event of paragraph element is handled first, and then the div element's event is handled. It means that in bubbling, the inner element's event is handled first, and then the outermost element's event will be handled.

In **Capturing** the event of div element is handled first, and then the paragraph element's event is handled. It means that in capturing the outer element's event is handled first, and then the innermost element's event will be handled.

addEventListener(event, function, useCapture);

We can specify the propagation using the **useCapture** parameter. When it is set to false (which is its default value), then the event uses bubbling propagation, and when it is set to true, there is the capturing propagation.

We can understand the bubbling and capturing using an illustration.

### **Example**

In this example, there are two div elements. We can see the bubbling effect on the first div element and the capturing effect on the second div element.

When we double click the span element of the first div element, then the span element's event is handled first than the div element. It is called bubbling.

But when we double click the span element of the second div element, then the div element's event is handled first than the span element. It is called capturing.

<!DOCTYPE html**>**

**<html>**

**<head>**

**<style>**

div{

background-color: lightblue;

border: 2px solid red;

font-size: 25px;

text-align: center;

}

span{

border: 2px solid blue;

}

**</style>**

**</head>**

**<body>**

**<h1>** Bubbling **</h1>**

**<div** id = "d1"**>**

This is a div element.

**<br><br>**

**<span** id = "s1"**>** This is a span element. **</span>**

**</div>**

**<h1>** Capturing **</h1>**

**<div** id = "d2"**>** This is a div element.

**<br><br>**

**<span** id = "s2"**>** This is a span element. **</span>**

**</div>**

**<script>**

document.getElementById("d1").addEventListener("dblclick", function() {alert('You have double clicked on div element')}, false);

document.getElementById("s1").addEventListener("dblclick", function() {alert('You have double clicked on span element')}, false);

document.getElementById("d2").addEventListener("dblclick", function() {alert('You have double clicked on div element')}, true);

document.getElementById("s2").addEventListener("dblclick", function() {alert('You have double clicked on span element')}, true);

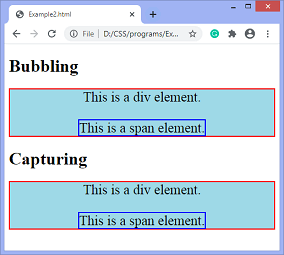
**</script>**

**</body>**

**</html>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=javascript-addeventlistener4)

**Output**



We have to double click the specific elements to see the effect.

# Filter In Java

# **Java Stream Filter**

Java stream provides a method filter() to filter stream elements on the basis of given predicate. Suppose you want to get only even elements of your list then you can do this easily with the help of filter method.

This method takes predicate as an argument and returns a stream of consisting of resulted elements.

## **Signature**

The signature of Stream filter() method is given below:

Stream<T> filter(Predicate<? **super** T> predicate)

### **Parameter**

**predicate:** It takes Predicate reference as an argument. Predicate is a functional interface. So, you can also pass lambda expression here.

### **Return**

It returns a new stream.

### **Java Stream filter() example**

In the following example, we are fetching and iterating filtered data.

**import** java.util.\*;

**class** Product{

**int** id;

    String name;

**float** price;

**public** Product(**int** id, String name, **float** price) {

**this**.id = id;

**this**.name = name;

**this**.price = price;

    }

}

**public** **class** JavaStreamExample {

**public** **static** **void** main(String[] args) {

        List<Product> productsList = **new** ArrayList<Product>();

        //Adding Products

        productsList.add(**new** Product(1,"HP Laptop",25000f));

        productsList.add(**new** Product(2,"Dell Laptop",30000f));

        productsList.add(**new** Product(3,"Lenevo Laptop",28000f));

        productsList.add(**new** Product(4,"Sony Laptop",28000f));

        productsList.add(**new** Product(5,"Apple Laptop",90000f));

        productsList.stream()

                    .filter(p ->p.price> 30000)   // filtering price

                    .map(pm ->pm.price)          // fetching price

                    .forEach(System.out::println);  // iterating price

    }

}

Output:

90000.0

### **Java Stream filter() example 2**

In the following example, we are fetching filtered data as a list.

**import** java.util.\*;

**import** java.util.stream.Collectors;

**class** Product{

**int** id;

    String name;

**float** price;

**public** Product(**int** id, String name, **float** price) {

**this**.id = id;

**this**.name = name;

**this**.price = price;

    }

}

**public** **class** JavaStreamExample {

**public** **static** **void** main(String[] args) {

        List<Product> productsList = **new** ArrayList<Product>();

        //Adding Products

        productsList.add(**new** Product(1,"HP Laptop",25000f));

        productsList.add(**new** Product(2,"Dell Laptop",30000f));

        productsList.add(**new** Product(3,"Lenevo Laptop",28000f));

        productsList.add(**new** Product(4,"Sony Laptop",28000f));

        productsList.add(**new** Product(5,"Apple Laptop",90000f));

        List<Float> pricesList =  productsList.stream()

                    .filter(p ->p.price> 30000)   // filtering price

                    .map(pm ->pm.price)          // fetching price

                    .collect(Collectors.toList());

        System.out.println(pricesList);

    }

}

Output:

[90000.0]

# Authentication Filter With Example

# **Authentication Filter**

We can perform authentication in filter. Here, we are going to check to password given by the user in filter class, if given password is admin, it will forward the request to the WelcomeAdmin servlet otherwise it will display error message.

### **Example of authenticating user using filter**

Let's see the simple example of authenticating user using filter.

Here, we have created 4 files:

index.html

MyFilter.java

AdminServlet.java

web.xml

**index.html**

<form action="servlet1">

Name:<input type="text" name="name"/><br/>

Password:<input type="password" name="password"/><br/>

<input type="submit" value="login">

</form>

**MyFilter.java**

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** javax.servlet.\*;

**public** **class** MyFilter **implements** Filter{

**public** **void** init(FilterConfig arg0) **throws** ServletException {}

**public** **void** doFilter(ServletRequest req, ServletResponse resp,

        FilterChain chain) **throws** IOException, ServletException {

    PrintWriter out=resp.getWriter();

    String password=req.getParameter("password");

**if**(password.equals("admin")){

    chain.doFilter(req, resp);//sends request to next resource

    }

**else**{

    out.print("username or password error!");

    RequestDispatcher rd=req.getRequestDispatcher("index.html");

    rd.include(req, resp);

    }

}

**public** **void** destroy() {}

}

**AdminServlet.java**

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.\*;

**public** **class** AdminServlet **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

        response.setContentType("text/html");

        PrintWriter out = response.getWriter();

        out.print("welcome ADMIN");

        out.close();

    }

}

**web.xml**

<web-app>

 <servlet>

    <servlet-name>AdminServlet</servlet-name>

    <servlet-**class**>AdminServlet</servlet-**class**>

  </servlet>

  <servlet-mapping>

    <servlet-name>AdminServlet</servlet-name>

    <url-pattern>/servlet1</url-pattern>

  </servlet-mapping>

 <filter>

  <filter-name>f1</filter-name>

  <filter-**class**>MyFilter</filter-**class**>

  </filter>

  <filter-mapping>

  <filter-name>f1</filter-name>

  <url-pattern>/servlet1</url-pattern>

  </filter-mapping>

</web-app>

# FilterConfig With Example

# **FilterConfig**

An object of FilterConfig is created by the web container. This object can be used to get the configuration information from the web.xml file.

## **Methods of FilterConfig interface**

There are following 4 methods in the FilterConfig interface.

**public void init(FilterConfig config):** init() method is invoked only once it is used to initialize the filter.

**public String getInitParameter(String parameterName):** Returns the parameter value for the specified parameter name.

**public java.util.Enumeration getInitParameterNames():** Returns an enumeration containing all the parameter names.

**public ServletContext getServletContext():** Returns the ServletContext object.

### **Example of FilterConfig**

In this example, if you change the param-value to no, request will be forwarded to the servlet otherwise filter will create the response with the message: this page is underprocessing. Let's see the simple example of FilterConfig. Here, we have created 4 files:

index.html

MyFilter.java

HelloServlet.java

web.xml

**index.html**

<a href="servlet1">click here</a>

**MyFilter.java**

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** javax.servlet.\*;

**public** **class** MyFilter **implements** Filter{

FilterConfig config;

**public** **void** init(FilterConfig config) **throws** ServletException {

**this**.config=config;

}

**public** **void** doFilter(ServletRequest req, ServletResponse resp,

    FilterChain chain) **throws** IOException, ServletException {

    PrintWriter out=resp.getWriter();

    String s=config.getInitParameter("construction");

**if**(s.equals("yes")){

         out.print("This page is under construction");

    }

**else**{

         chain.doFilter(req, resp);//sends request to next resource

    }

}

**public** **void** destroy() {}

}

**HelloServlet.java**

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.\*;

**public** **class** HelloServlet **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

        response.setContentType("text/html");

        PrintWriter out = response.getWriter();

        out.print("<br>welcome to servlet<br>");

    }

}

**web.xml**

<web-app>

 <servlet>

    <servlet-name>HelloServlet</servlet-name>

    <servlet-**class**>HelloServlet</servlet-**class**>

  </servlet>

  <servlet-mapping>

    <servlet-name>HelloServlet</servlet-name>

    <url-pattern>/servlet1</url-pattern>

  </servlet-mapping>

  <filter>

  <filter-name>f1</filter-name>

  <filter-**class**>MyFilter</filter-**class**>

  <init-param>

  <param-name>construction</param-name>

  <param-value>no</param-value>

  </init-param>

  </filter>

  <filter-mapping>

  <filter-name>f1</filter-name>

  <url-pattern>/servlet1</url-pattern>

  </filter-mapping>

</web-app>

# ServletInputStream And ServletOutputStream Class Example.

# **ServletOutputStream class**

[ServletOutputStream class](https://www.javatpoint.com/ServletOutputStream-class)

[Methods of ServletOutputStream class](https://www.javatpoint.com/ServletOutputStream-class#method)

[Example of ServletOutputStream class](https://www.javatpoint.com/ServletOutputStream-class)

**ServletOutputStream** class provides a stream to write binary data into the response. It is an abstract class.

The **getOutputStream()** method of **ServletResponse** interface returns the instance of ServletOutputStream class. It may be get as:

ServletOutputStream out=response.getOutputStream();

### **Methods of ServletOutputStream class**

The ServletOutputStream class provides print() and println() methods that are overloaded.

void print(boolean b){}

void print(char c){}

void print(int i){}

void print(long l){}

void print(float f){}

void print(double d){}

void print(String s){}

void println{}

void println(boolean b){}

void println(char c){}

void println(int i){}

void println(long l){}

void println(float f){}

void println(double d){}

void println(String s){}

# Basics Of JSP

# **JSP Tutorial**

**JSP** technology is used to create web application just like Servlet technology. It can be thought of as an extension to Servlet because it provides more functionality than servlet such as expression language, JSTL, etc.

A JSP page consists of HTML tags and JSP tags. The JSP pages are easier to maintain than Servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tags, etc.

### **Advantages of JSP over Servlet**

There are many advantages of JSP over the Servlet. They are as follows:

#### **Extension to Servlet**

JSP technology is the extension to Servlet technology. We can use all the features of the Servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.

#### **2) Easy to maintain**

JSP can be easily managed because we can easily separate our business logic with presentation logic. In Servlet technology, we mix our business logic with the presentation logic.

#### **3) Fast Development: No need to recompile and redeploy**

If JSP page is modified, we don't need to recompile and redeploy the project. The Servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

#### **4) Less code than Servlet**

In JSP, we can use many tags such as action tags, JSTL, custom tags, etc. that reduces the code. Moreover, we can use EL, implicit objects, etc.

### **The Lifecycle of a JSP Page**

The JSP pages follow these phases:

Translation of JSP Page

Compilation of JSP Page

Classloading (the classloader loads class file)

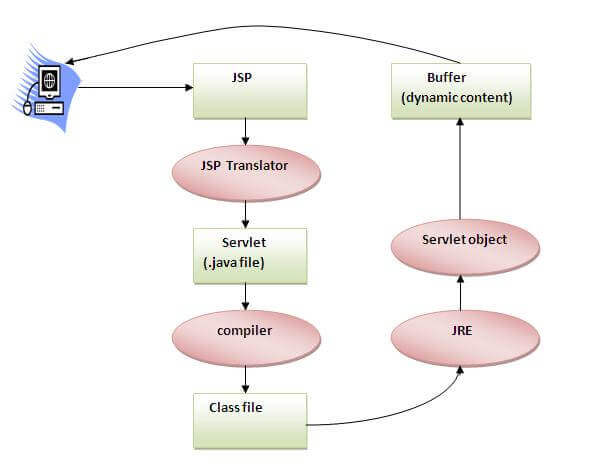
Instantiation (Object of the Generated Servlet is created).

Initialization ( the container invokes jspInit() method).

Request processing ( the container invokes \_jspService() method).

Destroy ( the container invokes jspDestroy() method).

#### **Note: jspInit(), \_jspService() and jspDestroy() are the life cycle methods of JSP.**



As depicted in the above diagram, JSP page is translated into Servlet by the help of JSP translator. The JSP translator is a part of the web server which is responsible for translating the JSP page into Servlet. After that, Servlet page is compiled by the compiler and gets converted into the class file. Moreover, all the processes that happen in Servlet are performed on JSP later like initialization, committing response to the browser and destroy.

### **Creating a simple JSP Page**

To create the first JSP page, write some HTML code as given below, and save it by .jsp extension. We have saved this file as index.jsp. Put it in a folder and paste the folder in the web-apps directory in apache tomcat to run the JSP page.

**index.jsp**

Let's see the simple example of JSP where we are using the scriptlet tag to put Java code in the JSP page. We will learn scriptlet tag later.

<html>

<body>

<% out.print(2\*5); %>

</body>

</html>

It will print **10** on the browser.

### **How to run a simple JSP Page?**

Follow the following steps to execute this JSP page:

Start the server

Put the JSP file in a folder and deploy on the server

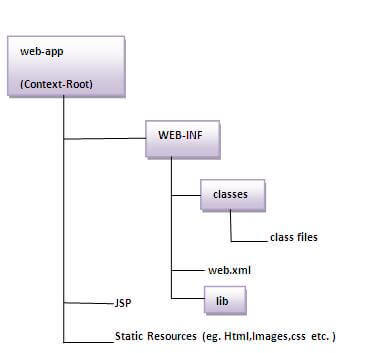
Visit the browser by the URL http://localhost:portno/contextRoot/jspfile, for example, http://localhost:8888/myapplication/index.jsp

### **Do I need to follow the directory structure to run a simple JSP?**

No, there is no need of directory structure if you don't have class files or TLD files. For example, put JSP files in a folder directly and deploy that folder. It will be running fine. However, if you are using Bean class, Servlet or TLD file, the directory structure is required.

### **The Directory structure of JSP**

The directory structure of JSP page is same as Servlet. We contain the JSP page outside the WEB-INF folder or in any directory.



# JSP API In Java

# **The JSP API**

[The JSP API](https://www.javatpoint.com/jsp-api)

[javax.servlet.jsp package](https://www.javatpoint.com/jsp-api#jsppackage)

[The JspPage interface](https://www.javatpoint.com/jsp-api#jsppage)

[The HttpJspPage interface](https://www.javatpoint.com/jsp-api#httpjsppage)

The JSP API consists of two packages:

javax.servlet.jsp

javax.servlet.jsp.tagext

## **javax.servlet.jsp package**

The javax.servlet.jsp package has two interfaces and classes.The two interfaces are as follows:

JspPage

HttpJspPage

The classes are as follows:

JspWriter

PageContext

JspFactory

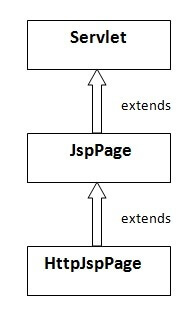
JspEngineInfo

JspException

JspError

## **The JspPage interface**

According to the JSP specification, all the generated servlet classes must implement the JspPage interface. It extends the Servlet interface. It provides two life cycle methods.



### **Methods of JspPage interface**

**public void jspInit():** It is invoked only once during the life cycle of the JSP when JSP page is requested firstly. It is used to perform initialization. It is same as the init() method of Servlet interface.

**public void jspDestroy():** It is invoked only once during the life cycle of the JSP before the JSP page is destroyed. It can be used to perform some clean up operation.

## **The HttpJspPage interface**

The HttpJspPage interface provides the one life cycle method of JSP. It extends the JspPage interface.

### **Method of HttpJspPage interface:**

**public void \_jspService():** It is invoked each time when request for the JSP page comes to the container. It is used to process the request. The underscore \_ signifies that you cannot override this method.

We will learn all other classes and interfaces later.

# JSP In Netbean

# **JSP using NetBeans IDE**

Back to: [JSP Tutorials for Beginners and Professionals](https://dotnettutorials.net/course/jsp-tutorials/)

## **JSP Web Application Development using NetBeans IDE**

In this article, I am going to show you the step-by-step process to create and run the **JSP Web Application using the NetBeans IDE**. Please read our previous article where discussed [**JSP Web Application Development using the MyEclipse IDE**](https://dotnettutorials.net/lesson/jsp-using-myeclipse-ide/). NetBeans IDE lets you quickly and easily develop Java desktop, mobile, and web applications, as well as HTML5 applications with HTML, JavaScript, and CSS. The IDE also provides a great set of tools for PHP and C/C++ developers.

Download NetBeans IDE from[**https://netbeans.org/downloads/6.1/index.html**](https://netbeans.org/downloads/6.1/index.html)

##### **JSP Web Application Development using NetBeans IDE:**

In this, we will create a simple JSP file and run it on NetBeans IDE using Apache Tomcat Server. Follow the below steps to create the servlet in the NetBeans IDE:

##### **Step-1:** **Create New Project**

Open NetBeans IDE and then select File -> New Project option from the context menu as shown in the below image.

It will open the New Project window. From this window, select Java web -> Web Application and then click on the Next button as shown in the below image.

Once you click on the Next button, a new window is generated with a default project name. Now change the project name to “HelloWorldJSP” and select the location where you want to create the project and then click on the “Next” button as shown in the below image.

Once you click on the Next button, a new window is generated for the server and setting the wizard. First, go to the server and click on “Add server”. Since I already set up the Tomcat Server, it is shown there but when you do it the first time it shows blank. And then click on the “Next” button as shown in the below image.

After clicking on Add, a list of servers opens. In this article, I used Tomcat Server 7.0.40. Choose your server and configure it. For Tomcat Server choose it and click on “Next”.

Now a window opens asking for the server location. Now, click on the Browse button and find the server location where you have stored the Apache Tomcat. Note that you need to provide the path up to the bin directory of Tomcat Server.

After providing the server path you will see two more options for username and password. It’s your choice whether you want your username and password or you want the default created by the Netbeans IDE. Now click on “Finish”.

Now a window is generated containing the default index.html file in Web Pages under HelloWorldJSP.

##### **Step-2: Creating JSP Page**

In order to create a JSP Page, right-click on your project, and then select **New -> JSP** option as shown in the below image.

It will open the New JSP window. Here, give the File Name as “HelloWorld” and then click on the “Finish” button as shown below.

Once you click on the “Finish” button a JSP page will create automatically where edit the following code and save it (Ctrl+S).

For changing the port number for your server, go to service right click on Apache Tomcat -> Properties

Change the port number and click on the “close” button as shown in the below image.

##### **Step-3: Run your Project**

For running your project right-click on the project and select “Run” as follows.

The following output is generated through the specified JSP file.

In the next article, I am going to discuss the [**JSP API**](https://dotnettutorials.net/lesson/jsp-api/). Here, in this article, I try to explain the step-by-step process to create and run the **JSP Web Application using NetBeans IDE**. I hope you enjoy this JSP using NetBeans IDE article.

# Scriptlet Tag In Jsp

# **JSP Scriptlet tag (Scripting elements)**

[Scripting elements](https://www.javatpoint.com/jsp-scriptlet-tag)

[JSP scriptlet tag](https://www.javatpoint.com/jsp-scriptlet-tag#scriptlet)

[Simple Example of JSP scriptlet tag](https://www.javatpoint.com/jsp-scriptlet-tag#scriptletex1)

[Example of JSP scriptlet tag that prints the user name](https://www.javatpoint.com/jsp-scriptlet-tag#scriptletex2)

In JSP, java code can be written inside the jsp page using the scriptlet tag. Let's see what are the scripting elements first.

## **JSP Scripting elements**

The scripting elements provides the ability to insert java code inside the jsp. There are three types of scripting elements:

ADVERTISEMENT

ADVERTISEMENT

scriptlet tag

expression tag

declaration tag

### **JSP scriptlet tag**

A scriptlet tag is used to execute java source code in JSP. Syntax is as follows:

<%  java source code %>

### **Example of JSP scriptlet tag**

In this example, we are displaying a welcome message.

**<html>**

**<body>**

**<**% out.print("welcome to jsp"); %**>**

**</body>**

**</html>**

### **Example of JSP scriptlet tag that prints the user name**

In this example, we have created two files index.html and welcome.jsp. The index.html file gets the username from the user and the welcome.jsp file prints the username with the welcome message.

*File: index.html*

**<html>**

**<body>**

**<form** action="welcome.jsp"**>**

**<input** type="text" name="uname"**>**

**<input** type="submit" value="go"**><br/>**

**</form>**

**</body>**

**</html>**

*File: welcome.jsp*

<html>

<body>

<%

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

out.print("welcome "+name);

%>

</form>

</body>

</html>

# Expression Tag In Jsp

# **JSP expression tag**

The code placed within **JSP expression tag** is written to the output stream of the response. So you need not write out.print() to write data. It is mainly used to print the values of variable or method.

### **Syntax of JSP expression tag**

**<**%=  statement %**>**

### **Example of JSP expression tag**

In this example of jsp expression tag, we are simply displaying a welcome message.

**<html>**

**<body>**

**<**%= "welcome to jsp" %**>**

**</body>**

**</html>**

#### **Note: Do not end your statement with semicolon in case of expression tag.**

### **Example of JSP expression tag that prints current time**

To display the current time, we have used the getTime() method of Calendar class. The getTime() is an instance method of Calendar class, so we have called it after getting the instance of Calendar class by the getInstance() method.

*index.jsp*

**<html>**

**<body>**

Current Time: **<**%= java.util.Calendar.getInstance().getTime() %**>**

**</body>**

**</html>**

### **Example of JSP expression tag that prints the user name**

In this example, we are printing the username using the expression tag. The index.html file gets the username and sends the request to the welcome.jsp file, which displays the username.

*File: index.jsp*

**<html>**

**<body>**

**<form** action="welcome.jsp"**>**

**<input** type="text" name="uname"**><br/>**

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

**</form>**

**</body>**

**</html>**

*File: welcome.jsp*

**<html>**

**<body>**

**<**%= "Welcome "+request.getParameter("uname") %**>**

**</body>**

**</html>**

# Declaration Tag In Jsp

# **JSP Declaration Tag**

[JSP declaration tag](https://www.javatpoint.com/jsp-declaration-tag)

[Difference between JSP scriptlet tag and JSP declaration tag](https://www.javatpoint.com/jsp-declaration-tag#diff)

[Example of JSP declaration tag that declares field](https://www.javatpoint.com/jsp-declaration-tag#declarationex1)

[Example of JSP declaration tag that declares method](https://www.javatpoint.com/jsp-declaration-tag#declarationex2)

The **JSP declaration tag** is used to declare fields and methods.

The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet.

So it doesn't get memory at each request.

#### **Syntax of JSP declaration tag**

The syntax of the declaration tag is as follows:

**<**%!  field or method declaration %**>**

### **Difference between JSP Scriptlet tag and Declaration tag**

|  |  |
| --- | --- |
| **Jsp Scriptlet Tag** | **Jsp Declaration Tag** |
| The jsp scriptlet tag can only declare variables not methods. | The jsp declaration tag can declare variables as well as methods. |
| The declaration of scriptlet tag is placed inside the \_jspService() method. | The declaration of jsp declaration tag is placed outside the \_jspService() method. |

### **Example of JSP declaration tag that declares field**

In this example of JSP declaration tag, we are declaring the field and printing the value of the declared field using the jsp expression tag.

### **index.jsp**

**<html>**

**<body>**

**<**%! int data=50; %**>**

**<**%= "Value of the variable is:"+data %**>**

**</body>**

**</html>**

### **Example of JSP declaration tag that declares method**

In this example of JSP declaration tag, we are defining the method which returns the cube of given number and calling this method from the jsp expression tag. But we can also use jsp scriptlet tag to call the declared method.

### **index.jsp**

**<html>**

**<body>**

**<**%!

int cube(int n){

return n\*n\*n\*;

}

%**>**

**<**%= "Cube of 3 is:"+cube(3) %**>**

**</body>**

**</html>**

# Out Object

# **JSP Implicit Objects**

[JSP Implicit Objects](https://www.javatpoint.com/jsp-implicit-objects)

[out implicit object](https://www.javatpoint.com/jsp-implicit-objects#out)

[Example of out implicit object](https://www.javatpoint.com/jsp-implicit-objects#outex)

There are **9 jsp implicit objects**. These objects are *created by the web container* that are available to all the jsp pages.

The available implicit objects are out, request, config, session, application etc.

A list of the 9 implicit objects is given below:

|  |  |
| --- | --- |
| **Object** | **Type** |
| out | JspWriter |
| request | HttpServletRequest |
| response | HttpServletResponse |
| config | ServletConfig |
| application | ServletContext |
| session | HttpSession |
| pageContext | PageContext |
| page | Object |
| exception | Throwable |

### **JSP out implicit object**

For writing any data to the buffer, JSP provides an implicit object named out. It is the object of JspWriter. In case of servlet you need to write:

PrintWriter out=response.getWriter();

But in JSP, you don't need to write this code.

### **Example of out implicit object**

In this example we are simply displaying date and time.

### **index.jsp**

<html>

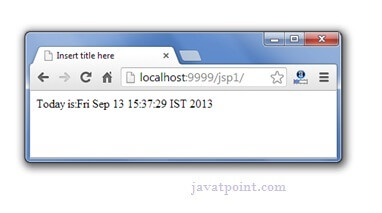
<body>

<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>

</body>

</html>

#### **Output**



# Request Object

# Request: Request() constructor

The **Request()** constructor creates a new [Request](https://developer.mozilla.org/en-US/docs/Web/API/Request) object.

## [Syntax](https://developer.mozilla.org/en-US/docs/Web/API/Request/Request#syntax)

JSCopy to Clipboard

new Request(input)

new Request(input, options)

### [Parameters](https://developer.mozilla.org/en-US/docs/Web/API/Request/Request#parameters)

[input](https://developer.mozilla.org/en-US/docs/Web/API/Request/Request#input)

Defines the resource that you wish to fetch. This can either be:

A string containing the URL of the resource you want to fetch. The URL may be relative to the base URL, which is the document's [baseURI](https://developer.mozilla.org/en-US/docs/Web/API/Node/baseURI) in a window context, or [WorkerGlobalScope.location](https://developer.mozilla.org/en-US/docs/Web/API/WorkerGlobalScope/location) in a worker context.

A [Request](https://developer.mozilla.org/en-US/docs/Web/API/Request) object, effectively creating a copy. Note the following behavioral updates to retain security while making the constructor less likely to throw exceptions:

If this object exists on another origin to the constructor call, the [Request.referrer](https://developer.mozilla.org/en-US/docs/Web/API/Request/referrer) is stripped out.

If this object has a [Request.mode](https://developer.mozilla.org/en-US/docs/Web/API/Request/mode) of navigate, the mode value is converted to same-origin.

[options](https://developer.mozilla.org/en-US/docs/Web/API/Request/Request#options) Optional

An object containing any custom settings that you want to apply to the request. The possible options are:

[method](https://developer.mozilla.org/en-US/docs/Web/API/Request/Request#method)

The request method, e.g., GET, POST. The default is GET.

[headers](https://developer.mozilla.org/en-US/docs/Web/API/Request/Request#headers)

Any headers you want to add to your request, contained within a [Headers](https://developer.mozilla.org/en-US/docs/Web/API/Headers) object or an object literal with [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String) values.

[body](https://developer.mozilla.org/en-US/docs/Web/API/Request/Request#body)

Any body that you want to add to your request: this can be a [Blob](https://developer.mozilla.org/en-US/docs/Web/API/Blob), an [ArrayBuffer](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/ArrayBuffer), a [TypedArray](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/TypedArray), a [DataView](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/DataView), a [FormData](https://developer.mozilla.org/en-US/docs/Web/API/FormData), a [URLSearchParams](https://developer.mozilla.org/en-US/docs/Web/API/URLSearchParams), a string, or a [ReadableStream](https://developer.mozilla.org/en-US/docs/Web/API/ReadableStream) object. Note that a request using the GET or HEAD method cannot have a body.

[mode](https://developer.mozilla.org/en-US/docs/Web/API/Request/Request#mode)

The mode you want to use for the request, e.g., cors, no-cors, same-origin, or navigate. The default is cors.

[credentials](https://developer.mozilla.org/en-US/docs/Web/API/Request/Request#credentials)

The request credentials you want to use for the request: omit, same-origin, or include. The default is same-origin.

[cache](https://developer.mozilla.org/en-US/docs/Web/API/Request/Request#cache)

The [cache mode](https://developer.mozilla.org/en-US/docs/Web/API/Request/cache) you want to use for the request.

[redirect](https://developer.mozilla.org/en-US/docs/Web/API/Request/Request#redirect)

The redirect mode to use: follow, error, or manual. The default is follow.

[referrer](https://developer.mozilla.org/en-US/docs/Web/API/Request/Request#referrer)

A string specifying no-referrer, client, or a URL. The default is about:client.

[referrerPolicy](https://developer.mozilla.org/en-US/docs/Web/API/Request/Request#referrerpolicy)

A string that changes how the referrer header is populated during certain actions (e.g., fetching subresources, prefetching, performing navigations).

[integrity](https://developer.mozilla.org/en-US/docs/Web/API/Request/Request#integrity)

Contains the [subresource integrity](https://developer.mozilla.org/en-US/docs/Web/Security/Subresource_Integrity) value of the request (e.g., sha256-BpfBw7ivV8q 2jLiT13fxDYAe2tJllusRSZ273h2nFSE=).

[keepalive](https://developer.mozilla.org/en-US/docs/Web/API/Request/Request#keepalive)

A boolean that indicates whether to make a persistent connection for multiple requests/responses.

[signal](https://developer.mozilla.org/en-US/docs/Web/API/Request/Request#signal)

An [AbortSignal](https://developer.mozilla.org/en-US/docs/Web/API/AbortSignal) object which can be used to communicate with/abort a request.

[priority](https://developer.mozilla.org/en-US/docs/Web/API/Request/Request#priority)

Specifies the priority of the fetch request relative to other requests of the same type. Must be one of the following strings:

high: A high priority fetch request relative to other requests of the same type.

low: A low priority fetch request relative to other requests of the same type.

auto: Automatically determine the priority of the fetch request relative to other requests of the same type (default).

If you construct a new Request from an existing Request, any options you set in an options argument for the new request replace any corresponding options set in the original Request. For example:

JSCopy to Clipboard

const oldRequest = new Request(

"https://github.com/mdn/content/issues/12959",

{ headers: { From: "webmaster@example.org" } },

);

oldRequest.headers.get("From"); // "webmaster@example.org"

const newRequest = new Request(oldRequest, {

headers: { From: "developer@example.org" },

});

newRequest.headers.get("From"); // "developer@example.org"

## [Errors](https://developer.mozilla.org/en-US/docs/Web/API/Request/Request#errors)

| **Type** | **Description** |
| --- | --- |
| TypeError | Since [Firefox 43](https://developer.mozilla.org/en-US/docs/Mozilla/Firefox/Releases/43), Request() will throw a TypeError if the URL has credentials, such as http://user:password@example.com. |

## [Examples](https://developer.mozilla.org/en-US/docs/Web/API/Request/Request#examples)

In our [Fetch Request example](https://github.com/mdn/dom-examples/tree/main/fetch/fetch-request) (see [Fetch Request live](https://mdn.github.io/dom-examples/fetch/fetch-request/)) we create a new Request object using the constructor, then fetch it using a [fetch()](https://developer.mozilla.org/en-US/docs/Web/API/fetch) call. Since we are fetching an image, we run [Response.blob](https://developer.mozilla.org/en-US/docs/Web/API/Response/blob) on the response to give it the proper MIME type so it will be handled properly, then create an Object URL of it and display it in an [<img>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/img) element.

JSCopy to Clipboard

const myImage = document.querySelector("img");

const myRequest = new Request("flowers.jpg");

fetch(myRequest)

.then((response) => response.blob())

.then((response) => {

const objectURL = URL.createObjectURL(response);

myImage.src = objectURL;

});

In our [Fetch Request with init example](https://github.com/mdn/dom-examples/tree/main/fetch/fetch-with-init-then-request) (see [Fetch Request init live](https://mdn.github.io/dom-examples/fetch/fetch-with-init-then-request/)) we do the same thing except that we pass in an options object when we invoke fetch():

JSCopy to Clipboard

const myImage = document.querySelector("img");

const myHeaders = new Headers();

myHeaders.append("Content-Type", "image/jpeg");

const myOptions = {

method: "GET",

headers: myHeaders,

mode: "cors",

cache: "default",

};

const myRequest = new Request("flowers.jpg", myOptions);

fetch(myRequest).then((response) => {

// ...

});

Note that you could also pass myOptions into the fetch call to get the same effect, e.g.:

JSCopy to Clipboard

fetch(myRequest, myOptions).then((response) => {

// ...

});

You can also use an object literal as headers in myOptions.

JSCopy to Clipboard

const myOptions = {

method: "GET",

headers: {

"Content-Type": "image/jpeg",

},

mode: "cors",

cache: "default",

};

const myRequest = new Request("flowers.jpg", myOptions);

You may also pass a [Request](https://developer.mozilla.org/en-US/docs/Web/API/Request) object to the Request() constructor to create a copy of the Request (This is similar to calling the [clone()](https://developer.mozilla.org/en-US/docs/Web/API/Request/clone) method.)

JSCopy to Clipboard

const copy = new Request(myRequest);

# Config Object

# **4) JSP config implicit object**

In JSP, config is an implicit object of type *ServletConfig*. This object can be used to get initialization parameter for a particular JSP page. The config object is created by the web container for each jsp page.

Generally, it is used to get initialization parameter from the web.xml file.

### **Example of config implicit object:**

**index.html**

**<form** action="welcome"**>**

**<input** type="text" name="uname"**>**

**<input** type="submit" value="go"**><br/>**

**</form>**

**web.xml file**

**<web-app>**

**<servlet>**

**<servlet-name>**sonoojaiswal**</servlet-name>**

**<jsp-file>**/welcome.jsp**</jsp-file>**

**<init-param>**

**<param-name>**dname**</param-name>**

**<param-value>**sun.jdbc.odbc.JdbcOdbcDriver**</param-value>**

**</init-param>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>**sonoojaiswal**</servlet-name>**

**<url-pattern>**/welcome**</url-pattern>**

**</servlet-mapping>**

**</web-app>**

**welcome.jsp**

**<**%

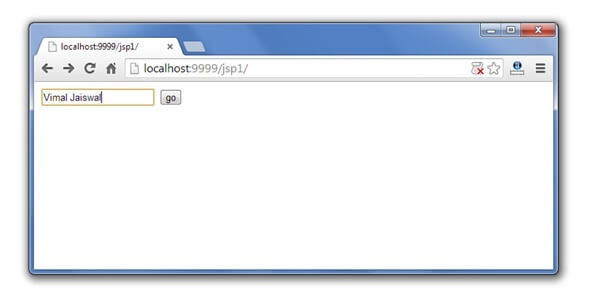
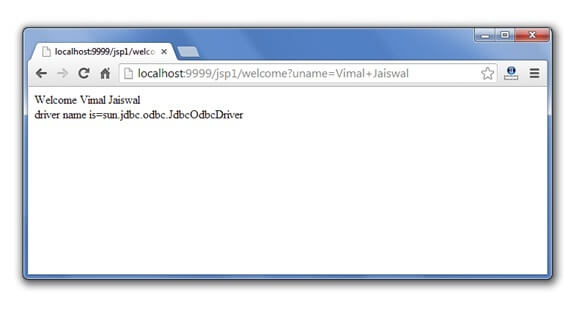
out.print("Welcome "+request.getParameter("uname"));

String driver=config.getInitParameter("dname");

out.print("driver name is="+driver);

%**>**

#### **Output**

# Session Object