UNIT # 2: ADVANCED TOPICS IN SERVLET

➢ INTRODUCTION

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?

Java Servlets are programs that run on a Web or Application server and act as a middle layer between a requests coming from a Web browser or other HTTP client and databases or applications on the HTTP server.

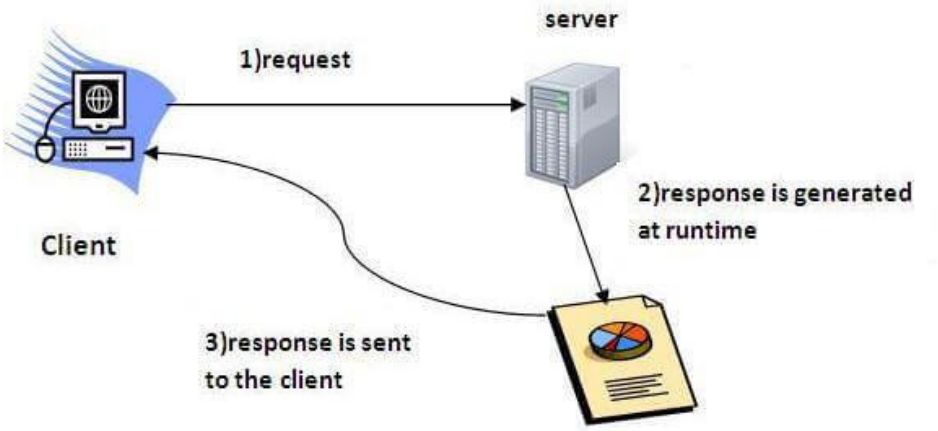
Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But Servlets offer several advantages in comparison with the CGI.

- Performance is significantly better.
- Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.
- Servlets are platform-independent because they are written in Java.
- Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So servlets are trusted.
- The full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seen already.

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.

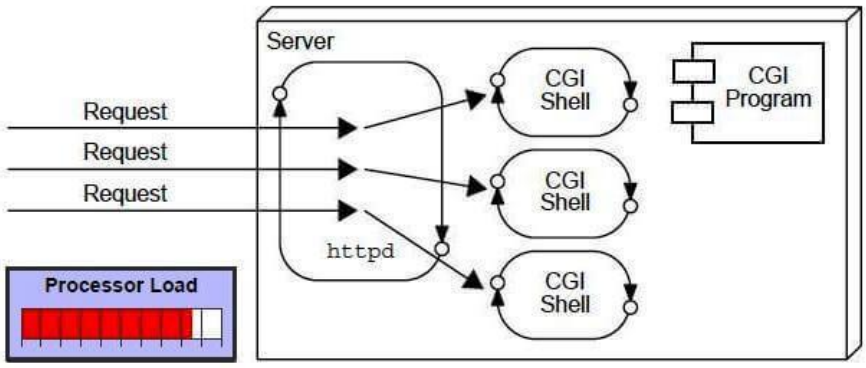


What is a web application?

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.

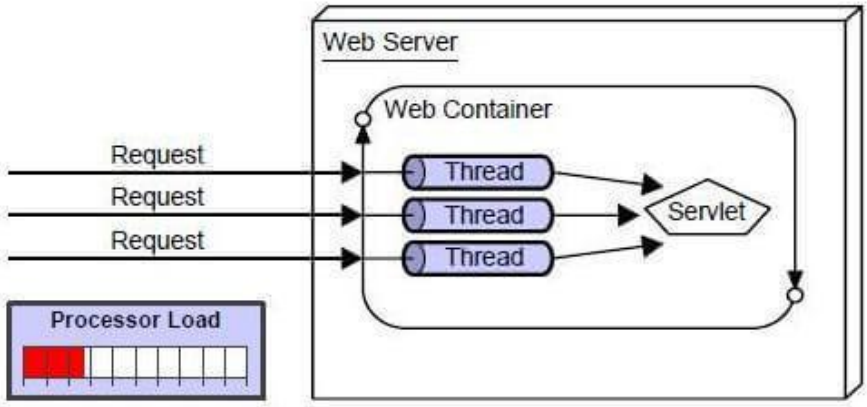


Disadvantages of CGI

There are many problems in CGI technology:

1. If the number of clients increases, it takes more time for sending the response.
2. For each request, it starts a process, and the web server is limited to start processes.
3. It uses platform dependent language e.g. C, C++, perl.

Advantages of Servlet

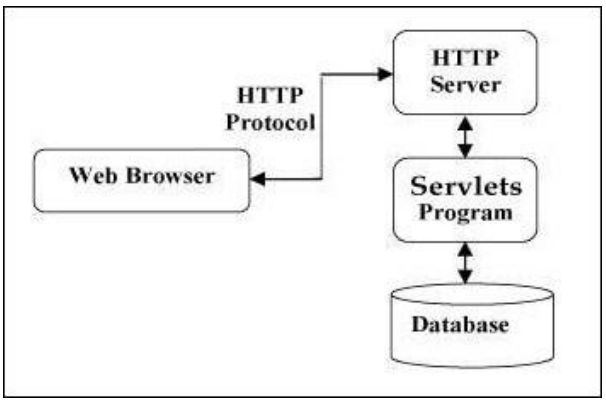


There are many advantages of Servlet over CGI. The web container creates threads for handling the multiple requests to the Servlet. Threads have many benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The advantages of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

1. **Better performance:** because it creates a thread for each request, not process.
2. **Portability:** because it uses Java language.
3. **Robust:** JVM manages Servlets, so we don't need to worry about the memory leak, garbage collection, etc.
4. **Secure:** because it uses java language.

Servlets Architecture

The following diagram shows the position of Servlets in a Web Application.



Servlets Tasks

Servlets perform the following major tasks −

- Read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.

- Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.
- Process the data and generate the results. This process may require talking to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.
- Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.
- Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

Servlets Packages

Java Servlets are Java classes run by a web server that has an interpreter that supports the Java Servlet specification.

Servlets can be created using the javax.servlet and javax.servlet.http packages, which are a standard part of the Java's enterprise edition, an expanded version of the Java class library that supports large-scale development projects.

These classes implement the Java Servlet and JSP specifications. At the time of writing this tutorial, the versions are Java Servlet 2.5 and JSP 2.1.

Java servlets have been created and compiled just like any other Java class. After you install the servlet packages and add them to your computer's Classpath, you can compile servlets with the JDK's Java compiler or any other current compiler.

➢ SERVLET LIFE CYCLE

The entire life cycle of a Servlet is managed by the Servlet container which uses the javax.servlet.Servlet interface to understand the Servlet object and manage it. So, before creating a Servlet object, let's first understand the life cycle of the Servlet object which is actually understanding how the Servlet container manages the Servlet object.

Stages of the Servlet Life Cycle: The Servlet life cycle mainly goes through four stages,
- Loading a Servlet.
- Initializing the Servlet.
- Request handling.
- Destroying the Servlet.

Let's look at each of these stages in details:

Loading a Servlet: The first stage of the Servlet lifecycle involves loading and initializing the Servlet by the Servlet container. The Web container or Servlet Container can load the Servlet at either of the following two stages :

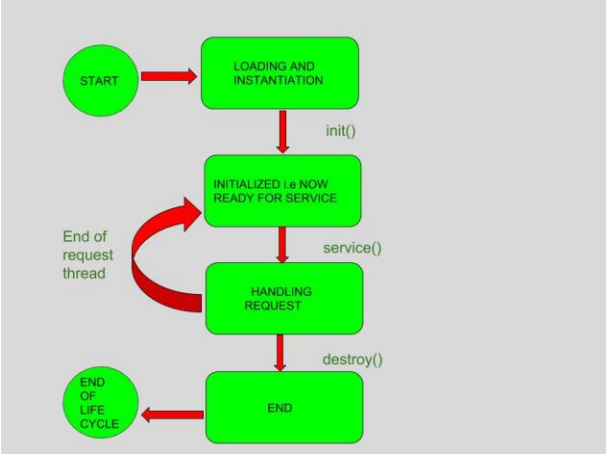
Initializing the context, on configuring the Servlet with a zero or positive integer value.

If the Servlet is not preceding stage, it may delay the loading process until the Web container determines that this Servlet is needed to service a request.

The Servlet container performs two operations in this stage :

Loading : Loads the Servlet class.

Instantiation : Creates an instance of the Servlet. To create a new instance of the Servlet, the container uses the no-argument constructor.



1. **Initializing a Servlet:** After the Servlet is instantiated successfully, the Servlet container initializes the instantiated Servlet object. The container initializes the Servlet object by invoking the Servlet.init(ServletConfig) method which accepts ServletConfig object reference as parameter.

   The Servlet container invokes the Servlet.init(ServletConfig) method only once, immediately after the Servlet.init(ServletConfig) object is instantiated successfully. This method is used to initialize the resources, such as JDBC datasource.

   Now, if the Servlet fails to initialize, then it informs the Servlet container by throwing the ServletException or UnavailableException.

2. **Handling request:** After initialization, the Servlet instance is ready to serve the client requests. The Servlet container performs the following operations when the Servlet instance is located to service a request :

   It creates the ServletRequest and ServletResponse objects. In this case, if this is a HTTP request, then the Web container creates HttpServletRequest and HttpServletResponse objects which are subtypes of the ServletRequest and ServletResponse objects respectively.

   After creating the request and response objects it invokes the Servlet.service(ServletRequest, ServletResponse) method by passing the request and response objects.

   The service() method while processing the request may throw the ServletException or UnavailableException or IOException.

3. **Destroying a Servlet:** When a Servlet container decides to destroy the Servlet, it performs the following operations,

It allows all the threads currently running in the service method of the Servlet instance to complete their jobs and get released.

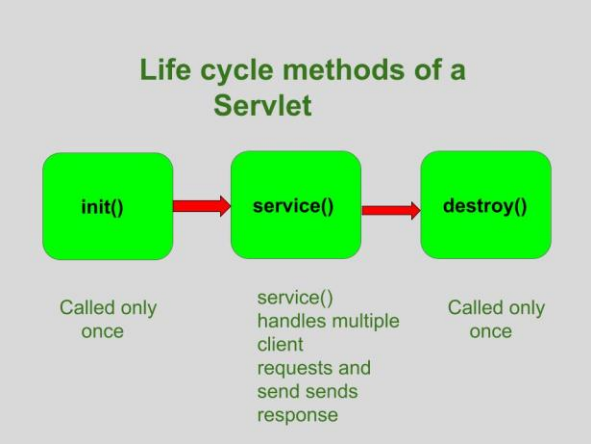
After currently running threads have completed their jobs, the Servlet container calls the destroy() method on the Servlet instance.

After the destroy() method is executed, the Servlet container releases all the references of this Servlet instance so that it becomes eligible for garbage collection.

Servlet Life Cycle Methods

There are three life cycle methods of a Servlet :

- init()
- service()
- destroy()



Let's look at each of these methods in details:

1. **init() method:** The Servlet.init() method is called by the Servlet container to indicate that this Servlet instance is instantiated successfully and is about to put into service.

```
//init() method


public class MyServlet implements Servlet{
  public void init(ServletConfig config) throws ServletException {
      //initialization code
  }
   //rest of code
}
```

2. **service() method:** The service() method of the Servlet is invoked to inform the Servlet about the client requests.

   This method uses ServletRequest object to collect the data requested by the client.

   This method uses ServletResponse object to generate the output content.

   // service() method

```
public class MyServlet implements Servlet{
   public void service(ServletRequest res, ServletResponse res)
   throws ServletException, IOException {
         // request handling code
   }
   // rest of code
}
```

3. **destroy() method:** The destroy() method runs only once during the lifetime of a Servlet and signals the end of the Servlet instance.

   //destroy() method

   public void destroy()

   As soon as the destroy() method is activated, the Servlet container releases the Servlet instance.

- **CODING A HTTPSERVLET**

  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:

1. By implementing Servlet interface,
2. By inheriting GenericServlet class, (or)
3. 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:

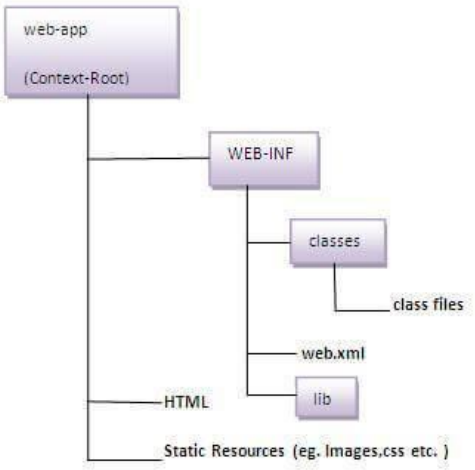
1. Create a directory structure
2. Create a Servlet
3. Compile the Servlet

4. Create a deployment descriptor
5. Start the server and deploy the project
6. Access the servlet

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.

1. By implementing the Servlet interface
2. By inheriting the GenericServlet class
3. 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

1. set classpath
2. 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.

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).

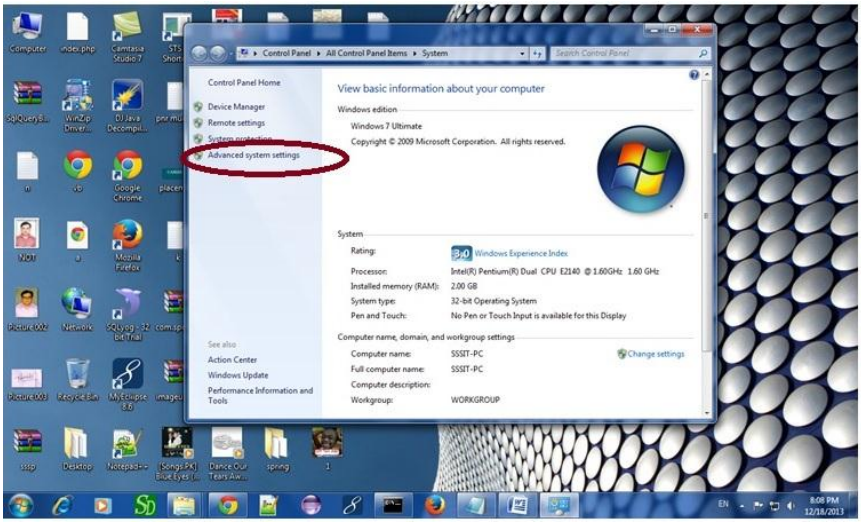
1) How to set JAVA_HOME in environment variable?

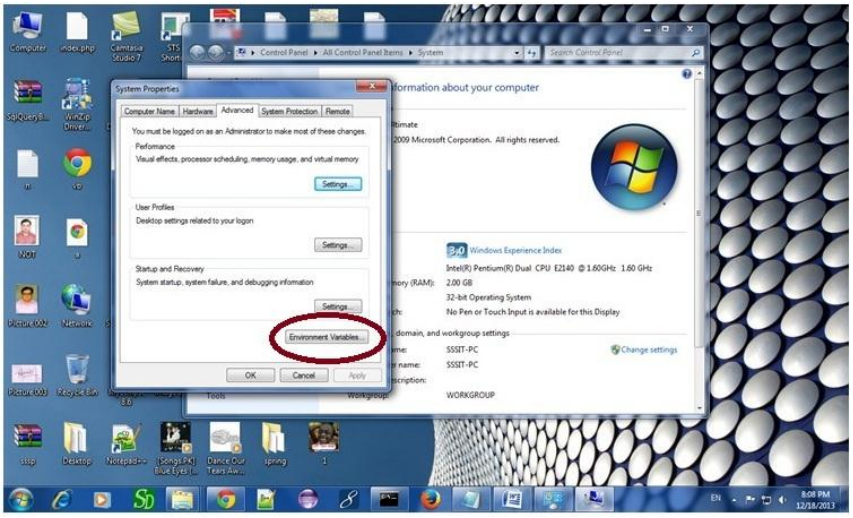
To start Apache Tomcat server JAVA_HOME and JRE_HOME must be set in Environment variables.

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:

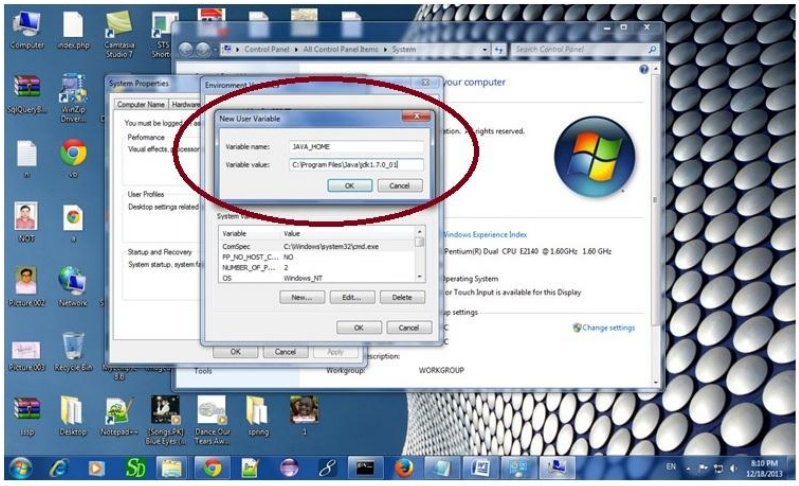
Click on advanced system settings tab then environment variables:





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:

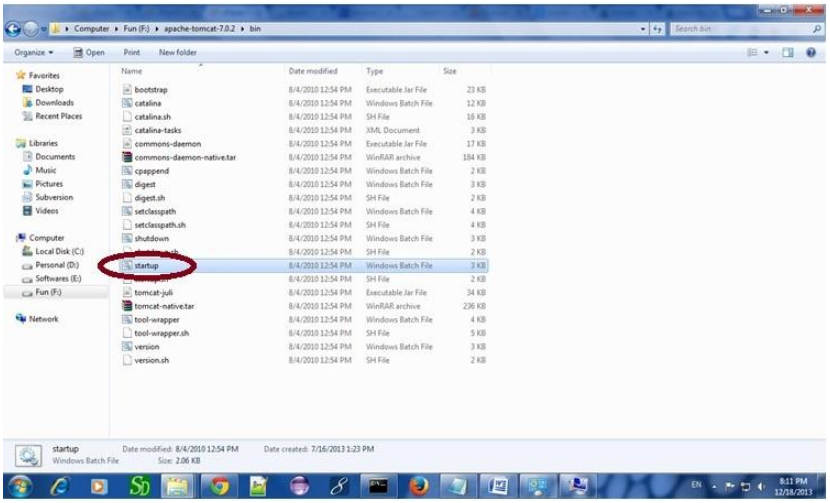


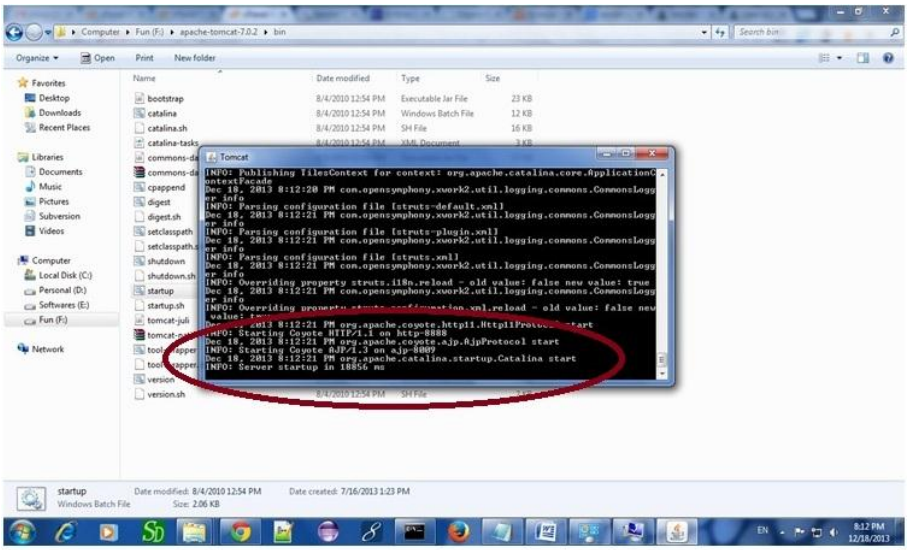
There must not be semicolon (;) at the end of the path.

After setting the JAVA_HOME double click on the startup.bat file in apache tomcat/bin.
Note: There are two types of tomcat available:

1. Apache tomcat that needs to extract only (no need to install)
2. Apache tomcat that needs to install

It is the example of apache tomcat that needs to extract only.





Now server is started successfully.

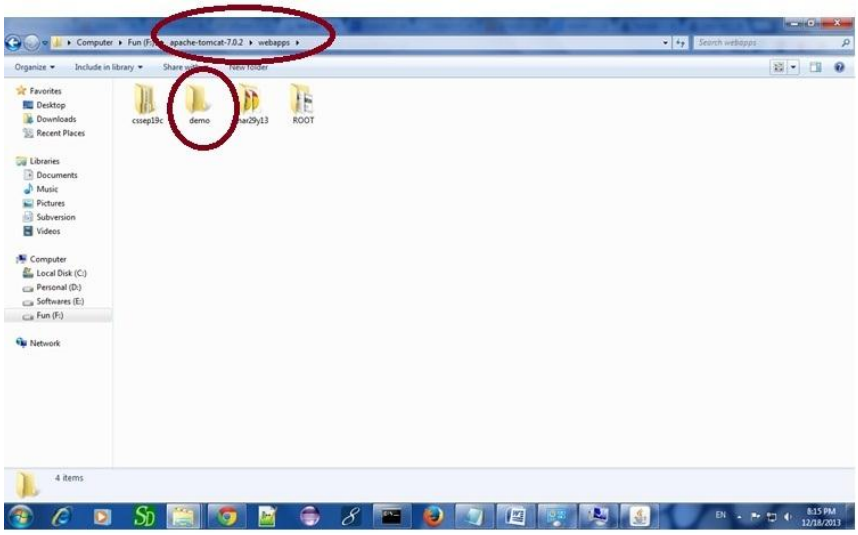
2) How to change port number of apache tomcat

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.

3) 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:

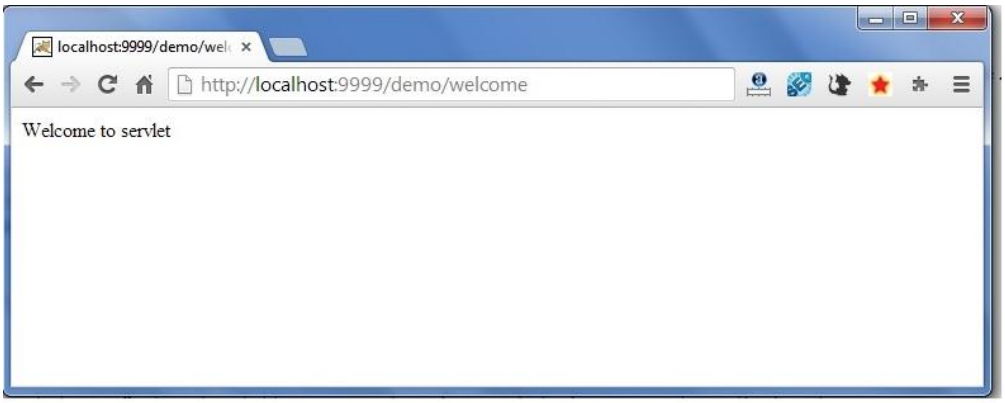
```
projectfolder> jar cvf myproject.war *
```

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

1. http://localhost:9999/demo/welcome



- **SERVLET CONFIGUARATION**

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

1. **public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.

2. **public Enumeration getInitParameterNames():**Returns an enumeration of all the initialization parameter names.
3. **public String getServletName():**Returns the name of the servlet.
4. **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**

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.

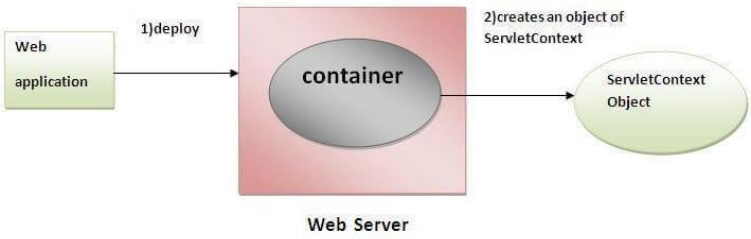
Advantage of ServletContext

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:

1. The object of ServletContext provides an interface between the container and servlet.
2. The ServletContext object can be used to get configuration information from the web.xml file.
3. The ServletContext object can be used to set, get or remove attribute from the web.xml file.
4. 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.

1. **public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.

2. **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters.
3. **public void setAttribute(String name,Object object):**sets the given object in the application scope.
4. **public Object getAttribute(String name):**Returns the attribute for the specified name.
5. **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters as an Enumeration of String objects.
6. **public void removeAttribute(String name):**Removes the attribute with the given name from the servlet context.

How to get the object of ServletContext interface

1. **getServletContext() method** of ServletConfig interface returns the object of ServletContext.
2. **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>
```

- **EVENTLISTENER**

Events are basically occurrence of something. Changing the state of an object is known as an event.

We can perform some important tasks at the occurrence of these exceptions, such as counting total and current logged-in users, creating tables of the database at time of deploying the project, creating database connection object etc.

There are many Event classes and Listener interfaces in the javax.servlet and javax.servlet.http packages.

Event classes

The event classes are as follows:

1. ServletRequestEvent
2. ServletContextEvent
3. ServletRequestAttributeEvent
4. ServletContextAttributeEvent
5. HttpSessionEvent
6. HttpSessionBindingEvent

Event interfaces

The event interfaces are as follows:

1. ServletRequestListener
2. ServletRequestAttributeListener
3. ServletContextListener
4. ServletContextAttributeListener
5. HttpSessionListener
6. HttpSessionAttributeListener
7. HttpSessionBindingListener
8. HttpSessionActivationListener

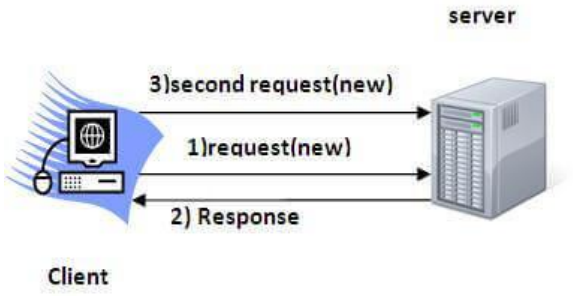
- **SESSION TRACKING :**

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:

1. **User Authorization**
2. **Cookies**
3. **Hidden Form Field**
4. **URL Rewriting**
5. **HttpSession**

1. **User Authorization**

We can bind the objects on HttpSession instance and get the objects by using setAttribute and getAttribute methods.

In the previous page, we have learnt about what is HttpSession, How to store and get data from session object etc.

Here, we are going to create a real world login and logout application without using database code. We are assuming that password is admin123.

In this example, we are creating 3 links: login, logout and profile. User can't go to profile page until he/she is logged in. If user is logged out, he need to login again to visit profile.

In this application, we have created following files.

- index.html
- link.html
- login.html
- LoginServlet.java
- LogoutServlet.java
- ProfileServlet.java
- web.xml

File: index.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Servlet Login Example</title>
</head>
<body>

<h1>Login App using HttpSession</h1>
<a href="login.html">Login</a>|
<a href="LogoutServlet">Logout</a>|
<a href="ProfileServlet">Profile</a>

</body>
</html>
```

File: link.html

```
<a href="login.html">Login</a> |
<a href="LogoutServlet">Logout</a> |
<a href="ProfileServlet">Profile</a>
<hr>
```

File: login.html

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

File: LoginServlet.java

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
public class LoginServlet extends HttpServlet {
   protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out=response.getWriter();
      request.getRequestDispatcher("link.html").include(request, response);

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

      if(password.equals("admin123")){
      out.print("Welcome, "+name);
      HttpSession session=request.getSession();
      session.setAttribute("name",name);
      }
      else{
         out.print("Sorry, username or password error!");
```

```
            request.getRequestDispatcher("login.html").include(request, response);
        }
        out.close();
    }
}
```

File: LogoutServlet.java

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
public class LogoutServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)

                        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();

        request.getRequestDispatcher("link.html").include(request, response);

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

        out.print("You are successfully logged out!");

        out.close();
    }
}
```

File: ProfileServlet.java

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
public class ProfileServlet extends HttpServlet {
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out=response.getWriter();
      request.getRequestDispatcher("link.html").include(request, response);

      HttpSession session=request.getSession(false);
      if(session!=null){
      String name=(String)session.getAttribute("name");

      out.print("Hello, "+name+" Welcome to Profile");
      }
      else{
         out.print("Please login first");
         request.getRequestDispatcher("login.html").include(request, response);
      }
      out.close();
   }
}
```

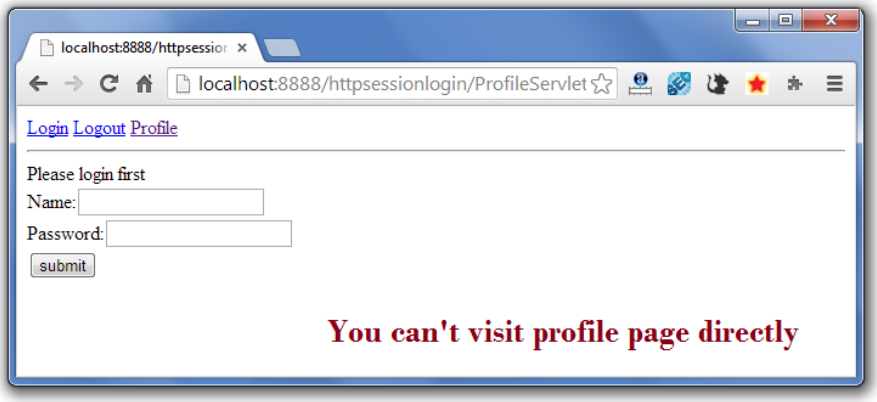
File: web.xml

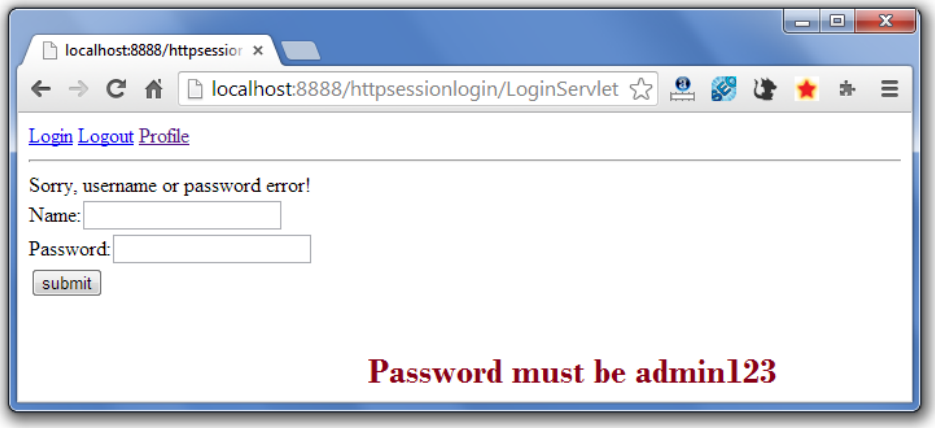
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

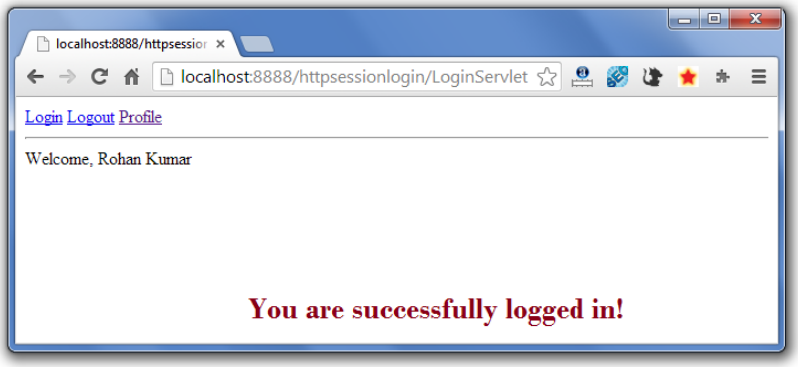
```
xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">

  <servlet>
    <description></description>
    <display-name>LoginServlet</display-name>
    <servlet-name>LoginServlet</servlet-name>
    <servlet-class>LoginServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>LoginServlet</servlet-name>
    <url-pattern>/LoginServlet</url-pattern>
  </servlet-mapping>
  <servlet>
    <description></description>
    <display-name>ProfileServlet</display-name>
    <servlet-name>ProfileServlet</servlet-name>
    <servlet-class>ProfileServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ProfileServlet</servlet-name>
    <url-pattern>/ProfileServlet</url-pattern>
  </servlet-mapping>
  <servlet>
    <description></description>
    <display-name>LogoutServlet</display-name>
    <servlet-name>LogoutServlet</servlet-name>
    <servlet-class>LogoutServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>LogoutServlet</servlet-name>
    <url-pattern>/LogoutServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

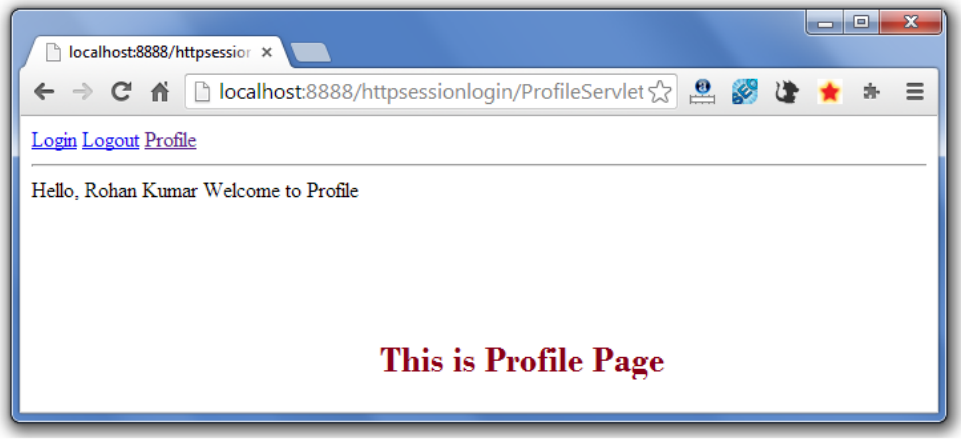
Output

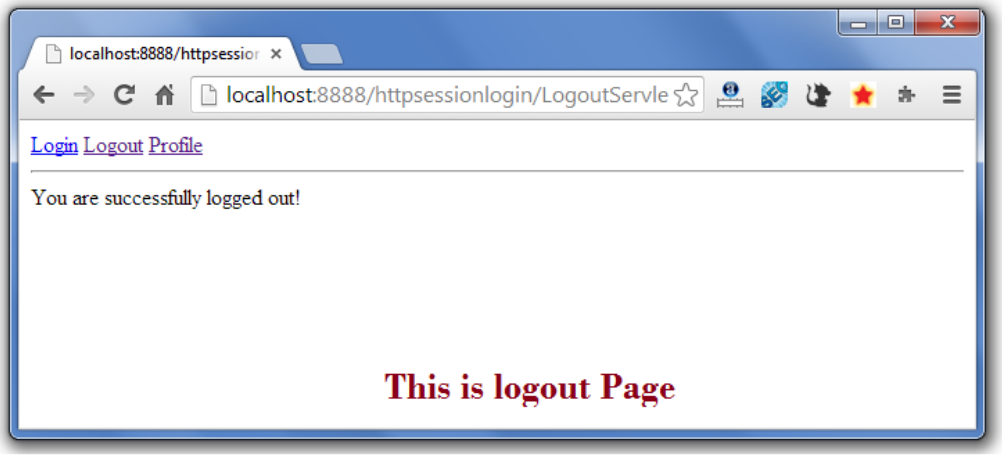

Insert title here
localhost:8888/httpsessionlogin/
Login App using HttpSession
Login Logout Profile


localhost:8888/httpsessior
localhost:8888/httpsessionlogin/ProfileServlet
Login Logout Profile
Please login first
Name:
Password:
submit
You can't visit profile page directly


localhost:8888/httpsessior
localhost:8888/httpsessionlogin/LoginServlet
Login Logout Profile
Sorry, username or password error!
Name:
Password:
submit
Password must be admin123


localhost:8888/httpsessior
localhost:8888/httpsessionlogin/LoginServlet
Login Logout Profile
Welcome, Rohan Kumar
You are successfully logged in!





2. Hidden Form Field

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

1. It will always work whether cookie is disabled or not.

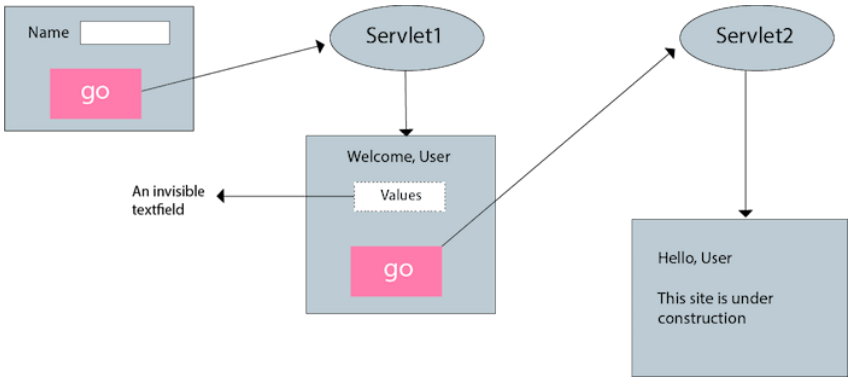
Disadvantage of Hidden Form Field:

1.It is maintained at server side.

2. Extra form submission is required on each pages.
3. 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

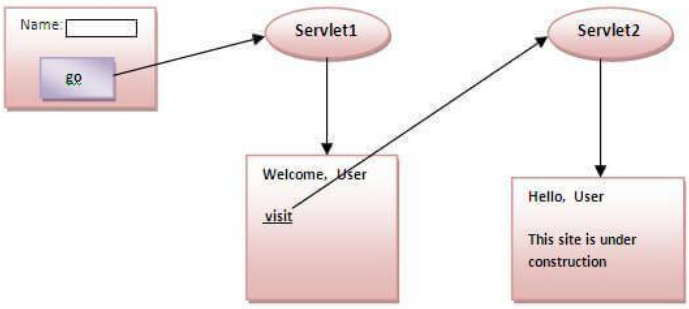
```
<web-app>

<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>
```

3. URL Rewriting

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

1. It will always work whether cookie is disabled or not (browser independent).
2. Extra form submission is not required on each pages.

Disadvantage of URL Rewriting

1. It will work only with links.
2. 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);}
            }


        }
```

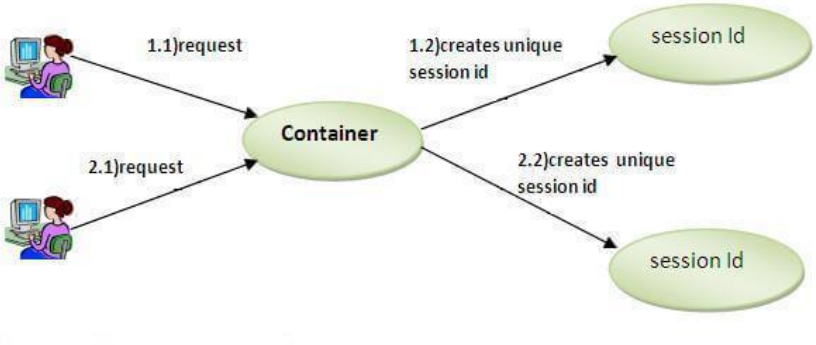
web.xml

```
<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>
```

4. **The Session Tracking API**

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:

1. bind objects
2. 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:

1. **public HttpSession getSession():**Returns the current session associated with this request, or if the request does not have a session, creates one.
2. **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

1. **public String getId():**Returns a string containing the unique identifier value.
2. **public long getCreationTime():**Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
3. **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.
4. **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>
```

❖ DATABASE CONNECTIVITY :

- **Relational Database**

What is a relational database?

A relational database is a collection of information that organizes data in predefined relationships where data is stored in one or more tables (or "relations") of columns and rows, making it easy to see and understand how different data structures relate to each other. Relationships are a logical connection between different tables, established on the basis of interaction among these tables.

Relational database defined

A relational database (RDB) is a way of structuring information in tables, rows, and columns. An RDB has the ability to establish links—or relationships–between information by joining tables, which makes it easy to understand and gain insights about the relationship between various data points.

The relational database model

Developed by E.F. Codd from IBM in the 1970s, the relational database model allows any table to be related to another table using a common attribute. Instead of using hierarchical structures to

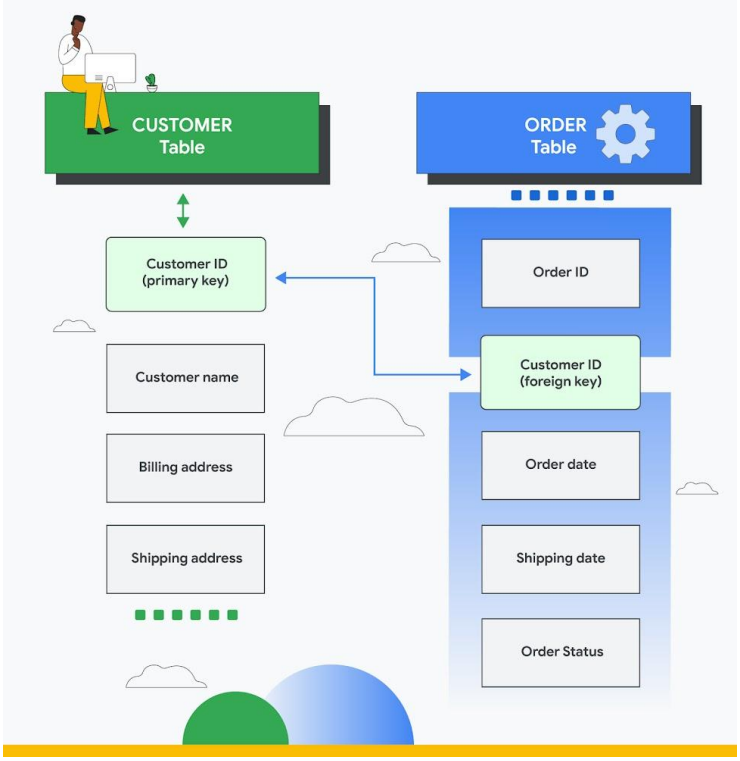
organize data, Codd proposed a shift to using a data model where data is stored, accessed, and related in tables without reorganizing the tables that contain them.

Think of the relational database as a collection of spreadsheet files that help businesses organize, manage, and relate data. In the relational database model, each “spreadsheet” is a table that stores information, represented as columns (attributes) and rows (records or tuples).

Attributes (columns) specify a data type, and each record (or row) contains the value of that specific data type. All tables in a relational database have an attribute known as the primary key, which is a unique identifier of a row, and each row can be used to create a relationship between different tables using a foreign key—a reference to a primary key of another existing table.

Let’s take a look at how the relational database model works in practice:

Say you have a Customer table and an Order table.



The Customer table contains data about the customer:

- Customer ID (primary key)
- Customer name
- Billing address
- Shipping address

In the Customer table, the customer ID is a primary key that uniquely identifies who the customer is in the relational database. No other customer would have the same Customer ID.

The Order table contains transactional information about an order:

- Order ID (primary key)
- Customer ID (foreign key)
- Order date
- Shipping date
- Order status

Here, the primary key to identify a specific order is the Order ID. You can connect a customer with an order by using a foreign key to link the customer ID from the Customer table.

The two tables are now related based on the shared customer ID, which means you can query both tables to create formal reports or use the data for other applications. For instance, a retail branch manager could generate a report about all customers who made a purchase on a specific date or figure out which customers had orders that had a delayed delivery date in the last month.

The above explanation is meant to be simple. But relational databases also excel at showing very complex relationships between data, allowing you to reference data in more tables as long as the data conforms to the predefined relational schema of your database.

As the data is organized as pre-defined relationships, you can query the data declaratively. A declarative query is a way to define what you want to extract from the system without expressing how the system should compute the result. This is at the heart of a relational system as opposed to other systems.

Examples of relational databases

Now that you understand how relational databases work, you can begin to learn about the many relational database management systems that use the relational database model. A relational database management system (RDBMS) is a program used to create, update, and manage relational databases. Some of the most well-known RDBMSs include MySQL, PostgreSQL, MariaDB, Microsoft SQL Server, and Oracle Database.

Benefits of relational databases

The main benefit of the relational database model is that it provides an intuitive way to represent data and allows easy access to related data points. As a result, relational databases are most

commonly used by organizations that need to manage large amounts of structured data, from tracking inventory to processing transactional data to application logging.

There are many other advantages to using relational databases to manage and store your data, including:

1. **Flexibility**

It's easy to add, update, or delete tables, relationships, and make other changes to data whenever you need without changing the overall database structure or impacting existing applications.

2. **ACID compliance**

Relational databases support ACID (Atomicity, Consistency, Isolation, Durability) performance to ensure data validity regardless of errors, failures, or other potential mishaps.

3. **Ease of use**

It's easy to run complex queries using SQL, which enables even non-technical users to learn how to interact with the database.

4. **Collaboration**

Multiple people can operate and access data simultaneously. Built-in locking prevents simultaneous access to data when it's being updated.

5. **Built-in security**

Role-based security ensures data access is limited to specific users.

6. **Database normalization**

Relational databases employ a design technique known as normalization that reduces data redundancy and improves data integrity.

Relational vs. non-relational databases

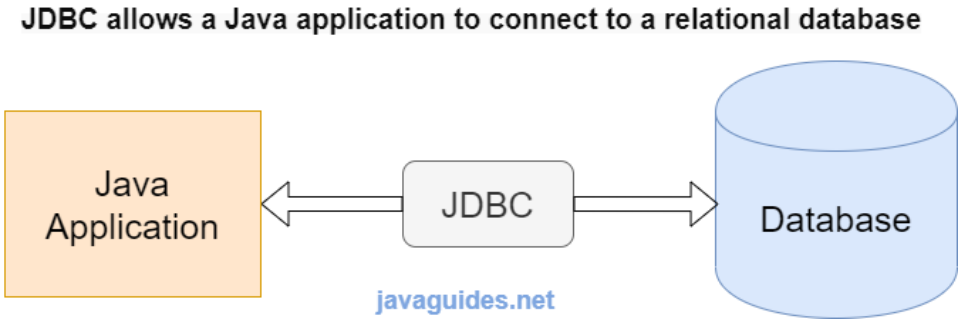
The main difference between relational and non-relational databases (NoSQL databases) is how data is stored and organized. Non-relational databases do not store data in a rule-based, tabular way. Instead, they store data as individual, unconnected files and can be used for complex, unstructured data types, such as documents or rich media files.

Unlike relational databases, NoSQL databases follow a flexible data model, making them ideal for storing data that changes frequently or for applications that handle diverse types of data.

- **The JDBC API**

What is JDBC?

Java Database Connectivity or JDBC API provides industry-standard and database-independent connectivity between the Java applications and relational database servers (relational databases, spreadsheets, and flat files).

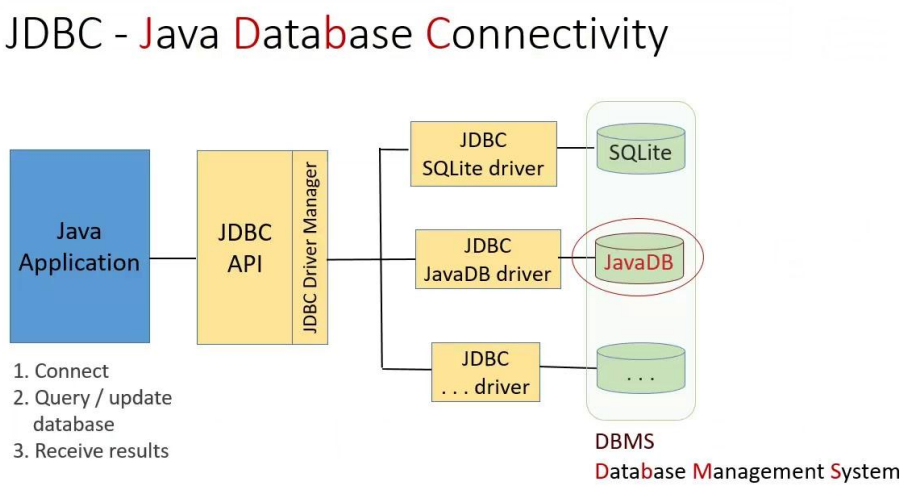


To keep it simple, JDBC allows a Java application to connect to a relational database. The major databases are supported such as Oracle, Microsoft SQL Server, DB2 and many others.

JDBC Flow

JDBC helps you to write Java applications that manage these three programming activities:

1. Connect to a data source, like a database
2. Send queries and update statements to the database
3. Retrieve and process the results received from the database in answer to your query



JDBC API

The JDBC API is comprised of two packages:

1. java.sql
2. javax.sql

We automatically get both packages when you download the Java Platform Standard Edition (Java SE) 8.

JDBC API consists of two parts – the first part is the JDBC API to be used by the application programmers. The second part is the low-level API to connect to a database server(JDBC Driver).

The first part of JDBC API is part of standard java packages in java.sql package. We use java.sql package API for accessing and processing data stored in a data source (usually a relational database) using the Java programming language.

For the second part is the JDBC driver(there are four different types of JDBC drivers) A JDBC driver is a set of Java classes that implement the JDBC interfaces, targeting a specific database. The JDBC interfaces come with standard Java, but the implementation of these interfaces is specific to the database you need to connect to. Such an implementation is called a JDBC driver.

Key Classes and Interfaces

1. JDBC Connection Interface
2. JDBC Statement Interface
3. JDBC PreparedStatement Interface
4. JDBC CallableStatement Interface
5. JDBC ResultSet Interface with Examples
6. JDBC ResultSetMetaData Interface
7. JDBC DatabaseMetaData Interface
8. JDBC DriverManager Class

JDBC Driver Types

To use the JDBC API with a particular database management system(MySQL, Oracle, etc), we need a JDBC technology-based driver to mediate between JDBC technology and the database. Depending on various factors, a driver might be written purely in the Java programming language or in a mixture of the Java programming language and Java Native Interface (JNI) native methods.

Install a JDBC driver from the vendor of your database. A JDBC driver is a set of Java classes that implement the JDBC interfaces, targeting a specific database. The JDBC interfaces come

with standard Java, but the implementation of these interfaces is specific to the database you need to connect to. Such an implementation is called a JDBC driver.

There are 4 different types of JDBC drivers:

Type 1: JDBC-ODBC bridge driver

Type 2: Java + Native code driver

Type 3: All Java + Middleware translation driver

Type 4: All Java driver.

If you are using Java DB (Apache Derby database), it already comes with a JDBC driver. If you are using MySQL, install the latest version of Connector/J.

Type 1: JDBC-ODBC bridge driver

Drivers that implement the JDBC API as a mapping to another data access API, such as ODBC (Open Database Connectivity). Drivers of this type are generally dependent on a native library, which limits their portability.

Note: The JDBC-ODBC Bridge should be considered a transitional solution. It is not supported by Oracle. Consider using this only if your DBMS does not offer a Java-only JDBC driver.

Type 2: Java + Native code driver

Drivers that are written partly in the Java programming language and partly in native code. These drivers use a native client library specific to the data source to which they connect. Again, because of the native code, their portability is limited. Oracle's OCI (Oracle Call Interface) client-side driver is an example of a Type 2 driver.

Type 3: All Java + Middleware translation driver

Type 3: Drivers that use a pure Java client and communicate with a middleware server using a database-independent protocol. The middleware server then communicates the client's requests to the data source.

Type 4: All Java driver

Type 4: Drivers that are pure Java and implement the network protocol for a specific data source. The client connects directly to the data source.

MySQL Connector/J is a Type 4 driver.

How to install a JDBC driver?

Installing a JDBC driver generally consists of copying the driver to your computer, then add the location of it to your classpath. In addition, many JDBC drivers other than Type 4 drivers require you to install a client-side API. No other special configuration is usually needed.

Fundamental Steps in JDBC

The fundamental steps involved in the process of connecting to a database and executing a query consist of the following:

1. Import JDBC packages
2. Load and register the JDBC driver // This step is not required in Java 6 and in JDBC 4.0
3. Open a connection to the database.
4. Create a statement object to perform a query.
5. Execute the statement object and return a query resultset.
6. Process the resultset.
7. Close the resultset and statement objects. // This step is not required because we use a try-with-resource statement to auto-close the resources
8. Close the connection. // This step is not required because we use a try-with-resource statement to auto-close the resources

From the above steps, we actually require below five steps to connect a Java application to the database (example: MySQL):

1. Import JDBC packages
2. Open a connection to the database.
3. Create a statement object to perform a query.
4. Execute the statement object and return a query resultset.
5. Process the resultset.

Key points

1. From JDBC 4.0, we don't need to include 'Class.forName()' in our code to load JDBC driver. JDBC 4.0 drivers that are found in your classpath are automatically loaded.
2. We have used try-with-resources statements to automatically close JDBC resources.

Step 1. Import JDBC packages

This is for making the JDBC API classes immediately available to the application program. The following import statement should be included in the program irrespective of the JDBC driver being used:

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

For individual JDBC API classes imports:

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

Step 2. Open a connection to the database

We use the getConnection() method of the DriverManager class to connect to the database.

Syntax: Here are overloaded getConnection() methods available:

```
Connection java.sql.DriverManager.getConnection(String url) throws SQLException
Connection java.sql.DriverManager.getConnection(String url, String username, String
password) throws SQLException
Connection java.sql.DriverManager.getConnection(String url, Properties info) throws
SQLException
```

Example: The following lines of code illustrate using the getConnection() method to connect to a MySQL database:

```
try (Connection connection = DriverManager
            .getConnection("jdbc:mysql://localhost:3306/mysql_database?useSSL=false",
"root", "root");
```

Step 3. Create a statement object to perform a query

We can use the createStatement() method is invoked on the current Connection object to create a SQL Statement.

Syntax:

```
public Statement createStatement() throws SQLException
```

Example: Let's create a Statement object using a connection.createStatement() method:

```
// Step 3:Create a statement using connection object
 Statement stmt = connection.createStatement();
```

Step 4. Execute the statement object and return a query resultset

Let's use the executeQuery() method of Statement interface is used to execute SQL statements.

Syntax:

```
public ResultSet executeQuery(String query) throws SQLException
```

Example:

```
// Step 4: Execute the query or update query
ResultSet rs = stmt.executeQuery(QUERY));
```

Step 5. Process the resultset

Below snippet shows how to process ResultSet object using while loop:

```
// Step 4: Process the ResultSet object.
            while (rs.next()) {
                int id = rs.getInt("id");
                String name = rs.getString("name");
                String email = rs.getString("email");
                String country = rs.getString("country");
                String password = rs.getString("password");
                System.out.println(id + "," + name + "," + email + "," + country +
"," + password);
            }
```

Complete Code

Let's put all the steps together and here is the complete example with output:

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

/**
 * Select Statement JDBC Example
 * @author Ramesh Fadatare
 *
 */
public class SelectStatementExample {
    private static final String QUERY = "select id,name,email,country,password from
Users";

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

```
        // using try-with-resources to avoid closing resources (boilerplate code)

        // Step 1: Establishing a Connection
        try (Connection connection = DriverManager
            .getConnection("jdbc:mysql://localhost:3306/mysql_database?useSSL=false",
"root", "root");

            // Step 2:Create a statement using connection object
            Statement stmt = connection.createStatement();

            // Step 3: Execute the query or update query
            ResultSet rs = stmt.executeQuery(QUERY)) {

            // Step 4: Process the ResultSet object.
            while (rs.next()) {
                int id = rs.getInt("id");
                String name = rs.getString("name");
                String email = rs.getString("email");
                String country = rs.getString("country");
                String password = rs.getString("password");
                System.out.println(id + "," + name + "," + email + "," + country +
"," + password);
            }
        } catch (SQLException e) {
            printSQLException(e);
        }
        // Step 4: try-with-resource statement will auto close the connection.
    }

    public static void printSQLException(SQLException ex) {
        for (Throwable e: ex) {
            if (e instanceof SQLException) {
                e.printStackTrace(System.err);
                System.err.println("SQLState: " + ((SQLException) e).getSQLState());
                System.err.println("Error Code: " + ((SQLException)
e).getErrorCode());
                System.err.println("Message: " + e.getMessage());
                Throwable t = ex.getCause();
                while (t != null) {
                    System.out.println("Cause: " + t);
                    t = t.getCause();
                }
            }
        }
    }
}
```

Output:

```
1,Ram,tony@gmail.com,US,secret
3,Pramod,pramod@gmail.com,India,123
4,Deepa,deepa@gmail.com,India,123
```

```
5,Tom,top@gmail.com,India,123
```

- **Transaction**

Transaction represents a single unit of work.

The ACID properties describes the transaction management well. ACID stands for Atomicity, Consistency, isolation and durability.

Atomicity means either all successful or none.

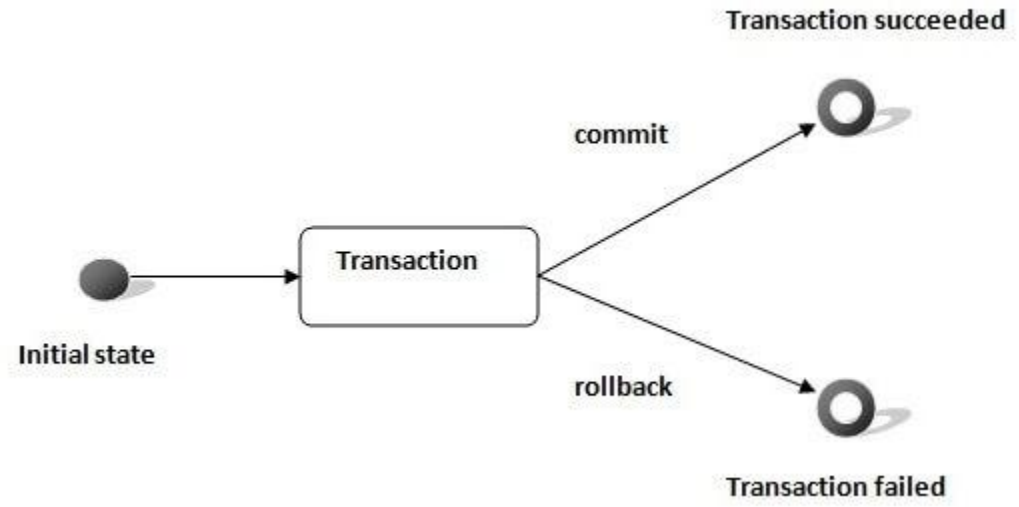
Consistency ensures bringing the database from one consistent state to another consistent state.

Isolation ensures that transaction is isolated from other transaction.

Durability means once a transaction has been committed, it will remain so, even in the event of errors, power loss etc.

Advantage of Transaction Mangaement

fast performance It makes the performance fast because database is hit at the time of commit.



In JDBC, Connection interface provides methods to manage transaction.

Method	Description
void setAutoCommit(boolean status)	It is true bydefault means each transaction is committed bydefault.
void commit()	commits the transaction.
void rollback()	cancels the transaction.

Simple example of transaction management in jdbc using Statement

Let's see the simple example of transaction management using Statement.

```
import java.sql.*;
class FetchRecords{
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");
con.setAutoCommit(false);

Statement stmt=con.createStatement();
stmt.executeUpdate("insert into user420 values(190,'abhi',40000)");
stmt.executeUpdate("insert into user420 values(191,'umesh',50000)");

con.commit();
con.close();
}}
```

If you see the table emp400, you will see that 2 records has been added.

Example of transaction management in jdbc using PreparedStatement

Let's see the simple example of transaction management using PreparedStatement.

```
import java.sql.*;
import java.io.*;
class TM{
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");
con.setAutoCommit(false);
```

```
PreparedStatement ps=con.prepareStatement("insert into user420 values(?,?,?)");

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
while(true){

System.out.println("enter id");
String s1=br.readLine();
int id=Integer.parseInt(s1);

System.out.println("enter name");
String name=br.readLine();

System.out.println("enter salary");
String s3=br.readLine();
int salary=Integer.parseInt(s3);

ps.setInt(1,id);
ps.setString(2,name);
ps.setInt(3,salary);
ps.executeUpdate();

System.out.println("commit/rollback");
String answer=br.readLine();
if(answer.equals("commit")){
con.commit();
}
if(answer.equals("rollback")){
con.rollback();
}


System.out.println("Want to add more records y/n");
String ans=br.readLine();
if(ans.equals("n")){
```

```
break;
}

}
con.commit();
System.out.println("record successfully saved");

con.close();//before closing connection commit() is called
}catch(Exception e){System.out.println(e);}

}}
```

It will ask to add more records until you press n. If you press n, transaction is committed.

➢ ADVANCED JDBC TECHNIQUES

Now that we've covered the basics, let's talk about a few advanced techniques that use servlets and JDBC. First, we'll examine how servlets can access stored database procedures. Then we'll look at how servlets can fetch complicated data types, such as binary data (images, applications, etc.), large quantities of text, or even executable database-manipulation code, from a database.

Stored Procedures

Most RDBMS systems include some sort of internal programming language. One example is Oracle's PL/SQL. These languages allow database developers to embed procedural application code directly within a database and then call that code from other applications. RDMBS programming languages are often well suited to performing certain database actions; many existing database installations have a number of useful stored procedures already written and ready to go. Most introductions to JDBC tend to skip over this topic, so we'll cover it here briefly.

```
CREATE OR REPLACE PROCEDURE sp_interest
(id IN INTEGER
bal IN OUT FLOAT) IS
BEGIN
SELECT balance
INTO bal
FROM accounts
WHERE account_id = id;

bal := bal + bal * 0.03;

UPDATE accounts
SET balance = bal
WHERE account_id = id;
```

```
END;
```

This procedure executes a SQL statement, performs a calculation, and executes another SQL statement. It would be fairly simple to write the SQL to handle this (in fact, the transaction example earlier in this chapter does something similar), so why bother with this at all? There are several reasons:

1. Stored procedures are precompiled in the RDBMS, so they run faster than dynamic SQL.
2. Stored procedures execute entirely within the RDBMS, so they can perform multiple queries and updates without network traffic.
3. Stored procedures allow you to write database manipulation code once and use it across multiple applications in multiple languages.
4. Changes in the underlying table structures require changes only in the stored procedures that access them; applications using the database are unaffected.
5. Many older databases already have a lot of code written as stored procedures, and it would be nice to be able to leverage that effort.

The Oracle PL/SQL procedure in our example takes an input value, in this case an account ID, and returns an updated balance. While each database has its own syntax for accessing stored procedures, JDBC creates a standardized escape sequence for accessing stored procedures using the java.sql.CallableStatement class. The syntax for a procedure that doesn't return a result is "{call procedure_name(?,?)}". The syntax for a stored procedure that returns a result value is "{? = call procedure_name(?,?)}". The parameters inside the parentheses are optional.

Using the CallableStatement class is similar to using the PreparedStatement class:

```
CallableStatment cstmt = con.prepareCall("{call sp_interest(?,?)}");
cstmt.registerOutParameter(2, java.sql.Types.FLOAT);
cstmt.setInt(1, accountID);
cstmt.execute();
out.println("New Balance: " + cstmt.getFloat(2));
```

This code first creates a CallableStatement using the prepareCall() method of Connection. Because this stored procedure has an output parameter, it uses the registerOutParameter() method of CallableStatement to identify that parameter as an output parameter of type FLOAT. Finally, the code executes the stored procedure and uses the getFloat() method of CallableStatement to display the new balance. The get*XXX*() methods in CallableStatement interface are similar to those in the ResultSet interface.