



BIJU PATNAIK UNIVERSITY OF TECHNOLOGY,
ODISHA

Lecture Notes

On

J2EE

Prepared by,
Dr. Subhendu Kumar Rath,
BPUT, Odisha.

INTRODUCTION TO SERVLET

Java Servlets are server side Java programs that require either a Web Server or an Application Server for execution. Examples for Web Servers include Apache's Tomcat Server and Macromedia's JRun. Web Servers include IBM's Weblogic and BEA's Websphere server. Examples for other Server programs include Java Server Pages (JSPs) and Enterprise Java Beans (EJBs). In the forthcoming sections, we will get acquainted with Servlet fundamentals and other associated information required for creating and executing Java Servlets.

Servlets are server side components that provide a powerful mechanism for developing server side programs. Servlets are server as well as platform-independent. This leaves you free to select a "best of breed" strategy for your servers, platforms, and tools. Using servlets web developers can create fast and efficient server side application which can run on any servlet enabled web server. Servlets run entirely inside the Java Virtual Machine. Since the Servlet runs at server side so it does not check the browser for compatibility. Servlets can access the entire family of Java APIs, including the JDBC API to access enterprise databases. Servlets can also access a library of HTTP-specific calls, receive all the benefits of the mature java language including portability, performance, reusability, and crash protection. Today servlets are the popular choice for building interactive web applications.

Servlets are not designed for a specific protocols. It is different thing that they are most commonly used with the HTTP protocols Servlets uses the classes in the java packages javax.servlet and javax.servlet.http. Servlets provides a way of creating the sophisticated server side extensions in a server as they follow the standard framework and use the highly portable java language.

HTTP Servlet typically used to:

- Provide dynamic content like getting the results of a database query and returning to the client.
- Process and/or store the data submitted by the HTML.
- Manage information about the state of a stateless HTTP. e.g. an online shopping cart manages request for multiple concurrent customers.

METHOD OF SERVLET

A Generic servlet contains the following five methods:

1. **init()**

public void init(ServletConfig config) throws ServletException

The **init() method** is called only once by the servlet container throughout the life of a servlet. By this init() method the servlet get to know that it has been placed into service.

The servlet cannot be put into the service if

- The init() method does not return within a fix time set by the web server.
- It throws a ServletException
- Parameters - The init() method takes a **ServletConfig** object that contains the initialization parameters and servlet's configuration and throws a **ServletException** if an exception has occurred.

2. **service()**

public void service(ServletRequest req, ServletResponse res) throws ServletException,IOException

Once the servlet starts getting the requests, the `service()` method is called by the servlet container to respond. The servlet services the client's request with the help of two objects. These two objects **`javax.servlet.ServletRequest`** and **`javax.servlet.ServletResponse`** are passed by the servlet container.

The status code of the response always should be set for a servlet that throws or sends an error.

Parameters - The `service()` method takes the **`ServletRequest`** object that contains the client's request and the object **`ServletResponse`** contains the servlet's response. The `service()` method throws **`ServletException`** and **`IOExceptions`** exception.

`getServletConfig()`

`public ServletConfig getServletConfig()`

This method contains parameters for initialization and startup of the servlet and returns a **`ServletConfig`** object. This object is then passed to the `init` method. When this interface is implemented then it stores the **`ServletConfig`** object in order to return it. It is done by the generic class which implements this interface.

Returns - the `ServletConfig` object

`getServletInfo()`

`public String getServletInfo()`

The information about the servlet is returned by this method like version, author etc. This method returns a string which should be in the form of plain text and not any kind of markup.

Returns - a string that contains the information about the servlet

3. `destroy()`

`public void destroy()`

This method is called when we need to close the servlet. That is before removing a servlet instance from service, the servlet container calls the `destroy()` method. Once the servlet container calls the `destroy()` method, no service methods will be then called. That is after the exit of all the threads running in the servlet, the `destroy()` method is called. Hence, the servlet gets a chance to clean up all the resources like memory, threads etc which are being held.

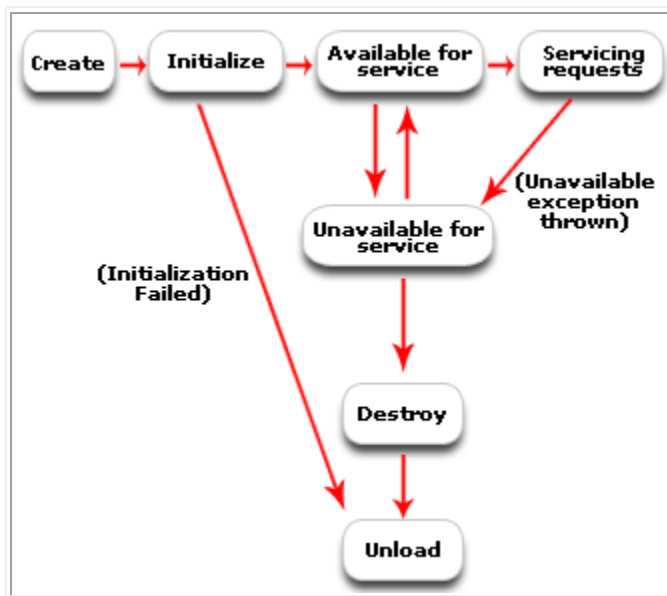
LIFE CYCLE OF SERVLET

The life cycle of a servlet can be categorized into four parts:

1. **Loading and Inatantiation:** The servlet container loads the servlet during startup or when the first request is made. The loading of the servlet depends on the attribute `<load-on-startup>` of web.xml file. If the attribute `<load-on-startup>` has a positive value then the servlet is load with loading of the container otherwise it load when the first request comes for service. After loading of the servlet, the container creates the instances of the servlet.
2. **Initialization:** After creating the instances, the servlet container calls the `init()` method and passes the servlet initialization parameters to the `init()` method. The `init()` must be called by the servlet container before the servlet can service any request. The initialization parameters persist untill the servlet is destroyed. The `init()` method is called only once throughout the life cycle of the servlet.

The servlet will be available for service if it is loaded successfully otherwise the servlet container unloads the servlet.

3. **Servicing the Request:** After successfully completing the initialization process, the servlet will be available for service. Servlet creates separate threads for each request. The servlet container calls the service() method for servicing any request. The service() method determines the kind of request and calls the appropriate method (doGet() or doPost()) for handling the request and sends response to the client using the methods of the response object.
4. **Destroying the Servlet:** If the servlet is no longer needed for servicing any request, the servlet container calls the destroy() method. Like the init() method this method is also called only once throughout the life cycle of the servlet. Calling the destroy() method indicates to the servlet container not to send the any request for service and the servlet releases all the resources associated with it. Java Virtual Machine claims for the memory associated with the resources for garbage collection.



Life Cycle of a Servlet

Advantages of Java Servlets

1. Portability
2. Powerful
3. Efficiency
4. Safety
5. Integration
6. Extensibility
7. Inexpensive

Each of the points are defined below:

1. Portability

As we know that the servlets are written in java and follow well known standardized APIs so they are highly portable across operating systems and server implementations. We can develop a servlet on Windows machine running the tomcat server or any other server and later we can deploy that servlet

effortlessly on any other operating system like Unix server running on the iPlanet/Netscape Application server. So servlets are **write once, run anywhere (WORA)** program.

2. Powerful

We can do several things with the servlets which were difficult or even impossible to do with CGI, for example the servlets can talk directly to the web server while the CGI programs can't do. Servlets can share data among each other, they even make the database connection pools easy to implement. They can maintain the session by using the session tracking mechanism which helps them to maintain information from request to request. It can do many other things which are difficult to implement in the CGI programs.

3.Efficiency

As compared to CGI the servlets invocation is highly efficient. When the servlet get loaded in the server, it remains in the server's memory as a single object instance. However with servlets there are N threads but only a single copy of the servlet class. Multiple concurrent requests are handled by separate threads so we can say that the servlets are highly scalable.

4.Safety

As servlets are written in java, servlets inherit the strong type safety of java language. Java's automatic garbage collection and a lack of pointers means that servlets are generally safe from memory management problems. In servlets we can easily handle the errors due to Java's exception handling mechanism. If any exception occurs then it will throw an exception.

5.Integration

Servlets are tightly integrated with the server. Servlet can use the server to translate the file paths, perform logging, check authorization, and MIME type mapping etc.

6.Extensibility

The servlet API is designed in such a way that it can be easily extensible. As it stands today, the servlet API support Http Servlets, but in later date it can be extended for another type of servlets.

7.Inexpensive

There are number of free web servers available for personal use or for commercial purpose. Web servers are relatively expensive. So by using the free available web servers you can add servlet support to it.

Basic Servlet Structure

As seen earlier, Java servlets are server side programs or to be more specific; web applications that run on servers that comply HTTP protocol. The javax.servlet and javax.servlet.http packages provide the necessary interfaces and classes to work with servlets. Servlets generally extend the HttpServlet class and override the doGet or the doPost methods. In addition, other methods such as init, service and destroy also called as life cycle methods might be used which will be discussed in the following section. The skeleton of a servlet is given in Figure

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

public class <<servlet name>> extends HttpServlet {

public void doGet (HttpServlet request, HttpServletResponse
response) throws ServletException, IOException {

// code for business logic here

// use request object to read client's requests

// user response object to throw output back to the client

} // close doGet
} // end program
```

Figure-Basic Servlet Structure

A Servlet Program

```
// Servlet program demonstrating it's life cycle
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class servlet_lifecycle extends HttpServlet {
    int i;

    public void init() throws ServletException
    {
        i=0; // initializing i value
    }

    // incrementing i value in doGet method
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        if (i==0)
        {
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet's Life Cycle</title>");
            out.println("</head>");
            out.println("<body>");
            out.print("<h1>i value initialized in init
method</h1>" + "<h1>" + i + "</h1>");
            out.println("</body>");
            out.println("</html>");
        }
    }
}
```

```

        i =i+1;
    if (i ==10)
        {
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet's Life Cycle</title>");
            out.println("</head>");
            out.println("<body>");
            out.print("<h1>i value reaches 10, hence calling
destroy method to reset it</h1>" + "<h1>" + i + "</h1>");
            out.println("</body>");
            out.println("</html>");

            destroy(); // call destroy method if i=10
        }

    if ( i < 10 ) // display incremented value of i
    {
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet's Life Cycle</title>");
        out.println("</head>");
        out.println("<body>");
        out.print("<h1>i value incremented in doGet</h1>" +
"<h1>" + i + "</h1>");
        out.println("</body>");
        out.println("</html>");
    }

}

public void destroy() // reset i value here
{
    i=0;
}
}

```

Output Screens

To appreciate the execution of the servlet life cycle methods, keep refreshing the browser (F5 in Windows). In the background, what actually happens is – with each refresh, the doGet method is called which increments i's value and displays the current value. Find below the screen shots (Figures 5 through 7) captured at random intervals. The procedure to run the servlets using a Web Server will be demonstrated in the next section (1.3.).

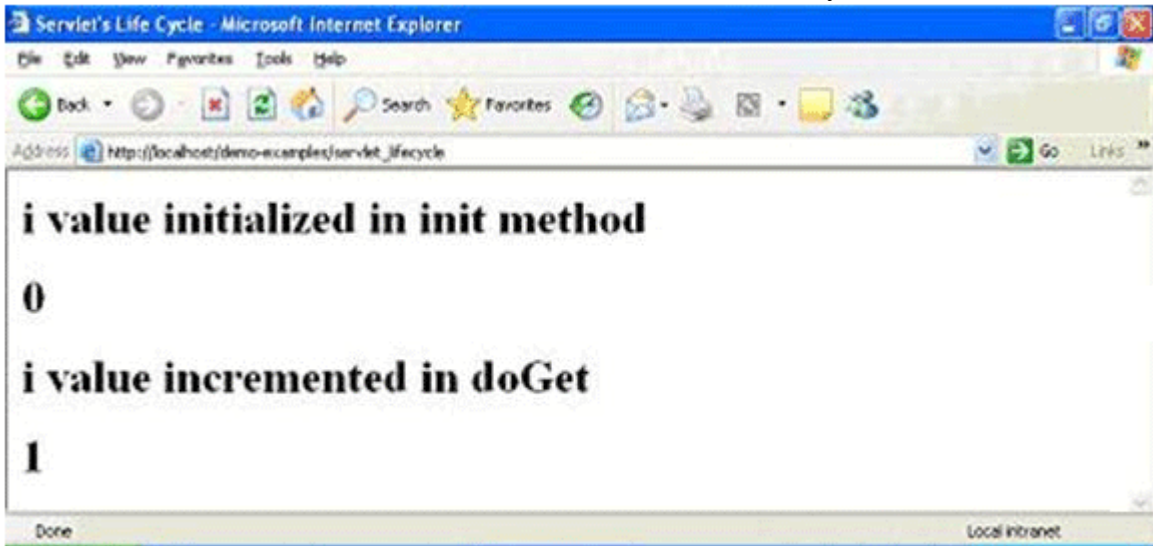


Figure- initial i value and incremented value

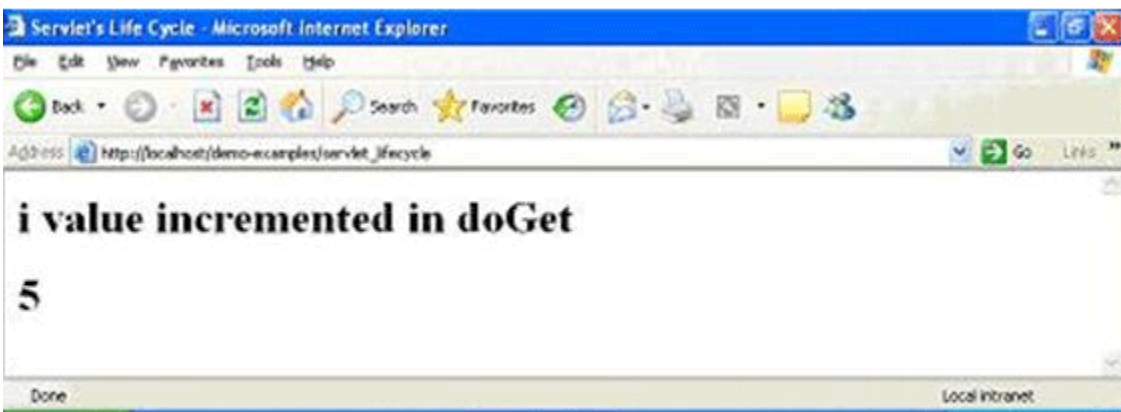


Figure- i value after repeated refreshing of browser

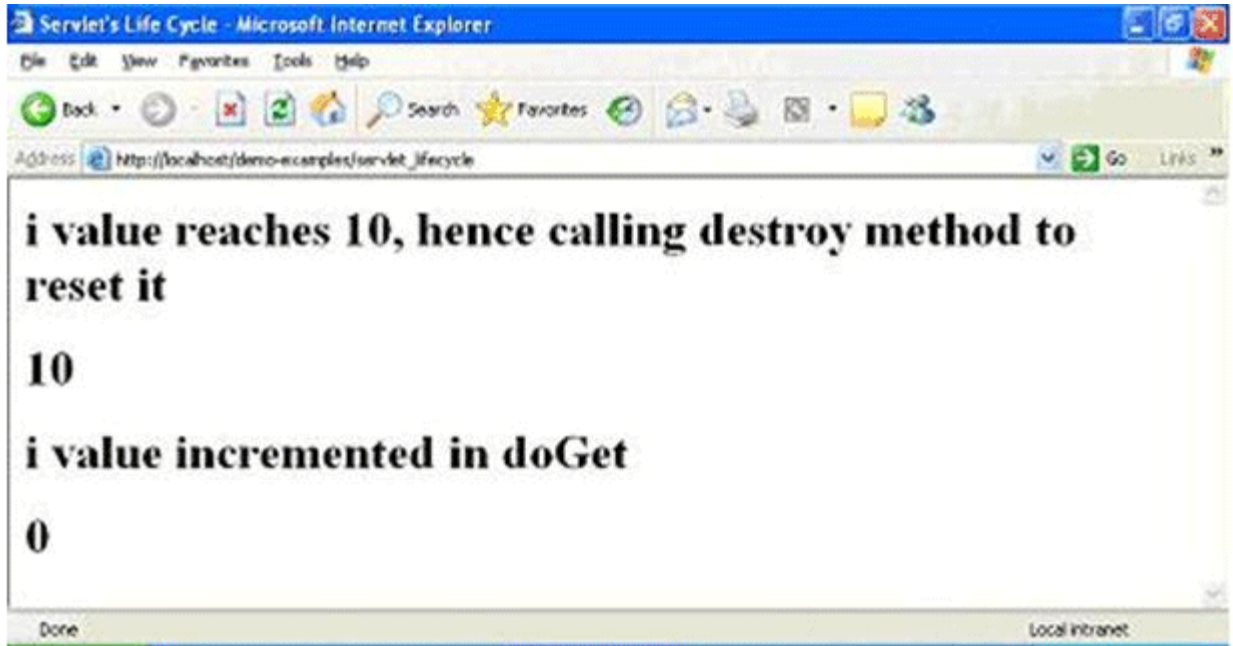


Figure- i value gets reset when its value equals 10

Installation, Configuration and running Servlets

In this section, we will see as how to install a WebServer, configure it and finally run servlets using this server. Throughout this tutorial, we will be using Apache's Tomcat server as the WebServer. Tomcat is not only an open and free server, but also the most preferred WebServer across the world. A few reasons we can attribute for its popularity is – Easy to install and configure, very less memory footprint, fast, powerful and portable. It is the ideal server for learning purpose.

1. Installation of Tomcat Server and JDK

As mentioned earlier, Apache's Tomcat Server is free software available for download @ www.apache.org. The current version of Tomcat Server is 6.0 (as of November 2007). This Server supports Java Servlets 2.5 and Java Server Pages (JSPs) 2.1 specifications. In case of doubt or confusion, you can refer to the abundant documentation repository available on this site.

Important software required for running this server is Sun's JDK (Java Development Kit) and JRE (Java Runtime Environment). The current version of JDK is 6.0. Like Tomcat, JDK is also free and is available for download at www.java.sun.com.

2. Configuring Tomcat Server

- Set JAVA_HOME variable - You have to set this variable which points to the base installation directory of JDK installation. (e.g. c:\program file\java\jdk1.6.0). You can either set this from the command prompt or from My Computer -> Properties -> Advanced -> Environment Variables.
- Specify the Server Port – You can change the server port from 8080 to 80 (if you wish to) by editing the server.xml file in the conf folder. The path would be something like this – c:\program files\apache software foundation\tomcat6\conf\server.xml

3. Run Tomcat Server

Once the above pre-requisites are taken care, you can test as whether the server is successfully installed as follows:

Step 1

- Go to C:\Program Files\Apache Software Foundation\Tomcat 6.0\bin and double click on tomcat6

OR

- Go to Start->Programs->Apache Tomcat 6.0 -> Monitor Tomcat. You will notice an icon appear on the right side of your Status Bar. Right click on this icon and click on Start service.

Step 2

- Open your Browser (e.g. MS Internet Explorer) and type the following URL :

http://localhost/ (If you have changed to port # to 80)

OR

- Open your Browser (e.g. MS Internet Explorer) and type the following URL :

http://localhost:8080/ (If you have NOT changed the default port #)

In either case, you should get a page similar to the one in Figure-8 which signifies that the Tomcat Server is successfully running on your machine.

4. Compile and Execute your Servlet

This section through a step by step (and illustration) approach explains as how to compile and then run a servlet using Tomcat Server. Though this explanation is specific to Tomcat, the procedure explained holds true for other Web servers too (e.g. JRun, Caucho's Resin).

Step 1 – Compile your servlet program

The first step is to compile your servlet program. The procedure is no different from that of writing and compiling a java program. But, the point to be noted is that neither the javax.servlet.* nor the javax.servlet.http.* is part of the standard JDK. It has to be exclusively added in the CLASSPATH. The set of classes required for writing servlets is available in a jar file called servlet-api.jar. This jar file can be downloaded from several sources. However, the easiest one is to use this jar file available with the Tomcat server (C:\Program Files\Apache Software Foundation\Tomcat 6.0\lib\servlet-api.jar). You need to include this path in CLASSPATH. Once you have done this, you will be able to successfully compile your servlet program. Ensure that the class file is created successfully.

Step 2 – Create your Web application folder

The next step is to create your web application folder. The name of the folder can be any valid and logical name that represents your application (e.g. bank_apps, airline_tickets_booking, shopping_cart,etc). But the most important criterion is that this folder should be created under webapps folder. The path would be similar or close to this - C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps. For demo purpose, let us create a folder called demo-examples under the webapps folder.

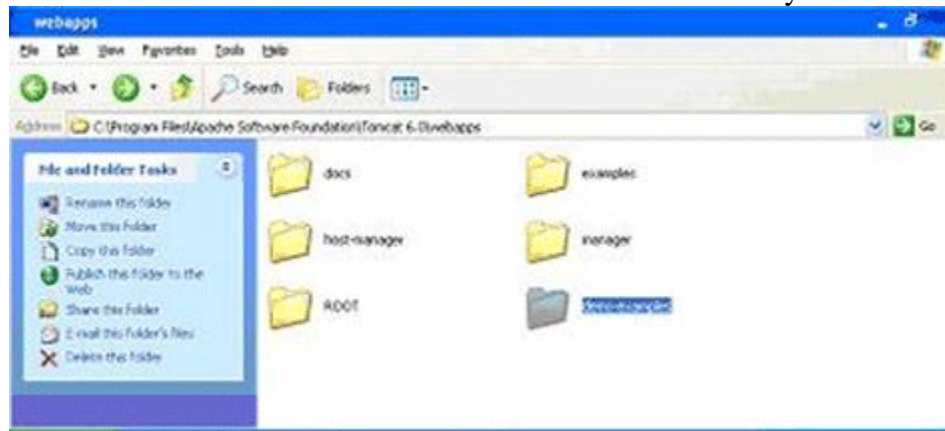


Figure- depicts the same.

Step 3 – Create the WEB-INF folder

The third step is to create the WEB-INF folder. This folder should be created under your web application folder that you created in the previous step. Figure-10 shows the WEB-INF folder being placed under the demo-examples folder.

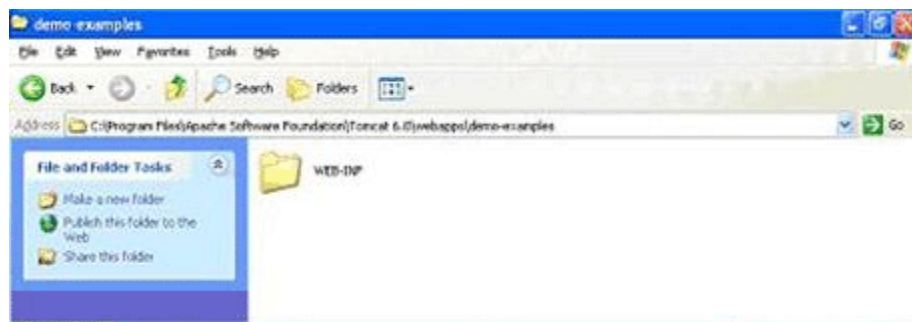


Figure – WEB-INF folder inside web application folder

Step 4 – Create the web.xml file and the classes folder

The fourth step is to create the web.xml file and the classes folder. Ensure that the web.xml and classes folder are created under the WEB-INF folder. Figure-11 shows this file and folder being placed under the WEB-INF folder.

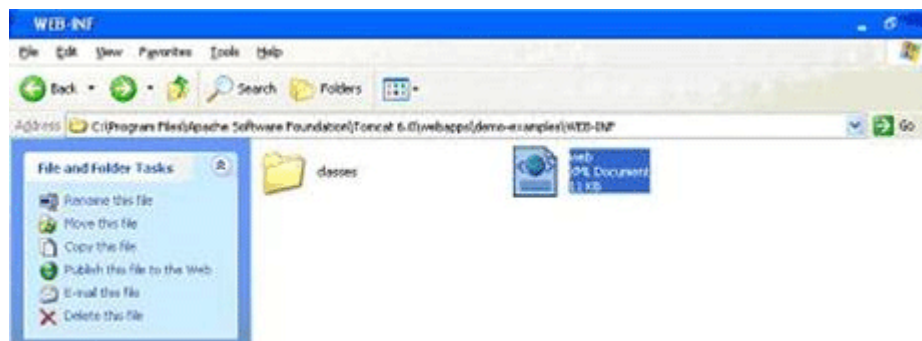


Figure – web.xml file and the classes folder

Note – Instead of creating the web.xml file an easy way would be to copy an existing web.xml file (e.g. C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\examples\WEB-INF) and paste it into this folder. You can later edit this file and add relevant information to your web

application.

Step 5 – Copy the servlet class to the classes folder

We need to copy the servlet class file to the classes folder in order to run the servlet that we created. All you need to do is copy the servlet class file (the file we obtained from Step 1) to this folder. Figure-12 shows the servlet_lifecycle (refer section 1.2.3.) class being placed in the classes folder.

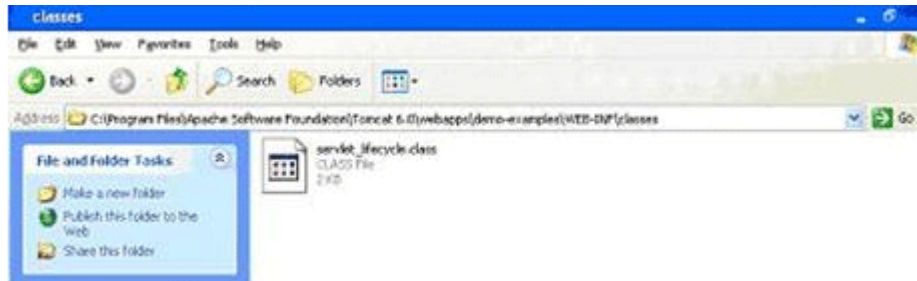
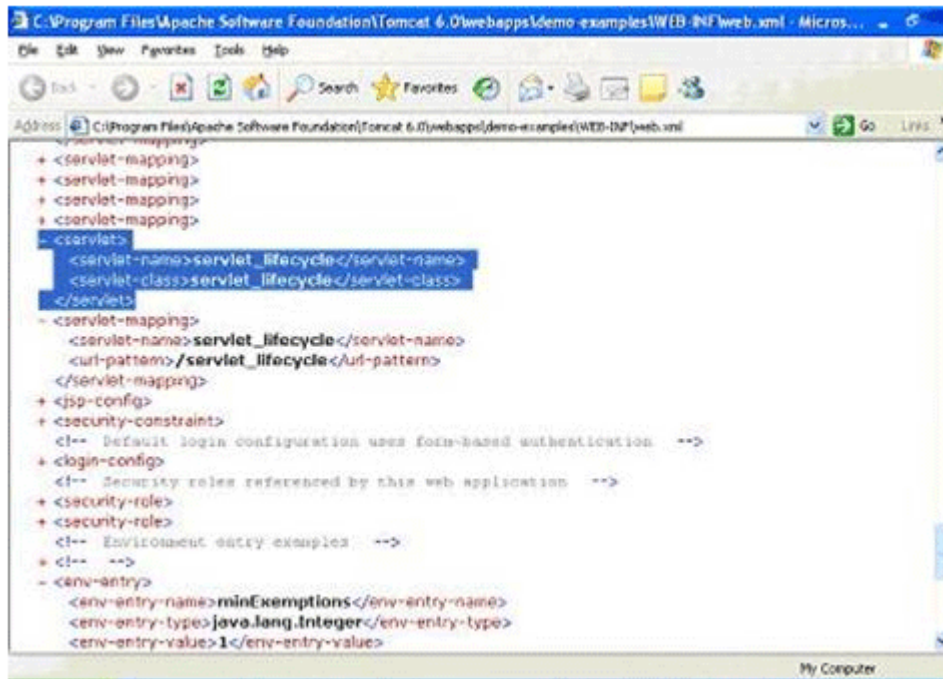


Figure – servlet class file placed under classes folder

Step 6 – Edit web.xml to include servlet's name and url pattern

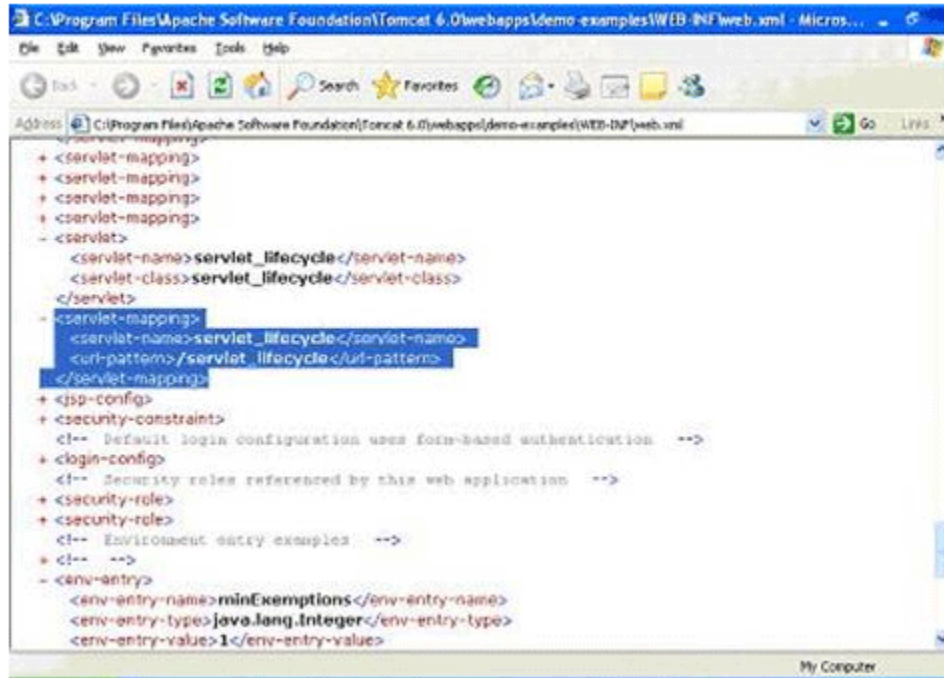
This step involves two actions viz. including the servlet's name and then mentioning the url pattern. Let us first see as how to include the servlet's name in the web.xml file. Open the web.xml file and include the servlet's name as shown in Figure-13.



Figure– Include servlet's name using the <servlet> </servlet> tag

Note – The servlet-name need not be the same as that of the class name. You can give a different name (or alias) to the actual servlet. This is one of the main reasons as why this tag is used for.

Next, include the url pattern using the <servlet-mapping> </servlet-mapping> tag. The url pattern defines as how a user can access the servlet from the browser. Figure-14 shows the url pattern entry for our current servlet.



```

+ <servlet-mapping>
+ <servlet-mapping>
+ <servlet-mapping>
+ <servlet-mapping>
- <servlet>
  <servlet-name> servlet_lifecycle </servlet-name>
  <servlet-class> servlet_lifecycle </servlet-class>
</servlet>
- <servlet-mapping>
  <servlet-name> servlet_lifecycle </servlet-name>
  <url-pattern> /servlet_lifecycle </url-pattern>
</servlet-mapping>
+ <jsp-config>
+ <security-constraint>
  <!-- Default login configuration uses form-based authentication -->
+ <login-config>
  <!-- Security roles referenced by this web application -->
+ <security-role>
+ <security-role>
  <!-- Environment entry examples -->
+ <!-- -->
- <env-entry>
  <env-entry-name> minExemptions </env-entry-name>
  <env-entry-type> java.lang.Integer </env-entry-type>
  <env-entry-value> 1 </env-entry-value>

```

Figure – Include url-pattern using the <servlet-mapping> </servlet-mapping> tag

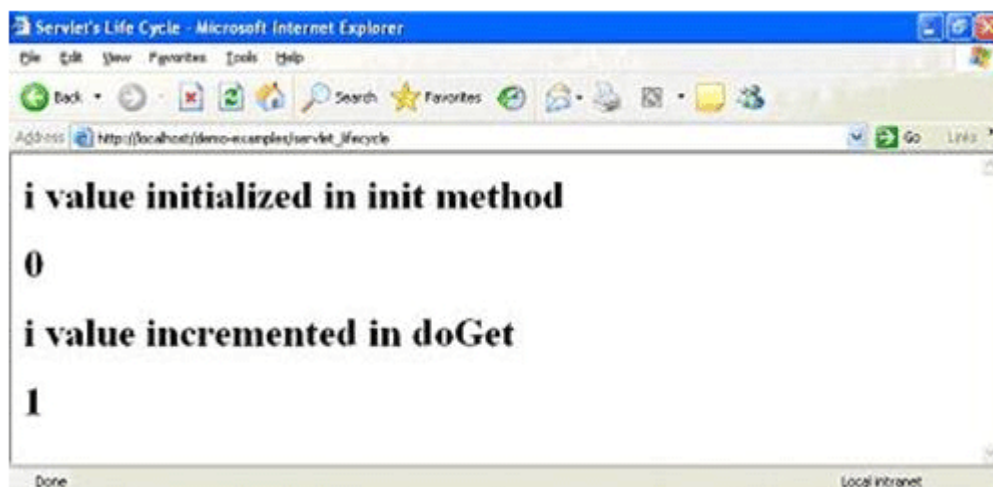
Note – Please remember that the path given in the url-pattern is a relative path. This means that this path is w.r.t. your web applications folder (demo-examples in this case).

Step 7 – Run Tomcat server and then execute your Servlet

This step again involves two actions viz. running the Web Server and then executing the servlet. To run the server, follow the steps explained in Section 1.3.3.

After ensuring that the web server is running successfully, you can run your servlet. To do this, open your web browser and enter the url as specified in the web.xml file. The complete url that needs to be entered in the browser is:

http://localhost/demo-examples/servlet_lifecycle



```

Servlet's Life Cycle - Microsoft Internet Explorer
Address http://localhost/demo-examples/servlet_lifecycle
i value initialized in init method
0
i value incremented in doGet
1
Done
Local intranet

```

Figure – Our servlet's output!

Eureka! Here's the output of our first servlet. After a long and pain staking effort, we finally got an output! As mentioned in Section 1.2.3. you can keep refreshing the browser window and see for yourself as how i value is incremented (a proof that the doGet is called every time you re-invoke a servlet).

Displaying Date in Servlet

In this example we are going to show how we can display a current date and time on our browser. It is very easy to display it on our browser by using the **Date** class of the **java.util** package.

As we know that the our servlet extends the **HttpServlet** and overrides the *doGet()* method which it inherits from the **HttpServlet** class. The server invokes doGet() method whenever web server receives the GET request from the servlet. The doGet() method takes two arguments first is **HttpServletRequest** object and the second one is **HttpServletResponse** object and this method throws the **ServletException**.

Whenever the user sends the request to the server then server generates two objects, first is **HttpServletRequest** object and the second one is **HttpServletResponse** object. **HttpServletRequest** object represents the client's request and the **HttpServletResponse** represents the servlet's response.

Inside the doGet() method our servlet has first used the *setContentType()* method of the response object which sets the content type of the response to *text/html*. It is the standard MIME content type for the Html pages. The MIME type tells the browser what kind of data the browser is about to receive. After that it has used the method *getWriter()* of the response object to retrieve a **PrintWriter** object. To display the output on the browser we use the *println()* method of the **PrintWriter** class.

The code the program is given below:

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

public class DisplayingDate extends HttpServlet{
    public void doGet(HttpServletRequest request, HttpServletResponse
        response) throws ServletException, IOException{
        PrintWriter pw = response.getWriter();
        Date today = new Date();
        pw.println("<html>"+<body><h1>Today Date is</h1>");
        pw.println("<b>"+ today+"</b></body>"+ "</html>");
    }
}
```

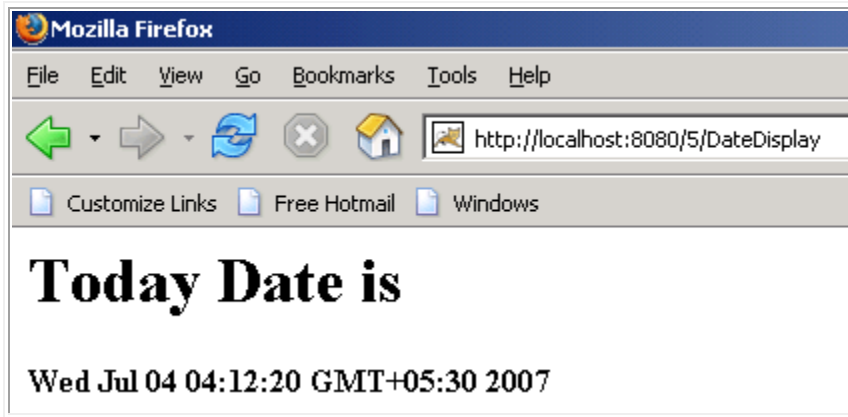
XML File for this program

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd"> -->

<web-app>
<servlet>
<servlet-name>Hello</servlet-name>
<servlet-class>DateDisplay</servlet-class>
</servlet>
```

```
<servlet-mapping>
<servlet-name>Hello</servlet-name>
<url-pattern>/DateDisplay</url-pattern>
</servlet-mapping>
</web-app>
```

The output of the program is given below:



A Holistic counter in Servlet

In this program we are going to make a such a servlet which will count the number it has been accessed and the number of threads created by the server.

In this example firstly we are going to create one class named as **HolisticCounterInServlet**. Now declare a variable counter of int with initial value 0, the value of this counter will be different for each servlet and create a **Hashtable** object. This object will be shared by all the threads in the container. Inside the **doGet()** method use the **getWriter()** method of the response object which will return the **PrintWriter** object.

The code of the program is given below:

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

public class HolisticCounter extends HttpServlet{
int counter = 0; //separate For Each Servlet
static Hashtable hashTable = new Hashtable(); //Shared by all the threads

public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
response.setContentType("text/html");
PrintWriter pw = response.getWriter();
counter++;
pw.println("This servlet has been accessed" + counter + "times<br>");
hashTable.put(this,this);
pw.println("There are currently" + hashTable.size() + "threads<br>");
```

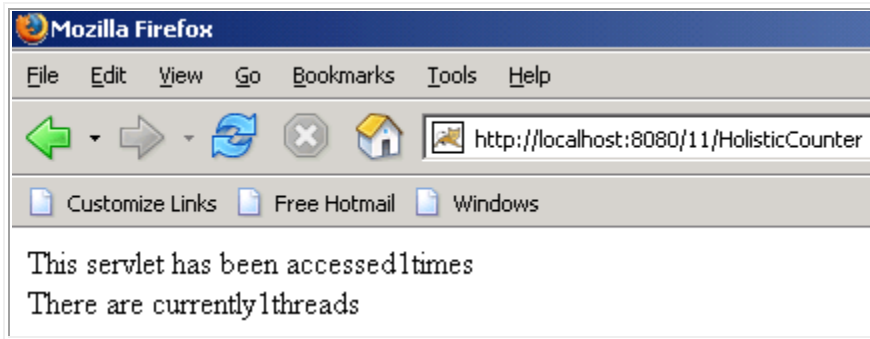


```

}
}

```

The output of the program is given below:



Counter in Init() Method

In this program we are going to make a such a servlet which will count and displays the number of times it has been accessed and by reading the init parameter to know from where the counting will begin.

In this program we are going to make use of the init method of the **Servlet** interface which takes one argument of ServletConfig. Firstly declare a variable counter which will have the initial value of the counter. The init() method accepts an object which implements ServletConfig interface. It uses the method getInitParameter() method of the ServletConfig interface to the value of the init parameter *initial* which we have defined in the deployment descriptor file. You need to *parse* the String value which you will get from the **getInitParameter()** method to a Integer.

The code of the program is given below:

```

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

public class CounterInInit extends HttpServlet {
    int counter;
    public void init(ServletConfig config) throws ServletException{
        super.init(config);
        String initialValue = config.getInitParameter("initial");
        try{
            counter = Integer.parseInt(initialValue);
        }
        catch(NumberFormatException e){
            counter = 0;
        }
    }
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {response.setContentType("text/html");
        PrintWriter pw = response.getWriter();
        counter++;
        pw.println("Since loading this servlet has been accessed" + counter + "times");
    }
}

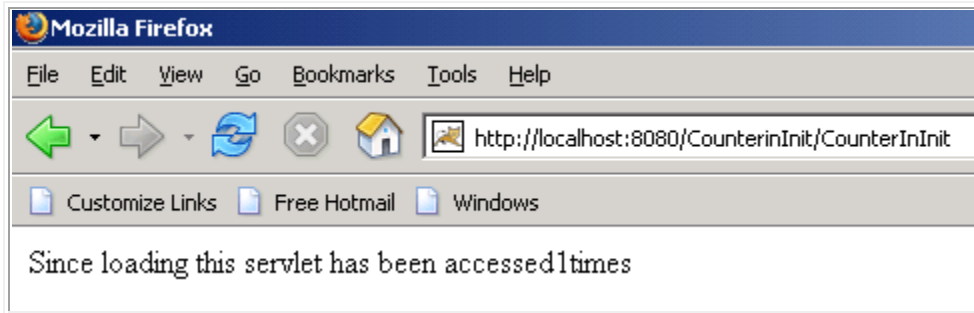
```

web.xml file for this program:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd"> -->

<web-app>
<servlet>
<servlet-name>Hello</servlet-name>
<servlet-class>CounterInInit</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>Hello</servlet-name>
<url-pattern>/CounterInInit</url-pattern>
</servlet-mapping>
</web-app>
```

The output of the program is given below:



Snooping the server

In this program we are going to tell you how can a use servlet to display information about its server.

Firstly we will create a class in which there will be **doGet()** method which takes two objects as arguments, first is request object and the second one is of response.

To display the name of the server you are using use the method **getServerName()** of the *ServletRequest* interface. To display the server port number use the method **getServerPort()**. You can also use other methods of the *ServletRequest* interface like **getProtocol()** to display the protocol you are using and many more methods depending on your needs.

The code of the program is given below:

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

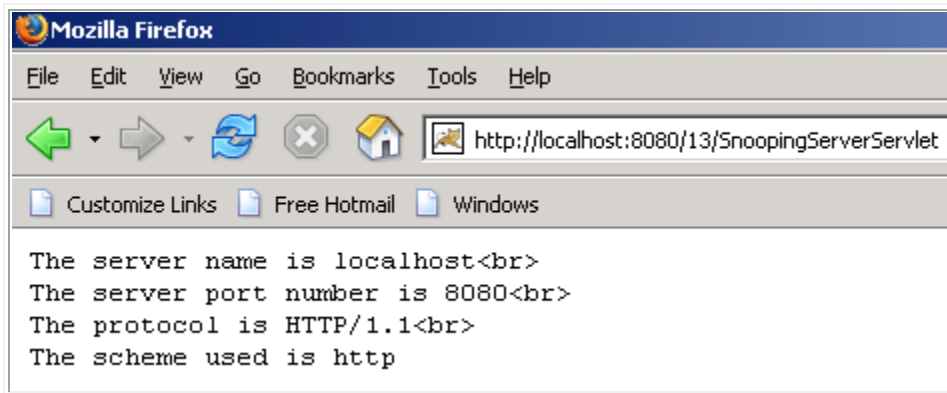
public class SnoopingServerServlet extends HttpServlet{
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        PrintWriter pw = response.getWriter();
        pw.println("The server name is " + request.getServerName() + "<br>");
        pw.println("The server port number is " + request.getServerPort()+ "<br>");
        pw.println("The protocol is " + request.getProtocol()+ "<br>");
        pw.println("The scheme used is " + request.getScheme());
    }
}
```

web.xml file for this program:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd"> -->

<web-app>
  <servlet>
    <servlet-name>Hello</servlet-name>
    <servlet-class>SnoopingServerServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Hello</servlet-name>
    <url-pattern>/SnoopingServerServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

The output of the program is given below:



Snooping Headers

In this program we are going to make a servlet which will retrieve all the Http request header.

To make a program over this firstly we need to make one class named **GettingSnoopingHeader**. In **HttpRequest** there are too many headers. To retrieve all the headers firstly we need to call the **getWriter()** which returns **PrintWriter** object and helps us to display all the headers. To get a header names call the method **getHeaderNames()** of the request object which will return the Enumeration of the headers. Now to retrieve all the headers from the Enumeration use the method **hasMoreElements()**. This method checks whether there are more headers or not. To display the output on your browser use the **PrintWriter** object.

The code of the program is given below:

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

public class HeaderSnoopServlet extends HttpServlet{
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    PrintWriter pw = response.getWriter();
    pw.println("Request Headers are");
  }
}
```


We are going to make one program on the dice roller in which the number in the dice will be selected randomly.

To make a program over this firstly we need to make a class **DiceRoller** in which we will have a **doGet()** method in which we will have our application logic. To make the dice working randomly use the **random()** method of the class **java.lang.Math**. To print the number on the browser call the method **getWriter()** of the response object which will return the **PrintWriter** object. Now by the object of the **PrintWriter** class print the values of the dice on the browser.

The code of the program is given below:

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

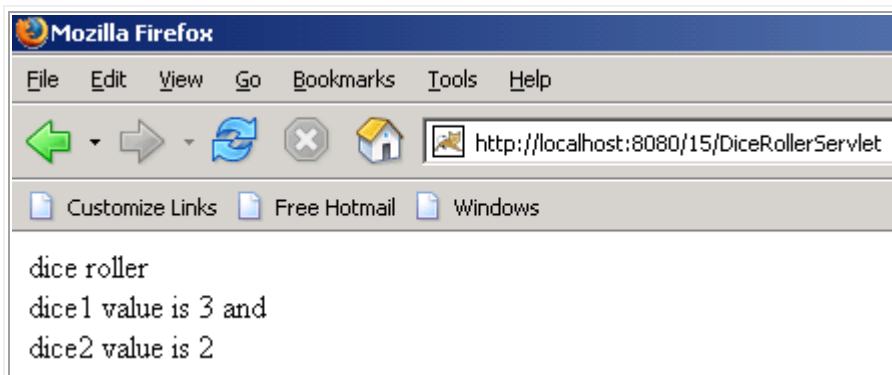
public class DiceRollerServlet extends HttpServlet{
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException{
    PrintWriter pw = response.getWriter();
    String dice1 = Integer.toString((int)(Math.random()*6)+1);
    String dice2 = Integer.toString((int)(Math.random()*6)+1);
    pw.println("<html><body>");
    pw.println("dice roller<br>");
    pw.println("dice1 value is " + dice1 + " and <br>dice2 value is " +dice2);
}
}
```

XML File for this program:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <servlet>
    <servlet-name>kailash</servlet-name>
    <servlet-class>DiceRollerServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Kailash</servlet-name>
    <url-pattern>/DiceRollerServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

The output of the program is given below:



Getting Init Parameter Names

In this example we are going to retrieve the init parameter values which we have given in the **web.xml** file.

Whenever the container makes a servlet it always reads its deployment descriptor file i.e. web.xml. Container creates name/value pairs for the **ServletConfig** object. Once the parameters are in **ServletConfig** they will never be read again by the Container.

The main job of the **ServletConfig** object is to give the init parameters.

To retrieve the init parameters in the program firstly we have made one class named **GettingInitParameterNames**. The container calls the servlet's **service()** method then depending on the type of request, the service method calls either the **doGet()** or the **doPost()**. By default it will be **doGet()** method. Now inside the **doGet()** method use **getWriter()** method of the **response** object which will return an object of the **PrintWriter** class which helps us to print the content on the browser.

To retrieve all the values of the init parameter use method **getInitParameterNames()** which will return the Enumeration of the init parameters.

The code of the program is given below:

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

public class InitServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        PrintWriter pw = response.getWriter();
        pw.print("Init Parameters are : ");
        Enumeration enumeration = getServletConfig().getInitParameterNames();
        while(enumeration.hasMoreElements()){
            pw.print(enumeration.nextElement() + " ");
        }
        pw.println("\nThe email address is " + getServletConfig().getInitParameter("AdminEmail"));
        pw.println("The address is " + getServletConfig().getInitParameter("Address"));
        pw.println("The phone no is " + getServletConfig().getInitParameter("PhoneNo"));
    }
}
```

web.xml file of this program:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <servlet>
    <init-param>
      <param-name>AdminEmail</param-name>
      <param-value>kailash@yahoo.co.in</param-value>
    </init-param>
    <init-param>
      <param-name>Address</param-name>
      <param-value>BBSR</param-value>
    </init-param>
    <init-param>
```

```

<param-name>PhoneNo</param-name>
<param-value>9853271986</param-value>
</init-param>
  <servlet-name>Kailash</servlet-name>
  <servlet-class>InitServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>Kailash</servlet-name>
<url-pattern>/InitServlet</url-pattern>
</servlet-mapping>
</web-app>

```

The output of the program is given below:

TRY YOURSELF AND GET THE O/P

Passing Parameter Using Html Form

This is a very simple example in which we are going to display the name on the browser which we have entered from the Html page.

To get the desired result firstly we have to make one html form which will have only one field named as name in which we will enter the name. And we will also have one submit button, on pressing the submit button the request will go to the server and the result will be displayed to us.

In the servlet which will work as a controller here picks the value from the html page by using the method `getParameter()`. The output will be displayed to you by the object of the **PrintWriter** class.

The code of the program is given below:

```

<html>

<head>
<title>New Page 1</title>
</head>

<body>

<h2>Login</h2>
<p>Please enter your username and password</p>
<form method="GET" action="/htmlform/LoginServlet">
  <p> Username <input type="text" name="username" size="20"></p>
  <p> Password <input type="text" name="password" size="20"></p>
  <p><input type="submit" value="Submit" name="B1"></p>
</form>
<p>&nbsp;</p>

</body>

</html>

```

LoginServlet.java

```

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

public class LoginServlet extends HttpServlet{
  public void doGet(HttpServletRequest request, HttpServletResponse response)

```

```

throws ServletException, IOException {

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String name = request.getParameter("username");
    String pass = request.getParameter("password");
    out.println("<html>");
    out.println("<body>");
    out.println("Thanks Mr." + " " + name + " " + "for visiting roseindia<br>" );
    out.println("Now you can see your password : " + " " + pass + "<br>");
    out.println("</body></html>");
}
}

```

web.xml file for this program:

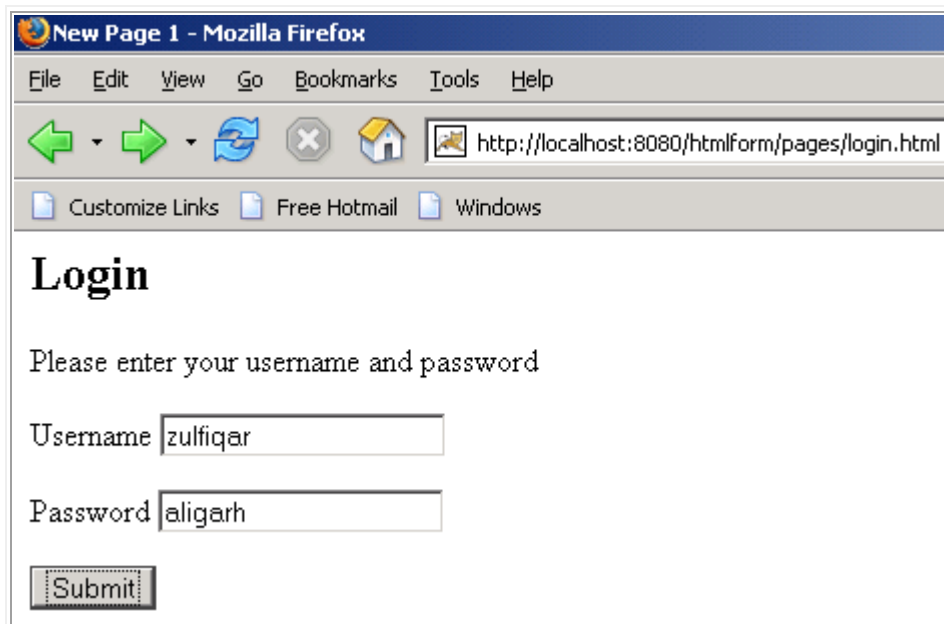
```

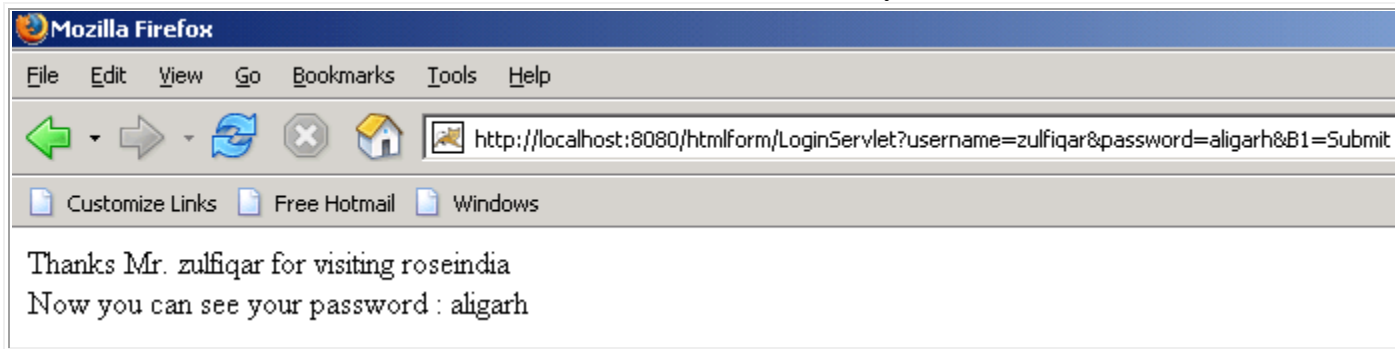
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd"> -->

<web-app>
  <servlet>
    <servlet-name>Hello</servlet-name>
    <servlet-class>LoginServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Hello</servlet-name>
    <url-pattern>/LoginServlet</url-pattern>
  </servlet-mapping>
</web-app>

```

The output of the program is given below:





Time Updater in Servlet

In this program we are going to make one program on servlet which will keep on updating the time in every second and the result will be displayed to you.

To make this servlet firstly we need to make a class named TimeUpdater. The name of the class should be such that it becomes easy to understand what the program is going to do. Call the method **getWriter()** method of the **response** object which will return a **PrintWriter** object. Use the method **getHeader()** of the response object to add a new header. We can also use **setHeader()** in place of **getHeader()**. The **setHeader()** method overrides the previous set header. Now by using the **PrintWriter** object display the result on the browser.

The code of the program is given below:

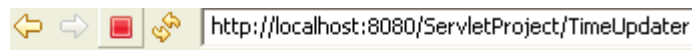
```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class TimeUpdater extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
PrintWriter pw = response.getWriter();

response.addHeader("Refresh", "1");
pw.println(new Date().toString());
}
}
```

The output of the program is given below:



Tue Feb 20 11:17:50 GMT+05:30 2007

Send Redirect in Servlet

When we want that someone else should handle the response of our servlet, then there we should use **sendRedirect()** method.

In send Redirect whenever the client makes any request it goes to the container, there the container decides whether the concerned servlet can handle the request or not. If not then the servlet decides


```
}
```

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

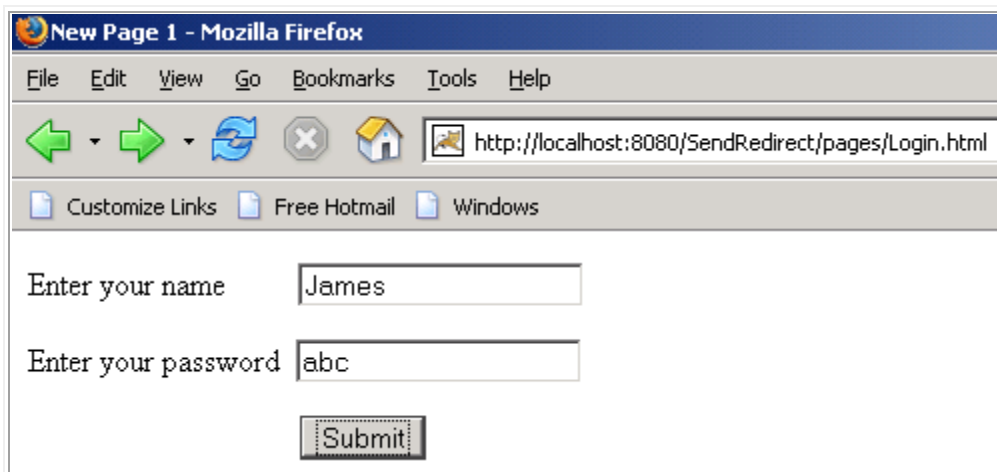
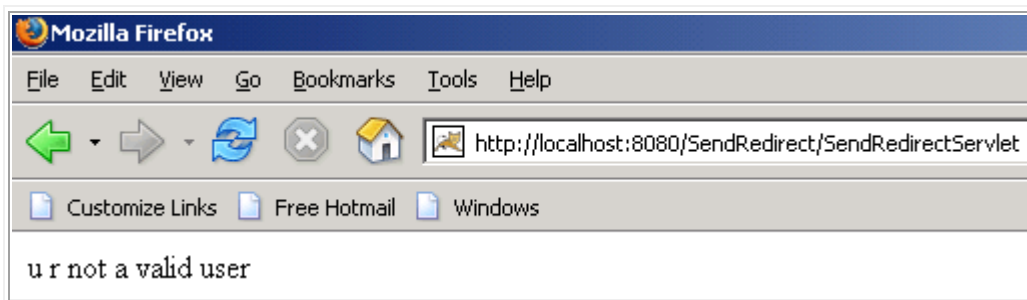
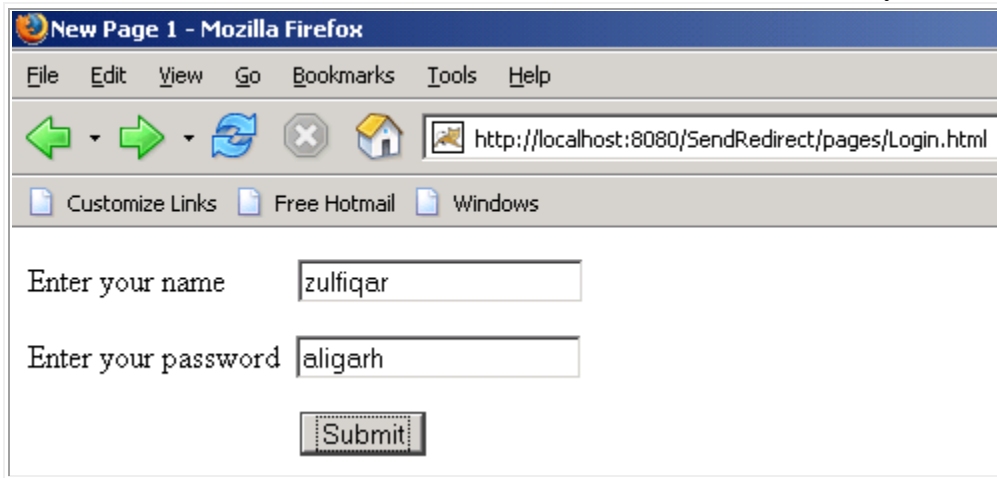
public class ValidUserServlet extends HttpServlet{
protected void doGet(HttpServletRequest request, HttpServletResponse response)
                    throws ServletException, IOException {
    PrintWriter pw = response.getWriter();
    pw.println("Welcome to roseindia.net " + " ");
    pw.println("how are you");
}
}
```

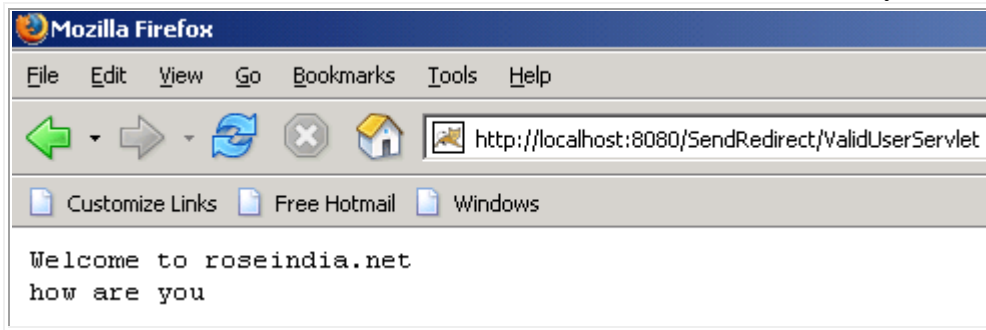
web.xml file for this program:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <servlet>
    <servlet-name>Kailash</servlet-name>
    <servlet-class>SendRedirectServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Kailash</servlet-name>
    <url-pattern>/SendRedirectServlet</url-pattern>
  </servlet-mapping>
  <servlet>
    <servlet-name>Hello</servlet-name>
    <servlet-class>ValidUserServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Hello</servlet-name>
    <url-pattern>/ValidUserServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

The output of the program is given below:





Session Tracking

As we know that the Http is a *stateless* protocol, means that it can't persist the information. It always treats each request as a new request. In Http client makes a connection to the server, sends the request., gets the response, and closes the connection.

In session management client first make a request for any servlet or any page, the container receives the request and generate a unique session ID and gives it back to the client along with the response. This ID gets stores on the client machine. Thereafter when the client request again sends a request to the server then it also sends the session Id with the request. There the container sees the Id and sends back the request.

Session Tracking can be done in three ways:

1. **Hidden Form Fields:** This is one of the way to support the session tracking. As we know by the name, that in this fields are added to an HTML form which are not displayed in the client's request. The hidden form field are sent back to the server when the form is submitted. In hidden form fields the html entry will be like this : `<input type = "hidden" name = "name" value="">`. This means that when you submit the form, the specified name and value will be get included in get or post method. In this session ID information would be embedded within the form as a hidden field and submitted with the Http POST command.
2. **URL Rewriting:** This is another way to support the session tracking. **URLRewriting** can be used in place where we don't want to use cookies. It is used to maintain the session. Whenever the browser sends a request then it is always interpreted as a new request because http protocol is a stateless protocol as it is not persistent. Whenever we want that our request object to stay alive till we decide to end the request object then, there we use the concept of session tracking. In session tracking firstly a session object is created when the first request goes to the server. Then server creates a token which will be used to maintain the session. The token is transmitted to the client by the response object and gets stored on the client machine. By default the server creates a cookie and the cookie get stored on the client machine.
3. **Cookies:** When cookie based session management is used, a token is generated which contains user's information, is sent to the browser by the server. The cookie is sent back to the server when the user sends a new request. By this cookie, the server is able to identify the user. In this way the session is maintained. Cookie is nothing but a name- value pair, which is stored on the client machine. By default the cookie is implemented in most of the browsers. If we want then we can also disable the cookie. For security reasons, cookie based session management uses two types of cookies.

To Determine whether the Session is New or Old

In this program we are going to make one servlet on session in which we will check whether the session is new or old.

To make this program firstly we need to make one class named **CheckingTheSession**. Inside the **doGet()** method, which takes two objects one of request and second of response. Inside this method call

the method `getWriter()` of the response object. Use `getSession()` of the request object, which returns the `HttpSession` object. Now by using the `HttpSession` we can find out whether the session is new or old.

The code of the program is given below:

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

public class CheckingTheSession extends HttpServlet{
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

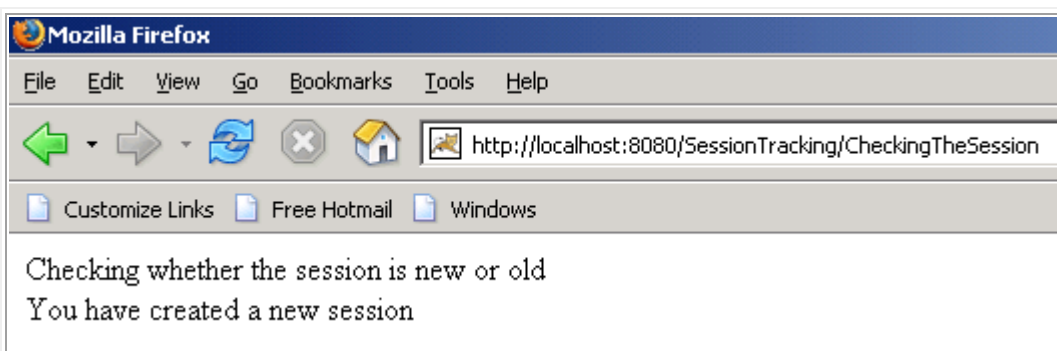
        response.setContentType("text/html");
        PrintWriter pw = response.getWriter();
        pw.println("Checking whether the session is new or old<br>");
        HttpSession session = request.getSession();
        if(session.isNew()){
            pw.println("You have created a new session");
        }
        else{
            pw.println("Session already exists");
        }
    }
}
```

web.xml file for this program:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd" -->

<web-app>
  <servlet>
    <servlet-name>Hello</servlet-name>
    <servlet-class>CheckingTheSession</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Hello</servlet-name>
    <url-pattern>/CheckingTheSession</url-pattern>
  </servlet-mapping>
</web-app>
```

The output of the program is given below:



Session Last Accessed Time Example

This example illustrates to find current access time of session and last access time of session. Sessions are used to maintain state and user identity across multiple page requests. An implementation of HttpSession represents the server's view of the session. The server considers a session to be new until it has been joined by the client. Until the client joins the session, isNew() method returns true.

Here is the source code of LastAccessTime.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.net.*;
import java.util.*;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class LastAccessTime extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        HttpSession session = request.getSession(true);
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String head;
        Integer count = new Integer(0);
        if (session.isNew()) {
            head = "New Session Value ";
        } else {
            head = "Old Session value";
            Integer oldcount = (Integer)session.getValue("count");
            if (oldcount != null) {
                count = new Integer(oldcount.intValue() + 1);
            }
        }
        session.putValue("count", count);
        out.println("<HTML><BODY BGCOLOR=\"#FDF5E6\">\n" +
            "<H2 ALIGN=\"CENTER\">" + head + "</H2>\n" +
            "<H4 ALIGN=\"CENTER\">Session Access Time:</H4>\n" +
            "<TABLE BORDER=1 ALIGN=CENTER>\n" + "<TR BGCOLOR=\"pink\">\n" +
            "<TD>Session Creation Time\n" + "<TD>" +
            new Date(session.getCreationTime()) + "\n" +
            "<TR BGCOLOR=\"pink\">\n" + "<TD>Last Session Access Time\n" +
            "<TD>" + new Date(session.getLastAccessedTime()) +
            "</TABLE>\n" + "</BODY></HTML>");
    }
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        doGet(request, response);
    }
}
```

Description of code: In the above servlet, isNew() method is used to find whether session is new or old. The getCreationTime() method is used to find the time when session was created. The getLastAccessedTime() method is used to find when last time session was accessed by the user.

Here is the mapping of servlet ("LastAccessTime.java") in the web.xml file:

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

Running the servlet by this url: <http://localhost:8080/CodingDiaryExample/LastAccessTime> displays the output like below:

http://localhost:8080/CodingDiaryExample/LastAccessTime

Windows Media Windows

New Session Value

Session Access Time:

Session Creation Time	Wed Jun 25 18:47:17 GMT+05:30 2008
Last Session Access Time	Wed Jun 25 18:47:17 GMT+05:30 2008

When user re-calls the servlet the creation time will be same but last accessed time will be changed as shown in the following figure:

Old Session value

Session Access Time:

Session Creation Time	Wed Jun 25 18:47:17 GMT+05:30 2008
Last Session Access Time	Wed Jun 25 18:48:46 GMT+05:30 2008

Display session value Using Servlet

Sometime while developing web application it is necessary to interact with the different values of the Session object. In this example we will explore the different values of the Session object and then learn how to use it in our programming code.

You will learn how to find all the session related information like:

- **getId.** This method is used to find the identifier of the session which is unique.

- **isNew.** This method is used when find, whether session is newly created or preexisted. If session has never seen by user then this method return "true" but if session is preexisted then it return "false".
- **getCreationTime.** This method is used to find the creation time of session. To use of this method we can find the following details about session i.e. day, month, date, time, GMT(**Greenwich Mean Time**) and year will be displayed.
- **getLastAccessedTime.** This method is used to find the last accessed time of session. It returns the time, in milliseconds.
- **getMaxInactiveInterval.** This method returns the total time, in seconds, during which session remains active if user does not accesses the session for this maximum time interval. After this time the session will be invalidated automatically. A negative value indicates that the session should never timeout.

Here is the sample code for `HttpSessionDisplay.java`:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.net.*;
import java.util.*;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class HttpSessionDisplay extends HttpServlet {
    String head;
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        HttpSession session = request.getSession(true);
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        Integer count = new Integer(0);
        if (session.isNew()) {
            head = "New Session Value";
        } else {
            head = "Old Session value";
            Integer oldcount = (Integer)session.getValue("count");
            if (oldcount != null) {
                count = new Integer(oldcount.intValue() + 1);
            }
        }
        session.putValue("count", count);
        out.println("<HTML><BODY BGCOLOR=\"pink\">\n" +
            "<H2 ALIGN=\"CENTER\">" + head + "</H2>\n" +
            "<H3 ALIGN=\"CENTER\">Description about Session:</H3>\n" +
            "<TABLE BORDER=1 ALIGN=CENTER>\n" + "<TR BGCOLOR=\"voilet\">\n" +
            "  <TH>Information Type<TH>Session Value\n" + "<TR>\n" + "  <TD>ID\n" +
            "<TD>" + session.getId() + "\n" +
            "<TR>\n" + "  <TD>Session Creation Time\n" +
            "  <TD>" + new Date(session.getCreationTime()) + "\n" +
            "<TR>\n" + "  <TD>Last Session Access Time\n" + "  <TD>" +
            new Date(session.getLastAccessedTime()) + "\n" +
            "<TR>\n" + "  <TD>Number of Previous Session Accesses\n" +
            "  <TD>" + count + "\n" +
```

```

        "</TABLE>\n" + "</BODY></HTML>");
    }
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        doGet(request, response);
    }
}

```

Here is the mapping of class file in web.xml.

```

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

```

Run this example by this url: <http://localhost:8080/CodingDiaryExample/HttpSessionDisplay>

Output:

In case of first time accessing of servlet the following session value will be displayed

New Session Value	
Description about Session:	
Information Type	Session Value
ID	39B14C30ABB7E2E376180FFEA2C56BA9
Session Creation Time	Wed Jun 25 17:34:03 GMT+05:30 2008
Last Session Access Time	Wed Jun 25 17:34:03 GMT+05:30 2008
Number of Previous Session Accesses	0

In case of preexist session if user accessed the servlet then the following session value will be displayed.

Old Session value	
Description about Session:	
Information Type	Session Value
ID	39B14C30ABB7E2E376180FFEA2C56BA9
Session Creation Time	Wed Jun 25 17:34:03 GMT+05:30 2008
Last Session Access Time	Wed Jun 25 17:34:03 GMT+05:30 2008
Number of Previous Session Accesses	1

Hit Counter Servlet Example

This example illustrates about counting how many times the servlet is accessed. When first time servlet (**CounterServlet**) runs then session is created and value of the counter will be zero and after again accessing of servlet the counter value will be increased by one. In this program **isNew()** method is used whether session is new or old and **getValue()** method is used to get the value of counter.

Here is the source code of CounterServlet.java:

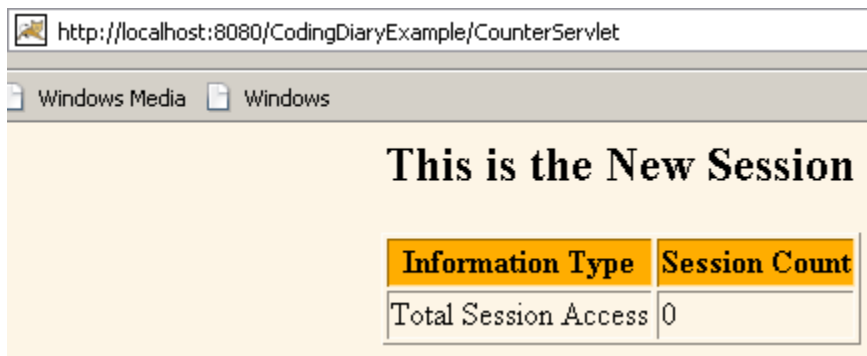
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class CounterServlet extends HttpServlet{
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        HttpSession session = request.getSession(true);
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        Integer count = new Integer(0);
        String head;
        if (session.isNew()) {
            head = "This is the New Session";
        } else {
            head = "This is the old Session";
            Integer oldcount =(Integer)session.getValue("count");
            if (oldcount != null) {
                count = new Integer(oldcount.intValue() + 1);
            }
        }
        session.putValue("count", count);
        out.println("<HTML><BODY BGCOLOR=#FDF5E6>\n" +
            "<H2 ALIGN=#CENTER>" + head + "</H2>\n" + "<TABLE BORDER=1 ALIGN=CENTER>\n"
            + "<TR BGCOLOR=#FFAD00>\n"
            + " <TH>Information Type<TH>Session Count\n"
            + "<TR>\n" + " <TD>Total Session Accesses\n" +
            "<TD>" + count + "\n" +
            "</TABLE>\n"
            + "</BODY></HTML>" );
    }
}
```

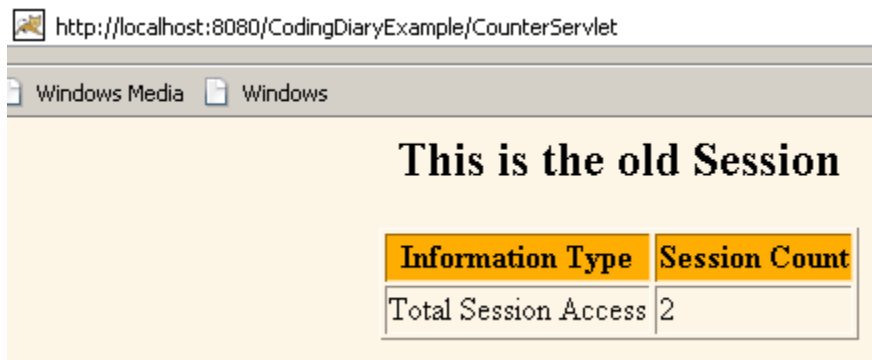
Mapping of Servlet ("CounterServlet.java") in web.xml file

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

Running the servlet by this url: <http://localhost:8080/CodingDiaryExample/CounterServlet> displays the figure below:



When servlet is hit two times by the user the counter value will be increased by two as shown in figure below:



Inserting Data In Database table using Statement

In this program we are going to insert the data in the database from our java program in the table stored in the database.

To accomplish our goal we first have to make a class named as **ServletInsertingData**, which must extends the abstract **HttpServlet** class, the name of the class should be such that other person can understand what this program is going to perform. The logic of the program will be written inside the **doGet()** method that takes two arguments, first is **HttpServletRequest** interface and the second one is the **HttpServletResponse** interface and this method can throw **ServletException**.

Inside this method call the **getWriter()** method of the **PrintWriter** class. We can insert the data in the database only and only if there is a connectivity between our database and the java program. To

establish the connection between our database and the java program we first need to call the method **forName()**, which is static in nature of the class **Class**. It takes one argument which tells about the database driver we are going to use. Now use the static method **getConnection()** of the **DriverManager** class. This method takes three arguments and returns the *Connection* object. SQL statements are executed and results are returned within the context of a connection. Now your connection has been established. Now use the method **createStatement()** of the *Connection* object which will return the *Statement* object. This object is used for executing a static SQL statement and obtaining the results produced by it. We have to insert a values into the table so we need to write a query for inserting the values into the table. This query we will write inside the **executeUpdate()** method of the *Statement* object. This method returns int value.

If the record will get inserted in the table then output will show "record has been inserted" otherwise "sorry! Failure".

The code of the program is given below:

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

public class DataInsertion extends HttpServlet{
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws
        ServletException, IOException{

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String url = "jdbc:mysql://localhost/kailash?user=root&password=admin";
        Connection conn;
        ResultSet rs;
        try{
            Class.forName("org.gjt.mm.mysql.Driver");
            conn = DriverManager.getConnection(url);
            Statement statement = conn.createStatement();
            String query = "insert into emp_sal values('kailash', 15000)";
            int i = statement.executeUpdate(query);
            if(i!=0){
                out.println("The record has been inserted");
            }
            else{
                out.println("Sorry! Failure");
            }
            rs = statement.executeQuery("select * from emp_sal");
            while(rs.next()){
                out.println("<p><table>" + rs.getString(1) + " " + rs.getInt(2) + "</p></table>");
            }
            rs.close();
            statement.close();
        }
        catch (Exception e){
            System.out.println(e);
        }
    }
}
```

XML File for this program

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
```

```
"http://java.sun.com/dtd/web-app_2_3.dtd"> -->

<web-app>
  <servlet>
    <servlet-name>Hello</servlet-name>
    <servlet-class>DataInsertion</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Hello</servlet-name>
    <url-pattern>/DataInsertion</url-pattern>
  </servlet-mapping>
</web-app>
```

Table in the database before Insertion:

```
mysql> select * from
emp_sal;
Empty set (0.02 sec)
```

The output of the program is given below:

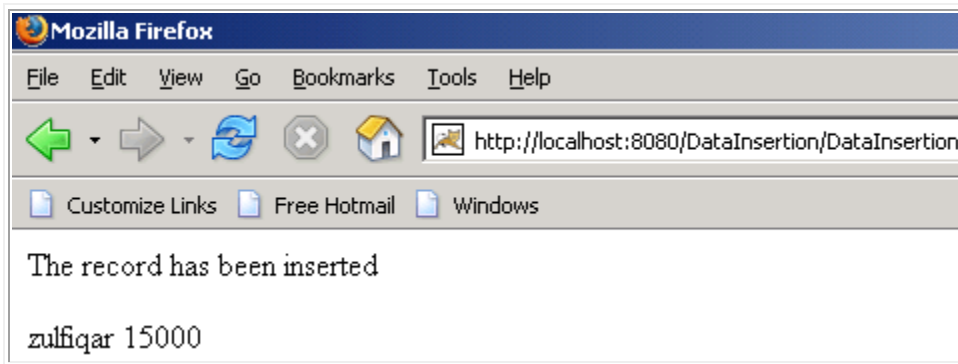


Table in the database after Insertion:

```
mysql> select * from
emp_sal;
+-----+-----+
| EmpName | salary |
+-----+-----+
| kailash | 15000  |
+-----+-----+
1 row in set (0.02 sec)
```

Retrieving Data from the table using Statement

In this program we are going to fetch the data from the database in the table from our java program.

To accomplish our goal we first have to make a class named as **ServletFetchingData** which must extends the abstract **HttpServlet** class, the name of the class should be such that the other person can understand what this program is going to perform. The logic of the program will be written inside the **doGet()** method which takes two arguments, first is **HttpServletRequest** interface and the second one is the **HttpServletResponse** interface and this method can throw **ServletException**.

Inside this method call the **getWriter()** method of the **PrintWriter** class. We can retrieve the data from the database only and only if there is a connectivity between our database and the java program. To

establish the connection between our database and the java program we firstly need to call the method **forName()** which is static in nature of the class **ClassLoader**. It takes one argument which tells about the database driver we are going to use. Now use the static method **getConnection()** of the **DriverManager** class. This method takes three arguments and returns the *Connection* object. SQL statements are executed and results are returned within the context of a connection. Now your connection has been established. Now use the method **createStatement()** of the *Connection* object which will return the *Statement* object. This object is used for executing a static SQL statement and obtaining the results produced by it. As we need to retrieve the data from the table so we need to write a query to select all the records from the table. This query will be passed in the **executeQuery()** method of *Statement* object, which returns the *ResultSet* object. Now the data will be retrieved by using the **getString()** method of the *ResultSet* object.

The code of the program is given below:

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

public class ServletFetchingDataFromDatabase1 extends HttpServlet{
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws
        ServletException, IOException{

        response.setContentType("text/html");
        PrintWriter pw = response.getWriter();
        String connectionURL = "jdbc:mysql://localhost/kailash";
        Connection connection=null;
        try{
            Class.forName("org.gjt.mm.mysql.Driver");
            connection = DriverManager.getConnection(connectionURL, "root", "admin");
            Statement st = connection.createStatement();
            ResultSet rs = st.executeQuery("Select * from emp_sal");
            while(rs.next()){
                pw.println("EmpName" + "    " + "EmpSalary" + "<br>");
                pw.println(rs.getString(1) + "    " + rs.getString(2) + "<br>");
            }
        }
        catch (Exception e){
            pw.println(e);
        }
    }
}
```

XML File for this program:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd"> -->

<web-app>
  <servlet>
    <servlet-name>Hello</servlet-name>
    <servlet-class>ServletFetchingDataFromDatabase</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Hello</servlet-name>
    <url-pattern>/ServletFetchingDataFromDatabase</url-pattern>
  </servlet-mapping>
```

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

The output of the program is given below:

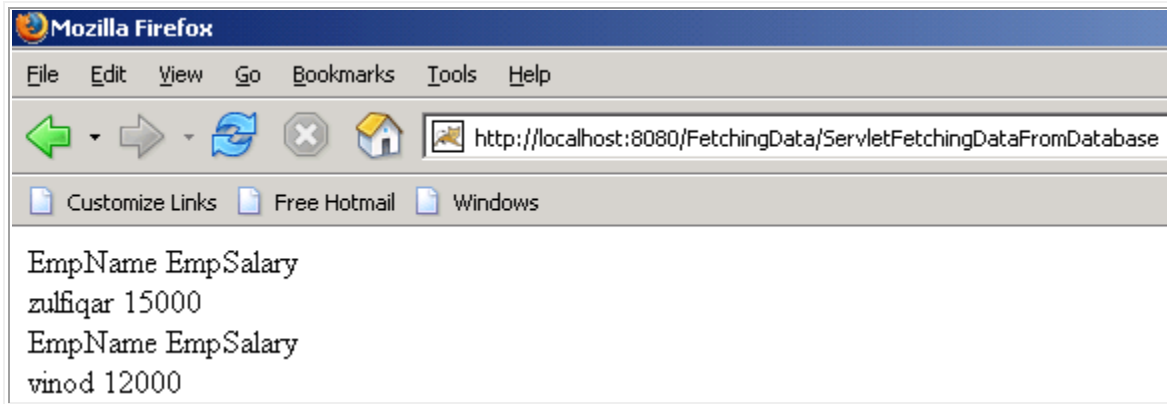


Table in the database:

```
mysql> select * from
emp_sal;
+-----+-----+
| EmpName | salary |
+-----+-----+
| kailash | 15000 |
| vinod   | 12000 |
+-----+-----+
2 rows in set (0.00 sec)
```

Inserting data from the HTML page to the database

In this program we are going to make program in which we are going to insert the values in the database table from the html form.

To make our program working we need to make one html form in which we will have two fields, one is for the name and the other one is for entering the password. At last we will have the submit form, clicking on which the values will be passed to the server.

The values which we have entered in the Html form will be retrieved by the server side program which we are going to write. To accomplish our goal we first have to make a class named as **ServletInsertingDataUsingHtml** which must extends the abstract **HttpServlet** class, the name of the class should be such that the other person can understand what this program is going to perform. The logic of the program will be written inside the **doGet()** method which takes two arguments, first is **HttpServletRequest** interface and the second one is the **HttpServletResponse** interface and this method can throw **ServletException**.

Inside this method call the **getWriter()** method of the **PrintWriter** class. We can insert the data in the database only and only if there is a connectivity between our database and the java program. To establish the connection between our database and the java program we firstly need to call the method **forName()** which is static in nature of the class **Class**. It takes one argument which tells about the database driver we are going to use. Now use the static method **getConnection()** of the **DriverManager** class. This method takes three arguments and returns the **Connection** object. SQL statements are executed and results are returned within the context of a connection. Now your connection has been established. Now use the method **prepareStatement()** of the **Connection** object which will return the **PreparedStatement** object and takes one a query which we want to fire as its input.

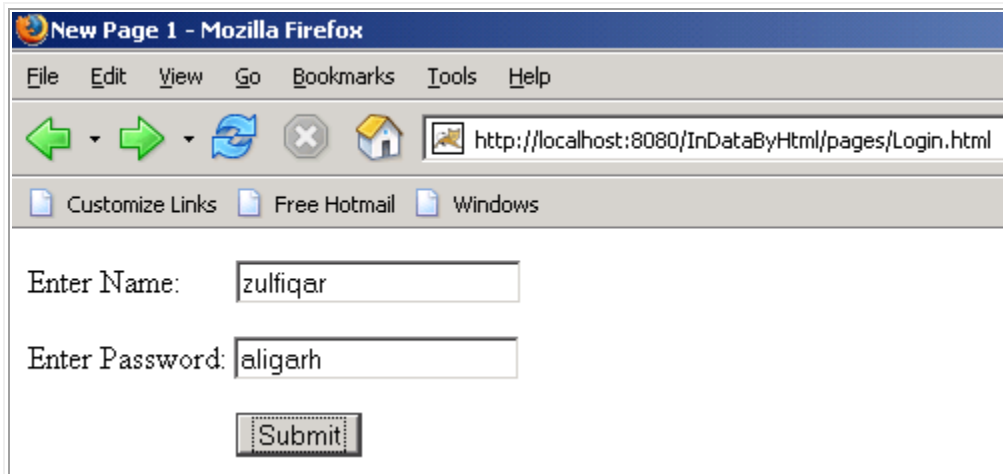

```
    }
    else{
        pw.println("failed to insert the data");
    }
}
catch (Exception e){
    pw.println(e);
}
}
```

web.xml file for this program:

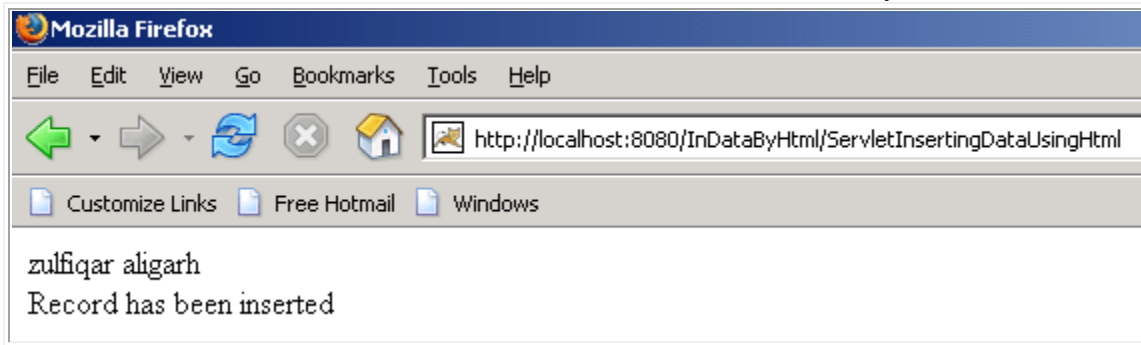
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <servlet>
    <servlet-name>Kailash</servlet-name>
    <servlet-class>ServletInsertingDataUsingHtml</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Kailash</servlet-name>
    <url-pattern>/ServletInsertingDataUsingHtml</url-pattern>
  </servlet-mapping>
</web-app>
```

The output of the program is given below:



This is the output of the above input.



Retrieving Data from the table using PreparedStatement

In this program we are going to fetch the data from the database in the table from our java program using **PreparedStatement**.

To accomplish our goal we first have to make a class named as **ServletFetchingDataFromDatabase** which must extend the abstract **HttpServlet** class, the name of the class should be such that the other person can understand what this program is going to perform. The logic of the program will be written inside the **doGet()** method which takes two arguments, first is **HttpServletRequest** interface and the second one is the **HttpServletResponse** interface and this method can throw **ServletException**.

Inside this method call the **getWriter()** method of the **PrintWriter** class. We can retrieve the data from the database only and only if there is a connectivity between our database and the java program. To establish the connection between our database and the java program we firstly need to call the method **forName()** which is static in nature of the class **ClassLoader**. It takes one argument which tells about the database driver we are going to use. Now use the static method **getConnection()** of the **DriverManager** class. This method takes three arguments and returns the *Connection* object. SQL statements are executed and results are returned within the context of a connection. Now your connection has been established. Now use the method **prepareStatement()** of the *Connection* object which will return the *PreparedStatement* object and takes a query as its parameter. In this query we will write the task we want to perform. The *ResultSet* object will be retrieved by using the **executeQuery()** method of the *PreparedStatement* object. Now the data will be retrieved by using the **getString()** method of the *ResultSet* object.

The code of the program is given below:

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

public class ServletFetchingDataFromDatabase extends HttpServlet{
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws
        ServletException, IOException{

        response.setContentType("text/html");
        PrintWriter pw = response.getWriter();
        String connectionURL = "jdbc:mysql://localhost/kailash";
        Connection connection=null;
        try{
            Class.forName("org.gjt.mm.mysql.Driver");
            connection = DriverManager.getConnection(connectionURL, "root", "admin");
            PreparedStatement pst = connection.prepareStatement("Select * from emp_sal");
            ResultSet rs = pst.executeQuery();
            while(rs.next()){
```

```

        pw.println(rs.getString(1) + " " + rs.getString(2)+"<br>");
    }
}
catch (Exception e){
    pw.println(e);
}
pw.println("hello");
}
}

```

The output of the program is given below:

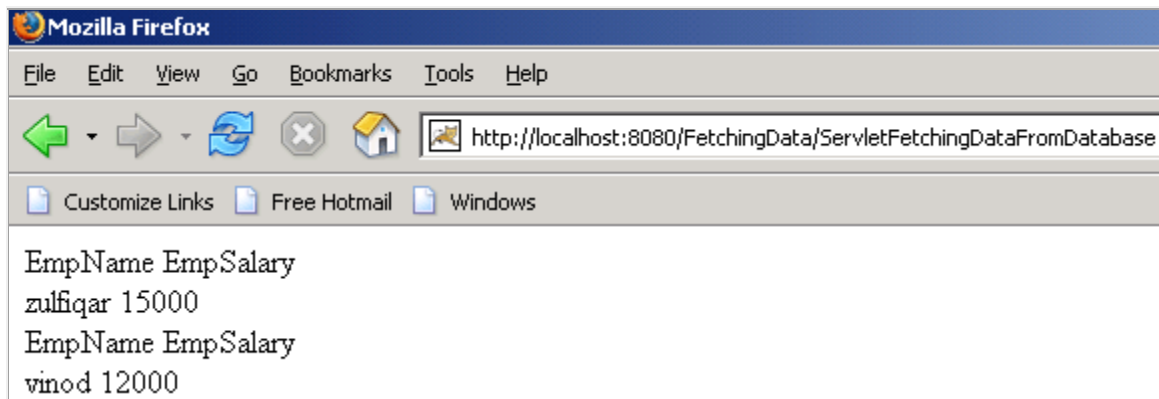


Table in the database:

```

mysql> select * from emp_sal;
+-----+-----+
| EmpName | salary |
+-----+-----+
| kailash | 15000 |
| vinod   | 12000 |
+-----+-----+
2 rows in set (0.00 sec)

```

Getting Columns Names using Servlets

Consider a situation where there is a need to know about the name of the columns without touching our database. As we are the programmers so why we need to worry about the database. We want to do the manipulation by sitting on our computer through our program without going into the database.

In this example we are going to exactly the same as we said above. To make this possible we need to make a class named **ServletGettingColumnsNames**, the name of the program should be such that if in future there is any need to make any change in the program, you can easily understand in which program you have to make a change. Now inside the **doGet()** method use the **getWriter()** method of the response object and its returns the **PrintWriter** object, which helps us to write on the browser. To get a column names from the database there is a need for the connection between the database and the java program. After the establishment of the connection with the database pass a query for retrieving all the records from the database and this will return the *PreparedStatement* object. To get the column names from the database we firstly need a reference of *ResultSetMetaData* object and we will get it only when if we have the *ResultSet* object. To get the object of the *ResultSet* we will call the method **executeQuery()** of the *PreparedStatement* interface. Now we have the object of the *ResultSet*. By the help of the *ResultSet* we can get the object of *ResultSetMetaData*. We will get it by calling the method **getMetaData()** of the

ResultSet interface. The names of the columns will be retrieved by the method **getColumnNames()** of the *ResultSetMetaData* interface. The output will be displayed to you by the **PrintWriter** object.

The code of the program is given below:

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

public class ServletGettingColumnsNames extends HttpServlet{
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException{

        response.setContentType("text/html");
        PrintWriter pw = response.getWriter();
        String connectionURL = "jdbc:mysql://localhost/kailash";
        Connection connection=null;
        try{
            Class.forName("org.gjt.mm.mysql.Driver");
            connection = DriverManager.getConnection(connectionURL, "root", "admin");
            PreparedStatement pst = connection.prepareStatement("select * from emp_details");
            ResultSet rs = pst.executeQuery();
            ResultSetMetaData rsmd = rs.getMetaData();
            int noOfColumns = rsmd.getColumnCount();
            //It shows the number of columns
            pw.println("The number of columns are " + noOfColumns + "<br>");
            //It shows the name of the columns
            pw.println("The name of the columns are: <br>");
            for(int i =1; i<=noOfColumns;i++){
                String names = rsmd.getColumnName(i);
                pw.println(names);
            }
        }
        catch(Exception e){
            pw.println("The exception is " + e);
        }
    }
}
```

XML File for this program:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
<servlet>
<servlet-name>Kailash</servlet-name>
<servlet-class>ServletGettingColumnsNames</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>Kailash</servlet-name>
<url-pattern>/ServletGettingColumnsNames</url-pattern>
</servlet-mapping>
</web-app>
```

The output of the program is given below:

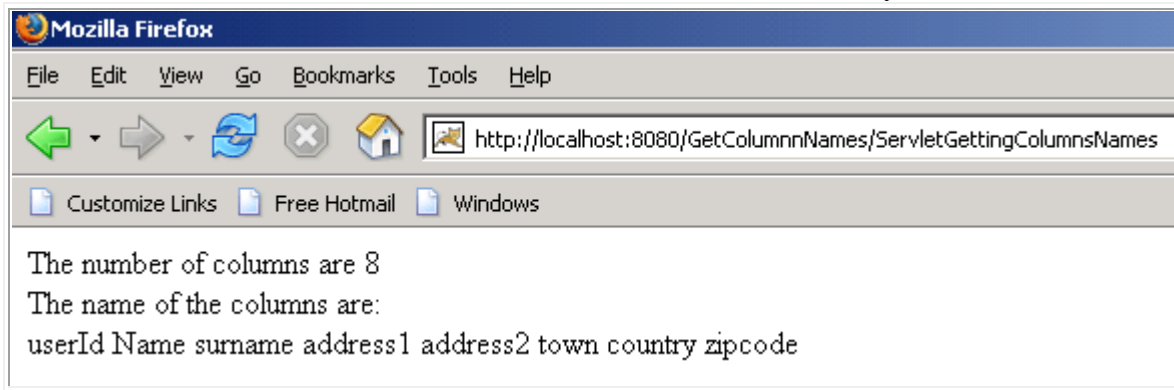


Table in the database:

```
mysql> select * from emp_details;
+-----+-----+-----+-----+-----+-----+-----+
+-----+-----
-+
| userId | Name | surname | address1 | address2 | town |
country |          |          |          |          |      |
|
+-----+-----+-----+-----+-----+-----+
+-----+-----
-+
| 73979 | kailash | Ahmed | Moradabad | Delhi | Okhla |
India |          |          |          |          |      |
|
+-----+-----+-----+-----+-----+-----+
+-----+-----
-+
1 row in set (0.00 sec)
```

Getting Number of Columns

Consider a situation where there is a need to know about the number of columns in the table without touching our database. As we are the programmers so why we should worry about the database. We want to do the manipulation by sitting on our computer through our program without going into the database.

In this example we are going to exactly the same as we said above. To make this possible we need to make a class named **ServletGettingNoOfColumns**, the name of the program should be such that if in future there is any need to make any change in the program, you can easily understand in which program you have to make a change. As we know that in Servlet the main logic of the program is written inside the *service* method and in turn the *service* method calls the **doGet()** method. Now inside the **doGet()** method use the **getWriter()** method of the response object and its returns the **PrintWriter** object, which helps us to write on the browser. To get the number of columns from the database table there is a need for the connection between the database and the java program. After the establishment of the connection with the database, pass a query for retrieving all the records from the database and this will return the *PreparedStatement* object. To get the number of columns from the database we firstly need a reference of *ResultSetMetaData* object and we will get it only when if we have the *ResultSet* object with us. To get the object of the *ResultSet* we will call the method **executeQuery()** of the *PreparedStatement* interface. Now we have the object of the *ResultSet*. By the help of the *ResultSet* we can get the object of *ResultSetMetaData*. We will get it by calling the method **getMetaData()** of the *ResultSet* interface. The

number of columns in the databasd table will be retrieved by the method **getColumnCount()** of the *ResultSetMetaData* interface. This method will return the integer type of value. The number of columns will be displayed on the browser by the **PrintWriter** object.

The code of the program is given below:

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

public class ServletGettingNoOfColumns extends HttpServlet{
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws
        ServletException, IOException{

        response.setContentType("text/html");
        PrintWriter pw = response.getWriter();
        String connectionURL = "jdbc:mysql://localhost/kailash";
        Connection connection=null;
        try{
            Class.forName("com.mysql.jdbc.Driver").newInstance();
            connection = DriverManager.getConnection(connectionURL, "root", "admin");
            PreparedStatement pst = connection.prepareStatement("select * from emp_details");
            ResultSet rs = pst.executeQuery();
            ResultSetMetaData rsmd = rs.getMetaData();
            int noOfColumns = rsmd.getColumnCount();
            //It shows the number of columns
            pw.println("The number of columns are " + noOfColumns);
        }
        catch(Exception e){
            pw.println("The exception is " + e);
        }
    }
}
```

web.xml file for this program:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <servlet>
    <servlet-name>Kailash</servlet-name>
    <servlet-class>ServletGettingNoOfColumns</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Kailash</servlet-name>
    <url-pattern>/ServletGettingNoOfColumns</url-pattern>
  </servlet-mapping>
</web-app>
```

Table emp_details in the database:

```
mysql> select * from emp_details;
+-----+-----+-----+-----+-----+-----+-----+
+-----+
--+
| userId | Name | surname | address1 | address2 | town | country |
```

```

zipcod
e
+-----+-----+-----+-----+-----+-----+-----+
+-----
--+
| 86372 | Kailash | Ahmed | Moradabad | Rohini | Rohini | Delhi
|                                             110025
|
+-----+-----+-----+-----+-----+-----+-----+
+-----

```

How to Delete a Table in MySQL

Consider a situation where we need to delete a table from a database.

To delete a table from the database firstly we need to make a connection with the database. When the connection has been established pass a query for deleting a table inside the **prepareStatement()** method and it will return the **PreparedStatement** object. Now call the method **executeUpdate()** of the **PreparedStatement** interface which will helps us to know the status of the program.

The code of the program is given below:

```

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

public class ServletDeletingTable extends HttpServlet{
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws
        ServletException, IOException{

        response.setContentType("text/html");
        PrintWriter pw = response.getWriter();
        String connectionURL = "jdbc:mysql://localhost/kailash";
        Connection connection;
        try{
            Class.forName("org.gjt.mm.mysql.Driver");
            connection = DriverManager.getConnection(connectionURL, "root", "admin");
            PreparedStatement pst = connection.prepareStatement("drop table emp_sal");
            int i = pst.executeUpdate();
            if (i==0){
                pw.println("Table has been deleted");
            }
            else{
                pw.println("Table has not been deleted");
            }
        }
        catch(Exception e){
            pw.println("The exception is " + e);
        }
    }
}

```

XML File for this program:

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app

```



```

PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <servlet>
    <servlet-name>Kailash</servlet-name>
    <servlet-class>ServletDeletingTable</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Kailash</servlet-name>
    <url-pattern>/ServletDeletingTable</url-pattern>
  </servlet-mapping>
</web-app>

```

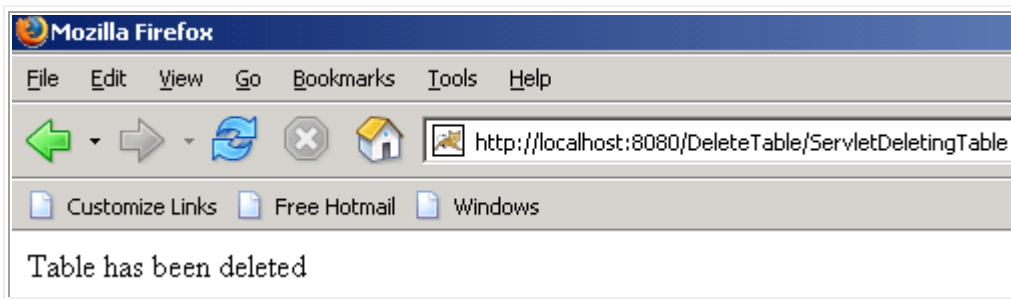
Table in the database before deletion:

```

mysql> select * from emp_sa
+-----+-----+
| EmpName | salary |
+-----+-----+
| kailash | 15000  |
| vinod   | 12000  |
+-----+-----+
2 rows in set (0.00 sec)

```

The output of the program is given below:



Natural Left Join

In this program we are going to join the two table by using the servlets and the result will be displayed in the browser. This join will be natural left join.

To join the tables firstly it is important to make a connection between the java class and the database. In our program we are using the **MySql** database. To join the table in a natural left join manner it is important to have those tables in our database. First of all make a class named **ServletNaturalJoiningTables**. The name of the class should be such that it becomes clear that what the program is going to do. The logic of the program will be written inside the **doGet()** method which takes two arguments **HttpServletRequest** and **HttpServletResponse**. call the method **getWriter()** of the **PrintWriter** class, which is responsible for writing the contents on the browser. Our priority is to join the two tables so pass a query in **prepareStatement()** method which will return the **PreparedStatement** object.

The result will be displayed to you by the object of the **PrintWriter** class.

The code of the program is given below:

```

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

public class ServletNaturalJoiningTables extends HttpServlet{
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException{

        response.setContentType("text/html");
        PrintWriter pw = response.getWriter();
        String connectionURL = "jdbc:mysql://localhost/kailash";
        Connection connection;
        try{
            Class.forName("org.gjt.mm.mysql.Driver");
            connection = DriverManager.getConnection(connectionURL, "root", "admin");
            PreparedStatement pst = connection.prepareStatement("SELECT * FROM "+"emp_details
                NATURAL LEFT JOIN "+"emp_sal

            ResultSet rs = pst.executeQuery();
            pw.println("UserId" + "\t" + "Firstname" + "\t" + "Salary"+"<br>");
            while(rs.next()){
                String id = rs.getString("userId");
                String name = rs.getString("Name");
                String sal = rs.getString("salary");
                pw.println(id + "\t\t" + name + "\t\t" + sal + "<br>");
            }
        }
        catch (Exception e) {
            pw.println("The statement is not executed");
        }
    }
}

```

web.xml file for this program:

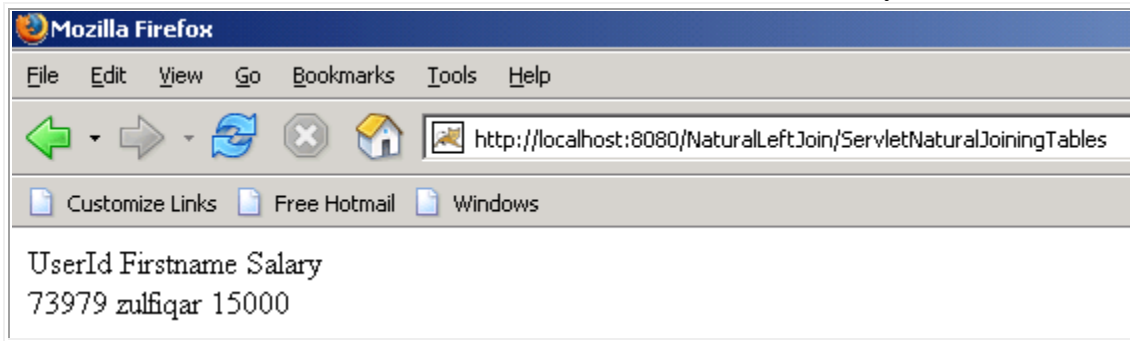
```

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
    <servlet>
        <servlet-name>Kailash</servlet-name>
        <servlet-class>ServletNaturalJoiningTables</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>Kailash</servlet-name>
        <url-pattern>/ServletNaturalJoiningTables</url-pattern>
    </servlet-mapping>
</web-app>

```

The output of the program is given below:



Select Color

In this program we are going to selected the various color and on the basis of the selection the output will be displayed to the user.

To make this program firstly we need to make one html page. Inside the page we will have one select option in which we will have our colors. We will also have a submit, clicking on which the values we have entered will be transferred to the server.

On the server we will create a session. The values which we have entered in the html form will be retrieved by the `getParameterValues()` of the `request` object. It returns the array of String. We will check the condition if there is any session available or not. If yes then we will set the attribute by using the `setAttribute()` method of the `HttpSession` object. The attribute we have set will be retrieved by the `getAttribute` method of the `HttpSession` object in the next page and the value will be displayed on the browser by the `PrintWriter` object.

The code of the program is given below:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Select the list of colors</title>
</head>
<body>
    <form action = "/ServletProject/ColorPage">
        <select name = "colors" size = 5 multiple>
            <option selected>Green</option>
            <option>Red</option>
            <option>Yellow</option>
            <option>Blue</option>
            <option>Black</option>
        </select>
        <input type = "submit" name = "submit">
    </form>
</body>
</html>
```

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

```

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

/**
 * Servlet implementation class for Servlet: ColorPage
 */
public class ColorPage extends javax.servlet.http.HttpServlet implements
javax.servlet.Servlet {
    /* (non-Java-doc)
    * @see javax.servlet.http.HttpServlet#HttpServlet()
    */
    public ColorPage() {
        super();
    }

    /* (non-Java-doc)
    * @see javax.servlet.http.HttpServlet#doGet
      (HttpServletRequest request, HttpServletResponse response)
    */
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        // TODO Auto-generated method stub
        response.setContentType("text/html");
        PrintWriter pw = response.getWriter();
        HttpSession session = request.getSession();
        String colors[] = request.getParameterValues("colors");
        if(session!=null)
        {
            session.setAttribute("color",colors);
            session.setMaxInactiveInterval(60);
        }
        pw.println("<html><body bgcolor =cyan>");
        for(int i = 0; i<colors.length; i++)
        {
            pw.println("The selected colors are" + colors[i]+ "<br>");
        }
        pw.println("<form action = /ServletProject/GetColors>");
        pw.println("<input type = submit name= submit>");
        pw.println("</form></body></html>");
    }

    /* (non-Java-doc)
    * @see javax.servlet.http.HttpServlet#doPost(HttpServletRequest request,
      HttpServletResponse response)
    */
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        // TODO Auto-generated method stub
    }
}

```

```

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

```

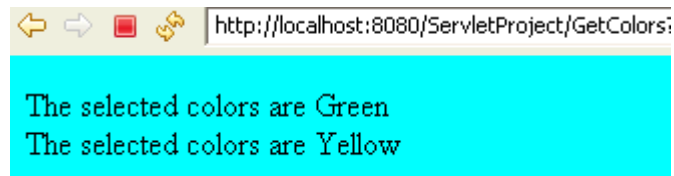
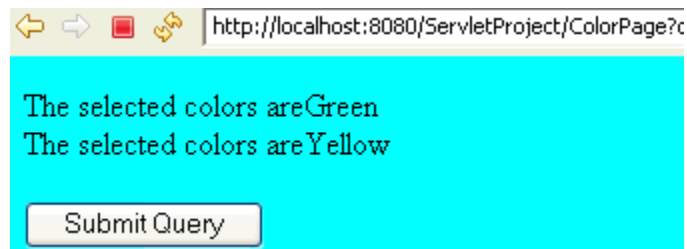
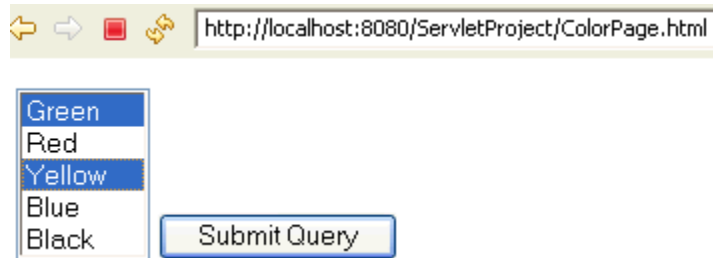
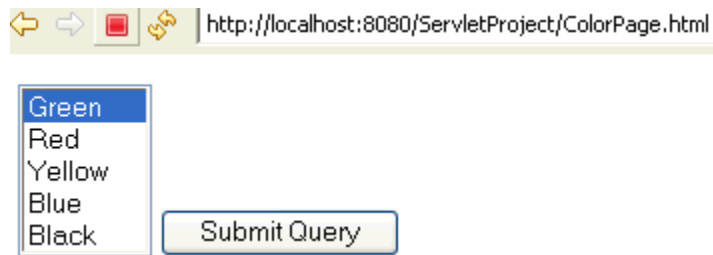
```
import javax.servlet.http.*;

/**
 * Servlet implementation class for Servlet: GetColors
 *
 */
public class GetColors extends HttpServlet {
    /* (non-Java-doc)
    * @see javax.servlet.http.HttpServlet#HttpServlet()
    */
    public GetColors() {
        super();
    }

    /* (non-Java-doc)
    * @see javax.servlet.http.HttpServlet#doGet(
    HttpServletRequest request, HttpServletResponse response)
    */
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        // TODO Auto-generated method stub
        response.setContentType("text/html");
        PrintWriter pw = response.getWriter();
        HttpSession session = request.getSession(false);
        if(session == null)
        {
            pw.println("No session is available");
            pw.println("We are creating a session for you. Creating.....");
            session = request.getSession();
        }
        else
        {
            String getColors[] = (String[])session.getAttribute("color");
            pw.println("<html><body bgcolor = cyan>");
            for(int i= 0; i<getColors.length;i++)
            {
                pw.println("The selected colors are " + getColors[i] + "<br>");
            }
            pw.println("<html><body>");
        }
    }

    /* (non-Java-doc)
    * @see javax.servlet.http.HttpServlet#doPost(
    HttpServletRequest request, HttpServletResponse response)
    */
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        // TODO Auto-generated method stub
    }
}
```

The output of the program is given below:



How to connect to MySql Database from Servlet?



In this example we will show you how to connect to MySQL database and perform select operation. You will learn the JDBC steps necessary to connect to the MySQL Database and execute the query.

Here we are using the MySQL jdbc driver for making the connection. You can download the jdbc driver for MySQL from <http://dev.mysql.com/downloads/connector/j/5.1.html> and then put the driver jar file into the classpath.

You have to first create the a table in MySQL database and then connect it through JDBC to show all the records present there.

MySql Table Structure:

```
CREATE TABLE `servlet` (
```

```

`id` int(11) NOT NULL auto_increment,
`name` varchar(256) default NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT
CHARSET=latin1;

/*Data for the table `servlet` */

insert into `servlet`(`id`,`name`) values
(1,'sandeep'),(2,'amit'),(3,'anusmita'),(4,'vineet');

```

Here is the code of Example:

```

// *DataBase Connectivity from the Servlet.
import java.io.*;
import java.util.*;
import javax.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class DBConnection extends HttpServlet {
    public void service(HttpServletRequest request,HttpServletResponse response)
        throws IOException, ServletException{

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>Servlet JDBC</title></head>");
        out.println("<body>");
        out.println("<h1>Servlet JDBC</h1>");
        out.println("</body></html>");
        // connecting to database
        Connection con = null;
        Statement stmt = null;
        ResultSet rs = null;
        try {
            Class.forName("com.mysql.jdbc.Driver");
            con =DriverManager.getConnection ("jdbc:mysql://192.168.10.59:3306/example",
                                                "root", "root");

            stmt = con.createStatement();
            rs = stmt.executeQuery("SELECT * FROM servlet");
            // displaying records
            while(rs.next()){
                out.print(rs.getObject(1).toString());
                out.print("\t\t\t");
                out.print(rs.getObject(2).toString());
                out.print("<br>");
            }
        } catch (SQLException e) {
            throw new ServletException("Servlet Could not display records.", e);
        } catch (ClassNotFoundException e) {
            throw new ServletException("JDBC Driver not found.", e);
        }
    }
}

```

```

    } finally {
      try {
        if(rs != null) {
          rs.close();
          rs = null;
        }
        if(stmt != null) {
          stmt.close();
          stmt = null;
        }
        if(con != null) {
          con.close();
          con = null;
        }
      } catch (SQLException e) {}
    }
    out.close();
  }
}

```

Program Description:

The following query is used to fetch the records from database and display on the screen.

```

    stmt = con.createStatement();
    rs = stmt.executeQuery("SELECT * FROM servlet");
    // displaying records
    while(rs.next()){
        out.print(rs.getObject(1).toString()); //You can also use
rs.getString(1);
        out.print("\t\t\t");
        out.print(rs.getObject(2).toString()); //You can also use
rs.getString(2);
        out.print("<br>");
    }

```

Other JDBC statement you can understand easily.

Output:

Servlet JDBC

```

1 sandeep
2 arnit
3 anusmita
4 vineet

```

Download Source Code**Refresh a Web Page Using In Servlet**

In this simplified example we develop an application to Refresh a web Page using Servlet. We create two file **timer.html** and **timer.java**. When a web page ("timer.html") run on browser then it will call to Servlet ("timer.java") and refresh this web page and print the current Date and Time after 10 sec on the browser as a output.

Step 1: Create a web page(timer.html) to call a Servlets.

timer.html

```
<HTML>
<HEAD>
  <TITLE>Refresh Servlet Timer</TITLE>
  <META NAME="Generator" CONTENT="EditPlus">
  <META NAME="Author" CONTENT="">
  <META NAME="Keywords" CONTENT="">
  <META NAME="Description" CONTENT="">
  <style type="text/css">
A:link {text-decoration: none;
  padding: 3px 7px;
  margin-right: 3px;

  border-bottom: none;

  color: #2d2b2b; }
A:visited {text-decoration: underline;
  padding: 3px 7px;
  margin-right: 3px;

  color: #2d2b2b; }
A:active {text-decoration: none}
A:hover {text-decoration: none;
  padding: 3px 7px;
  margin-right: 3px;
  border: 0px;

  color: #2d2b2b; }
</style>

</HEAD>

<BODY>
<br><br><br> <br><br><br>
<table width="200px" height="100px" align="center" bgcolor="#BBFFFF" border=0>

  <tr>
    <td style="text-align:top;" valign="middle" align="center" border=0>
      <a href="timer" ><b>Refresh Servlet Timer</b></a>
    </td>
  </tr>
</tr>
```


Date and Time Refresh After 10 Seconds.

The current time is: Wed Jul 16 17:08:29 GMT+05:30 2008

How to get client's address in a servlet

This is detailed java code to get client's address in a servlet. In this example we have used method `getRemoteAddr()` of the `ServletRequest` interface which returns IP address of the client in the string format.

Syntax of the method : `java.lang.String getRemoteAddr()`

We have used a jsp page that is used to send a request to a servlet that execute the request and find the ID address of the client's request. Before run this code create a new directory named "user" in the tomcat-6.0.16/webapps and paste WEB-INF directory in same directory.

get_address.jsp

```
<%@page language="java" session="true"
contentType="text/html;charset=ISO-8859-1" %>
<b><font color="blue">Please Enter your Full Name
here:</font></b><br>
<form name="frm" method="get"
action=" ../user/GetAddress">
  <table border = "0">
    <tr align="left" valign="top">
      <td>First Name:</td>
      <td><input type="text" name = "name"
/></td>
    </tr>
    <tr align="left" valign="top">
      <td></td>
      <td><input type="submit" name="submit"
value="submit"/></td>
    </tr>
  </table>
</form>
```

Save this code as a .jsp file named "**get_address.jsp**" in the directory Tomcat-6.0.16/webapps/user/ and you can run this jsp page with following url in address bar of the browser "http://localhost:8080/user/get_address.jsp"

GetAddress.java

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class GetAddress extends HttpServlet {
  public void doGet(HttpServletRequest request,HttpServletResponse response)
```

```

throws IOException, ServletException{
response.setContentType("text/html");
PrintWriter out = response.getWriter();
String name = request.getParameter("name");
    out.println("<h3>You have entered name : "
+ name + "<br>");
    out.println("<b><font color='blue'>IP Address
of request : </font></b>"
    +request.getRemoteAddr()+"<h3>");
}
}

```

Compile this java code and save .class file in directory C:\apache-tomcat-6.0.16\webapps\user\WEB-INF\classes.

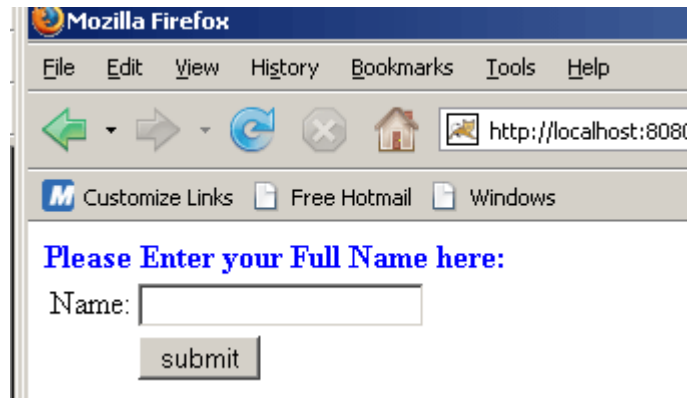
web.xml

```

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

```

This is web .xml file use to map servlet. When run jsp page in the browser.....



User enters first name and click on submit button.....



You have entered name : Mahendra Singh
IP Address of request : 127.0.0.1

Client Auto Refresh in Servlets

This section illustrates you how client gets auto refresh.

We are providing you an example which explains you clearly. In the example, We have created the session by `request.getSession()` method. The method `response.addHeader("Refresh", "15")` refreshes the servlet after every 15 seconds till the servlet gets destroy.

Here is the code of ClientAutoServlet.java

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import java.io.*;

    public class ClientAutoServlet extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse
response)throws ServletException, java.io.IOException {
        HttpSession session = request.getSession();
        Long times = (Long) session.getAttribute("times");
        if (times == null)
            session.setAttribute("times", new Long(0));
        long value = 1;
        if (times != null)
            value = (times.longValue()) + 1;
        if (value < 6)
            response.addHeader("Refresh", "15");
            response.setContentType("text/html");
            java.io.PrintWriter out = response.getWriter();
            out.println("<html><head><title>Client Auto Refresh Example
                </title></head><body>");
            out.println("You've visited this page " + value + " times.");
            session.setAttribute("times", new Long(value));
            out.println("</body></html>");
        }
    }
}
```

In web.xml, do the servlet-mapping

```
<servlet>
<servlet-
name>ClientAutoServlet</servlet-
```

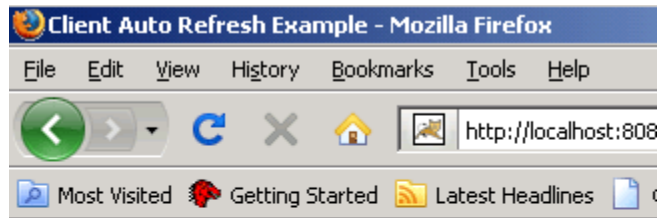
```

name>
<servlet-
class>ClientAutoServlet</servlet-class>
</servlet>

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

```

Output will be displayed as:



You've visited this page 5 times.

After 15 seconds, page will get refresh. Output will be:



You've visited this page 6 times.

Simple Web Application using Servlet

In this section, we have developed a simple web application in Servlet . In this application user can add, update and delete all the user information. We run this application on Apache Tomcat Server. To Download the latest version of Server click on the link <http://www.roseindia.net/struts/struts2/struts-2-download.shtml>. Download the code and Run the application on browser with the url. <http://localhost:8080/servlet/index.jsp>

Brief description of the flow of the application:

- 1) This application having a link "Add New User" to add new User in the database.
- 2) In this application having two button's "Edit" and "Delete".
 "Edit" : This button updates the user Information and

"Delete": This button Deletes the user information.

Step 1: Create a Home Page ("home.jsp") to view all the user.

<u>Add New User</u>								
User Id	UserName	First Name	Last Name	City	Sate	Country	Edit	Delete
1	vineet	Vineet	Bansal	New Delhi	Delhi	India	Edit	Delete
2	amar	Amar	Kumar	New Delhi	Delhi	India	Edit	Delete
3	amit	Amit	Kumar	New Delhi	Delhi	India	Edit	Delete
4	santosh	Santosh	Kumar	New Delhi	Delhi	India	Edit	Delete

Step:2 Create a web page ("addUser.jsp") to add the new user.

Add User

First Name

Last Name

UserName

Password

City

State

Country

Step:3 Create a web page ("edit.jsp") to edit the user.

Upload Image using Servlet

This application illustrates how to upload an image using servlet.

In this example program a form is displayed to user, where user can browse the image file which is to be uploaded on the server. Once the submit button is clicked the form data is posted to a servlet. Servlet then processes the uploaded data and saves the image on the server.

Following programming code is developed in the application:

1. **Form** **JSP:**
Form JSP file is used to display the form to the user.
2. **Servlet** **(UploadImage.java):**
The UploadImage servlet is used to process the form data. After processing the form data, image is saved in the images directory of the server. Servlet also saves the path of the image into database. Finally the uploaded image is displayed on the browser with success message.

The Complete Application is as follows:

Uploaded image shows on the browser:



image inserted successfully

Source Code of uploadImage.jsp

```
<html>
<head><title>Image Upload</title></head>
<body>
  <form action="/example/UploadImage" method="post" enctype="multipart/form-data"
        name="productForm" id="productForm"><br><br>
    <table width="400px" align="center" border=0 style="background-color:ffeef;">
      <tr>
        <td align="center" colspan=2 style="font-weight:bold;font-size:20pt;">
          Image Details</td>
      </tr>
      <tr>
        <td align="center" colspan=2>&nbsp;&nbsp;&nbsp;</td>
      </tr>
      <tr>
        <td>Image Link: </td>
        <td>
          <input type="file" name="file" id="file">
        </td>
      </tr>
      <tr>
        <td></td>
        <td><input type="submit" name="Submit" value="Submit"></td>
      </tr>
    </table>
  </form>
</body>
</html>
```



```

String reg = "[.*]";
String replacingtext = "";
System.out.println("Text before replacing is:-" + itemName);
Pattern pattern = Pattern.compile(reg);
Matcher matcher = pattern.matcher(itemName);
StringBuffer buffer = new StringBuffer();

while (matcher.find()) {
    matcher.appendReplacement(buffer, replacingtext);
}
int IndexOf = itemName.indexOf(".");
String domainName = itemName.substring(IndexOf);
System.out.println("domainName: "+domainName);

String finalimage = buffer.toString()+"_"+r+domainName;
System.out.println("Final Image=== "+finalimage);

File savedFile = new File("C:/apache-tomcat-6.0.16/
                            webapps/example/"+images\\"+finalimage);

item.write(savedFile);
out.println("<html>");
out.println("<body>");
out.println("<table><tr><td>");
out.println("<img src=images/"+finalimage+">");
out.println("</td></tr></table>");

Connection conn = null;
String url = "jdbc:mysql://localhost:3306/";
String dbName = "test";
String driver = "com.mysql.jdbc.Driver";
String username = "root";
String userPassword = "root";
String strQuery = null;
String strQuery1 = null;
String imgLen="";

try {
    System.out.println("itemName::::: "+itemName);
    Class.forName(driver).newInstance();
    conn = DriverManager.getConnection(url+dbName,username,userPassword);
    Statement st = conn.createStatement();
    strQuery = "insert into testimage set image='"+finalimage+"'";
    int rs = st.executeUpdate(strQuery);
    System.out.println("Query Executed Successfully++++++++++++++++");
    out.println("image inserted successfully");
    out.println("</body>");
    out.println("</html>");
} catch (Exception e) {
    System.out.println(e.getMessage());
} finally {
    conn.close();
}
} catch (Exception e) {
    e.printStackTrace();
}
}

```

```
}  
  }  
}  }
```