Servlets provide a component-based, platform-independent method for building Webbased applications, without the performance limitations of CGI programs. Servlets have access to the entire family of Java APIs, including the JDBC API to access enterprise databases. This tutorial will teach you how to use Java Servlets to develop your web based applications in simple and easy steps.

Why to Learn Servlet?

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

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

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

Applications of Servlet

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

Prerequisites

We assume you have good understanding of the Java programming language. It will be great if you have a basic understanding of web application and how internet works.

What are Servlets?

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

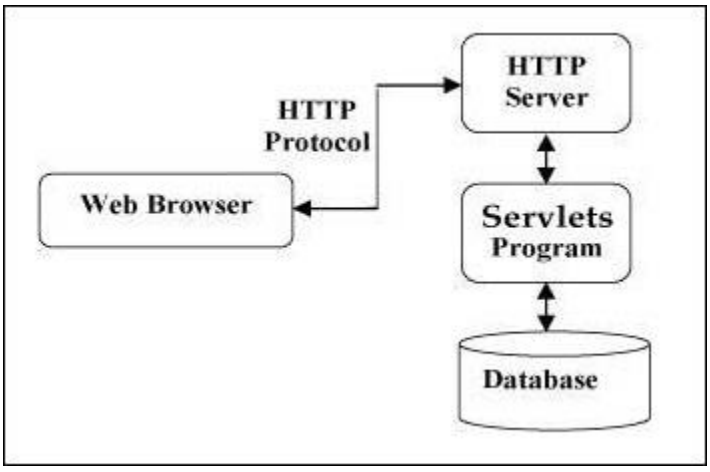
Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

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

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

Servlets Architecture

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



Servlets Tasks

Servlets perform the following major tasks −

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

Servlets Packages

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

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

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

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

What is Next?

I would take you step by step to set up your environment to start with Servlets. So fasten your belt for a nice drive with Servlets. I'm sure you are going to enjoy this tutorial very much.

Advantage of Servlet Over CGI

CGI (Common Gateway Interface)

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

Disadvantages of CGI

There are many problems in CGI technology:

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

Advantages of Servlet

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

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

Servlets - Life Cycle

A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet.

- The servlet is initialized by calling the **init()** method.
- The servlet calls **service()** method to process a client's request.
- The servlet is terminated by calling the **destroy()** method.
- Finally, servlet is garbage collected by the garbage collector of the JVM.

Now let us discuss the life cycle methods in detail.

The init() Method

The init method is called only once. It is called only when the servlet is created, and not called for any user requests afterwards. So, it is used for one-time initializations, just as with the init method of applets.

The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.

When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate. The init() method simply creates or loads some data that will be used throughout the life of the servlet.

The init method definition looks like this −

```
public void init() throws ServletException {
   // Initialization code...
}
```

The service() Method

The service() method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client.

Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

Here is the signature of this method −

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

The service () method is called by the container and service method invokes doGet, doPost, doPut, doDelete, etc. methods as appropriate. So you have nothing to do with service() method but you override either doGet() or doPost() depending on what type of request you receive from the client.

The doGet() and doPost() are most frequently used methods with in each service request. Here is the signature of these two methods.

The doGet() Method

A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
   // Servlet code
}
```

The doPost() Method

A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
   // Servlet code
}
```

The destroy() Method

The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

After the destroy() method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this −

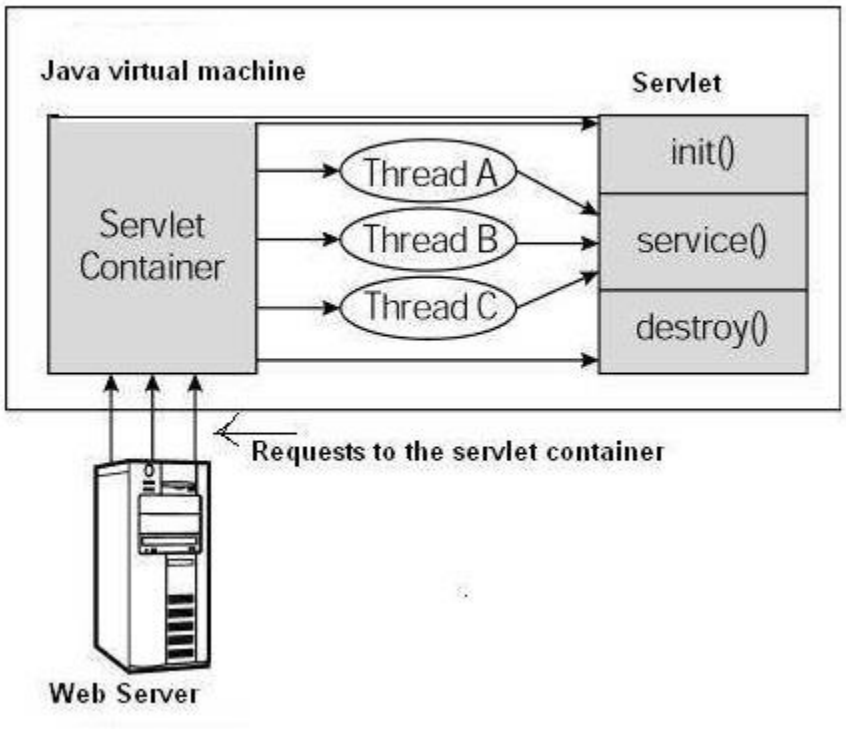
```
public void destroy() {
   // Finalization code...
}
```

Architecture Diagram

The following figure depicts a typical servlet life-cycle scenario.

- First the HTTP requests coming to the server are delegated to the servlet container.

- The servlet container loads the servlet before invoking the service() method.
- Then the servlet container handles multiple requests by spawning multiple threads, each thread executing the service() method of a single instance of the servlet.



Request and Response Object in Servlet

ServletRequest Interface

1. ServletRequest Interface

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

Methods of ServletRequest interface

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

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

ServletResponse Interface

The servlet container is connected to the web server that receives Http Requests from client on a certain port. When client sends a request to web server, the servlet container creates `HttpServletRequest` and `HttpServletResponse` objects and passes them as an argument to the servlet service() method.

The response object allows you to format and send the response back to the client. First we will see the commonly used methods in the ServletReponse interface and then we will see an example.

Method of ServletResponse interface

1) `String getCharacterEncoding()`: It returns the name of the MIME charset used in body of the response sent to the client.
2) `String getContentType()`: It returns the response content type. e.g. text, html etc.
3) `ServletOutputStream getOutputStream()`: Returns a ServletOutputStream suitable for writing binary data in the response.
4) `java.io.PrintWriter getWriter()`: Returns the PrintWriter object.
5) `void setCharacterEncoding(java.lang.String charset)`: Set the MIME charset (character encoding) of the response.

10) `void flushBuffer()`: Forces any content in the buffer to be written to the client.

To get the complete list of methods. Refer the official documentation.

index.html

```
<form action="mydetails" method="get">
User name: <input type="text" name="uname">
<input type="submit" value="login">
</form>
```

MyServletDemo.java

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class MyServletDemo extends HttpServlet{
   public void doGet(HttpServletRequest req,HttpServletResponse res)
   throws ServletException,IOException
   {
     res.setContentType("text/html");
     PrintWriter pwriter=res.getWriter();
     String name=req.getParameter("uname");
     pwriter.println("User Details Page:");
     pwriter.println("Hello "+name);
     pwriter.close();  }}
```

Servlets - Cookies Handling

Cookies are text files stored on the client computer and they are kept for various information tracking purpose. Java Servlets transparently supports HTTP cookies.

There are three steps involved in identifying returning users −

- Server script sends a set of cookies to the browser. For example name, age, or identification number etc.
- Browser stores this information on local machine for future use.
- When next time browser sends any request to web server then it sends those cookies information to the server and server uses that information to identify the user.

A servlet will then have access to the cookie through the request method *request.getCookies()* which returns an array of *Cookie* objects.

Servlet Cookies Methods

Following is the list of useful methods which you can use while manipulating cookies in servlet.

Sr.No.	Method & Description
1	**public void setDomain(String pattern)** This method sets the domain to which cookie applies, for example tutorialspoint.com.
2	**public String getDomain()** This method gets the domain to which cookie applies, for example tutorialspoint.com.
3	**public void setMaxAge(int expiry)** This method sets how much time (in seconds) should elapse before the cookie expires. If you don't set this, the cookie will last only for the current session. **public int getMaxAge()** This method returns the maximum age of the cookie, specified in seconds, By default, -1 indicating the cookie will persist until browser shutdown.
4	
5	**public String getName()**This method returns the name of the cookie. The name cannot be changed after creation.
6	**public void setValue(String newValue)** This method sets the value associated with the cookie
7	**public String getValue()** This method gets the value associated with the cookie.

Setting Cookies with Servlet

Setting cookies with servlet involves three steps −

(1) Creating a Cookie object − You call the Cookie constructor with a cookie name and a cookie value, both of which are strings.

```
Cookie cookie = new Cookie("key","value");
```

Keep in mind, neither the name nor the value should contain white space or any of the following characters −

```
[ ] ( ) = , " / ? @ : ;
```

(2) Setting the maximum age − You use setMaxAge to specify how long (in seconds) the cookie should be valid. Following would set up a cookie for 24 hours.

```
cookie.setMaxAge(60 * 60 * 24);
```

(3) Sending the Cookie into the HTTP response headers − You use response.addCookie to add cookies in the HTTP response header as follows −

```
response.addCookie(cookie);
```

Example

Let us modify our Form Example to set the cookies for first and last name.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class HelloForm extends HttpServlet {

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

      // Create cookies for first and last names.
      Cookie firstName = new Cookie("first_name",
request.getParameter("first_name"));
      Cookie lastName = new Cookie("last_name",
request.getParameter("last_name"));

      // Set expiry date after 24 Hrs for both the cookies.
      firstName.setMaxAge(60*60*24);
      lastName.setMaxAge(60*60*24);

      // Add both the cookies in the response header.
```

```
      response.addCookie( firstName );
      response.addCookie( lastName );

      // Set response content type
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      String title = "Setting Cookies Example";
      String docType =
         "<!doctype html public \"-//w3c//dtd html 4.0 " +
"transitional//en\">\n";

      out.println(docType +
         "<html>\n" +
            "<head>
               <title>" + title + "</title>
            </head>\n" +

            "<body bgcolor = \"#f0f0f0\">\n" +
               "<h1 align = \"center\">" + title + "</h1>\n" +
               "<ul>\n" +
                  "  <li><b>First Name</b>: "
                  + request.getParameter("first_name") + "\n" +
                  "  <li><b>Last Name</b>: "
                  + request.getParameter("last_name") + "\n" +
               "</ul>\n" +
            "</body>
         </html>"
      );
   }
}
```

Compile the above servlet **HelloForm** and create appropriate entry in web.xml file and finally try following HTML page to call servlet.

```
<html>
   <body>
      <form action = "HelloForm" method = "GET">
         First Name: <input type = "text" name = "first_name">
         <br />
         Last Name: <input type = "text" name = "last_name" />
         <input type = "submit" value = "Submit" />
      </form>
   </body>
</html>
```

Keep above HTML content in a file Hello.htm and put it in <Tomcat-installationdirectory>/webapps/ROOT directory. When you would access *http://localhost:8080/Hello.htm*, here is the actual output of the above form.

First Name:

Last Name:

Try to enter First Name and Last Name and then click submit button. This would display first name and last name on your screen and same time it would set two cookies firstName and lastName which would be passed back to the server when next time you would press Submit button.

Next section would explain you how you would access these cookies back in your web application.

Reading Cookies with Servlet

To read cookies, you need to create an array of *javax.servlet.http.Cookie* objects by calling the **getCookies()** method of *HttpServletRequest*. Then cycle through the array, and use getName() and getValue() methods to access each cookie and associated value.

Example

Let us read cookies which we have set in previous example −

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class ReadCookies extends HttpServlet {

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

      Cookie cookie = null;
      Cookie[] cookies = null;

      // Get an array of Cookies associated with this domain
      cookies = request.getCookies();

      // Set response content type
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      String title = "Reading Cookies Example";
      String docType =
         "<!doctype html public \"-//w3c//dtd html 4.0 " +
         "transitional//en\">\n";

      out.println(docType +
         "<html>\n" +
         "<head><title>" + title + "</title></head>\n" +
         "<body bgcolor = \"#f0f0f0\">\n" );

      if( cookies != null ) {
         out.println("<h2> Found Cookies Name and Value</h2>");
```

```
         for (int i = 0; i < cookies.length; i++) {
            cookie = cookies[i];
            out.print("Name : " + cookie.getName( ) + ",  ");
            out.print("Value: " + cookie.getValue( ) + " <br/>");
         }
      } else {
         out.println("<h2>No cookies founds</h2>");
      }
      out.println("</body>");
      out.println("</html>");
   }
}
```

Compile above servlet **ReadCookies** and create appropriate entry in web.xml file. If you would have set first_name cookie as "John" and last_name cookie as "Player" then running *http://localhost:8080/ReadCookies* would display the following result −

Found Cookies Name and Value

```
Name : first_name, Value: John

Name : last_name,  Value: Player
```

Delete Cookies with Servlet

To delete cookies is very simple. If you want to delete a cookie then you simply need to follow up following three steps −

- Read an already existing cookie and store it in Cookie object.
- Set cookie age as zero using **setMaxAge()** method to delete an existing cookie
- Add this cookie back into response header.

Example

The following example would delete and existing cookie named "first_name" and when you would run ReadCookies servlet next time it would return null value for first_name.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class DeleteCookies extends HttpServlet {

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

```
      Cookie cookie = null;
      Cookie[] cookies = null;

      // Get an array of Cookies associated with this domain
      cookies = request.getCookies();

      // Set response content type
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      String title = "Delete Cookies Example";
      String docType =
         "<!doctype html public \"-//w3c//dtd html 4.0 " +
"transitional//en\">\n";

      out.println(docType +
         "<html>\n" +
         "<head><title>" + title + "</title></head>\n" +
         "<body bgcolor = \"#f0f0f0\">\n" );

      if( cookies != null ) {
         out.println("<h2> Cookies Name and Value</h2>");

         for (int i = 0; i < cookies.length; i++) {
            cookie = cookies[i];

            if((cookie.getName( )).compareTo("first_name") == 0 ) {
               cookie.setMaxAge(0);
               response.addCookie(cookie);
               out.print("Deleted cookie : " + cookie.getName( ) + "<br/>");
            }
            out.print("Name : " + cookie.getName( ) + ",  ");
            out.print("Value: " + cookie.getValue( )+" <br/>");
         }
      } else {
         out.println("<h2>No cookies founds</h2>");
      }
      out.println("</body>");
      out.println("</html>");
   }   }
```

Compile above servlet **DeleteCookies** and create appropriate entry in web.xml file. Now running *http://localhost:8080/DeleteCookies* would display the following result −

Cookies Name and Value

```
Deleted cookie : first_name

Name : first_name, Value: John

Name : last_name,  Value: Player
```

Now try to run *http://localhost:8080/ReadCookies* and it would display only one cookie as follows –

4) HttpSession interface

In such case, container creates a session id for each user.The container uses this id to identify the particular user.An object of HttpSession can be used to perform two tasks:

1. bind objects
2. view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.

How to get the HttpSession object ?

The HttpServletRequest interface provides two methods to get the object of HttpSession:

1. **public HttpSession getSession():**Returns the current session associated with this request, or if the request does not have a session, creates one.
2. **public HttpSession getSession(boolean create):**Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.

Commonly used methods of HttpSession interface

1. **public String getId():**Returns a string containing the unique identifier value.
2. **public long getCreationTime():**Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
3. **public long getLastAccessedTime():**Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.
4. **public void invalidate():**Invalidates this session then unbinds any objects bound to it.

Example of using HttpSession

In this example, we are setting the attribute in the session scope in one servlet and getting that value from the session scope in another servlet. To set the attribute in the session scope, we have used the setAttribute() method of HttpSession interface and to get the attribute, we have used the getAttribute method.

index.html

```
1. <form action="servlet1">
2. Name:<input type="text" name="userName"/><br/>
3. <input type="submit" value="go"/>
4. </form>
```

FirstServlet.java

```
1. import java.io.*;
2. import javax.servlet.*;
3. import javax.servlet.http.*;
4.
5.
6. public class FirstServlet extends HttpServlet {
7.
8. public void doGet(HttpServletRequest request, HttpServletResponse response){
9.         try{
10.
11.        response.setContentType("text/html");
12.        PrintWriter out = response.getWriter();
13.
14.        String n=request.getParameter("userName");
15.        out.print("Welcome "+n);
16.
17.        HttpSession session=request.getSession();
18.        session.setAttribute("uname",n);
19.
20.        out.print("<a href='servlet2'>visit</a>");
21.
22.        out.close();
23.
24.              }catch(Exception e){System.out.println(e);}
25.    }
26.
27. }
```

SecondServlet.java

```
1.  import java.io.*;
2.  import javax.servlet.*;
3.  import javax.servlet.http.*;
4.
5.  public class SecondServlet extends HttpServlet {
6.
7.  public void doGet(HttpServletRequest request, HttpServletResponse response)
8.         try{
9.
10.        response.setContentType("text/html");
11.        PrintWriter out = response.getWriter();
12.         HttpSession session=request.getSession(false);
13.        String n=(String)session.getAttribute("uname");
14.        out.print("Hello "+n);
15.          out.close();         }catch(Exception e){System.out.println(e);}
16.    } }
```

Index.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<html>      <head>        <title>First Page</title>     </head>
  <body><center>
     <form action="NewServlet" method="get">
     <h1>Hello World!</h1>
     No: <input type="text" name="fno"><br><br>
     Name:<input type="text" name="fname"><br><br>
     Address:<input type="text" name="faddress"><br><br>
     <input type="submit" value="Add">
     </form>        </center>      </body>   </html>
```

NewServlet.java

```
import java.sql.*;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class NewServlet extends HttpServlet {
   protected void processRequest(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
      try {
      } finally {
         out.close();
         }      }
 @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException
   {
      response.setContentType("text/html");
      PrintWriter out=response.getWriter();
      out.println("No  "+request.getParameter("fno"));
      out.println("<br>");
      out.println("Name  "+request.getParameter("fname"));
      out.println("<br>");
```

```
   out.println("Address  "+request.getParameter("faddress"));
   out.println("<br>");
   int n=Integer.parseInt(request.getParameter("fno"));
   String name=request.getParameter("fname");
   String address=request.getParameter("faddress");
   try
   {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      Connection con=DriverManager.getConnection("jdbc:odbc:udms","system","system");
      Statement stat=con.createStatement();
      PreparedStatement ps=con.prepareStatement("insert into stud values(?,?,?)");
      ps.setInt(1,n);
      ps.setString(2,name);
      ps.setString(3,address);
      ps.executeUpdate();
      out.println("Record Inserted Successfully....");
      out.println("All records from table");
      ResultSet rs=stat.executeQuery("select * from stud");
      while(rs.next())
      {
         out.print("No "+rs.getInt(1));
         out.print("Name  "+rs.getString(2));
         out.print("Address  "+rs.getString(3));
         out.println("<br>");
      }
   out.println("<html>");
   out.println("<body>");
   out.println("<form action=index.jsp>");
   out.println("<input type=submit value=LogOut>");
   out.println("</form>");
   out.println("</body>");
   out.println("</html>");
      con.close();
   }
   catch(Exception e)
   {         }
      processRequest(request, response);       }
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
   processRequest(request, response);
}         }
```