Installing a Web Server

1. Install a sample web server, which supports Servlets/JSPs. A light weight web server is Apache Tomcat server. You can get the server from
   `http://tomcat.apache.org/`

2. Follow the installation directions and install the server on ccc

3. We will call the root of your installation as $TOMCAT_DIR
   `setenv TOMCAT_DIR /home/mmani/apache-tomcat-5.5.15`

4. Setup the configuration:
   a. Check the file $TOMCAT_DIR/conf/server.xml - You will see a line starting as <Connector port="8080" . You can renumber the port to any number say between 1200 and 20000. This is the port where you our web server will listen for connections. You can leave the port at default value itself.
   b. Let us set the path and class path necessary as:
      `setenv PATH ${PATH}:${TOMCAT_DIR}/bin`
      `setenv CLASSPATH`
      `${CLASSPATH}:${TOMCAT_DIR}/common/lib/servlet-api.jar`

5. Test our web server is installed correctly as:
   - Start up the web server with the script startup.sh
   - Suppose you start the web server from the machine ccc2.wpi.edu, go to the page http://ccc2.wpi.edu:8080 You will know whether the installation was correct.

6. Remember: Once you are down, you **MUST** shut down the web server and clean up any unnecessary java processes still running. It really slows down the machine otherwise. You can shut down the servlet by the script shutdown.sh You can check java processes running by
   `ps -u <loginName>`
   If you find any java processes running, kill them. For ease you can use
   `killall java`

Example Servlet

Now we are ready to test servlets. We will use the default directory for servlets, which is $TOMCAT_DIR/webapps/servlets-examples/WEB-INF/classes. The default directory for JSPs is $TOMCAT_DIR/webapps/jsp-examples/jsp2/el

Let us first test an example servlet. Make sure your web server is running. Now go to the URL http://ccc2.wpi.edu:8080/servlets-examples/servlet/HelloWorldExample You will see whether the servlet is working or not.

Let us create an example servlet. Remember that the default directory for servlets is $TOMCAT_DIR/webapps/servlets-examples/WEB-INF/classes Let us go to that directory. Here we will create a java file called TestServlet.java The source code is

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

public class TestServlet extends HttpServlet {

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

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<HTML>\n" +
                "<HEAD><TITLE>Hello</TITLE></HEAD>\n" +
                "<BODY BGCOLOR=\'WHITE\'>\n" +
                "<H1>Hello</H1>\n");
    out.println ("</BODY></HTML>");
  }

}
```

Note that it needs to have a method called doGet with exactly the same parameters and exceptions as above. Now compile it with `javac TestServlet.java` Now we have created a new servlet file We need to ensure that our web server can see the servlet file. For this we have to do the following:

- Open the file $TOMCAT_DIR/webapps/servlets-examples/WEB-INF/web.xml This file specifies all the servlets that are visible to the web server. Find a set of declarations such as

  ```
  <servlet>
      <servlet-name>...</servlet-name>
      <servlet-class>...</servlet-class>
  </servlet>
  ```

  The above declaration specifies for each servlet name, the class associated with it. Append at the end of these declarations something like

  ```
  <servlet>
      <servlet-name>MyTestServlet</servlet-name>
      <servlet-class>TestServlet</servlet-class>
  </servlet>
  ```

  This file also declares the URL-pattern to be specified to access a specific servlet. Find in the file a set of declarations such as

  ```
  <servlet-mapping>
      <servlet-name>...</servlet-name>
      <url-pattern>...</url-pattern>
  </servlet-mapping>
  ```

  Append at the end of such declarations something like

  ```
  <servlet-mapping>
      <servlet-name>MyTestServlet</servlet-name>
      <url-pattern>/servlet/FirstTestServlet</url-pattern>
  </servlet-mapping>
  ```

Now we are ready to test our servlet is accessible over the web. Restart the web server. Now go to the URL http://ccc2.wpi.edu:8080/servlets-examples/servlet/FirstTestServlet

You will see that the servlet is correctly executed.

HTML Forms

Now let us see how we can create a web form to handle our data. We will create a simple HTML form such as

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD><TITLE>HTML Test</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<H1>HTML Test</H1>
<FORM METHOD="POST" ACTION="http://ccc2.wpi.edu:8080/servlets-
examples/servlet/HelloServlet1">
<input type="text" name="sNumber"/>&amp;nbsp;&amp;nbsp;sNumber<br>
<input type="text" name="sName"/>&amp;nbsp;sName<br>
<input type="text" name="professor"/>professor<br>
<p><INPUT TYPE=SUBMIT value="Submit"/></p>
</FORM>
</BODY>
</HTML>
```

We will save the file as Hello1.html in the directory $TOMCAT_DIR/webapps/ROOT Now this document can be accessed via the URL: http://ccc2.wpi.edu:8080/Hello1.html

Main things to note is what is in between <FORM> and </FORM>. Note METHOD=”POST”. This is necessary and it says that this form posts the values that the user types in the form. The ACTION=”…” specifies where the form values should be sent to. We send it to our servlet. Note the <input …> Each input requires the user to do something. The type=”…” specifies what the type of the input is, some of the important types are: text, radio (for radio buttons), checkbox, submit and reset.

Note the input with type=”submit” and value=”Submit”. This creates a button “Submit”, when this button is clicked, the action is performed, and the values entered by the user are sent to the servlet.

Servlet to handle a post and use JDBC

Before we use JDBC, we should ensure that the JDBC driver can be downloaded by our servlet. The servlet sees only the classes that are available at $TOMCAT_DIR/shared/lib (it actually also sees classes at $TOMCAT_DIR/common/lib, but we do not need that for our project). To do this, go to the directory $TOMCAT_DIR/shared/lib Here we create a symbolic link to the JDBC driver (oracle JDBC driver) as:

```
ln -s $ORACLE_HOME/jdbc/lib/ojdbc14.jar ojdbc14.jar
```
This creates a symbolic link called ojdbc.jar which points to the JDBC jar file. (Remember that for ccc, $ORACLE_HOME is set to /usr/local/oracle/app/oracle/product/9.2.0.1

Now let us write a servlet that handles a post and performs the required action on the Oracle database. For example, how does HelloServlet1.java look like. For this we replace the previous HelloServlet1.java with the following code.

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

public class HelloServlet1 extends HttpServlet {
    Connection conn = null; Statement stmt = null;

  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<HTML>\n" +
                "<HEAD><TITLE>Hello</TITLE></HEAD>\n" +
                "<BODY BGCOLOR=\'WHITE\'>\n" +
                "<H1>Hello</H1>\n");

      try {

          getDBConnection (out);
          String sNumber = request.getParameter ("sNumber");
          String sName = request.getParameter ("sName");
          String professor = request.getParameter ("professor");
          String sql = "INSERT INTO Student (sNumber, sName,
professor)"
                       + " VALUES (" + sNumber + ",'" + sName + "','"
                       + professor + "')";
          out.println (sql);
          stmt.executeUpdate (sql);

      } catch (Exception e) {
            out.println ("some error happened");
      }

      out.println ("</BODY></HTML>");
  }

    void getDBConnection (PrintWriter out) throws Exception {

      try {
          Class.forName("oracle.jdbc.driver.OracleDriver");
          out.println ("Got driver");
          conn = DriverManager.getConnection
                  ("jdbc:oracle:thin:@oracle.wpi.edu:1521:CS",
                                  "mmani", "mmani");
```

```
            out.println("Got connected");
            stmt = conn.createStatement ();
        } catch (Exception e) { throw e; }
    }
}
```

Now we will compile the servlet, ensure that $TOMCAT_DIR/ webapps/servlets-examples/WEB-INF/web.xml has entries corresponding to this servlet, and restart the web server. Now we go to the URL: http://ccc2.wpi.edu:8080/Hello1.html

We type in values in the fields, and then click Submit. Look at the resulting HTML document, also log on to oracle and check if the intended operation was performed.

If you get this successfully, then you are ready to build your complete interface. Design it, implement it, and test it.