LS/EECS Building Ecommerce Applications

Lab 1: StudentCalc R1.0

(upload it by the deadline)

Objectives

-understand the development environment
-setup the development environment
-deploy and run simple html, jsp and servlets
-understand the APIs of the Servlet's HTTP request and response

Task A: Setup and test the Dev and Runtime Environment on Lab computers, skip 1-3)

1. Download and install Eclipse **"Eclipse IDE for Java Enterprise Developers,"** (4.10) from eclipse.org
    a. Need to install at least Java 8 on your computer, (JRE 1.8) https://docs.oracle.com/javase/8/docs/technotes/guides/install/install_overview.html
    b. Start Eclipse
    c. Explore the Perspectives
    d. Run your first Java program, Hello.java
2. Install Apache Tomcat (v8.5), https://tomcat.apache.org/tomcat-8.5-doc/index.html
3. Download and unzip Apache derby version 10.12.1.0 https://db.apache.org/derby/releases/release-10.12.1.1.cgi
4. Configure Eclipse with Tomcat (in Java EE perspective, explore the Servers tab and add the Tomcat directory)
5. Configure Eclipse with Tomcat (in Java EE perspective, explore the Servers tab and add the Tomcat directory)
6. Create your first Java EE project, **studentCalc1**
    a. The project should have a **web.xml** file (last page of the create wizard)
    b. Understand the structure of a Java EE project
7. Test your environment by writing, deploying, running, your first pages
    a. Html page: HelloFromHTML.html
    b. Jsp page: HelloFromJSP.jsp
    c. Java Faces: HelloFromFaces.jsp

Use pop-up menu to create the pages. For jsp/faces files, use the "Create JSP" meny, select the templates for xml for each.
In each page, add some HTML markup to say "Hello…"
Then use "Run on Server" menu. Notice the Server tab.

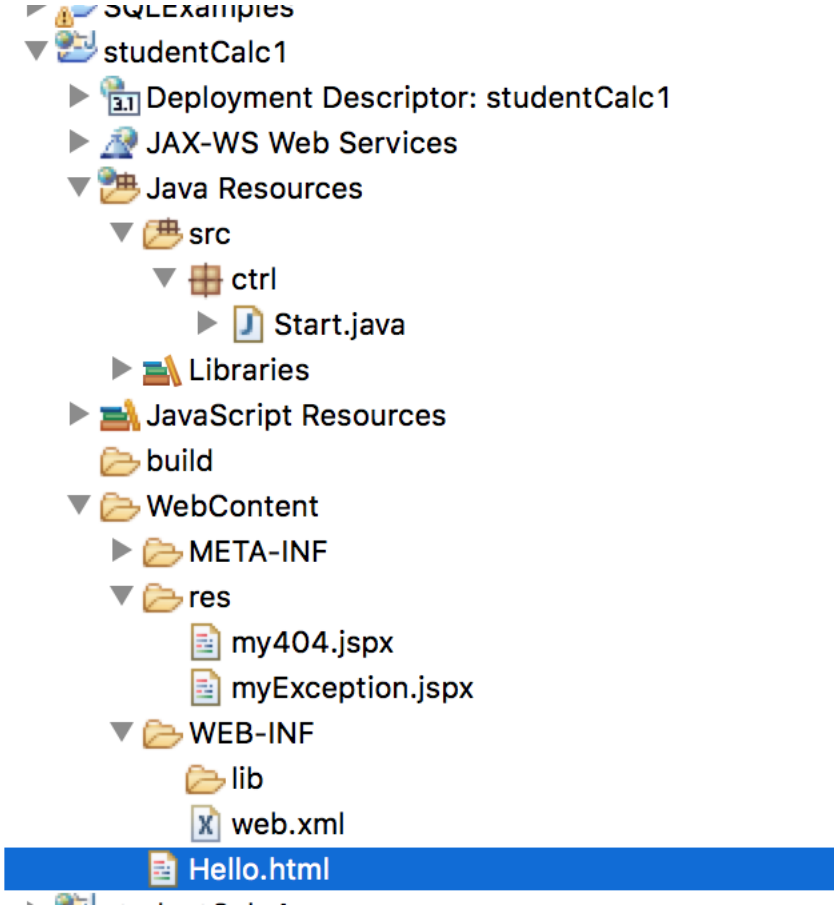

Fig. 1 A typical Java EE project in Eclipse: src folder contains the java files, including servlets; WebContent folder contains the html and js pages. Note the web.xml file and the error and exception pages.

Task B: Explore the servlet, request and response API

8. Create your first Servlet, **Start.java** to write "Hello" to the console (on server). To do that, after you create the servlet, in doGet method, add this line:

```
System.out.println("Hello, Got a GET request!");
```

- Invoke servlet at this URL: http://localhost:8080/studentCalc1/Start
- You can use an external browser, like Chrome, Firefox to access the servlet

7. By exploring the API of the servlet, add code to Start.java to do the following :

- Return the client ip and port to the browser
  - How would you implement an IP-filtering firewall?
- Return the request's http protocol and method
- Pass a query string in the URL and return the entire query string sent by the client (check the lecture slides how you pass parameters)
  - Use the getParameter of request to extract a named request parameter
- Embed spaces in the query string and note the URL encoding in the developer tool.

Notes:
To reply to the client (browser) with a message, in the servlet *Start.java*, *doGet()* method, you need to obtain a *writer* from *response* object and write a string…

```
response.setContentType("text/plain");
Writer resOut = response.getWriter();
resOut.write("Hello, World!\n");
```

To explore different elements of the http request message, use *get* methods on *request* object…

```
String clientIP = request.getRemoteAddr();
resOut.write("Client IP: " + clientIP + "\n");
String clientQueryString = request.getQueryString();
String foo = request.getParameter("foo");
resOut.write("Query Param foo=" + foo + "\n");
```

8.Modify Start.java to

- Change the annotation of your servlet so it responds to /Startup as well as /Start.
- Change the URL mapping again to include /Startup/*

Notes:
When you create the servlet Start.java, it is created with this annotation,
@WebServlet({ "/Start" }).
You can add additional paths to this annotation

- Output the various pieces of the URL: URI, context, servlet/extra path…..
- If the client visited your webapp via /Startup/YorkBank then force their browser to redirect to Start. Hint: use the sendRedirect method of the response object.

Notes:
response object has a method *redirect*…here is a hint of how to use it when in the url of the servlet you find "YorkBank"

response.sendRedirect(…)

the argument of *sendRedirect* should be the url of the servlet; you can hardcode it but more elegant is to compose the url of the servlet from servlet context, something like this:

String url= this.getServletContext().getContextPath() +"/Start";

Task C: Understanding Java EE Application Descriptor

9. Deployment Descriptor, web.xml
   Note that changing web.xml requires a server restart.

- Add Start to the welcome file list and test it.
- Add a *context parameter* and verify that you can access it in the servlet.
- Add an error-page for error-code 404 and point it at location */res/my404.jspx*. Test by visiting a non-existent page.
- Add an error-page for exception-type java.lang.Exception and point it at location */res/myException.jspx*. Test by triggering any exception in your servlet.

Notes:

You should have a *web.xml file* in your project.

Edit *web.xml* using the xml editor that opens the file (design view)
-add context parameter children. The parameters have name and value
Ex:

```
<context-param>
    <param-name>appName</param-name>
    <param-value>OSAP Calculator</param-value>
  </context-param>
```

```
<context-param>
  <param-name>principal</param-name>
  <param-value>1000</param-value>
</context-param>
```

Note: web.xml is loaded when the application is deployed, you might need to restart the server after you edit it.

You can read the context parameters from within the servlet, using the method getServletContext().getInitParameter("parameterName")//it returns a string that need to be converted…
Ex:

```
principal = Double.parseDouble(this.getServletContext().getInitParameter("principal"));
```

10. Error pages… (see Fig 1.)
   a) you have to create them in webContent/res folder in your project
   b) you have to edit the web.xml and add error page children, like these:

```
<error-page>
    <error-code>404</error-code>
    <location>/res/my404.jspx</location>
  </error-page>
  <error-page>
    <exception-type>java.lang.Exception</exception-type>
    <location>/res/myException.jspx</location>
  </error-page>
```

   c) then create the pages as shown in Fig. 1. The pages should display some meaningful content.

Task D: OSAP Calculator and Parameters

11. Modify Start.java to compute the OSAP monthly payments:

- Extract the values of the following request parameters: principal, period, and interest (the parameters come via URL). You can assume that any client-supplied value is valid; i.e. do not validate.
- If a parameter is missing, supply a default value obtained from a context parameter. This way, one can change the defaults w/o recompiling the servlet.

- Compute the monthly payment using the formula: $(r/12)*A/[1 - (1+ (r/12))^{-n}]$, where r is the annual interest rate, A is the present value (principal), and n is the period measured in months.
- Send the computed payment with an appropriate message to the client but format the amount so it is rounded to the nearest cent.

Notes:

You can pass the parameters to the servlet in the url of the get method
Ex:
http://localhost:8080/studentCalc1?principle=10000&interest=10&period=24
In the servler, doGet method, you retrieve the parameters using request.getParameter("parameterName") method

You can also pass the default values for principal, interest and period in the context-param of the web.xml…see Step 9 how to retrieve them..

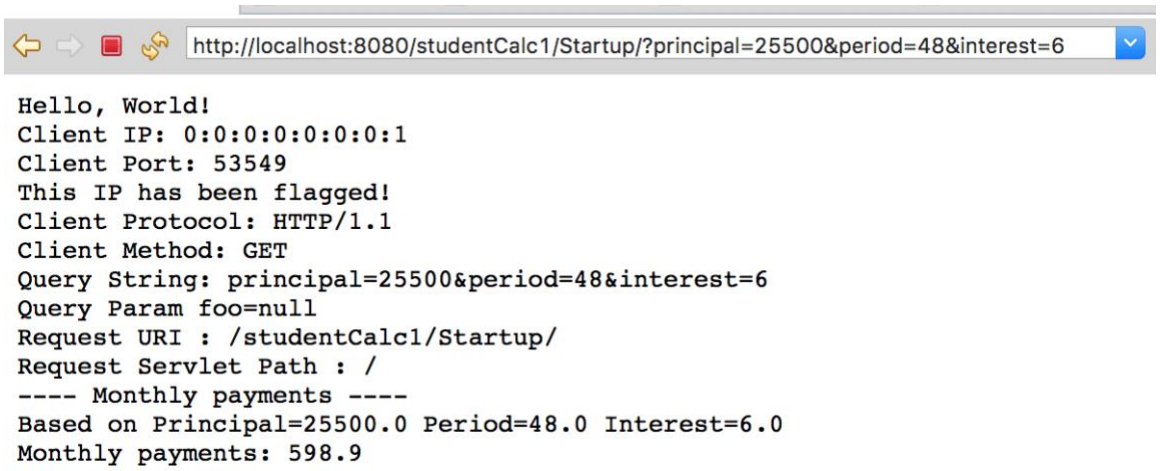

Fig 2. A snapshot of the output in the browser