COMP 3095 – Lab 3

HTTP Status Codes and Redirection Lab

In java web development it is sometimes necessary to set a status code for a web response, often called the response code. The response code effectively tells the client what the output represents. Or in other words, it indicates the status of the request: successful, page not found, access denied etc…

When to set the status code:

In general, a developer should set the status code before sending any output. This is because the status code always be set and identified before payload data returned to the client.

Redirection:

The sendRedirect() method of the HttpServletResponse interface can be used to redirect a request to another resource, it may be a servlet, jsp, or html file. It is primarily used to redirect to different servers or domains. This transfer of control task is delegated to the browser by the container. In other words, the redirect sends a header back to the browser (client). The header contains the resource url (location) to redirect to, then the browser initiates a new request to the given url.

The sendRedirect() accepts relative as well as absolute URLs and works on the client-side browser, to make another Http request.

Syntax of sendRedirect() method

```
public void sendRedirect(String URL)throws IOException;
```

Example of sendRedirect() method

```
response.sendRedirect("http://www.javatpoint.com");
```

In this lab we will investigate the use of HttpServletResponse sendRedirect() and HTTP status codes.

Creating Redirection

1. Open the **servletlabs** project in the previous tutorial.
2. Right click on the **WebContent** folder, select **New → HTML File**. Name the file: **index.html**.
3. Inside index.html add the following code:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Status Codes and Redirection</title>
</head>
<body>

    <fieldset>
    <legend>Please select a search engine:</legend>
    <form action="search" method="post">
        <input type="radio" name="searchEngine" value="Google" checked> Google
        <input type="radio" name="searchEngine" value="Yahoo" > Yahoo
        <input type="radio" name="searchEngine" value="Bing"  > Bing
        <input type="radio" name="searchEngine" value="Fake"  > Fake
        <input type="submit" value="Submit">
    </form>
    </fieldset>


</body>
</html>
```

4. Ensure that you have a valid web.xml file that lists the newly created index.html file as your welcome-file (application entry point):

```
<welcome-file-list>
  <welcome-file>index.html</welcome-file>
  <welcome-file>index.htm</welcome-file>
  <welcome-file>index.jsp</welcome-file>
  <welcome-file>default.html</welcome-file>
  <welcome-file>default.htm</welcome-file>
  <welcome-file>default.jsp</welcome-file>
</welcome-file-list>
```

Now that we have created an index file added it to our welcome-file-list, we can simply access our application with the following URL: **http://localhost:8080/servletlabs/**

5. Create a servlet to interact with the index html file and to act as the servlet-handler. Create a descriptive package name to store the servlet. Right click in **servletlabs** project, select **New→Servlet**. Name the servlet something descriptive and annotate with @WebServlet or alternatively define your servlet in your web.xml file (**hint**: review the index.html file above to determine proper servlet annotation url). Finally, store in the servlet in the package you've just created.

6. Add the following code to your servlet:

```
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        String name = request.getParameter("searchEngine");
        if (name.contains("Google")) {
            response.sendRedirect("http://www.google.ca");
        } else if (name.contains("Yahoo")) {
            response.sendRedirect("http://www.yahoo.ca");
        } else if (name.contains("Bing")) {
            response.sendRedirect("http://www.bing.ca");
        } else {
            response.sendError(HttpServletResponse.SC_NOT_FOUND,
                    "The search engine is fake and cannot be found");
        }

    }

}
```

7. Submit the form using different answer selections, validate the data send back and forth utilizing the network packet analyzer in Chrome (**Chrome Menu →More Tools → Developer Tools**). Validate status codes for the above. What status codes did you receive? Is this what was expected?

Exercises

1. Inspired from the example above, create a form that takes form data from a user (First name, Last name, Address, Postal Code, and Credit-Card number etc..) that when submitted, invokes a servlet directing the user to an appropriate credit-card web-site. To determine which web-site to direct the user, base your logic on the following credit-card account number regular expression information:

1. ^4[0-9]{12}(?:[0-9]{3})?$ → www.visa.com
2. ^5[1-5][0-9]{14}$ → www.mastercard.com
3. ^3[47][0-9]{13}$ → www.americanexpress.com/canada

***Note, use the account number to validate the credit-card issuer.*

****Create a separate class that handles the credit-card number validation, removing any such logic from any servlet.*

2. Treat all fields as mandatory in your form, validate all data that is provided, and in the event any data is missing, return a response issuing the appropriate status code (ie. **400-Bad Request / HttpServletResponse.SC_BAD_REQUEST)** and details of the error.

3. Create another servlet, that acts as a helper servlet (name it appropriately) that when invoked, simply displays help information (static text/html) educating the client of acceptable account numbers and their formats for each of the credit-card issuers named above.

4. In the client form created in step #1, now add an additional “Help” button that when invoked, sends a **redirect** to the application exposing your “Helper” servlet created in step #3, finally displaying the help information to the end-user through the browser.

5. Lastly create a war file of your project, deploy to your tomcat container through the Tomcat manager/admin console, validating your deployment runs and works successfully (ie. the application is running and functional).