**<u>TOMCAT NOTES</u>**:
Each and every time you re-compile/re-build your Java code you must copy the new `*`.class files from your Eclipse `bin` folder into the `Tomcat/webapps/lastnamefirstinitial/web-inf/classes` folder (including your *`lastnamefirstinitial`* package folder, and then reload the new files into Tomcat by going to `http://localhost:8080/manage` and clicking the "Reload" button in the "Commands" column (IF YOU FAIL TO MOVE THE FILES OR RELOAD THE PROJECT YOUR CHANGES WILL NOT BE PICKED UP BY THE SERVER)

All of your classes should exist in a *`webd4201.lastnamefirstinitial`* package (all lowercase).

**Be sure that you ZIP up your *`webd4201.lastnamefirstinitial`* folder that is in the `Tomcat/webapps` folder, with a copy of your source code (i.e. your Eclipse `src` folder) placed inside of it. Any other compression schemes other than *.zip will be penalized. This *.zip file should be submitted into DC Connect on or before the due date.**

For this assignment you are going to create several java classes/files, we will be building upon the existing code from previous assignments. It is suggested you use the Bradshaw jsp/servlet example as a start point. You are required to make a new project folder in your webapps folder in the Tomcat folder that is your `lastnamefirstinitial`. It is suggested that you copy the "`demo`" folder in the `webapps` folder, and then paste it in the same location, then rename the "`demo – Copy`" folder to your `lastnamefirstinitial.`

**BE SURE TO USE YOUR `webd4201.lastnamefirstinitial` (all lowercase) AS THE PACKAGE NAME FOR YOUR JAVA CLASSES (this will build a `webd4201.lastnamefirstinitial` folder in your build structure, this should already be the case from previous assignments, if not you are to change it so that it is.)**

Be sure that your generated html conforms to XHTML 1.0 Strict compliance; you must include the success image when compliance is met. In order for the http://validator.w3.org to work you must include the following two lines at the top of your `header.jsp` file:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
                 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
```

Submit complete program source code that satisfies the following requirements:

1. You are to continue to use a PostgreSQL database that runs on port `5432` (default), named `webd4201_db` that is owned by a user named `webd4201_admin` that has a password of `webd4201_password`

   NOTE: This should already be set up from previous assignments. If you do not set your database up with these values exactly for testing, your program will not run on your instructor's laptop for assessment (there will be penalties).

2. Your database should be preloaded with tables for `Users, Students`, `Faculty, Courses` and `Marks` (these will be required to get this assignment to work).

3. Modify your `StudentDA.java` file so that there is a method named as listed and will perform the following:
   - `authenticate()` (using a long student number and a String password ) – this method should return a Student object (if the student exists in the database). This method should throw a `NotFoundException` if the student number and password combination passed are not found in the database. This method should be static (i.e. it can be called as `Student.login(100111111, "password")`

4. Modify the Student.java class to have:
   a. class (i.e. static) method named `authenticate()` (that takes a long argument and a Sting) that call the similarly name method created in the `StudentDA.java` class (described above).

   NOTE: at this point you should be able to test your code by placing a
   `Student.login(100111111, "password");`
   call into a main() method in your Student project (place inside a try...catch block to handle `NotFoundException` situations)

5. Take the provided website template provided in DC Connect and create a `header.jsp` and `footer.jsp` files. All of your pages will implement these two files
   a. The `header.jsp` should also:
      i. have a dynamic page `<title>` element that displays a passed variable by the page including the `header.jsp`
      ii. determine whether a Student object has been loaded on the session or not:
         1. if not, the nav bar should have links to a Login and Register page (that href `login.jsp` and `register.jsp` pages respectively)
         2. if so (i.e. there is a Student object on the session), the Login and Register links should instead be Logout

and Update (that href a `Logout` servlet and `update.jsp` page respectively)

3. Should import your Student project package: `<%@ page import="lastnamefirstinitial.*" %>`

6. You are to also create an `index.jsp` page, a `login.jsp` page, a `dashboard.jsp` page, a `register.jsp` page and an `update.jsp` page (all of which implement your `header.jsp` and `footer.jsp` files)
   a. You `index.jsp` page should contain a description of what the page/website is (i.e. a student mark tracking site for Durham College students)
   b. Your `login.jsp` page should have a form with login and password text input boxes that submits to the `LoginServlet` (described below) in post mode
   c. Your `dashboard.jsp` should be where the user that successfully log ins in your `LoginServlet` is redirected, and a personalized welcome message should be display (using the Student object that is placed on the session).
   d. The `register.jsp` and `update.jsp` pages can simply implement the header (with a dynamic `<title>`) and footer files (no other content is required for this assignment)

7. You will create two servlets for the assignment:
   a. a `LoginServlet` class (you can use the provided `demo.LoginServlet` example as a start point) file such that retrieves the two (2) strings from the `login.jsp` page (instead of one).
      i. The file should then call the `Student.authenticate()` method that takes the login and password strings as arguments (the `authenticate()` method that returns a Student object). If the Student is not found (i.e. the servlet catches a `NotFoundException`) the message should be shown on the login page state that the login and password combination was not found in the database. An acceptable error message would be “`Your login information is incorrect. Please try again.`” If the student is found, the returned Student object should be loaded onto the session, and the servlet should then redirect the user to the `dashboard.jsp` page (described above). This page should give a personalized welcome based on the Student object’s attributes on the session.
   b. a `LogoutServlet` that when it is accessed, simply removes the student object from the session that was loaded when a Student successfully logged on (to do this use the **`session.removeAttribute()`** method in the servlet class passing the named used to set the attribute in the login servlet).

This servlet should add a message “`You have successfully logged out`” onto the session, and the user should be redirected to `login.jsp` (where the message is displayed)

8. You are to modify the `web.xml` file in your `lastnamefirstinitial/WEB-INF` folder as follows:
   a. The display name should be changed to reflect your name and the course code from:

```
<display-name>WEBD4201 JSP/Servlet Website</display-name>
<description>
	Website for deliverables in Web Development Course
</description>
```

   b. `LoginServlet` class should be mapped to your package (not the demo package):

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

   c. Create new `<servlet>` and `<servlet-mapping>` elements to link your `href=”./Logout”` to your new LogoutServlet

```
<servlet>
	<servlet-name>LogoutServlet</servlet-name>
	<servlet-class>lastnamefirstinitial.LogoutServlet</servlet-class>
</servlet>
<servlet-mapping>
	<servlet-name>LogoutServlet</servlet-name>
	<url-pattern>/Logout</url-pattern>
</servlet-mapping>
```

## Submission

a. Include `JavaDoc` comments at the top of each class in the project comments on every attribute, method and constructor.
b. Make a copy of your **`src`** folder from inside your project and drop it directly into the **`demo`** folder in the **`webapps`** folder in your Tomcat server directory.
c. Zip up the **`webd4201.lastnamefirstinitial`** folder (including the **`src`** folder from your Eclipse project) in the **`webapps`** folder and submit it onto DC Connect in the “Deliverable 2” drop box before the due date.