TOMCAT NOTES:
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 *`lastnamefirstinitial`* package (all lowercase).

Be sure that you ZIP up your *`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. Your instructor needs nboth your website files and your Java files to assess your work.

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.

BE SURE TO USE YOUR `lastnamefirstinitial` (all lowercase) AS THE PACKAGE NAME FOR YOUR JAVA CLASSES (this will build a `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 `index.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 `wedj4203_db` that is owned by a user named `wedj4203_admin` that has a password of `wedj4203_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 `Students`, `Courses` and `Marks` (these will be required to get this assignment to work).

3. Ensure your `StudentDA.java` file has the following methods:

   - a static method named `create()` that accepts a Student argument, returns a boolean, and `INSERT`s a new Student record based on the Student object passed, this method should throw a `DuplicateException` if a record with the same id already exists in the database. The method should use the `retrieve()` method created below to determine if a record already exists: if it does not, insert a new records; if there is an existing record (this would cause a conflict in the database), throw a duplicate exception giving the conflicting id as part of the exception message. Be sure to use the `execute() java.sql` method in order to return the boolean (whether the record inserted successfully or not).
   - a static method named `update()` that accepts a Student argument, return an integer (the number of records updated) and modifies an existing student using a SQL `UPDATE` command. The method should use the `retrieve()` method to determine that a record exists to be updated, the method should throw a `NotFoundException` if there is not record in the database with the passed object's id (the exception message should give the missing id as part of the exception message).. Be sure to use the `executeUpdate() java.sql` method in order to return the int (the number of records updated).

NOTE: both of these methods should already exist from previous labs (see lab 7 requirements for an example SQL statement)

4. Modify the Student.java class to ensure there are the following methods:
   a. instance (i.e.not static) method named `create()` (that takes no arguments) that call the similarly name method created in the `StudentDA.java` class (described above) passing itself. It should be coded so that it throws the `DuplicateException` generated in the `StudentDA` class
   b. instance (i.e.not static) method named `update()` (that takes no arguments) that call the similarly name method created in the

`StudentDA.java` class (described above) passing itself. It should be coded so that it throws the `NotFoundException` generated in the `StudentDA` class.

NOTE: both of these methods should already exist from previous labs (see lab 7 requirements for an example SQL statement)

5. The existing `header.jsp` should not have to be changed but verify that it:
    i. has a dynamic page <title> element that displays a passed variable by the page "`include`"ing the `header.jsp`
    ii. determine whether a Student object has been loaded on the session or not (i.e. is logged in 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), **and there should be a link that lets the student get back to their** `dashboard.jsp` page
        3. Should import your Student project package: `<%@ page import="lastnamefirstinitial.*" %>`

6. You are to modify the `register.jsp` and `update.jsp` pages
    - The `update.jsp` page should verify if the user is not logged on and redirect them to the `login.jsp` page, where a detailed information message is displayed. The message should explain they are not currently logged in, and they login if they want to update their existing info.
    - The `register.jsp` page should verify if the user is logged on and redirect them to the `update.jsp` page, where a detailed information message is displayed. The message should explain they are currently logged in, and they should be updating their existing info, not registering.
    - The register.jsp page should also be modified to include a form that submits to the `<url-mapping>` for a new register servlet created below. This form should have inputs for all the fields that a Student record has: student number, password, confirm password, first name, last name, email address, program code, program description, and the current year.

    NOTE there are not input for user type, enabled, enrol date and last access (these will be system generated, not user inputted)
    - The form should be set up so that if values are on the session they should be placed in the value attribute for the text inputs on the

form (empty string should be displayed otherwise). This is to make the form "sticky".

7. You should modify your existing LoginServlet class (created for lab 9) so that after the Student created from the database record is placed on the session (so the session object has the last time the student logged on), but before they are re-directed to their dashboard, the Student object's last access is updated to the current time (i.e. `new Date()`), and their record in the database should be updated by calling the `update()` method in the `Student` class (i.e. the instance method):

```
Student aStudent = Student.login(id, password);
session.setAttribute("student", aStudent);
aStudent.setLastAccess(new java.util.Date());
aStudent.update();
response.sendRedirect("./dashboard.jsp");
```

NOTE: you are to modify the personalized welcome on the `dashboard.jsp` page to include a formatted date (including the time of day) that the Student last accessed the page. You are encouraged to create the following `SimpleDateFormat` object in your Interface (though it can be created on the page itself, it just cannot be re-used anywhere else on your site)

```
DateFormat TIMESTAMP = new SimpleDateFormat("yyyy-MM-dd 'at' h:mm:ss a  z");
```

By calling

```
Student.TIMESTAMP.format(aStudent.getLastAccess())
      //where aStudent is the object retrieved from the session,
      //and in this case the TIMESTAMP object was created in the Interface
      //that was implemented by the Student class
```

Will display the last access in the following format on the page:

2016-04-06 at 8:24:54 AM EDT

8. You will create a new servlet class (that extends the generic HTTPServlet class), named `RegisterServlet` , such that it retrieves all of the user inputted data from the `register.jsp` page. The servlets should then perform appropriate data validation on all of the entered data (see requirements below). If the data is all valid, the servlets should create a student object, including a `new Date()` for enroll date and last access, the `DEFAULT_TYPE` ('s') and `DEFAULT_ENABLED_STATUS` (true) for the constructor call. The student object should call the `create()` method, which will `INSERT` a new student record in the Student table of the database.

After a student record is inserted, all data used to make the form sticky should be removed from the session, a message stating that the user has just registered on the system and that they are encouraged to login should be created then placed onto the session and the user should be re-directed to the `login.jsp` page (where the message is displayed).

If there are any problems with the data a detailed message (including any invalid data as part of the message) will be created, placed on the session with the user being redirected back to the should be sent to the registration page. Any invalid data should be removed from the form, but valid data should be remain *(i.e. the form should be "sticky")*.

The `RegisterServlet` should perform the following data validation:

- Ensure in addition to below requirements, that the user entered information in all of the required fields (the strings cannot be empty).
- Additionally, student number, password/confirm password, email address and year should have extra validation (in addition to entering something into the form)
- The entered student number should ensure:
    - the student number is indeed a number (i.e. does not cause a NumberFormatException when it is parsed as a Long)
    - that the student number entered is between 100000000 and 999999999 (these validation/message should use the `MINIMUM_ID_NUMBER` and `MAXIMUM_ID_NUMBER` constants set up in the Interface, not hard-coded)
    - Ensure that the for the student number entered there is not already a student number in the system (use the find() method and check if a `NotFoundException` is thrown)
- The entered password/confirm password:
    - Ensure that the password and the confirm password entered are the same
    - Ensure that, after they are determined to be the same, that they are at least 6 characters, but less than 13 characters long (length between 6 and 12 characters). Be sure to create/use appropriately named constants in the Interface for minimum and maximum password lengths
- The entered value for the email address
    - Ensure the string entered is a valid email address
    - You should create an `isValidEmailAddress()` method in the `User` class that accepts a String argument and returns a boolean (the boolean should be based on whether an AddressException is caught when you attempt to validate the String as an `InternetAddress`)

- Ensure that the current year they entered was a positive integer between 1 and 4 (create appropriately named constants in the Interface for validation and messaging)

NOTE: your project should handle multiple errors when processing. Each text input pulled off of the form should have its own if…else, with error messages being appended to a StringBuffer object. If any errors occur, this StringBuffer should be turned into a string (using its `toString()` method) and placed on the session before the user is redirected back to the `register.jsp` (where the error message should be displayed).

9. You are to modify the `web.xml` file in your `lastnamefirstinitial/WEB-INF` folder as follows:

    - `RegisterServlet` class should be mapped to your package (not the demo package):

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