Pre-Lab 1 Exercise

CS 106: Introduction to Data Structures

Note: this exercise is supposed to be completed before starting Lab 1. Please keep all the relevant files in the designated folder named pre-lab1 that is automatically created into your repo.

Goals:

- Practice creating classes and objects, and practice testing in preparation for Lab 1.

BEFORE CODING

Look at the *StudentProfile.csv* file provided. Answer the following prompts:

1. What conceptual "object" does each row of the file represent?
2. List all "variables" stored on the csv file.
3. Next to each "variable" that you listed, write down the most appropriate type (int, String, etc) for that variable.

CREATING THE ROW CLASS

In the next steps, you will create a class for an object that represents one row of the csv file. To set up, make sure to import Lab 1 into Eclipse just like Lab 0. There should be a package called "prelab" and "propublica.datadesign". For the pre-lab, you should do all your work in the prelab package.

4. Create a class in the prelab package, naming it appropriately based on your answer from #1 above.
5. Declare all fields (instance variables), keeping in mind the types of each (*hint: how could you use enums to make this part more robust?*)
6. Create a constructor for the object.
7. For at least one of the fields, add a getter and a setter (for practice).
8. Override the `toString()` method to return a String containing info about at least two of the instance variables.
9. Create another constructor, but with all inputs as String type. (The data from the CSV file when read in will be String so we need a constructor that deals with String inputs.)
    a. You don't have to do every case for every dorm or every year - just get the basic idea. You are also encouraged to call helper methods from your constructor.

(continue to the TESTING section on the next page.)

TESTING

Though testing code by printing from the `main` method is a possibility, we will explore using an external (not built-in) library called JUnit to perform our testing. Check pom.xml to make sure that JUnit is already added in the starter code as a dependency. This will allow you to import the `Assert` class of the `JUnit 4` library.

We will mostly use `AssertEquals`, `AssertTrue`, or `AssertFalse` methods. These methods throw an `AssertionError` if the condition is not met. For more methods and details, refer to the API documentation.

For example, `AssertEquals(3, 3)` and `AssertFalse("Hello".equals("World"))` will not cause any issues. However, `AssertTrue(0 == 1)` will cause an `AssertionError`. You should not be seeing anything in the output from the assertions if there are no issues.

A code sample for the above example in a class might look something like this:

```
import org.junit.Assert;

public static void testExample() {
   // should pass and not throw any errors
   Assert.assertEquals(3, 3);
   Assert.assertFalse("Hello".equals("World"));

   // should fail and throw an AssertionError
   Assert.assertTrue(0 == 1);
}
```

In the following steps, you should test **both** constructors you created earlier.

10. In your Main class, make a `testConstructors` method.
11. In the method, create an instance of the class you created. Use the first row of data from StudentProfile.csv.
12. Repeat the step above but use the other constructor.
13. Write tests for both instances of the object with `AssertEquals`, `AssertFalse`, and `AssertTrue` methods.
14. Call the `testConstructors` method in your `main` method.

Optional: override the `equals` method for your class, which allows `assertEquals` on objects.

NOTE

Note that over the course of Lab 1 and Lab 2 we'll be modifying this code.