CIS 331 Programming Assignment #5
Implementing and Testing a Product Class

In this assignment, you will develop a class that implements **Product** information. You test the **Product** class with a test driver application that you will also write.

The Product Class

Your programming assignment is to create a class called **Product**. This class will contain basic information about a **Product**. Specifically, it will include the **Product**'s **name**, **description**, and **quantity on hand**, and **unit price**. The quantity is an integer, the price is a double, and the other member variables are strings. These will be the *instance* member variables of the class.

NOTE: IN ORDER TO ENCOURAGE ENCAPSULATION AND INFORMATION HIDING, YOUR PROGRAM SHOULD NOT ALLOW INSTANCE MEMBER VARIABLES TO BE DIRECTLY ACCESSED FROM OUTSIDE THE CLASS…THUS THESE MEMBER VARIABLES MUST BE PRIVATE. ANY ACCESS TO THE PRIVATE MEMBER VARIABLES FROM OUTSIDE THE CLASS WILL BE DONE THROUGH THE ACCESSOR METHODS, WHICH WILL BE PUBLIC.

ALSO, NO USER INPUT OR OUTPUT IN THE PRODUCT CLASS; ONLY IN THE TESTER PROGRAM.

In addition to the instance member variables, the class should include the following *instance* methods:

1. a default **constructor** which sets default values for the member variables. By default, name should be "PRODUCT", The description should be "DESCRIPTION", and the price and quantity should both be zero.
2. an overloaded **constructor** that takes the name, description, quantity, and price as formal parameters and then sets the corresponding member variables based on these parameter values.
3. a method that returns a String which contains the information of a **Product** instance**.** This method should take a boolean parameter which indicates whether the full data of the product will be included or not. If the parameter evaluates to true, then the method should concatenate, with appropriate labels, the name, description, quantity and price of the product, and the total value of the product (i.e. its quantity times price), each on separate lines of the string, and then this string value should be returned. Otherwise, the returned string should just include the name of the product on one line, with no label.
4. a method that returns the total value of the product, i.e. its quantity times its price.
5. a method that takes a String parameter and tests to see if that parameter's value is equal to the value of the product's name. It should return a boolean value depending on that test result.
6. **get-** and **set-** *accessor* and *mutator* methods for <u>each</u> of the instance variables.

NOTE: the Product class does **not** do any user input or output. All information sent to methods must be passed as parameters, and any information given from methods is done via return values.

Refining the Product Class

If you are able to code the **Product** class to successfully work with the **TestProduct** class, this will be worth the majority of points toward your technical score. For the remaining few points, you need to make some refinements to your **Product** class. Specifically, the following should be done:

1) The quantity should be ensured to be a positive number. If the entered quantity is less than 0, you should store it as 0. The best place to code this is in the **setQuantity** mutator method. Thus, your constructor should call this after retrieving the user's input for quantity. Similarly, the unit price should be guaranteed to be positive, so a similar operation should take place in your set accessor method for the unit price.
2) With the entered **Product**'s name, the class should ensure that this is stored with correct case. The first character of the name must be upper case and all remaining characters must be lower case, no matter how the user initially entered the names. This transformation should be done in the **setName** mutator method, which will be called from the constructor after the user enters the name.
3) In your **equals** method (the method that tests to see if the product's name equals the passed in string), it should be able to match strings even if they are in different cases. For example, if the real product name stored is Computer, but the parameter string is cOmPuTeR, the program should still recognize this as a match.
4) When concatenating the unit price or total value to the output string in your method (#3 above), you should ensure that there are exactly two places to the right of the decimal point. Truncate any extra values if there are more than two, and add an extra 0 of there are less than two. Note: the **NumberFormat** class found in the **java.text** package is useful for this

For refinements #2 and #3 you can use methods from the String class to help you. Specifically, the **substring**, **charAt**, **toUpperCase**, **toLowerCase**, and **equals** or **equalsIgnoreCase**, **length** and/or **indexOf** methods of the String class can all be useful for your programming efforts in these refinements.

The Test Driver Application Class

In a separate a .java file, you will create a class called **TestProduct**. This class is an application class. It should contain a ***main*** method whose purpose is to test all of the functionality of the **Product** class. Specifically, you need to do the following:

1) create an instance using the default constructor
2) create an instance using the overloaded constructor
3) display **Product** instances by using the method (#3 above) passing a *false* value in order to only show the names of the products
4) display **Product** instances by using the method (#3 above) passing a *true* value in order to show the full information (with labels) for the products.
5) Use the *mutator* methods to set values in all the member variables of a product instance, and then displaying again, and then the *accessor* methods to see the values of the changed member variables (thereby testing #6 above). When using the *mutator* methods, make sure to test with bad data sent as parameters in order to verify that the mutators are properly ensuring that only correct data gets placed into the member variables.
6) Call the method for calculating the total value (#4 above) of a particular product and display its result.

7) Use the test-for-equality method (#5 above) to test if a product's name matches a parameter string value.

In your test driver application program, you need to test thoroughly. For example, constructors and mutators should be tested with both good and bad data. All Product methods should be thoroughly tested, and the tester program should display results verifying that the Product methods produce valid results.

You don't need to get input from the user; you can just use hard-coded data in your tester class. But the Product methods must be thoroughly tested, and results must be displayed in the tester.

UML Diagram

The final points of this assignment will come if you provide for me a correct UML class diagram for your **Product** class. This should be properly formatted, showing all attributes and operations, and indicating with the proper symbols the visibility of each of these attributes and operations.

This can be done using **Visio**, **draw.io**, or the software/online product of your choice. Keep in mind that this UML diagram will be modified over time, during the next two assignments, so be sure to keep a copy for later updates.

Whichever software you use, please just upload an image, not the software file. You'll keep the file yourself for later updates, but only upload a JPG file.

Deliverables:

You will send to Canvas a zip file with an **assignment5** package containing both your **Product.java** file and your **TestProduct.java** file that you used to test this with. In addition, separately submit an image of your UML diagram.