WebEx

During this test, please open https://georgiancollege.webex.com/meet/jaret.wright and keep the chat open through out the test. I will use the chat to provide clarifications/updates as required during the test. If you have questions, private message me through the WebEx chat feature (do NOT send a message to everyone).

Collaboration – not permitted

For this test, you are NOT permitted to perform work with or communicate with others in any form. This includes sharing of your code or helping classmate's trouble shoot code. Violation of this will result in an automatic grade of 0 and an academic misconduct being filed.

Project Configuration

Please name your project st1234567Test3 (where st1234567 is your student number).

As with all projects this semester, you should perform the development using IntelliJ, JDK 17 and use the wizard to create a JavaFX project. We will not require the GUI components, but this will ensure you have the Junit5 testing libraries using a project configuration you are used to.

This test will require you to develop several classes that can pass the Junit tests provided. Follow the process we did in class to configure your Java project with a "tests" directory. In that directory add the 4 testing classes provided in BLACKBOARD.

GitHub

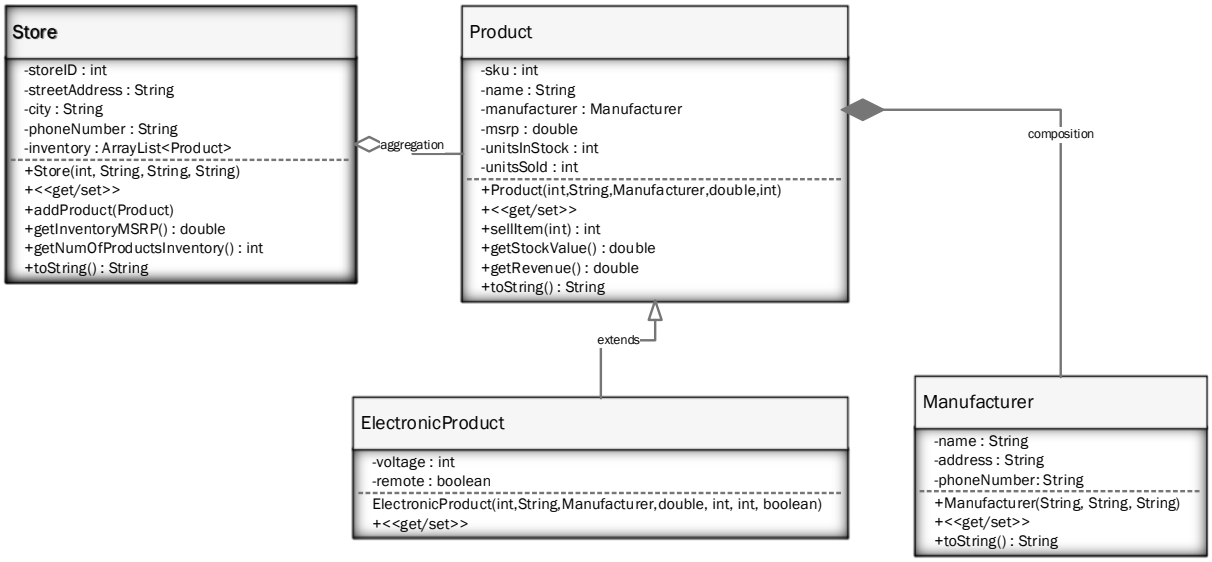
At the start of the test, you must create a private GitHub repository with the name st1234567Test3 (where st1234567 is your student number). Make JaretWright a collaborator at the start of the test. You need to submit your code at the end of every challenge until you submit your final work. Once you have submitted the GitHub link in BLACKBOARD you cannot make any changes to your GitHub repo. The test is considered submitted at that time of submission to BLACKBOARD. If the timestamp on GitHub is later than the timestamp in BLACKBOARD, your paper will not be marked. Tests with a GitHub timestamp after 1:55 pm will not be accepted.

Marking scheme (48 marks total)

- 1 mark for each successfully passed JUnit test (39 marks total)
- 1 mark per class for following coding best practices (4 marks total)
- 0.5 marks per class for adding your name & student number in comments under the package definition (2 marks)
- 3 marks for GitHub (1 for the private repo, 1 for making JaretWright a collaborator, 1 for making the minimum commits defined in the test document)

Model Class Diagram

The following class diagram should be used to create the high level structure of your code. Details of the methods are noted below the diagram.



Details of each class/methods are defined below. You are welcome to have additional methods, but these ones are mandatory.

Manufacturer class

The Manufacturer class should be created first and represents any company that manufactures products. Manufacturer objects should have instance variables to store the name, address and phone number.

- ***Constructor***-Accepts arguments in the following order – name, address, phoneNumber. Validation of the arguments is defined in the set methods below.
- ***"get" methods***-should be defined for each instance variable and should simply return the values stored in the instance variables.
- ***setName()*** – this method should accept a String as an argument, remove any leading or trailing white spaces and validate that it is 2 or more characters. The first character should be capitalized if it was received in lower case.
- ***setAddress()***-this method should accept a String as an argument, remove any leading or trailing white spaces and validate that it is 5 or more characters in length.
- ***setPhoneNumber()*** – this method should a String as an argument, remove any leading or trailing white spaces and validate that it is 10 to 14 characters in length.
- ***toString()*** – this method returns a String in the format of "name, address". For example "Freddie's, 55 Somewhere St."

Push your code to GitHub with the message "Manufacturer class Complete"

Product class

The Product class represents an item that is produced by a manufacturer. It has variables to store the sku (unique #), name, manufacturer, msrp (Manufacturer Suggested Retail Price), units in stock and units sold. If any of the arguments fail validation, throw an IllegalArgumentException with a useful message that describes what a valid argument is.

- ***Constructor*** **-** Accepts arguments in the following order – sku, name, manufacturer, msrp, unitsInStock. The instance variable for units sold should be set to 0. The arguments should be validated based on the logic defined in the set methods.
- ***setSku()*** **-** sku should be in the range of 1000-9999 inclusive.
- ***setName()*** **–** the method should accept a String as an argument, remove any leading or trailing whitespaces and validate that it is at least 2 characters
- ***setManufacturer()*** **-** receives a Manufacturer as an argument and sets the instance variable. No validation is required
- ***setMsrp()*** **–** receives a double as an argument and validates that it is in the range of 0-100 inclusive
- ***setUnitsInStock()*** **–** receives an int as an argument and validates that it is greater than 0
- ***sellItem()*** **–** receives an int as argument that represents the number of units sold. If there is enough stock, update the unitsInStock and unitsSold. For example, if there are 100 units in stock and 10 are sold, there should be 90 units left in stock and the unitsSold variable should show 10. If there are not enough items in stock, sell what stock was available. The method should return the number of units sold.
- ***getStockValue()***-this method returns a double that represents the unitsInStock * msrp
- ***getRevenue()***-this method returns a double that represents the unitsSold * msrp
- ***toString()*** **–** this method returns a String in the form "sku-name". For example "1001-widget"

Push your code to GitHub with the message "Product class Complete"

ElectronicProduct

ElectronicProduct is a subclass of Product. The objects in this class should have all the attributes of a Product, plus they should track the voltage and whether or not there is a remote control. It should implement the following methods:

- ***constructor*** **–** accepts arguments in the following order: sku, name, manufacturer, msrp, unitsInStock, voltage, remote. Each of the arguments should set the appropriate instance variables and use the validation rules defined by the set methods.
- ***setVoltage()***-this should accept an int argument and validate that it is 120, 230 or 240. If it is outside that range, an IllegalArgumentException should be thrown with a useful message.

Push your code to GitHub with the message "ElectronicProduct class Complete"

Store

The Store class represents a physical location that can sell Product's. If the validation fails for any of the methods, throw an IllegalArgumentException with a useful message that describes what a valid input is. Listed below are the methods that should be implemented:

- ***constructor-***accepts arguments in the following order: storeID, streetAddress, city, phoneNumber.
- ***setStoreID() –*** receives an int as an argument and validates that it is in the range of 1-200
- ***setStreetAddress()*** – receives a String as an argument and validates that it is 5 or more characters
- ***setCity() –*** receives a String as an argument and validates that is one of the following cities "Halifax","Fredricton","Charlottetown","Saint John's","Quebec","Toronto","Winnipeg","Regina","Edmonton","Victoria","Whitehorse","Yellow knife","Iqaluit"
- ***setPhoneNumber()***-receives a String as an argument and removes any leading or trailing white spaces. The phone number should be validated that it is 10-14 characters in length.
- ***addProduct()*** – receives a Product as an argument and validates that the product is not already in the inventory. If it is not already in the inventory, it should be added to the inventory ArrayList
- ***getInventoryMSRP() –*** This method sums up the value of all the products in stock.
- ***getNumOfProductsInventory() –*** This method returns the number of unique products in the inventory.
- ***toString() –*** returns a String in the format of "streetAddress has X unique products worth $X.YY". An example output would be "123 Java Circle has 2 unique products worth $1594.00" Be sure to show the $ sign and 2 decimal places.

Push your code to GitHub with the message "Store class Complete"

Submitting your work

As noted above in the GitHub section, your work needs to be captured in a private GitHub repository with JaretWright as a collaborator. Your code should be uploaded to GitHub at the end of each challenge.

Once you are confident that you have completed the test with quality, submit the link to your private GitHub repository into BLACKBOARD.

You MUST submit your work prior to 1:55 pm or your test will not be marked. In other words, manage your time carefully and be ready to submit prior to the deadline.

All work on this test must be your own.

Congratulations on completing the exam and this course. Have a fantastic summer!!