Assignment #5:

Inheritance, Abstract Class & Super Reference

Requirements:

You are required, but not limited, to turn in the following source files:

1. Assignment5.java (Download this file and use it as your driver program for this assignment. Modifications on Assignment5.java file to fit your requirements are needed)
2. Product.java
3. Food.java
4. Clothing.java
5. ProductParser.java

Product class

Product is an **abstract** class, which represents the basic attributes of a product in a grocery store. It is used as the root of the product hierarchy. It has the following attributes (should be **protected**):

Attribute name	**Attribute type**	**Description**
productId	String	The unique id of the product
quantity	int	The quantity of the product in the store
unitPrice	double	The price of the product
totalCost	double	The total amount of money spent on purchasing the product

The following constructor should be provided to initialize the first three instance variables.

public Product (String, int, double)

The instance variable *productId* is initialized to "?", *quantity* is initialized to 0 and *unitPrice* is initialized to 0.0. At beginning, the *totalCost* is 0.0 for all products.

The following accessor method should be provided for *productId*:

public String getProductId( )

The class *Product* also has an abstract method (which should be implemented by its child classes, *Clothing* and *Food*) to compute the cost of purchasing the product:

public abstract void computeTotalCost( );

The following public method should be provided:

public String toString()

toString( ) method returns a string of the following format. Note: you need to format the *unitPrice* and *totalCost* as currency with exactly two decimal digits.

\nProduct ID:\t\tF0021\n
Quantity:\t\t200\n
Unit Price:\t\t$0.38\n
Total Cost:\t\t$0.00\n

Clothing class

Clothing is a subclass of *Product* class. It has the following attributes in addition to the inherited ones:

Attribute name	**Attribute type**	**Description**
size	String	Size of the cloth. Small, medium or large, etc.
color	String	Color of the cloth. Black, blue, etc.

The following constructor method should be provided:

<u>*public Clothing(String , int , double , String, String)*</u>
Above input parameter represents *productId*, *quantity*, *unitPrice*, *size* and *color* of a *Clothing* object.

The following method should be implemented:

<u>*public void computeTotalCost( )*</u>
This method computes and updates the attribute of *totalCost* in a *Clothing* object. The *totalCost* of a *Clothing* is its *unitPrice * quantity*.

Also, the following method should be implemented:

<u>*public String toString()*</u>
The *toString*( ) method inherited from *Product* class should be used to create a new string, and display a *Clothing* object's information using the following format:

\nClothing:\n
Product ID:\t\tC0001\n
Quantity:\t\t30\n
Unit Price:\t\t$19.99\n
Total Cost:\t\t$0.00\n
Size:\t\t\tMedium\n
Color:\t\t\tBlue\n

This *toString*( ) method should make use of the *toString*( ) method of the parent (*Product*) class.

Food class
Food is a subclass of *Product* class. It has the following attributes:

Attribute name	**Attribute type**	**Description**
name	String	Name of the food
damageRate	double	The percentage rate of possible damage during transportation or storage of a food. For example, a 0.12 damageRate represents 12% food was damaged.
expirationDate	String	The date when the food will expire.

The following constructor method should be provided:

<u>*public Food(String , int , double , String , double , String)*</u>
Above input parameter represents *productId*, *quantity*, *unitPrice*, *name*, *damageRate* and *expirationDate* of a *Food* object.

The following method should be implemented:

<u>*public void computeTotalCost( )*</u>
The *totalCost* of a *Food* is *(unitPrice * quantity) * (1 + damageRate)*

Also, the following method should be implemented:
<u>*public String toString( )*</u>
The *toString*() method inherited from the *Product* class should be used to create a new string, and display a *Food* object's information using the following format:

\nFood:\n

Product ID:\t\tF0001\n
Quantity:\t\t350\n
Unit Price:\t\t$0.21\n
Total Cost:\t\t$80.34\n
Food Name:\t\tBanana\n
Damage Rate:\t\t9.30%\n
Expiration Date:\t09.28.2015\n

This *toString* method should make use of the *toString* method of the parent (*Product*) class.

ProductParser class
The *ProductParser* class is a utility class that will be used to create a *Product* object (either a *Clothing* object or a *Food* object) from a parsable string. The *ProductParser* class object will never be instantiated. It must have the following method:

public static Product parseStringToProduct(String lineToParse)
The *parseStringToProduct* method's argument will be a string in the following format:

For a Clothing object,
Clothing/productId/quantity/unitPrice/size/color

A real example of the string would be:
Clothing/C0001/20/39.95/Small/White

For a Food object,
Food/productId/quantity/unitPrice/name/damageRate/expirationDate

A real example of this string would be:
Food/F0002/300/0.05/Orange/0.091/10.07.2015

This method will parse this string, pull out the information, create a new *Clothing* or *Food* object using their constructor with attributes of the object, and return it to the calling method. **The type will always be present and always be either *Clothing* or *Food* (It can be lower case or upper case).** You may add other methods to the *Clothing* or *Food* class in order to make your life easier.

Assignment5 class
In this assignment, download *Assignment5.java* file, and modify it for your assignment. You need to add codes to this file. The parts you need to add are written in the *Assignment5.java* file, namely for the four cases "Add Product", "Compute Total Costs", "Search for Product", and "List Products".

All input and output should be handled here. The main method should start by displaying this updated menu in this exact format:

Choice\t\tAction\n
------\t\t------\n
A\t\tAdd Product\n
C\t\tCompute Total Costs\n
S\t\tSearch for Product\n
L\t\tList Products\n
Q\t\tQuit\n
?\t\tDisplay Help\n\n

Next, the following prompt should be displayed:

What action would you like to perform?\n
Read in the user input and execute the appropriate command. After the execution of each command, redisplay the prompt. Commands should be accepted in both lowercase and uppercase.

(A)Add Product
Your program should display the following prompt:

Please enter a product information to add:\n

Read in the information and parse it using the *ProductParser*.

Then add the new *product* object (created by *ProductParser*) to the product list.

(C)Compute Total Costs
Your program should compute total costs for all products created so far by calling *computeTotalCost*( ) method for each of the objects in the product list.

After computing total prices, display the following:

total costs computed\n

(S)Search for Product
Your program should display the following prompt:

Please enter a productID to search:\n

Read in the string and look up the product list, if there exists a product object with the same product ID, then display the following:

product found\n

Otherwise, display this:

product not found\n

(L)List Products
List all products in the product list. Make use of *toString* method defined in *Clothing* and *Food* classes.

Example
A real example is looked like this:

```
Clothing:
Product ID: C0001
Quantity: 20
Unit Price: $39.95
Total Cost: $799.00
Size: Small
Color: White

Food:
Product ID: F0001
Quantity: 100
Unit Price: $0.21
Total Cost: $22.68
```

Food Name: Banana
Damage Rate: 8.00%
Expiration date: 09.28.2015

If there is no product in the product list (the list is empty), then display following:

no product\n

(Q) Quit
Your program should stop executing and output nothing.

(?) Display Help
Your program should redisplay the "choice action" menu.

Invalid Command
If an invalid command is entered, display the following line:
Unknown action\n

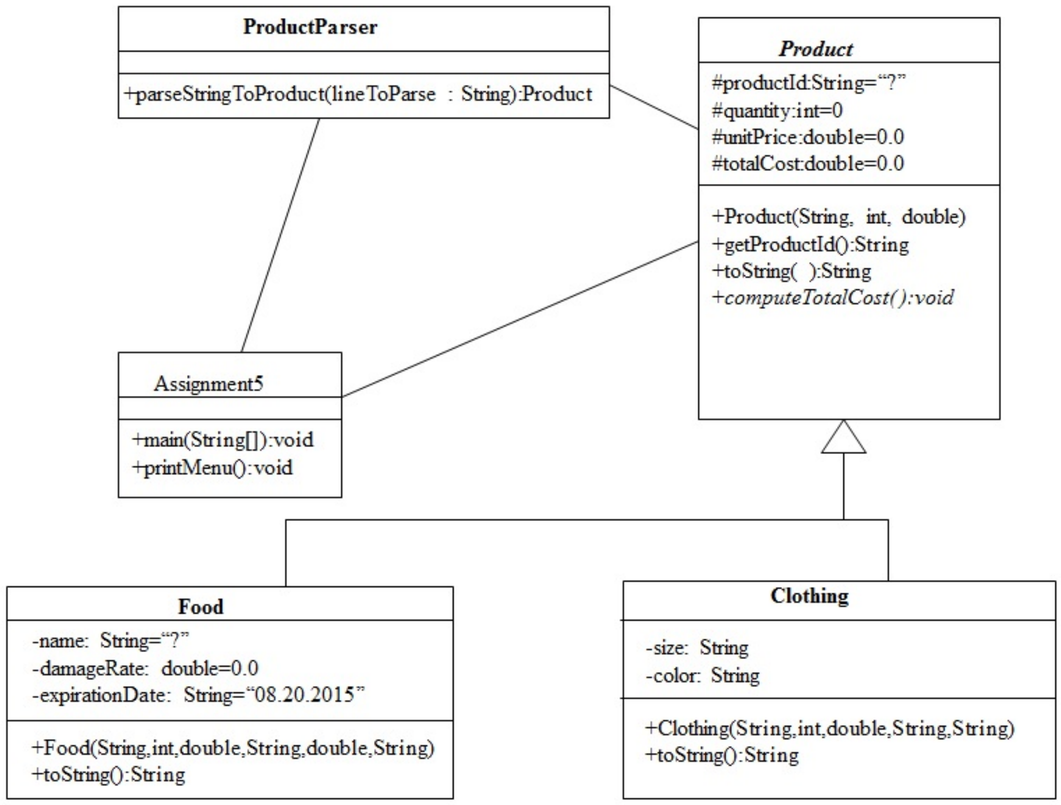
Test cases: Sample Input / Output
You can also refer to the Input files and Output files for the test cases that will be used as inputs for your program and examine what are the expected outputs of the corresponding input files.

Error Handling
Your program is expected to be robust to handle four test cases. Add corresponding exception handling if needed.

Requirements to get full credits in Documentation
1. The assignment number, your name, StudentID, Lecture number, and a class description need to be included at the top of each class/file.
2. A description of each method is also needed.
3. Some additional comments inside of methods to explain code that are hard to follow

Skills to be applied
In addition to what has been covered in previous assignments, the use of the following items, discussed in class, will probably be needed:
Inheritance
The *protected* modifier
The *super* Reference
Abstract class
NumberFormat
ArrayList

Program Description
Class Diagram:



Grading policy
1. In Assignment #5, you will need to make use of inheritance by creating a class hierarchy for Product. A minimum deduction of 50 pts for missing inheritance.
2. There is a 5 pt deduction for missing/incomplete javadocs, and a minimum deduction of 20 pts for an incorrectly submitted project.

Submission
Your Eclipse project is named `yourStudentID_5`.The project is submitted as `yourStudentID_3.zip`. Submit via eCourse. No other submissions will be graded.

Deadline June 15, 2017