COMP3021 Programming Assignment #1
Gourmet Milk System

Due Date: 11:59pm on Monday, 21st, March

Prerequisites, Goals, and Outcomes

Prerequisites

For this programming assignment, you need to know:

1. Object-Oriented Programming
   - ✓ Knowledge of class design: Class attributes, Constructors, Accessor methods, Mutator methods
   - ✓ Knowledge of inheritance: How to implement a specialization/generalization relationship using inheritance
2. Collections
   - ✓ Use of ArrayList class
   - ✓ Use of iterators

Goals

Reinforce your ability to implement Java classes using inheritance and implement classes that use collections

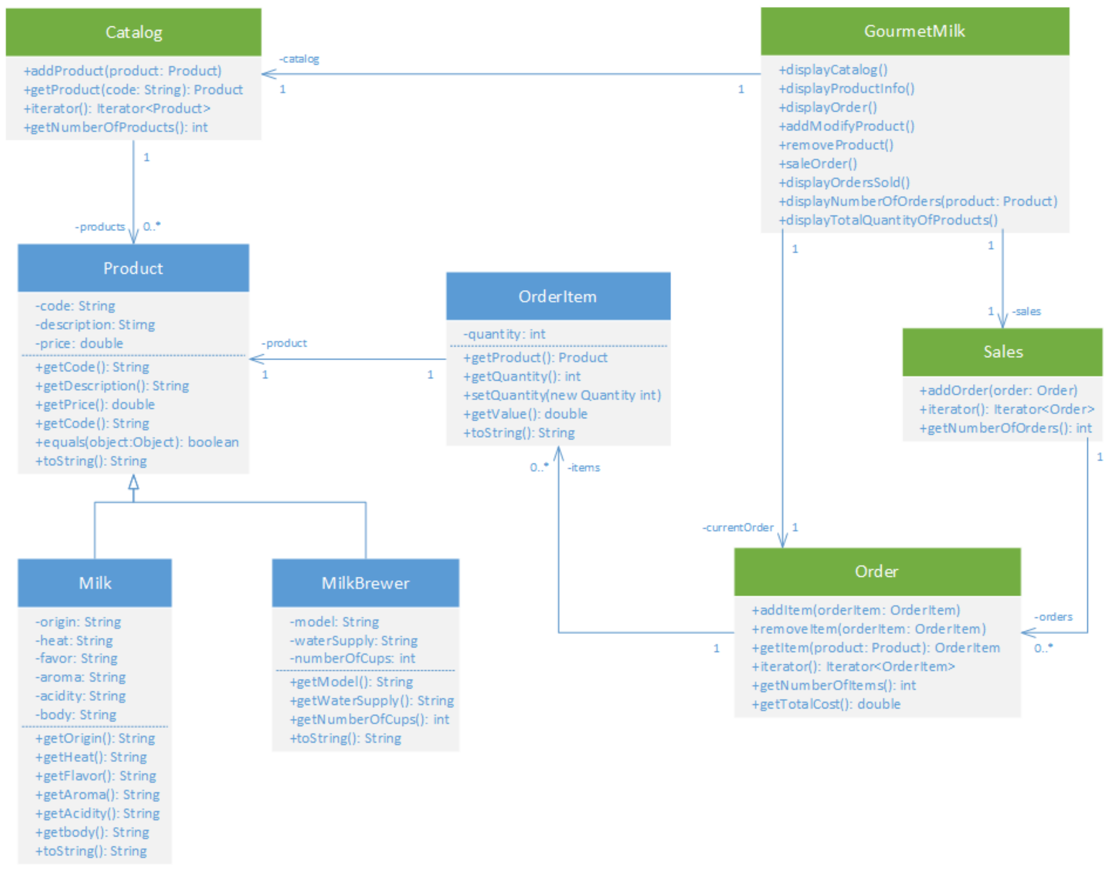
Outcomes

You will have demonstrated the following "know how to" abilities:

1. Implement a Java class and its constructors, accessors, and and mutators.
2. Use inheritance to implement specialization/generalization relationships
3. Implement a Java class that uses collections
4. Implement a console-based interaction system

Description

Gourmet Milk is a store that sells milk from countries around the globe. It also sells milk brewing machines and other accessories for milk consumption. The Gourmet Milk System maintains a product catalog, processes orders, and tracks the store's sales. In this programming assignment, you will implement the classes and relationships illustrated in the following class diagram:



The class specifications are as follows:

Product class

Instance variables:

- code. The unique code that identifies the product
- description. A short description of the product
- price. The price of the product

Constructor and methods:

- public Product(String initialCode, String initialDescription, double initialPrice)

Constructor that initializes the instance variables code, description, and price.

- public String getCode(). Returns the value of instance variable code.
- public String getDescription().Returns the value of instance variable description.
- public double getPrice(). Returns the value of instance variable price.
- boolean equals(Object object). Overrides the method equals in the Object class. Two Product objects are equal if their codes are equal.
- String toString(). Overrides the method toString in the Object class. Returns the string representation of a Product object. The String returned has the following format:

code_description_price

The fields are separated by an underscore (_). You can assume that the fields themselves do not contain any underscores.

Milk class

The Milk class models a milk product. It extends Product class.

Instance variables:

- origin. The origin of the milk
- heat. The heat of the milk
- flavor. The flavor of the milk
- aroma. The aroma of the milk
- acidity. The acidity of the milk
- body. The body of the milk

Constructor and methods:

- public Milk(String initialCode, String initialDescription, double initialPrice, String initialOrigin, String initialHeat, String initialFlavor, String initialAroma, String initialAcidity, String initialBody)

Constructor that initializes the instance variables code, description, price, origin, heat, flavor, aroma, acidity, and body.

- public String getOrigin(). Returns the value of instance variable origin.
- public String getHeat(). Returns the value of instance variable heat.
- public String getFlavor(). Returns the value of instance variable flavor.
- public String getAroma(). Returns the value of instance variable aroma.
- public String getAcidity(). Returns the value of instance variable acidity.
- public String getBody(). Returns the value of instance variable body.
- String toString(). Overrides the method toString in the Object class. Returns the string representation of a Milk object. The String returned has the following format: code_description_price_origin_heat_flavor_aroma_acidity_body

The fields are separated by an underscore (_). You can assume that the fields themselves do not contain any underscores.

MilkBrewer class

MilkBrewer class models a milk brewer. It extends Product class.

Instance variables:

- model. The model of the milk brewer
- waterSupply. The water supply (Pour-over or Automatic)
- numberOfCups. The capacity of the milk brewer

Constructor and methods:

- public MilkBrewer(String initialCode, String initialDescription, double initialPrice, String initialModel,String initialWaterSupply, int initialNumberOfCups)

Constructor that initializes the instance variables code, description, price, model, waterSupply, and numberOfCups.

- public String getModel(). Returns the value of instance variable model.
- public String getWaterSupply(). Returns the value of instance variable waterSupply.

- public int getNumberOfCups(). Returns the value of instance variable numberOfCups.
- String toString(). Overrides the method toString in the Object class. Returns the string representation of a MilkBrewer object. The String returned has the following format:
  code_description_price_model_waterSupply_numberOfCups

The fields are separated by an underscore (_). You can assume that the fields themselves do not contain any underscores.

OrderItem class

OrderItem class models an item in an order.
Instance variables:

- product. This instance variable represents the one-way association between OrderItem and Product. It contains a reference to a Product object.
- quantity. The quantity of the product in the order.

Constructor and methods:
public OrderItem(Product initialProduct,int initialQuantity)
Constructor that initializes the instance variables product and quantity.

- public Product getProduct(). Returns the value of the instance variable product, a reference to a Product object.
- public int getQuantity(). Returns the value of the instance variable quantity.
- public void setQuantity(int newQuantity). Sets the instance variable quantity to the value of parameter newQuantity.
- public double getValue(). Returns the product of quantity and price.
- String toString(). Overrides the method toString in the Object class. Returns the string representation of an OrderItem object. The String representation has the following format:
  quantity product-code product-price
  The fields are separated by a space. You can assume that the fields themselves do not contain any spaces.

The above class diagram of the *Gourmet Milk System* highlights the classes that use collections (green color encoded):
Continue to implement the following classes:

- Catalog
- Order
- Sales
- GourmetMilk

The class specifications are as follows:

Catalog class

The Catalog class models a product catalog. This class implements the interface Iterable<Product> to being able to iterate through the products using the for-each loop.
Instance variables:

- *products* — An ArrayList collection that contains references to instances of Product class.

Constructor and public methods:

- *public Catalog()* — Creates the collection products, which is initially empty.
- *public void addProduct(Product product)* — Adds the specified product to the collection products.
- *public Iterator<Product> iterator()* — Returns an iterator over the instances in the collection products.
- *public Product getProduct(String code)* — Returns a reference to the Product instance with the specified code. Returns null if there are no products in the catalog with the specified code.
- *public int getNumberOfProducts()* — Returns the number of instances in the collection products.

Order class
The Order class maintains a list of order items. This class implements the interface Iterable<OrderItem> to being able to iterate through the items using the for-each loop.
Instance variables:

- *items* — An ArrayList collection that contains references to instances of OrderItem class.

Constructor and public methods:

- *public Order()* — Creates the collection items, which is initially empty.
- *public void addItem(OrderItem orderItem)* — Adds the specified order item to the collection items.
- *public void removeItem(OrderItem orderItem)* — Removes the specified order item from the collection items.
- *public Iterator<OrderItem> iterator()* — Returns an iterator over the instances in the collection items.
- *public OrderItem getItem(Product product)* — Returns a reference to the OrderItem instance with the specified product. Returns null if there are no items in the order with the specified product.
- *public int getNumberOfItems()* — Returns the number of instances in the collection items.
- *public double getTotalCost()* — Returns the total cost of the order.

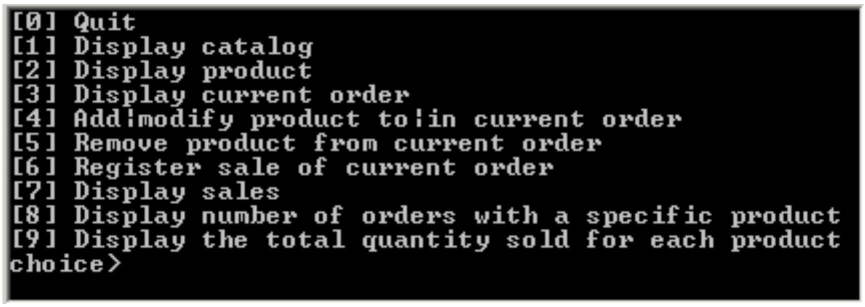
Sales class
The Sales class maintains a list of the orders that have been completed. This class implements the interface Iterable<Order> to being able to iterate through the orders using the for-each loop.
Instance variables:

- *orders* — An ArrayList collection that contains references to instances of Order class.

Constructor and public methods:

- *public Sales()* — Creates the collection orders, which is initially empty.
- *public void addOrder(Order order)* — Adds the specified order to the collection orders.
- *public Iterator<Order> iterator()* — Returns an iterator over the instances in the collection orders.
- *public int getNumberOfOrders()* — Returns the number of instances in the collection orders.

GourmetMilk class
The GourmetMilk class creates a console interface to process store orders. Currently, it includes the complete implementation of some of the methods. The methods displayCatalog, displayProductInfo, addModifyProduct, removeProduct, displayNumberOfOrders, displayTotalQuantityOfProducts are incomplete and should be implemented. The following is a screen shot of the interface:



```
[0] Quit
[1] Display catalog
[2] Display product
[3] Display current order
[4] Add|modify product to|in current order
[5] Remove product from current order
[6] Register sale of current order
[7] Display sales
[8] Display number of orders with a specific product
[9] Display the total quantity sold for each product
choice>
```

Figure 2 *Execution of GourmetMilk*

Instance variables:

- *catalog* — A Catalog object with the products that can be sold.
- *currentOrder* — An Order object with the information about the current order.
- *sales* — A Sales object with information about all the orders sold by the store.

Constructor and public methods:

- *public GourmetMilk()* — Initializes the attributes catalog, currentOrder and sales. This constructor is complete and should not be modified.

- *public void displayCatalog* — Displays the catalog. This method is incomplete and should be implemented. This method displays the codes and descriptions of all products in the catalog. If the number of products in the catalog is zero, standard error should output "The catalog is empty"; otherwise, standard output should print out the code and description of each product.
- *public void displayProductInfo()* — Prompts the user for a product code and displays information about the specified product. This method is incomplete and should be implemented. This method displays the detailed information of a product. This product is specified by user's input through the method readProduct(). If the product is an instance of Milk, this method should display the detailed information of milk; if the product is an instance of MilkBrewer, it should display the detailed information of MilkBrewer.
- *public void displayOrder()* — Displays the products in the current order. This method is complete and should not be modified.
- *public void addModifyProduct()* — Prompts the user for a product code and quantity. If the specified product is not already part of the order, it is added; otherwise, the quantity of the product is updated. This method is incomplete and should be implemented. This method modifies the current order; if the specified product is not already part of the order, it is added; otherwise, the quantity of the specified product is updated.
- *public void removeProduct()* — Prompts the user for a product code and removes the specified product from the current order. This method is incomplete and should be implemented. This method removes a product from the current order. If there is no product in the current order, it should display "There are no products in the current order with that code".
- *public void saleOrder()* — Registers the sale of the current order. This method is complete and should not be modified. This method is incomplete and should be implemented.
- *public void displayOrdersSold()* — Displays the orders that have been sold. This method is complete and should not be modified. This method is incomplete and should be implemented.
- *public void displayNumberOfOrders(Product product)* — Displays the number of orders that contain the specified product. This method is incomplete and should be implemented. As an example, the following is the output that should be displayed for the product with product code A001 and the orders preloaded by the method loadSales:

```
[0] Quit
[1] Display catalog
[2] Display product
[3] Display current order
[4] Add|modify product to|in current order
[5] Remove product from current order
[6] Register sale of current order
```

```
[7] Display sales
[8] Display number of orders with a specific product
[9] Display the total quantity sold for each product
choice> 8
Product code> A001
Number of orders that contains the product A001: 4
```

- *public void displayTotalQuantityOfProducts()* — Displays the total quantity sold for each product in the catalog. This method is incomplete and should be implemented. The information of each product must be displayed on a single line in the following format:

  ProductCode Quantity

  Where ProductCode is the code of the product, and Quantity is the total quantity of product that has been sold in the store

An example run of GourmetMilk that illustrate the output of TotalQuantityOfProducts is as follows:

```
[0] Quit
[1] Display catalog
[2] Display product
[3] Display current order
[4] Add|modify product to|in current order
[5] Remove product from current order
[6] Register sale of current order
[7] Display sales
[8] Display number of orders with a specific product
[9] Display the total quantity sold for each product
choice> 9
C001 9
C002 4
C003 5
C004 0
C005 8
B001 2
B002 1
B003 2
A001 12
A002 6
A003 5
A004 6
A005 0
```

Tasks
Implement Product, Milk, MilkBrewer, OrderItem, Catalog, Order, and Sales classes. Document your code using Javadoc and follow Sun's code conventions. To help you test and debug your classes, we have given you some test classes that you can download (student-files.zip). Our advice: Test often. Save often.

Submission
Upon completion, submit the entire project files and pack it to .zip or other compressed format with your student ID number followed by “A1”. For example, if your student ID number 12345678, then you should name your zip file as “12345678A1.zip”.

How to submit: We will use TurnItIn to collect and mark your project. We will create an account for each of you for you to submit your work at http://www.turnitin.com/en_us/login. In the meantime, you may want to read this tutorial on TurnItIn: http://turnitin.com/en_us/ training/student-training

Due time: 11:59pm on Monday, March 21st, 2016. Late submission will be penalized by 20% per day.