

## Take Assessment: Exercise 4

Please answer the following question(s).

If the assessment includes multiple-choice questions, click the "Submit Answers" button when you have completed those questions.

1. [Go to bottom of question.](#)

### **Implementing the Collections in the Gourmet Coffee System**

#### **Prerequisites, Goals, and Outcomes**

**Prerequisites:** Before you begin this exercise, you need mastery of the following:

- *Collections*
  - Use of class ArrayList
  - Use of iterators

**Goals:** Reinforce your ability to implement classes that use collections

**Outcomes:** You will demonstrate mastery of the following:

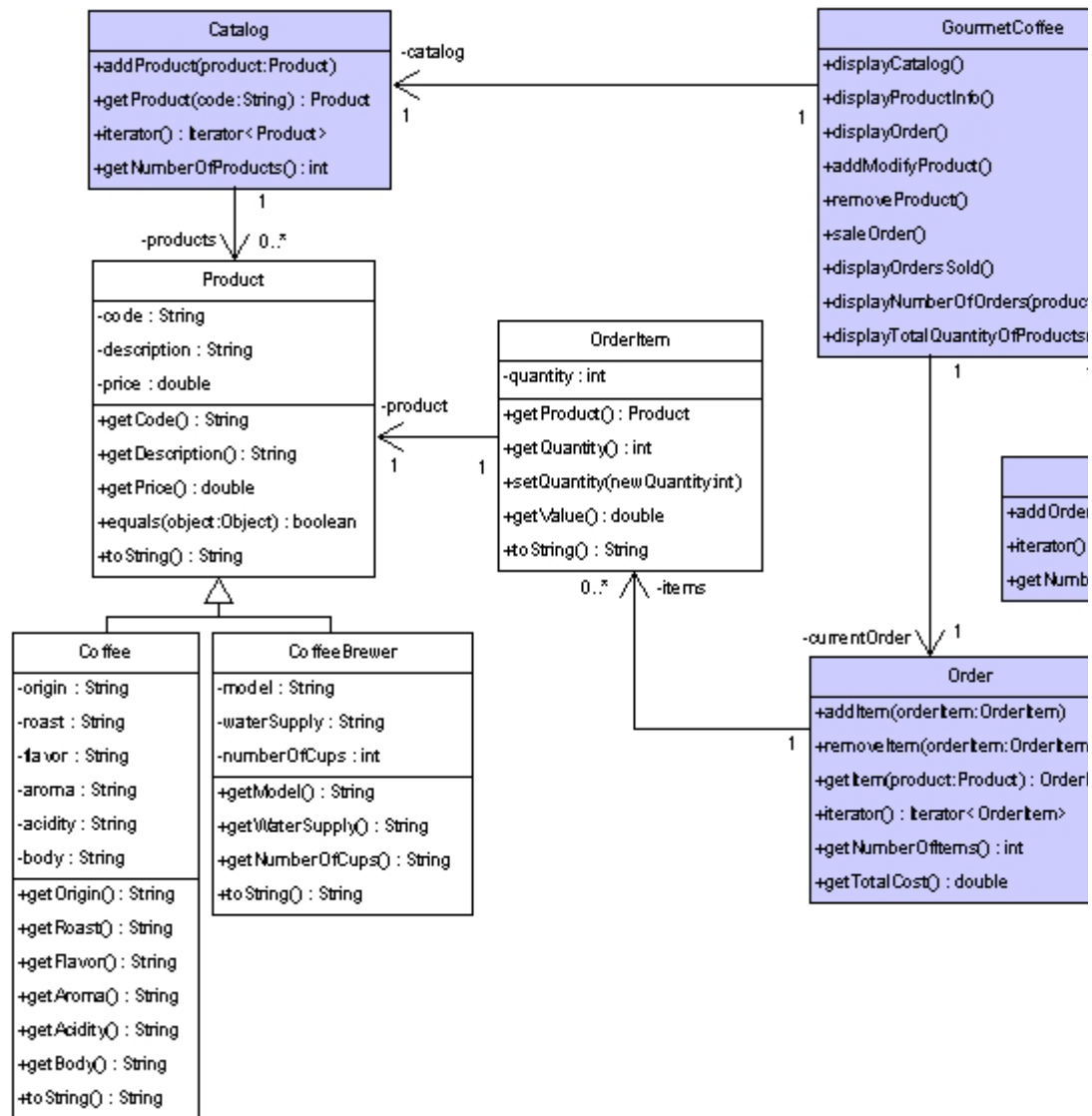
- Implementing a Java class that uses collections

#### **Background**

In this assignment, you will implement the classes in the *Gourmet Coffee System* that use collections.

#### **Description**

The following class diagram of the *Gourmet Coffee System* highlights the classes that use collections:



**Figure 1** *Gourmet Coffee System class diagram*

Complete implementations of the following classes are provided in the student archive:

- Coffee
- CoffeeBrewer
- Product
- OrderItem
- GourmetCoffee

In this assignment, you will implement the following classes:

- Catalog
- Order
- Sales
- GourmetCoffee

The class specifications are as follows:

### Class Catalog

The class Catalog 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 class `Product`.

*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`.

### **Class Order**

The class `Order` 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 class `OrderItem`.

*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.

### **Class Sales**

The class `Sales` 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 class `Order`.

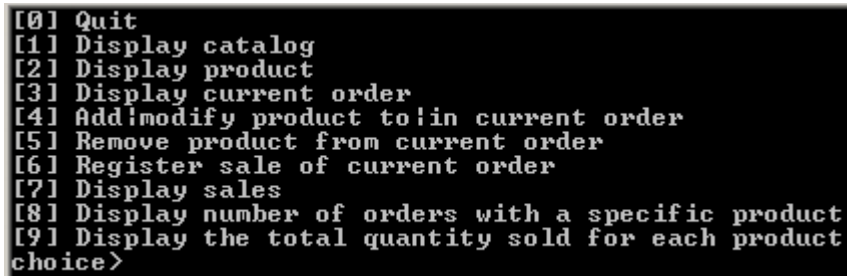
*Constructor and public methods:*

- *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.

### Class GourmetCoffee

The class `GourmetCoffee` creates a console interface to process store orders. Currently, it includes the complete implementation of some of the methods. The methods `displayNumberOfOrders` and `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 *GourmetCoffee*

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 GourmetCoffeeSolution()` — 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 complete and should not be modified.
- `public void displayProductInfo()` — Prompts the user for a product code and displays information about the specified product. This method is complete and should not be modified.
- `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 complete and should not be modified.
- `public void removeProduct()` — Prompts the user for a product code and removes the specified product from the current order. This method is complete and should not be modified.
- `public void saleOrder()` — Registers the sale of the current order. This method is complete and should not be modified.
- `public void displayOrdersSold()` — Displays the orders that have been sold. This method is complete and should not be modified.
- `public void displayNumberOfOrders(Product product)` — Displays the number of orders that contain the specified product. This method is incomplete and should be implemented.

- *public void displayTotalQuantityOfProducts()* — Displays the total quantity sold for each product in the catalog. This method is incomplete and should be implemented.

### Test driver classes

Complete implementations of the following test drivers are provided in the student archive:

- Class TestCatalog
- Class TestOrder
- Class TestSales

### Files

The following files are needed to complete this assignment:

- [student-files.zip](#) — Download this file. This archive contains the following:
  - Class files
    - *Coffee.class*
    - *CoffeeBrewer.class*
    - *Product.class*
    - *OrderItem.class*
  - Documentation
    - *Coffee.html*
    - *CoffeeBrewer.html*
    - *Product.html*
    - *OrderItem.html*
  - Java files
    - *GourmetCoffee.java* — An incomplete implementation.
    - *TestCatalog.java* — A complete implementation.
    - *TestOrder.java* — A complete implementation.
    - *TestSales.java* — A complete implementation.

### Tasks

Implement classes Catalog, Order, and Sales. Document your code using Javadoc and follow Sun's code conventions. The following steps will guide you through this assignment. Work incrementally and test each increment. Save often.

1. **Extract** the files by issuing the following command at the command prompt.

C:\>unzip student-files.zip

2. **Then**, implement class Catalog from scratch. Use TestCatalog to test your implementation.
3. **Next**, implement class Order from scratch. Use TestOrder to test your implementation.
5. **Then**, implement class Sales from scratch. Use TestSales to test your implementation.
6. **Finally**, complete class GourmetCoffee. It uses a Catalog object created by method GourmetCoffee.loadCatalog and a Sales object generated by method GourmetCoffee.loadSales. To complete class GourmetCoffee, implement the following methods:
  - *public void displayNumberOfOrders(Product product)* — This method displays the number of orders in the sales object that contain the specified product. Compile and execute the class GourmetCoffee. Verify that the method *displayNumberOfOrders* works correctly. 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()* — This method displays the total quantity sold for each product in the catalog. The information of each product must be displayed on a single line in the following format:

*ProductCode Quantity*

The following is a description of the information included in the format above:

- *ProductCode* — the code of the product
- *Quantity* — the total quantity of product that has been sold in the store

Compile and execute the class `GourmetCoffee`. Verify that the method `displayTotalQuantityOfProducts` works correctly. The following is the output that should be displayed for 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> **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

**Submission**

Upon completion, submit **only** the following.

1. Catalog.java
2. Order.java
3. Sales.java
4. GourmetCoffee.java

[Go to top of question.](#)

---

File to submit:

Upload File

Forward File

Refresh

Ready for Grading

---

[Go to top of assessment.](#)

© Copyright 2006 iCarnegie, Inc. All rights reserved.