Assignment 2: Coffee Shop Purchasing App and Order Processing App

Deadline: Friday, February 23 at 11:59pm

Description

Your assignment is to write two programs; one called **CoffeePurchasingApp** (save it in a file named **CoffeePurchasingApp.java**) that simulates a coffee-shop purchasing app, and another program called **CoffeeShopProcessingApp** (save it in a file named **CoffeeShopProcessingApp.java**) that simulates an app that processes the orders that are placed using the purchasing app.

To place an order, the user will run the purchasing app. After an order is placed, the purchasing app may then be used to check on the status of an order.

The processing app will be used to process orders received from the purchasing app.

The two apps will communicate via files. For example, the purchasing app will write the user's purchase to a file, and the processing app will then read the order from that same file.

Purchasing Program Requirements

Your purchasing program must run as follows.

a. Display the following welcome message to the user:

```
Welcome to Java Code Coffee Shop Purchasing App!
```

b. Display the following menu, prompt the user for an option, and read that option into your program. The user will select an option by entering the number that corresponds to that action (e.g. 1). We will refer to this menu as the **purchasing menu**.

```
--- Purchasing Menu ---
1. Place an order
2. Check order status
3. Exit

Enter an action:
```

c. If the user chooses option 1, then display the menu on the screen, with each item and the item's cost listed on a numbered line (you do not need to show the "Type" column - see the sample runs below for an example). Here are the items and the cost of each item:

Item	Cost
Coffee	$1.50
Latte	$3.50
Cappuccino	$3.25
Espresso	$2.00
Scone	$2.50
Muffin	$3.00

Prompt the user to enter a name for the order. Then prompt the user to select an item. The user will select an item by entering the number that corresponds to the desired item. Read the user's input and store it in your program.

Prompt the user to enter the quantity for the selected item. Read the user's input and store it in your program.

If the user selects a drink, offer a pastry, and vice versa.

Lastly, prompt the user for a coupon code. If the code entered by the user matches one of the valid codes, then give the user a 20% discount. If the user enters an invalid code, then display, "Sorry, that code is invalid." and continue. Also, the user may skip entering a coupon code by hitting the Enter key when prompted for a code.

Here are the valid coupon codes:
`COFFEE1, LATTE23, I<3JAVA, SCONEZ, MUFFIN!LOLOL`

Calculate the total cost of the order.

Note that each order has an order number; this number is displayed in the order summary and will be stored in the **orders.txt** file (more on that below). There should be a limit of 10 orders for each time you run the program. Order numbers are in the range 1 – 10. If the user places 10 orders during one session, and then tries to place order 11, display an appropriate message and take the user back to the purchasing menu.

Display an order summary and prompt the user to submit or cancel the order. **Format all dollar amounts to two decimal places in the output**. Also, be sure to display each dollar amount with a dollar sign ($). Below is an example summary. [Note: no taxes will be included in this assignment]

```
--- Order 1 Summary ---
1     Latte      $3.50
2     Scone      $5.00

Total Cost:           $8.50
20% Discount:         $1.70
Final Cost:           $6.80

Please enter 1 to confirm or 0 to cancel:
```

If the user enters a 0, cancel the order and take the user back to the purchasing menu.

If the user enters a 1, then submit the order and print, “Order [order number] has been placed”. The way that you will submit the order is by writing the order details to a file called **orders.txt**. The information that you should write to the file for each order is: name of the user, name of each item order, a quantity of each item ordered. You may choose the format for **orders.txt**. Below is the suggested format with an example.

Suggested format (note that this is a single line/row of data):

```
[order number],[name for the order],[coffee quantity],[latte
quantity],[cappuccino quantity],[espresso quantity],[scone quantity],[muffin
quantity]
```

Example with the suggested format:

```
1,Dave,0,1,0,0,2,0
```

The above format is a suggestion; you may use whatever format you like.

d. If the user chooses option 2, then your program should display the status of the user’s order. To do this, your program will read from a file called **status.txt**. If this file does not exist, then display the message “No orders are in at this time.” If the file exists and contains lines of text, then read the lines and display all lines.

e. If the user chooses option 3, then your program should display the message, “Goodbye!” and then terminate.

Processing Program Requirements

Your processing program must run as follows.

a. Display the welcome message, “Welcome to the Java Code Coffee Shop Processing program!”

b. Display the following menu, prompt the user for an option, and read that option into your program. The user will select an option by entering the number that corresponds to that action (e.g. 1). We will refer to this menu as the **processing menu**.

```
--- Processing Menu ---
1. View orders
2. Receive orders
3. Begin processing an order
4. Complete processing an order
5. View order history
6. Run a Report
7. Exit

Enter an action:
```

c. If the user chooses option 1, then your program should display all the current orders. If no orders exist, then display an appropriate message (e.g. “There are no orders at this time”).

d. If the user chooses option 2, then your program needs to receive the current orders (if any exist). This entails the following:

1. Read the orders into the processing app and store each one in a `String` array
2. Erase the contents of **orders.txt**
3. For each order that was accepted, write a status message in **status.txt** that has the following format:
   `Order [order number] has been received!`

   Example:
   ```
   Order 1 has been received!
   Order 2 has been received!
   Order 3 has been received!
   ```

Therefore, your processing program will move the orders from the **orders.txt** file into the program (and store each order in a `String` array that has size 10).

[To simplify things, it is recommended that you store "order *n*" in the array at index *n – 1*. For example, if you read in an order from **orders.txt** that has order number 6, store this order in your array at index 5.]

e. If the user chooses option 3, then your program should allow the user to begin processing one of the received orders. The program will display each of the received orders that are not being processed, and the user will choose one of those orders.

For example:

```
Choose one of the received orders to process:
1. Order 2
2. Order 4
3. Order 5

Which order would you like to start processing? <3>

Order 5 is now processing!
```

In the processing app, an order can be in one of three possible states: "received", "processing", "processed". How will your program keep track of the state of each order? One way is to use an array of `int` values (let's refer to this array as **`orderStatus`**) – one value for each order. When an order is received, set that order's value in **`orderStatus`** to 1. When that order is being processed, set its **`orderStatus`** to 2. And when the order has completed processing, set its **`orderStatus`** to 3.

When processing of an order is started, write an "order processing" message to **status.txt**, as in this example:

```
Order 5 is now processing!
```

Also, set the **`orderStatus`** of the order to 2.

If no orders are received but not processing, then display an appropriate message.

e. If the user chooses option 4, display a menu of all processing orders. The user will choose one of these orders, and the following actions will occur:

(1) append the order data for the selected order to the file **order-history.txt**; and
(2) write a status to the file **status.txt**; and
(3) update the **orderStatus** of the order to 3.

Example:

```
Here are the orders that are processing:
1. Order 5
2. Order 8

Which order would you like to finish processing? <2>

Order 8 has been processed.
```

Write a line such as, "Order 8 has been processed!" to **status.txt**.

e. Option 5 will allow the user to view order history.

If the user chooses option 5, then display the order history in a table format.

Below is an example of viewing the history. Note that each row represents a single order.

```
--- Order History ---

Customer   Coffee     Latte      Cappuccino       Espresso   Scone      Muffin
--------   ------     -----      ----------       --------   -----      ------
Fred            2         0               0              0       1            1
Lisa            0         1               2              0       1            0
Justin          1         3               0              4       0            0
Fred           10         0               0              0       0            0
```

f. Option 6 will allow the user to run reports. A report will aggregate the orders in the order history with respect to a specified field, which can be the name of a customer or the name of an item.

A report on a customer will look like this:

```
--- Report Menu ---

1. Customer
2. Coffee
3. Latte
4. Cappuccino
5. Espresso
6. Scone
7. Muffin
```

```
Choose a field: 1
Enter the customer's name: Dave

Coffee   Latte   Cappuccino   Espresso   Scone   Muffin   Total Items
------   -----   ----------   --------   ------  ------   -----------
     0       2            0          3       0        1             6
```

A report on an item will look like this:

```
--- Report ---
1. Customer
2. Coffee
3. Latte
4. Cappuccino
5. Espresso
6. Scone
7. Muffin

Choose a field: 3

Item:           Latte
Total Sold:     5
```

Coding Requirements

1. You must store the received orders in a `String` array in the processing app.

2. You must include a **multi-line comment** at the top of each source file that includes your name, the class name (CS401), the semester (Spring 2018), and the assignment name (Assignment 2), with one of these items per line.

3. You must print each of your calculated dollar amounts (e.g. tax amount) with exactly two decimal places (There is more than one way to format numbers in output; see sections 3.10 and 3.11 in the textbook).

4. You must use the `printf` method to print the "order history" in the processing app and also to print the customer reports in the processing app.

5. Use good programming style.

Assumption

You may assume that the user will enter valid inputs when prompted, except in the case when the use is prompted for a coupon code.

Program Demo

For the demo, the contents of the files (**orders.txt**, **status.txt**, and **order-history.txt**) is given during various points while the programs are running. The marker "[empty]" indicates the file is empty.
Also, user input is in **bold, red** font.

```
-----------------------------------------------------------------
 orders.txt:
[empty]

status.txt:
[empty]

order-history.txt:
[empty]
 -----------------------------------------------------------------



-----------------------------------------------------------------
First, let's run the purchasing app to place two orders.
-----------------------------------------------------------------


Welcome to Java Code Coffee Shop Purchasing App!

--- Purchasing Menu ---
1. Place an order
2. Check order status
3. Exit

Enter an action: 2

Order status:
No orders are in at this time.


--- Purchasing Menu ---
1. Place an order
2. Check order status
3. Exit

Enter an action: 1

Here is our menu:
1. Coffee       $1.50
2. Latte        $3.50
3. Cappuccino   $3.25
4. Espresso     $2.00
5. Scone        $2.50
```

```
6. Muffin       $3.00

Enter a name for the order: Dave

Enter the item number: 3

Enter the quantity: 1

Would you like to add a pastry?: 1

Great! Which one would you like? 5

And how many? 2

Enter a coupon code (or press Enter to skip): SCONEZ

Your code has been accepted! You will receive 20% off!

--- Order 1 Summary ---
1     Cappuccino      $3.25
2     Scone           $5.00

Total Cost:           $8.25
20% Discount:         $1.65
Final Cost:           $6.60

Please enter 1 to confirm or 0 to cancel: 1

Order 1 has been placed.


--- Purchasing Menu ---
1. Place an order
2. Check order status
3. Exit

Enter an action: 1

Here is our menu:
1. Coffee       $1.50
2. Latte        $3.50
3. Cappuccino   $3.25
4. Espresso     $2.00
5. Scone        $2.50
6. Muffin       $3.00

Enter a name for the order: Moltar

Enter the item number: 6

Enter the quantity: 3
```

```
Would you like to add a coffee or espresso drink?: 0

Enter a coupon code (or press Enter to skip):

--- Order 2 Summary ---
3     Muffin     $9.00

Total Cost:           $9.00
Final Cost:           $9.00

Please enter 1 to confirm or 0 to cancel: 1

Order 2 has been placed.


--- Purchasing Menu ---
1. Place an order
2. Check order status
3. Exit

Enter an action: 3

Goodbye!
```

```
orders.txt:
1, Dave, 0, 0, 1, 0, 2, 0
2, Moltar, 0, 0, 0, 0, 0, 3

status.txt:
[empty]

order-history.txt:
[empty]
```

Now let's run the processing app to process those two orders.

```
Welcome to Java Code Coffee Shop Processing App!

--- Processing Menu ---
1. View orders
2. Receive orders
3. Begin processing an order
4. Complete processing an order
5. View order history
6. Run a Report
7. Exit

Enter an action: 1

--- Current orders ---
1, Dave, 0, 0, 1, 0, 2, 0
2, Moltar, 0, 0, 0, 0, 0, 3


--- Processing Menu ---
1. View orders
2. Receive orders
3. Begin processing an order
4. Complete processing an order
5. View order history
6. Run a Report
7. Exit

Enter an action: 2

Order 1 has been received!
Order 2 has been received!
```

---

orders.txt:
[empty]

status.txt:
Order 1 has been received!
Order 2 has been received!

order-history.txt:
[empty]

```
--- Processing Menu ---
1. View orders
2. Receive orders
3. Begin processing an order
4. Complete processing an order
5. View order history
6. Run a Report
7. Exit

Enter an action: 3


Choose one of the received orders to process:
1. Order 1
2. Order 2

Which order would you like to start processing? 2

Order 2 is now processing!


-----------------------------------------------------------------
 orders.txt:
 [empty]

 status.txt:
  Order 1 has been received!
  Order 2 is now processing!

 order-history.txt:
 [empty]
-----------------------------------------------------------------


--- Processing Menu ---
1. View orders
2. Receive orders
3. Begin processing an order
4. Complete processing an order
5. View order history
6. Run a Report
7. Exit

Enter an action: 4

Here are the orders that are processing:
1. Order 2
```

```
Which order would you like to finish processing? 2

Order 2 has been processed!


-----------------------------------------------------------------
 orders.txt:
 [empty]

  status.txt:
   Order 1 has been received!
   Order 2 has been processed!

  order-history.txt:
 Moltar, 0, 0, 0, 0, 0, 3

 -----------------------------------------------------------------


--- Processing Menu ---
1. View orders
2. Receive orders
3. Begin processing an order
4. Complete processing an order
5. View order history
6. Run a Report
7. Exit

Enter an action: 3


Choose one of the received orders to process:
1. Order 1

Which order would you like to start processing? 1

Order 1 is now processing!



-----------------------------------------------------------------
 orders.txt:
 [empty]

  status.txt:
   Order 1 is now processing!
   Order 2 has been processed!
```

order-history.txt:
Moltar, 0, 0, 0, 0, 0, 3

--- Processing Menu ---
1. View orders
2. Receive orders
3. Begin processing an order
4. Complete processing an order
5. View order history
6. Run a Report
7. Exit

Enter an action: **4**

Here are the orders that are processing:
1. Order 1

Which order would you like to finish processing? **1**

Order 1 has been processed!

---

orders.txt:
[empty]

status.txt:
Order 1 has been processed!
Order 2 has been processed!

order-history.txt:
Moltar, 0, 0, 0, 0, 0, 3
Dave, 0, 0, 1, 0, 2, 0

Now let's go back to the purchasing app to check on the status of the orders.

```
Welcome to Java Code Coffee Shop Purchasing App!

--- Purchasing Menu ---
1. Place an order
2. Check order status
3. Exit

Enter an action: 2

Order status:
Order 1 has been processed!
Order 2 has been processed!
```

Let's return to the processing app to run some reports

```
Welcome to Java Code Coffee Shop Processing App!

--- Processing Menu ---
1. View orders
2. Receive orders
3. Begin processing an order
4. Complete processing an order
5. View order history
6. Run a Report
7. Exit

Enter an action: 5


--- Order History ---

Customer  Coffee  Latte  Cappuccino  Espresso  Scone  Muffin
--------  ------  -----  ----------  --------  -----  ------
Moltar         0      0           0         0      0       3
Dave           0      0           1         0      2       0


--- Processing Menu ---
1. View orders
2. Receive orders
3. Begin processing an order
```

```
4. Complete processing an order
5. View order history
6. Run a Report
7. Exit

Enter an action: 6


--- Report Menu ---
1. Customer
2. Coffee
3. Latte
4. Cappuccino
5. Espresso
6. Scone
7. Muffin

Choose a field: 1
Enter the customer's name: Dave

Coffee  Latte  Cappuccino  Espresso  Scone  Muffin  Total Items
------  -----  ----------  --------  ------ ------  -----------
     0      0           1         0       2      0            3


--- Processing Menu ---
1. View orders
2. Receive orders
3. Begin processing an order
4. Complete processing an order
5. View order history
6. Run a Report
7. Exit

Enter an action: 6

--- Report Menu ---
1. Customer
2. Coffee
3. Latte
4. Cappuccino
5. Espresso
6. Scone
7. Muffin

Choose a field: 6
```

```
Item:            Scone
Total Sold:      2


--- Processing Menu ---
1. View orders
2. Receive orders
3. Begin processing an order
4. Complete processing an order
5. View order history
6. Run a Report
7. Exit

Enter an action: 7

Goodbye!
```