FLORIDA SOUTHERN COLLEGE

Data Structures
CSC 3280

Fall 2020

Program 1: Moc Mart
Assigned: Tuesday, September 8, 2020
Due: Friday, September 18, 2020 by 11:55 PM

Purpose:

1. The first purpose of this program is for you to REVIEW your Java and programming an array of objects.
2. The second purpose of this program is for you to review file I/O (input/output).
3. Finally, you are to implement binary search in the program.

Read Carefully:

- This program is worth 7% of your final grade.

- **WARNING**: This is an individual assignment; you must solve it by yourself. Please review the definition of cheating in the syllabus, the FSC Honor Code, and the serious consequences of those found cheating.
    - **The FSC Honor Code pledge MUST be written as a comment at the top of your program**.

- When is the assignment due? The date is written very clearly above.
    - **Note:** once the clock becomes 11:55 PM, the submission will be closed! Therefore, in reality, you must submit by 11:54 and 59 seconds.

- LATE SUBMISSION: you are allowed to make a late submission according to the rules defined in the syllabus. Please see course syllabus for more information.

- **Canvas Submission**:
    - This assignment must be submitted online via Canvas.
    - You should submit a single ZIP file, which has **ONLY** your Java files inside it.
        - Do NOT submit the full NetBeans Project folder…ONLY the Java files.
    - Within NetBeans, your project should be named as follows:
        - `MocMart`
        - And when you make the project, NetBeans will automatically create a package with that same name. This is what we want.
- Your zip should be named YourFSCIDnumber.zip

Program 1: Moc Mart

The Problem

Your going to write a program to simulate the Moc Mart, with customers coming to purchase items and more. Your program will:

- manages all products in the store, along with the stock (quantity) for that product
- manages the reordering of depleted products (once the stock for that item reaches zero)
- manage the total sales to all customers.

During the program, the following things can occur (just as in a normal store):

- New products can be added to the list of products. For example, perhaps the store doesn't initially carry Cheerios. So this new product, Cheerios, can be added as a new product for Moc Mart. Note: when new products are added for the first time, they are initialized with a specific stock amount associated with that product. It is guaranteed that newly added items will always have a quantity of at least one unit.
- Customers can come throughout the day to purchase items from the store. If the items are available, they are purchased, and the stock of those specific products are all reduced by the quantity purchased. If a product is not available (stock is zero), the customer simply does not purchase that product.
- A purchase occurs when a customer finds at least one of the items they are looking for. All such purchases will need to be saved as described later in this write up. However, if a customer enters the store and all of the desired items have a stock of zero, this customer did not make a purchase, and therefore, does not need to be saved.
- As customers purchase products, the quantity for that product is clearly reduced. Once the inventory level for a product reaches zero, no more product can be purchased (of that item) until the RESTOCK command occurs.
- Reorders can be made, and this will happen whenever the RESTOCK command is read from the input file. Products that are reordered will have their stock increased based on a fixed restock quantity associated with that particular product.

You will use File I/O to read input from a file and then print the output to a file. To be clear, you will read COMMANDS from an input file. Example commands are ADDITEM, RESTOCK, CUSTOMER, etc. Then, depending on the command, you will either add a new item, restock the items, process a customer/sale, etc. But instead of printing to the console window (screen), you will print to an output file.

**Sample input and output files have been provided to you with the 7z zip file.*

Implementation

For this program, you will create **two** Classes (UML diagram shown below):

MocMartSale

Data Members
`private String firstName`
`private String lastName`
`private int numItemsOnList`
`private int[] itemsPurchased`
`private static int numSales`

Operations/Methods public String getFirstName( ) public String getLastName( ) public int getNumItemsOnList( ) public int[] getItemsPurchased( ) public static int get numSales( ) public void setFirstName(String firstName) public void setLastName(String lastName) public void setNumItemsOnList(int numItemsOnList) public void setItemsPurchased(int[] itemsPurchased) public static void setNumSales (int numSales)

MocMartProduct

Data Members
`private int itemNum`
`private String itemName`
`private double itemPrice`
`private int quantity`
`private int restockQuantity`
`private static int numProducts`

Operations/Methods public int getItemNum( ) public String getItemName( ) public double getItemPrice( ) public int getQuantity( ) public int getRestockQuantity( ) public static int getNumProducts ( ) public void setItemNum(int itemNum) public void setItemName(String itemName) public void setItemPrice(double itemPrice) public void setQuantity(int quantity) public void setRestockQuantity(int restockQuantity) public static void setNumProducts (int numProducts)

*Note: the methods given above are just an example. You may certainly add methods if you feel that is best for your program.

You will use the `MocMartProduct` class to create nodes of type `MocMartProduct`, and you will use the `MocMartSale` class to create nodes of type `MocMartSale`.

The first two lines of the input file are as follows:
Line 1: the maximum number of products that can be in Moc Mart
Line 2: the maximum number of sales recorded by the system

You must read these values into appropriate variables.

Next, you will use these new values to make two new arrays:

1. An array of `MocMartProduct` object ***references***
2. An array of `MocMartSale` object ***references***

- *See pages 72 - 77 of Chapter 9 slides (from 2290) for help on array of object references*

To help you, here is the code to do this:

```
MocMartProduct[] products = new MocMartProduct[maxProducts];
MocMartSale[] sales = new MocMartSale[maxSales];
```

What do these lines do? They create an array of **references**. Note: each reference has a default value of **`null`**. Until now, we have NOT created any objects. For example, `MocMartProduct` objects are only created when we see the command `ADDITEM`. At that time, a new `MocMartProduct` object will be created and the reference for that object will be saved in the appropriate location of the array.

Input File Specifications

You will read in input from a file, "**`MocMart.in`**". Have this AUTOMATED. Do not ask the user to enter "**`MocMart.in`**". You should read in this automatically.

> ***Note:*** a **`*.in`** file is just a regular text file. Also, a `*.out` file is a regular text file. Do not be scared by this. Instead of calling the input and output file as input.txt and output.txt, it is better for those files to have a good name. Therefore, the input file will be called `MocMart.in` and the output file will be called `MocMart.out`.

The first line of the file will be an integer, *d*, representing the maximum number of products in the Moc Mart. The second line of the file will be an integer representing the maximum possible number of recorded sales for the Moc Mart. Each of the remaining lines will have a command, which is then followed by relevant data as described below (and this relevant data will be on the same line as the command).

The commands you will have to implement are as follows:

- ADDITEM – Makes a new product which is added to the store. The command will be followed by the following information all on the same line: `itemNum`, an integer representing the item number for this product; `itemName`, the name of the item, no longer than 20 characters; `unitPrice`, the price of a single item; `stockQty`, the initial amount of stock for this product (guaranteed to be at least 1); `restockQty`, the quantity by which the product should be restocked whenever a reorder is placed.

  When you read the ADDITEM command, you must make a new object of type `MocMartProduct`. Next, you must scan the additional information (listed above) and save this information into the correct data members of the new object. Each item will be unique. Meaning, the input file is guaranteed to not have duplicate

items added to the store inventory via the ADDITEM command. Finally, you must save the reference of this object inside the appropriate index of the student array.

*__Note__: this array **must** stay in sorted order based on the ID of the product. So the product with the smallest ID value will be at index 0, the product with the next smallest at index 1, and so on. Therefore, when you save the object reference into the array, you must first find the correct sorted index. After you find the correct insertion location, if there is no object at that location, then you can easily save the object reference at this index. **BUT, IF** there is an object at the index that you find, you must **SHIFT** that object, **AND** all objects to the right of it, one space to the right. This is how you make a new space in the array for the new object.

Example: if your array already has 6 product object references, from index 0 to index 5. And now you want to add a new product object reference at index 4 (based on the sorted location). Before you can add the new product object reference at index 4, you must SHIFT the product object references at index 4 and index 5 to index 5 and index 6. Basically you need to move them one position over to the right. Now, you have an empty spot at index 4 for the new product object reference.

***Note: you are NOT allowed to use ArrayLists for this program. The whole purpose here is for you to deal with shifting and see how problematic it is with respect to efficiency.*

- FINDITEM – This command will be followed by an integer on the same line. This integer represents the Item number of the item you must search for. You must perform a **Binary Search** on your array of `MocMartProduct` object references. If the product is found, you should print the required information for the product. (see output).

  *See output for different printing possibilities.

  **Note: I want to confirm that you are using Binary Search for this task. Therefore, you should print each "mid" that is visited during the binary search (See output).

- RESTOCK – This command will have no other information on the line. When this command is read, all items that have a stock of zero must be restocked. You do this as follows:
  - You must iterate over all objects in the products array.

- You will check the stock of each item. If the stock is zero, you will reset the stock using the `restockQty` variable.

- CUSTOMER – This command is for customers attempting to make purchases. It is followed by the following on the same line: a first name, then a last name, each no longer than 20 characters; an integer, *n*, representing **two times** (2x) the number of different products the customer wants to purchase, followed by *n* integers, which will be pairs of item numbers and the respective quantity purchased of that item number. For example, **if n were six**, this means the customer wants **three** items. Pretend those following six integers were as follows: 5437 2 8126 1 9828 4. This means the customer wants 2 units of product 5437, 1 unit of 8126, and 4 units of product number 9828.

  As mentioned previously, a purchase occurs when a customer finds, and of course then buys, one of the items on their shopping list. When this happens, a new object of type `MocMartSale` must be made, which records this sale. The data members of the object must be saved correctly. One member of the `MocMartSale` node is `itemsPurchased`. This is an array that must be made based on the size (number) on the original shopping list. If the product is found and purchased, the item number is added to the corresponding cell of this array, along with the quantity purchased. We assume that if the stock for a given item is available, the customer will purchase the full desired quantity on their shopping list (of that item). If the product is not found, a zero is recorded in that cell of the array.

  Example: Based on the three-item example above, if the first and third items were available, with the second being unavailable (no stock), the `itemsPurchased` array would have SIX cells as follows: 5437, 2, 8126, 0, 9828, 4. This shows that 2 units of 5437 were purchased, and 4 units of 9828 were purchased; the zero after item 8126 simply shows that the item was not available.

  Finally, once all appropriate information is saved into the data members of the `MocMartSale` object, a reference to this object is then added to the next available position in the sales array. Note: a line of output is printed regardless of whether or not a purchase was made. Refer to sample output for examples.

- INVENTORY – This command will have no other information on the line. When this command is read, the current Moc Mart inventory (each item with its respective stock) is printed to the file in ascending order (the order that the product list is already in). See sample output file for specific formatting of this command. If there

store does not have any inventory, an appropriate error message is printed (see sample output).

- PRINTSUMMARY – This command will have no other information on the line. When this command is read, a report of all sales, for that given day, will be printed to the output file. Please refer to sample output for exact specifications. Once the header information is printed (see sample output), all sales (nodes) found in the sales array will need to be printed. This will proceed as follows:
  - You need to iterate over the entire sales array.
  - For every Sale (node) in the sales array, you need to print the Sale number (starting at 1), along with the first and last name of the customer (in that order). You then need to print the list of items that were successfully found and then purchased. Finally, you need to print the total amount paid. Please refer to the EXACT output format for specifics.
  - Lastly, you must print out the total sales (dollar amount) for the given day.

Output Format

Your program must output to a file, called "**`MocMart.out`**". **You must follow the program specifications exactly.** You will lose points for formatting errors and spelling.

When examining the output file, you should notice that the INVENTORY command and the PRINTSALESSUMMARY command results in printing a "semi-formatted" line of text. To make everything clear and to guarantee the correctness of your format, we are providing those exact code needed to print those two lines below:

Here is the literal for printing inventory items in the INVENTORY command:

```
"\t| Item %6d | %-20s | $%7.2f | %4d unit(s) |%n"
```

And here is the literal for printing customer items in the PRINTSALESSUMMARY command:

```
"\t\t| Item %6d | %-20s | $%7.2f (x%4d) |%n"
```

Note: You need to use formatted printing to print these lines of output.

Sample and Output Files

A sample input file, with corresponding output file, has been provided to you with the 7z zip file (the files have too many lines to paste here).

NOTE: These files do NOT test every possible scenario. That is the job of the programmer to come think up the possible cases and test for them accordingly. You can be sure that the input file we grade with will be very large with the expectation that it will test all possible cases. It is recommended that you spend time thinking of various test cases and build your own input file for testing purposes.

*****WARNING*****

Your program MUST adhere to the EXACT format shown in the sample output file (spacing capitalization, use of dollar signs, periods, punctuation, etc). For grading, we will use very large input files, resulting in very large output files. As such, text comparison programs will be used to compare your output to the correct output. If, for example, you have two spaces between in the output when there should be only one space, this will show up as an error even through you may have the program correct. You WILL get points off if this is the case, which is why this is being explained in detail. Minimum deduction will be 10% of the grade, as I will be forced to go to text editing of your program in order to give you an accurate grade. Again, your output MUST ADHERE EXACTLY to the sample output.

Grading Details

Your program will be graded upon the following criteria:

1) Adhering to the implementation specifications listed on this write-up.
2) Your algorithmic design.
3) Correctness.
4) **Use of Classes, Objects, and Arrays of Objects. If your program is missing these elements, you will get a zero. Period.**
5) The frequency and utility of the comments in the code, as well as the use of white space for easy readability. (We're not kidding here. If your code is poorly commented and spaced and works perfectly, you could earn as low as 80-85% on it.)
6) Compatibility to the **newest version** of NetBeans. (If your program does not compile in NetBeans, you will get a large deduction from your grade.)
7) Your program should include a header comment with the following information: your name, **email**, course number, section number, assignment title, and date.
8) Your output MUST adhere to the EXACT output format shown in the sample output file.

Deliverables

You should submit a zip file with THREE files inside:

1. `MocMartProduct.java`
2. `MocMartSale.java`
3. `MocMart.java` (this is your main program)

***These three files should all be INSIDE the same package called **`mocmart`**. If they are not in this specific package, you will lose points.

NOTE: your name, ID, section number AND EMAIL should be included as comments in all files!

Suggestions:

- Read AND fully understand this document BEFORE starting the program!
- Read and fully understand the `IOexample.java` program provided to you.
- Now, make your new project in NetBeans and name it MocMart.
  - You can use the "Java with Ant" option when making new projects
  - The create main class box should be checked
  - And it should say "mocmart.MocMart"
  - Now you have a project and you have your main class, `MocMart.java`, although it is, for all intents and purposes, empty.
- Next, start by making the two classes to create nodes:
  - `MocMartProduct` and `MocMartSale`
  - So make those two classes, with all the data members, constructors, and accessor/mutator methods.
- Now, after you have made these two classes, begin to edit your main class:
  - `MocMart.java`
  - This is your main program that will read the commands from the file, process the commands, and print to the output file.
  - Use the `IOexample.java` program to help you create your main to read from a file and write to a file.
  - Finally, one by one, implement the commands (ADDITEM, CUSTOMER, etc).

Hope this helps.

Final suggestion: START EARLY!