

*#7*

# CPT 187—OBJECT-ORIENTED LOGIC & DESIGN

Spring-Program #7

Due Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**PROGRAM DESCRIPTION**

It’s time to get a bit more practical about your **InventoryData** class: wasn’t it nice that the file *inventoryPricing.dat* had already been sorted by part number? If you were doing this for real, you would probably not be so lucky! Most of the time you will not have the luxury of assuming that the file is sorted just the way you need it. So, in this new version of your inventory class, you are going to update your logic to take that into account. You will be working with a different file, *masterInventory.dat*, which is not sorted by part number. Also, you are going to add some functionality to compute the total price for how many items the customer wants to purchase.

## **SPECIFIC DIRECTIONS**

Start out with your Program #6 (after making any necessary changes that your feedback identified. You don’t want to lose the same points again, and my policy is to increase the point deduction if you don’t seem to be learning from your mistakes!). Now, you need to expand your class to incorporate the bubble sort method to put all the information in order after the arrays have been loaded. The bubble sort flowchart is available as a Student Handout in the Content area, and this is the algorithm that you must use. Don’t go searching for a bubble sort elsewhere. When it is time to swap elements, you can just program that into the sort method, instead of calling another method as the flowchart shows. (Of course, you may create a swap method if you prefer.) Not that other than fixing problems with program #6 this new method is the only change you should need to make to your **InventoryData** class (plus the swap method if you so choose).

The new data file is ***masterInventory.dat***, and it has two fields per record, just like the *inventory.dat* file, but is not sorted. You should have found it zipped up with these directions. Note: do not make any changes to this file to enable your program to work, because I will be using a similar but completely different file to evaluate your program.

**RECORD DESCRIPTION (masterInventory.dat):**

|  |  |
| --- | --- |
| partNum | int |
| price | double |

After the data has been loaded into arrays and sorted, your program should explain to the user how the program will work. Then, start asking for part numbers. For every part number that you find, report the correct price. If not found, report that the part number is invalid. As before, your program needs to report the search results independently for the two search algorithms. In fact, that’s even more important now, since the data file was not sorted. You need to prove that both search algorithms are still working after you rearranged the arrays.

Next, if the part number was found, you have more work to do. Ask the user for how many pieces they want to order for that part number. Then, report a short summary table of the cost. Report the part number, its price, how many they want to purchase, and the total cost for those parts. This output must be formatted for ease of reference and understanding, using a header for each answer. For example, if the parts clerk enters a part number of 15804, and a quantity of 15, then you should report results like this:

PART# PRICE QTY COST

15804 28.14 15 422.10

Don’t even think of coding some ugly-looking sentence with the answers embedded in it! And remember, monetary values in the US should be shown with two decimal places. Don’t tell the user that the cost is 422.1 (or worse, 23.2747). Allow the user to search for as many parts as they want, reporting the results of each one separately just like the one above, and quit only when the user wishes to.

When the user quits the program, report the following summary data: total number of parts that were sought (don’t count the two search methods separately, just the count of part numbers entered); the number of parts that were found; and the number of invalid part numbers entered. These last two should add up to the total.

As always, be sure that you have named the project, package, and .Java files correctly. See the Coding Standards handout for the required names.

## **TURN IN**

A complete Class Description and a zipped copy of the complete solution directory.