Software Development Challenge 2018

Software Development Challenge 2018

The Software Development Competition is intended to offer college students an opportunity to showcase their development talent. Students will be eligible to win a cash prize and will be recommended as candidates for our internship and full-time positions. The challenge follows the format of existing programming competitions/contests (see codechef.com, topcoder.com, etc.). For dates and prizes, see the sign-up website: https://www.tylertech.com/softwaredevchallenge2018

You can expect a small challenge to solve programmatically with varying levels of complexity designed for different grade levels, and a writing component to document your processes. We design the challenge complexity levels to meet approximately a 4-hour effort for completion. There will be three types of tasks: a required task, three optional advanced tasks, and an extra credit task to solve programmatically using **Java or C#** only**.**

Table of Contents

Programming Challenge: The Assignment

You are opening your own coffee shop! Starting a new business can be expensive, so you have decided to write your own program for taking and tracking orders. Customers can come up with the craziest drink combinations and you'll need to be ready for just about anything. Drink type, size, and any add-ons will all impact the price of the order. Consider building in the advanced tasks below to provide better service to your customers by offering things like food or special discounts. Your job is to write a program that will take these factors into consideration and provide a price for each order.

Required Task: This task is mandatory for submission.

For any given drink order, provide a correct price to the customer. This price should account for the variety of options applicable to a drink such as type, size, and add-ons. As part of the input file, you will be provided with information about the price options for different sized drinks, types of drinks and add-ons such as an extra shot of espresso, flavored syrup, or whipped cream. For any given combination of these things, your program should be able to output a price for the order. A few constraints do apply: customers shouldn't be able to order an add-on without ordering a drink and customers must specify a size for their drink.

Additional Tasks: You may optionally solve these challenges for extra points.

1. Implement the ability to use food items. The input for food items has been defined in the Inputs section.

2. Allow purchase of "meals" that include a food and a drink item as a combo. Allow your program to apply a 10% discount automatically when items can be grouped into combos. This discount should apply for each pair of drink and food items in the order. For example, if a customer orders two large coffees and one croissant, the discount should only apply to one of the coffees and the croissant. To be generous with your customers, you should discount the most expensive drink when calculating the total order amount.

3. Allow the system to optionally pass a payment method. Provide a 5% discount to customers that pay with cash, instead of a credit card. This discount would be in addition to the discounts given for combo meals and should be applied after other discounts have been calculated.

Extra Credit:

Create a command line interface to add, delete, or modify your shop's menu. Please be certain to clearly document your extra credit in the write up.

Inputs

Your program will need to load and parse a configuration input file located in a subdirectory from your executable. The path passed to your program as the first input argument will take the form: "./inputs/input.X.json" where "X" is some number between 1 and 100 that denotes the test number. Keep track of this number as it will be required for the name of your output file.

An example JSON configuration file will be made available to you. This file will contain all of the information needed to build your drink and food menu, in the following format:

DrinkExtras: Array of objects that defines the extra options available for all drinks on the menu. Each object represents one drink extra. These extras are optional and may not necessarily be included in the order.

Name: string. Name of the add-on.

Price: decimal. Price for the add-on.

Drinks: Array of objects. Each object represents one drink.

Name: string. Name of the drink.
Sizes: object[].
Name: string. Name of the size (e.g.: Small, Medium, Large).
Price: decimal. Price to charge the customer for this size.

Food: Array of objects. Each object represents one food item.

Name: string. Name of food item (e.g.: Bagel).
Sizes: object[]. Sizes available for that item.
Name: string. Name of the size (e.g.: half, whole).
Price: decimal. Price for the size option.
Extras: object[]. Any extra options available for the food option.
- *Name*: string. Name of the extra (cream cheese)
- *Price*: decimal. Price for the option.

After configuration has been loaded, your program should prompt for orders. For simplicity, you can depend on an order always being entered with the following format:

order [item] [size] [extra1]...[extraN], [item] [size] ...

Example:

order coffee medium whipcream flavorshot, coffee small, bagel half butter

Note: item extras are optional

Output

Your program will be expected to write a JSON file into a sub-directory from your executable. The path of your output file will take the form: " ./outputs/FIRSTNAME_LASTNAME.X.json", where "FIRSTNAME" is your first name, "LASTNAME" is your last name, and "X" is the number loaded from the input file's name.

The JSON file should contain an object array named Orders. Orders will contain objects with the following properties:

- *Price*: The total order price.
- *Error:* If applicable, an error message explaining why an order couldn't be processed and totaled.

An example output file has been commented and provided for your understanding. Your final output file should not include any comments.

Packaging and Submission

Submissions must be in the form of a Zip file containing all your documents. The name of that file should follow the following pattern:

<last-name>-<first-name>-<four-digit-combination-of-your-choice>.zip

At the root level of your Zip file, the following should be present:

- A PDF document named “WriteUp.pdf”.
- A text document named “README.txt”.
- A directory containing all your source code files name “Source”.

The “WriteUp.pdf” document should be short (ideally 1 page) and it should contain a description of the problem-solving process that you went through. You can describe the algorithm that you used and discuss why you chose that specific algorithm. You should go over any complications that you experienced while working on this problem and the different approaches you took to solve them; feel free to discuss what worked and what did not. In the end, you can describe if there was anything new that you learned from this project.

The “README.txt” document should contain the necessary information needed to compile and run your program. It should provide configuration and installation instructions as well as a summary of known bugs (if any). You are expected to provide contact information and a copyright and licensing statement. If necessary, a “credits and acknowledgement” section must be included.

The “Source” directory should contain only your source code. Please do not provide any pre-built executables as your solution will be compiled and tested with in-house utilities. This assignment can be coded using Java or C#. We will use the latest version of the corresponding compiler to compile your code indifferent to the programming language you choose. If exceptions apply to your solution, please indicate so in the “README.txt” file.

Grading Criteria

The challenge will include a mandatory basic task, three optional advanced tasks, and an optional “extra credit” task. A solution is only required for the basic task to qualify, as each of the different tasks will be tested separately. The tests will be executed using different input files for the basic task, optional advanced tasks, and optional “extra credit” task. The same input files and tests will be used across all submitted solutions. During these tests you will be graded on error handling for bad data, result accuracy, and execution speed.

You will also be graded for the overall code organization in your program. The expectation is that you will make good use of comments and use meaningful variable names in case somebody inspects your code. In the event of a tie, we will evaluate other aspects of your code such as creativity, originality, and performance of the algorithms used.

Written Component (max 15 pts)

Points earned correspond to rubric descriptions.

Points	Description
4	The Readme file is missing any of the defined components and/or the submission file does not conform to the described format and/or the Source directory contains more than source code. The Readme file is not clear in describing the tasks completed and how to compile each. The WriteUp document does not highlight complications encountered and key steps in the problems solving process applied. The WriteUp subject matter jumps around, lacking flow.
8	The Readme file includes all of the defined components and the submission file conforms to the described format. The Readme file is clear in describing the tasks completed and how to compile each. The WriteUp document somewhat highlights complications encountered and key steps in the problems solving process applied. The WriteUp subject matter jumps around a bit, lacking flow.
12	The Readme file includes all of the defined components, the submission file conforms to the described format, and the submitted file's directory conforms to the described format. The Readme file is well organized and clear in describing the tasks completed and how to compile each. The WriteUp document highlights complications encountered and key steps in the problems solving process applied. The WriteUp subject matter is organized with a natural flow.
15	The Readme file includes all of the defined components, the submission file conforms to the described format, and the submitted file's directory conforms to the described format. The Readme file is well organized and clear in describing the tasks completed and how to compile each, including information regarding exceptions (if any). The WriteUp document highlights complications encountered and key steps in the problems solving process applied, and shares lessons learned. The WriteUp subject matter is well organized with a natural flow and does not exceed 1 page in length.

Completed Task(s) (max 89)

Points are earned for each task completed, with detailed breakdown of points earned by criteria as described for each task.

- The Required Task is a maximum of 65 points awarded
- The Advanced Tasks have a maximum of 9 points awarded
- Code organization points apply to all completed task(s).
- Extra Credit is a maximum of 5 points awarded.

Points	Task	Description
65 Max	Required Task	0-10 = Program runs and processes the Required Task input file bug free. 0-45 = Required Task output file result is accurate as per success criteria. 0-10 = Program finishes executing within a time requirement with accurate output file result.
9 Max	Advanced Tasks	0-6 = Advanced Tasks output file result is accurate as per success criteria. (2 points maximum per advanced task) 0-3 = Program finishes executing within a time requirement with accurate output file result. (1 point maximum per advanced task.)
10 Total	Code Organization	0-2 = Source code compiles and program can run. 0-4 = Code is well organized and easy to read. 0-2 = Code is well commented. 0-2 = Code used meaningful variable names.
5 Total	Extra Credit Task	0-5 = Create a command line interface to add, delete, or modify your shop's menu. Please be certain to clearly document your extra credit in the write up.

104 total available points. Consideration for grade level is incorporated into judgment of rubric criteria.