ASSIGNMENT 1

Advanced Programming

TABLE OF CONTENTS

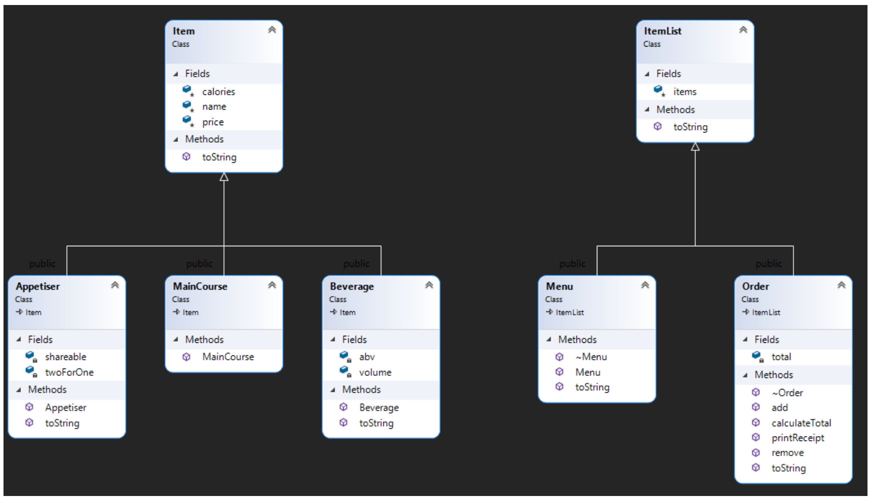
JACK HARTMAN
19702197

APPLICATION DESIGN

DESCRIPTION

The program that I have created allows for a CSV file containing all the items in a menu to be read and sorted into separate objects and into one list using inheritance and polymorphism, once the items have been initialised into the list they can add/remove to an order list by the user through commands. Once the user has added all the items, they want to the order they can check out the order which will print a receipt to a text file which all the items in the order and the final total with discounts displayed.

IMPLEMENTATION



The class structure within the program has followed the brief exactly with additional functions in the order class for adding and removing functions for the items to the order. All the Item types are derived from the Item class and the Menu and Order classes are derived from the Item List class to keep in line with the brief.

Inside the abstract classes, I used virtual classes on the toString methods to allow them to easily be accessed through the derived classes.

In the program where the classes were located I used methods and constructors to separate and call code when they are needed, this is also to prevent repeating of code, for example, the toString methods in the classes were to allow it to be called from different commands instead of having to have the same code in a file multiple times. I decided that keeping the methods in the header file would be better for me to organise and keep track of where things are since there weren't too many methods however if there were more I could have used more CPP files to further separate the methods from the classes to make it easier to read.

While creating my program I kept testing each part I worked on before moving on to make sure it worked, this way I could keep on top of any errors that could arise, this also made it easier to diagnose what could be causing an error which would reduce my debugging and fixing time and possibly reducing the amount of time to create and finish the program. To check for errors I used breakpoints to check through each variable I am testing to make sure they have the correct value at the point of testing, this way I could see where the variables are being set wrong to make it easier to fix.

I used operators to sort the items by price and sort two different integer inputs, I did this by using a sort function and the operators < and > this checks to see if the value next to them is higher or lower and depending on the result it will move the item a certain way to result in an ascending or descending order.

For printing and sorting each item in vectors in multiple functions I opted for using for loops to cycle through each item to either sort, get the price, evaluate, and store values which work effectively when working with single loops. While looping for each item I used points to refer to the objects in ram, this way I was not using more ram to copy the object I would just refer to the original object already stored in ram in the menu thus reducing the amount of ram I used.

ADDITIONAL TASKS

From the brief, there are three different additional tasks of which I completed all of them.
For the first task which involves allowing users to add and remove multiple tasks at once, I passed through the parameters after removing the command from it to pass through all the item IDs in an easy to use format, for the removal I had to work it a little different as after I checked that all the inputted values were valid I sorted them in decreasing order to make it remove the items closest to the end so I didn't end up trying to delete an item that is no longer in that position.

The second task involves calculate total function to include the 2-4-1 discounts and add the items which cost the lowest first, to do this I looped through all the 2-4-1 items and added the prices to a vector while counting the number of eligible items in the order. Once counting through the order, I looped through the collected prices after sorting them into ascending order to pull the cheapest ones first this was the cheapest items are always first by sorting.

The final task involves adding extra commands to the menu to sort in ascending and descending order, I did this by getting all the items and sorting them into their own categories to keep the menu tidy and split into organised categories. After splitting the items up, I sorted each category by price depending on the user input using the sort function then once sorted it I start to add all the items to the end message.

EVALUATION

TEST CASES

I tested my program by running through each input and checking the output to make sure it was correct with what the expected output should be and to bug hunt for unexpected errors, for example checking the two for one discount worked I added two items which are compatible and tested it applied the discount.

Test Case	Input Values	Expected Output	Passed
Add Appetiser to order	add 1	Appetiser object got from menu and added to items vector in order.	Y
Add Main Course to order	add 4	Main Course object got from menu and added to items vector in order.	Y
Add beverage to order	add 8	Beverage object got from menu and added to items vector in order.	Y
Remove item from order	remove 1	Removes object in position 0 of items in order	Y
Add multiple items	add 1 2 3	Collects objects from menu in positions then add them to order items vector	Y
Check 2-4-1 discount	add 2 2	Adds two appetisers to order then calculates new total with discount applied	Y
Check checkout works as intended	add 2 2 6 checkout	Gets items from order then writes to text file and finishes program	Y
Remove item from empty order	remove 1	Program checks order list to see if it's over 0 if not it displays error message	Y
Checkout with empty order	checkout	Program checks order list to see if it's over 0 if not it displays error message	Y

TIME COMPLEXITY

Since there aren't any many nested loops all loops are mostly O(N), however during the reading of the menu file I have a nested loop for reading each split string which has a time complexity of O(N^2) which works effectively for the program.