Multi-paradigm Programming
Shop Assignment

David Sheils – G00364696

Introduction: Objectives of the assignment

The purpose of this assignment is to model a shop, based on the shop program(s) developed in the lecture series "Multi-Paradigm Programming".

In particular, the aim is to extend the functionality as follows:

The csv from which the shop is to be created must have an opening cash balance

- There should be two methods of reading in a customers order:
- CSV mode. In this mode, customer details will be loaded from a CSV.
- Interactive mode, in which the customer order is entered via the console/terminal.

After processing the order, the cash balance of both the shop and the customer should be updated.

This functionality should be implemented (ideally identically) in C and in Java.

The key learning of this assignment should be demonstrated, comparing the approaches in the object-oriented solution (in Java) and in the procedural solution (in C).

Business Logic and overall operation of programs

On running of the program, the **shop will be created and populated** from a csv file ("stock.csv")
This is done by reading the first line, which is an integer into the variable that will contain the opening cash balance of the shop. This variable will be updated (i.e.) increased after each transaction.
The program will then **get the customer details**. The user is offered two modes by which this task may be accomplished:

1. "**CSV Mode**". In this mode, the customer details are read in from a csv file ("orders.csv"). The first line contains two values, a string representing the name of the customer and the value (as a double) of the customers opening balance. The remaining lines will contain the names of the products and the quantity which they wish to order. These are read into the variable/object that will represent the customer details.
2. "**Interactive Mode**". In this mode, the user plays the role of the customer, entering their name and their cash budget, when prompted, then entering the name of the product and the quantity of which they wish to purchase. The customer may enter up to 10 items; if they are requesting less than 10 items, they type "end" (all lowercase) to exit and to allow the program to process their order.

Please note that it is assumed that (a) the customer does not know the price of any item they wish to purchase or (b) whether the item is in stock. Where an item is in stock,

however, it is essential that (in both modes) the entered product name must be spelled exactly as it is in the stock list ("stock.csv").

The **processing of the customer's order** is done as follows:

1. The customers opening cash balance is read.
2. The program then cycles through each item in the list of items the customer has ordered.
3. The program checks the name of the item against the name of each item in its stock (hence the necessity for an item to be spelled/entered exactly as it is spelled in the stock csv. If the item is found:
4. The program will then check that the quantity of the item the customer has sought is less than or equal to the quantity in stock. If the quantity sought is higher than the number in stock, the quantity that will be sold to the customer is capped at the amount in stock.
5. Then the program checks if the customer can afford the purchase of the item in the quantity required (capped where necessary), by checking the cost (i.e. the price of the item multiplied by the quantity sought) against the customers budget. If the total cost of the item exceeds the customers budget, the customer is told that they have insufficient funds for the transaction and the program moves to process the next item in the customers orders list.
6. After each item is processed, the customers budget is decreased by the cost of the item and the shops cash balance is increased by the same amount.
7. At the end of the processing cycle, the program will output the final shops cash balance and the final customers balance.

Implementation of Functionality in C

This implementation is a single program.

The main method of the program roughly follows the business logic outlined above.

The user is given a choice of getting the customer details either by reading the CSV or by entering details through the command line.

There are two separate subroutines/procedures to get the customer details:

- getCustomerLive()
- getCustomerCSV()

The program contains the following additional subroutines/methods/procedures:

- printProduct()
- createAndStockShop()
- printShop()
- printCustomer()

The Customer, the Product and the Product Stock (i.e. a product plus the amount of items in stock) are modelled using the *struct* composite data type. A struct may be viewed as a basic form of a class. However, a key difference is that when a struct is passed as an argument to a function, any changes to the state will not be reflected in the original variable. This is also the case with the ProductStock array, which is represented in the c implementation as a Struct composed of (up to) 20 Product structs.

The full code for the c implementation can be viewed here :
https://github.com/davesheils/multiParadigmProgramming2019/blob/master/projectFinalForSubmission/shopInCFinal.c

Implementation in Java

Although the functionality is almost identical, the implementation in Java is quite different. Rather than a single script the Java implementation is created from a set of Java objects that interact with each other. The objects are:

- Runner - The main method for the program
- Shop – The class that models the shop
- NewCustomer – The class that models the customer
- Product - The class that models the product
- ProductStock – The class that contains a product and the quantity in stock.

A class in Java and other object oriented programs may be viewed as a C-style struct, in that it is a data-type that is constructed of other data types (including classes). However, classes are more than just a data type. One crucial difference is that classes can have a built in functions/methods which can be designed to update the state of the class ("setter" methods) or provide information about it's state ("getter methods"). Therefore, rather than having external methods/procedures, the state of the shop and the customer classes are updated by methods internal to the class:

Examples from the java implementation:
Shop.getCash()
Shop.processOrder()
NewCustomer.getDetailsCSV()
NewCustomer.getDetailsLive()

etc.

Whereas the elements of a class are accessible from any part of the program (e.g. product.name = "Catfood"), access to the elements of a class can be controlled by the building (or not) of getter or setter methods. Finally, and very importantly, a class will maintain it's updated state when passed to a function. So, for example, shop.processOrder(customer) will return a customer with an updated closing balance after the order has been processed.

The code for the java implementation can be viewed online at:

https://github.com/davesheils/multiParadigmProgramming2019/tree/master/projectFinalForSubmission/javaFiles