Adam Clark
Senior in Computer Science @ USM
adam.clark2@maine.edu

Write-Up for Dev Challenge

Description of the problem

My task is to develop a point of sale system for a small business coffee shop. The system should take orders at a minimum. To solve this problem, I used C# and GitHub. My code is stored online here: https://github.com/adamclark2/CoffeeShop-POS it also displays the ReadMe file in a nice graphical layout.

User Stories

Organizing the problem into several user stories helps people understand the problem better and enables better time management. My user stories are:

Allow a cashier to order a drink from the system.
Allow a cashier to order multiple items from the system (foods & drinks).
Allow a customer to retrieve a receipt from the system in .json format.
Allow a cashier/customer to get a list of all the things on the menu.
Allow a cashier to enter payment information as cash, credit card, or check.
Allow a cashier to add and remove things from the menu.

Cool UX Features

Some programs have cool features that enhance the user experience. Some features I have are:

Allowing a customer to use multiple payment methods
Allowing a cashier to select “cash in full” when accepting payment, allowing them to avoid entering in the number.

Allowing a cashier to select “no payment” when an item is given to somebody to allow for receipt printing.
Allowing a cashier to enter multiple items in a single order instead of entering in multiple orders
Allowing the user to call a help command to get a list of available actions
Allowing the user to enter multiple commands without exiting and re-launching the program

Complications

*Discount to cash tenders not implemented
*Meal deals not implemented
*You can add a drink/food and have it without a size
    Implication: You have an item in the menu you can't order
    Quick Fix: Add a size via the command

*I also output the file `Adam.Clark.X.receipt.json` this is a copy of the receipt object
    I used this for debug but it might be useful to have, so I'm keeping it in here

*Automated Test Issues
    Tests in the makefile may crash the program if a tender isn't specified
    Tests I added work fine (On my system)
    Those issues only happen in the `make test` command, manually things work fine

Description of the problem-solving process

To solve this process, I used object-oriented programming and design patterns. Some of the patterns I used include:

Singleton: An object that has one instance for the entire program.
Example: DaoFactory
Why: The DaoFactory only exists once for the program. It is used to track an ItemDao so all parts of the program can access it. DaoFactory also uses the factory design principle.

Data Abstraction: Separate the data access logic from the model
Example: ItemDao and JsonDao
Why: The models are simpler to understand and new DataAccessObjects can be created to enable different data access methods. Let's say we wanted to use SQL instead of JSON, we'd create another DAO for it and the rest of the program would work fine.

Program Usage

Let's say you want to order a medium coffee with whipcream & flavorshot, a small coffee, and a bagel half with butter. You have the exact amount of cash and you want to exit right after. Run the commands:

```
order coffee medium whipcream flavorshot, coffee small, bagel half butter
3
Exit
```

See `examples/operations.1.txt`

Let's say you want to remove the drink 'coffee' run the command:

```
menu-remove-drink "coffee"
```

See `examples/operations.4.txt` for how menu-remove works

Let's say you want to get help information. Run the command:

```
help
```

See ScreenshotsAndExamples/help.txt for a sample output of this command

Let's say you want to add tea to the menu with 3 sizes and a new drink-extra "sweetener stuff" run the series of commands:

```
menu-add-drink "tea"
menu-add-drink-size "tea" "small" "2.50"
menu-add-drink-size "tea" "medium" "3.50"
menu-add-drink-size "tea" "large" "4.50"
menu-add-drink-extra "sweetener stuff" 2.50
```

See `examples/operations.5.txt` for how menu-add works for foods & drinks

Algorithms Used
Advanced algorithms weren't needed to complete this project. However, I did do some clever tricks in the code. For example, I used recursion to implement getting payment methods from the user.

I also used currying to add item to the menu. Currying is essentially taking a method and 'fixing' one of its parameters to make a new function. By currying the function
`public static void menuAddItemDelegate(string type, string args)`

I turned it into a:
`public delegate void ConsoleMenuDelegate(string args)`
which allowed me to use one function for multiple console commands. The code I'm talking about is in `DrinkFoodMenuMaintenance.addMenuOptions()`. The operator => is used to define a function inline. The function is called a lambda.

To use multiple console commands I used C#'s built in Dictionary. This is a key-value store. In this case the key was the command the user entered and the value was a function that can handle the user command. This removed the need for a giant switch statement in the code.