Int projectNumber = 5;

System.out.println(“Documentation for project” + projectNumber);

System.out.println(“OpenOrder”);

Ionel Radu Constantin E-Business, An 1, 2022.

Contents

[1.Introduction 3](#_Toc119100340)

[2. Explanation of each class. 3](#_Toc119100341)

[2.1 Menu class contains mostly static methods used to access the menu for the restaurant from a given text file. 3](#_Toc119100342)

[3. Functions 4](#_Toc119100343)

[3.1 getUserString() 4](#_Toc119100344)

[3.2 getUserInt() 5](#_Toc119100345)

[3.3 getUserInt(String gen) 5](#_Toc119100346)

[3.4 DisplayOptions() 6](#_Toc119100347)

[3.5 GetOpenTables() 6](#_Toc119100348)

[3.6 AskForOrder(int TableNumber) 7](#_Toc119100349)

[3.7 IsInteger(String str) 7](#_Toc119100350)

[3.8 addAllOrders(int[] ints) 8](#_Toc119100351)

[3.9 updateReport(HashMap<String, Integer> map) 9](#_Toc119100352)

[4. Class Order 9](#_Toc119100353)

[4.1 Payment(int tableNumber) 10](#_Toc119100354)

[4.2 WriteTO(ArrayList<String> list, int tableNumber) 11](#_Toc119100355)

[4.3 ToAdd(int tableNumber, String arg) 12](#_Toc119100356)

[4.4 ToOpen(int tableNumber) 13](#_Toc119100357)

[4.5 Clear() 13](#_Toc119100358)

[4.6 ToEdit() 14](#_Toc119100359)

[4.7 ToDelete() 14](#_Toc119100360)

[4.8 Display() 15](#_Toc119100361)

[4.9 isPaid() 15](#_Toc119100362)

[5. Main 15](#_Toc119100363)

[5.1 Orders 15](#_Toc119100364)

[5.2 Administrative 18](#_Toc119100365)

[5.3 Exit 18](#_Toc119100366)

[6. References. 18](#_Toc119100367)

# 1.Introduction

Sa se implementeze o aplicatie Java pentru gestiunea comenzilor din cadrul unui

restaurant, care presupune lucrul cu fisiere, vectori si colectii in Java. Se va asigura

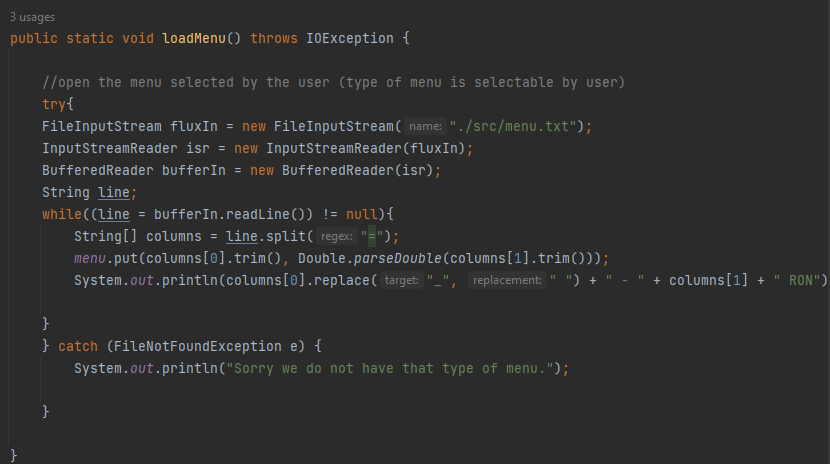
persistenta datelor, utilizand fisiere (si/sau baze de date). Aplicatia permite realizarea unor

situatii (rapoarte) privind valoarea comenzilor zilnice, cele mai vandute produse, etc

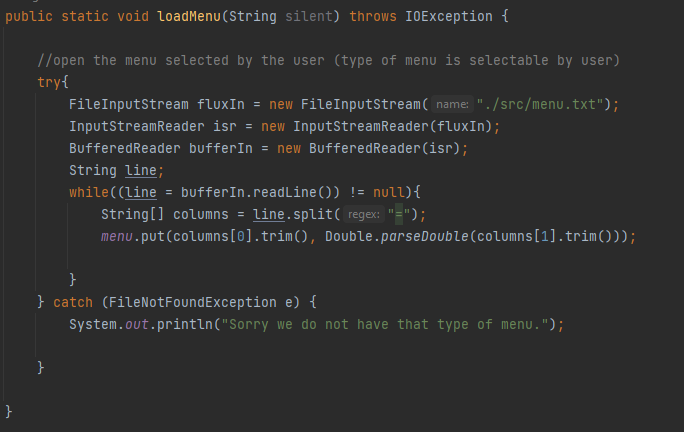
Implement a Java application to process orders for a restaurant. The main focus of the project is to convre

## 2. Explanation of each class.

### 2.1 Menu class contains mostly static methods used to access the menu for the restaurant from a given text file.



The method loadMenu() reads the menu.txt file and each line of the file is split into two parts, based on the “=” character. This results in an array of strings, containing the name of the food and its price. After, the information is stored in the menu variable, in the form of <key,value> pairs, in a HashMap. Lastly, it prints the information in a more user friendly way.

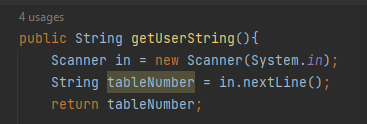


The second method is another loadMenu() but an overloaded method this time. The difference is that here, the menu is loaded but without it being also printed for the user at the same time.

## 3. Functions

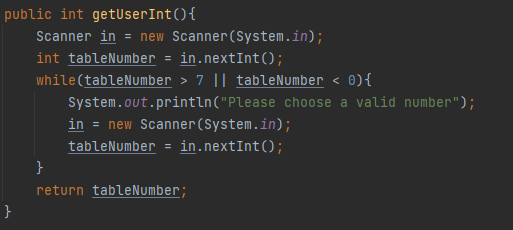
Functions is a class that stores different methods used across the program.

### 3.1 getUserString()



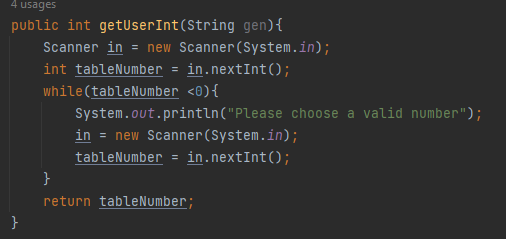
getUserString() is a method that returns a String with the user’s input. Because the program interacts with the user, I thought it would be helpful to have this method implemented, in order to get String input faster.

### 3.2 getUserInt()



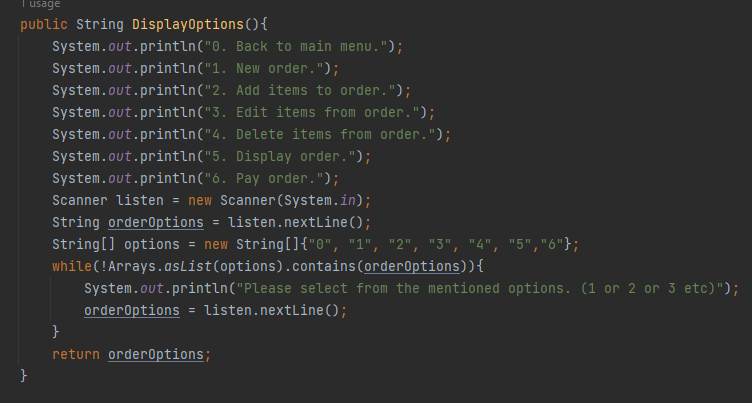
getUserInt() waits for the user to input a number between 1 and 6. If the user enters anything else, a message appears to guide the user to select a valid number. This method was also used in order to save time when interacting with the user.

### 3.3 getUserInt(String gen)



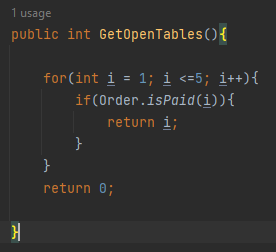
Another overload method, used to get integers from the user, that are not only between 1 and 6.

### 3.4 DisplayOptions()



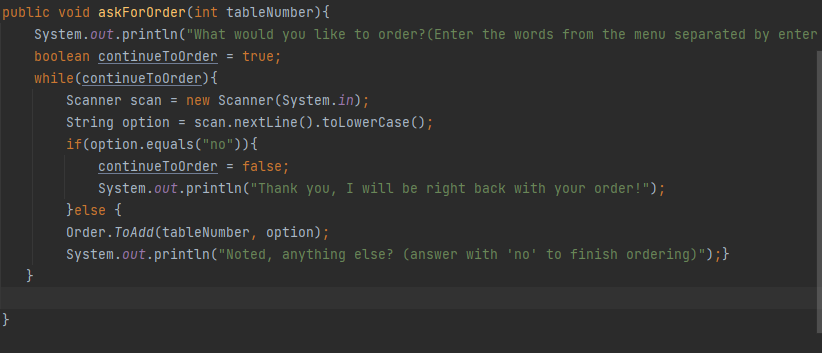
This method displays the options available in the first menu of the program (called Orders). The user can pick between options 0 to 6, and if they pick something else, the program asks them to choose from the mentioned options.

### 3.5 GetOpenTables()



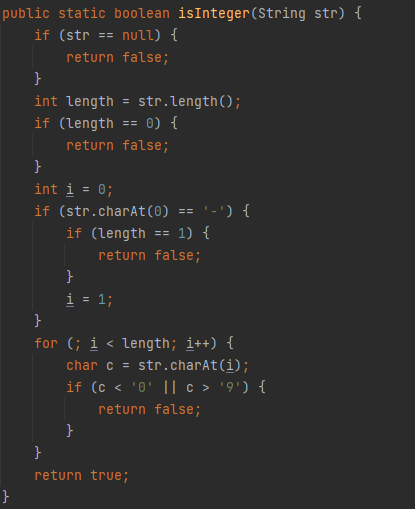
GetOpenTables() verifies if the existing orders are paid. If they are paid, the order is empty and another customer can order, otherwise the new customer will have to wait for an empty table.

### 3.6 AskForOrder(int TableNumber)



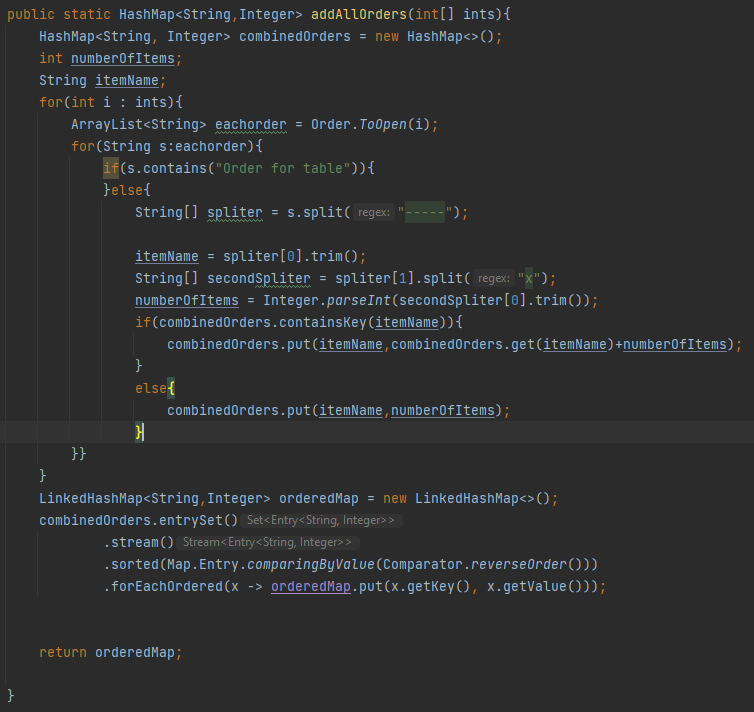
This method asks the users for input when a new order is generated. They can input as many items from the menu as they want, and in order to stop the order, the users will answer with “no” and the “food” will be registered on the order.

### 3.7 IsInteger(String str)



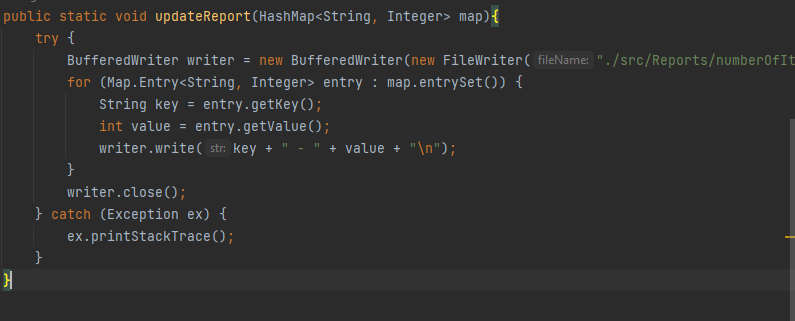
As the name suggests, this method checks if a string can be converted into an interger. There was no specific need for this method, as there are other ways to check the same thing, but I wanted to try the logic behind it to see how it works.

### 3.8 addAllOrders(int[] ints)



addAllOrders is a bit tricky. It takes a list of ints (the list contains the number of tables that we have). For each of the numbers in the int, it opens the files where the orders are. If any line contains “order for table” it is left alone, otherwise the string that is found is split into multiple parts in order to add all the items that were sold. Then, it stores those values in a HashMap in order to have a total number of orders for each type of food. Lastly, the HashMap is ordered and it is returned.

### 3.9 updateReport(HashMap<String, Integer> map)

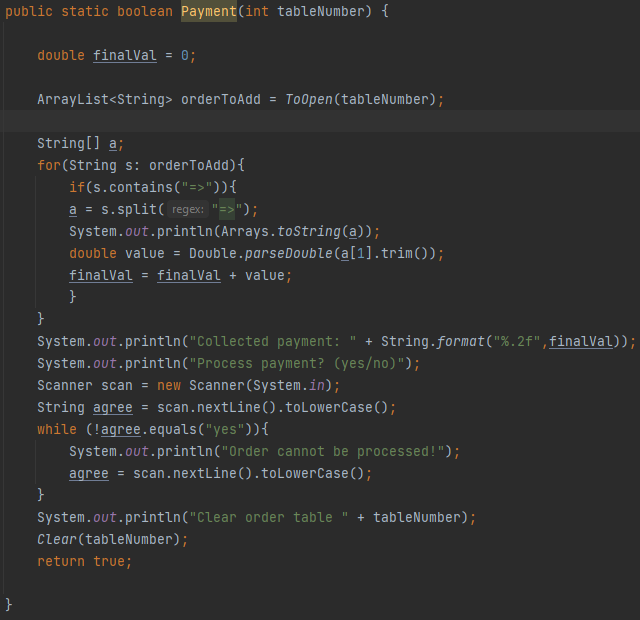


This method updates the report file with the latest information about the situation in the restaurant. The report contains the most ordered dishes, a useful information in order to stock up the restaurant as needed.

## 4. Class Order

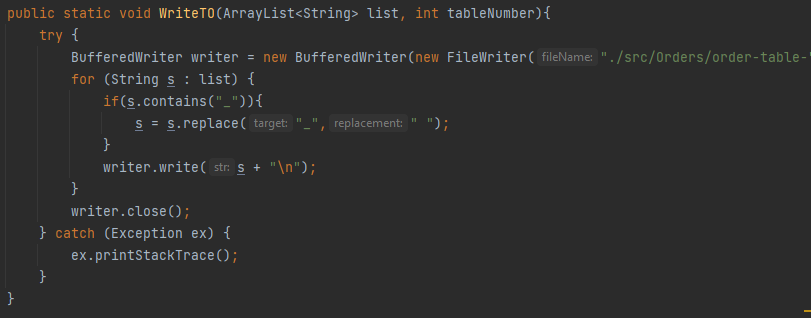
Here are a few methods that are used to manipulate the orders of the restaurant.

### 4.1 Payment(int tableNumber)



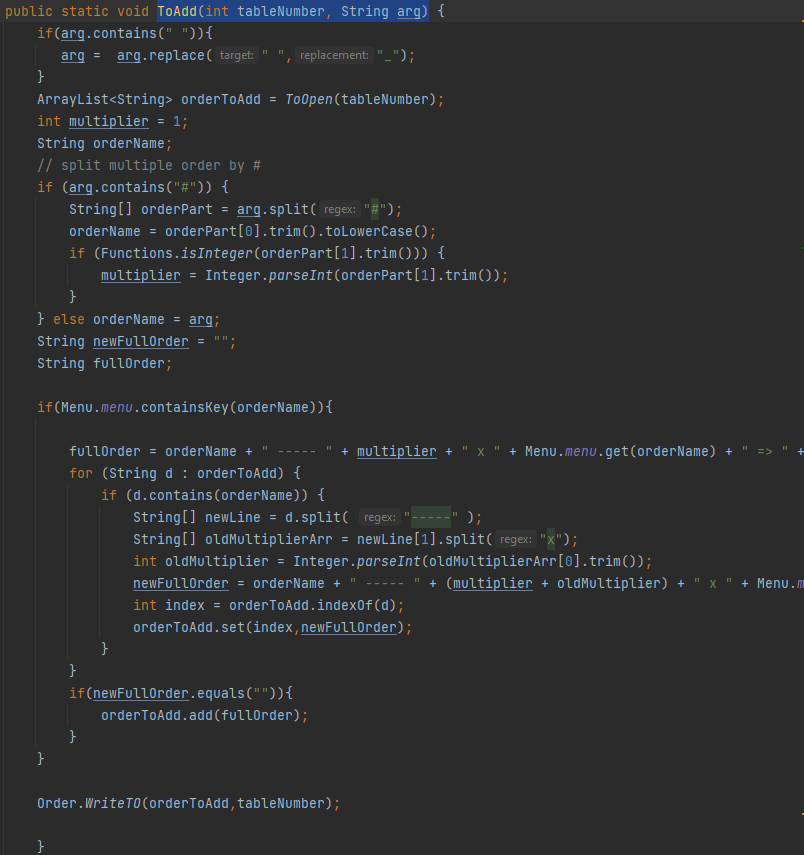
This method is used to simulate a payment. The order is loaded intro an ArrayList of type string, then by using string split methods we get the final value for the whole order. Then the system asks the operator if the user has presented the money. If the answer is yes, the order is cleared and the function returns true.

### 4.2 WriteTO(ArrayList<String> list, int tableNumber)



WriteTo method is used to write the information from a list inside a file.

### 4.3 ToAdd(int tableNumber, String arg)



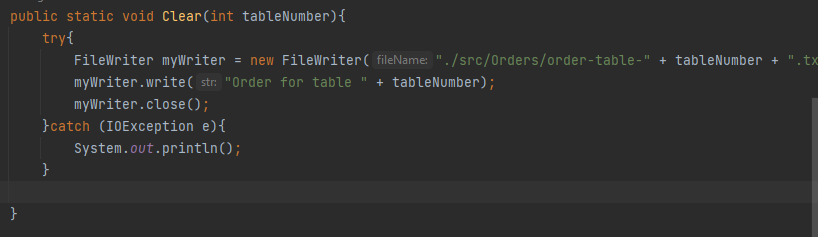
ToAdd is used to add items to an existing order. First the method checks if the argument contains any spaces. If it does, then the space is replaced with “\_” as this is how the menu understand the input. Then, the input is verified if it contains “#” as this symbol is used to introduce multiple orders at a time. For example hotdog#3 introduces 3 hotdogs at once. Then the function also adds these information to the order file of the table that was selected by the user.

### 4.4 ToOpen(int tableNumber)



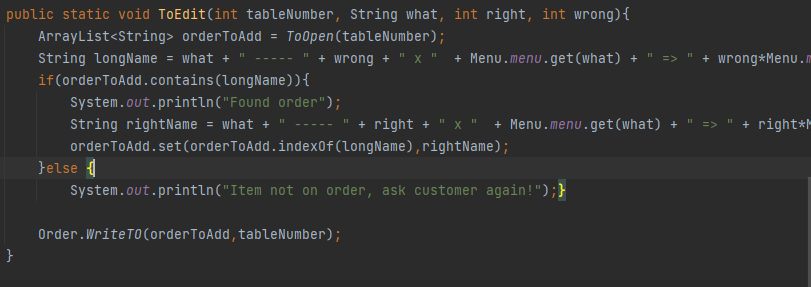
ToOpen is used to read the contents from a file and store them into a list for further use.

### 4.5 Clear()



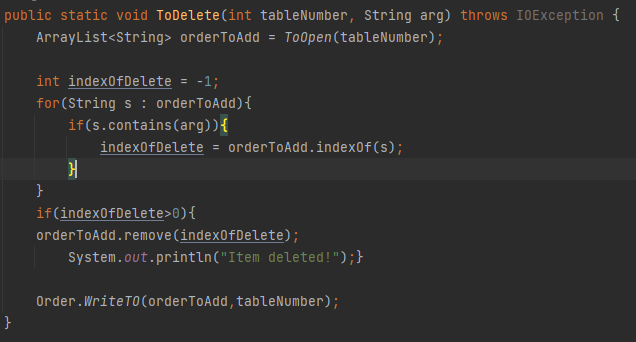
Creates a new order over the old one and enters the header containing the table number.

### 4.6 ToEdit()



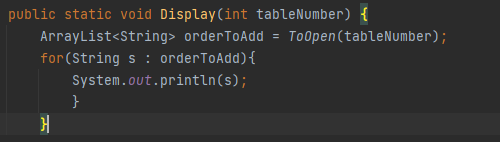
ToEdit takes 4 parameters. First is the table number that we want to edit, second parameter is what item we want to edit, third parameter is the right number that should be on the order and the 4th parameter is the wrong number that already exists on the order and needs to be modified.

### 4.7 ToDelete()

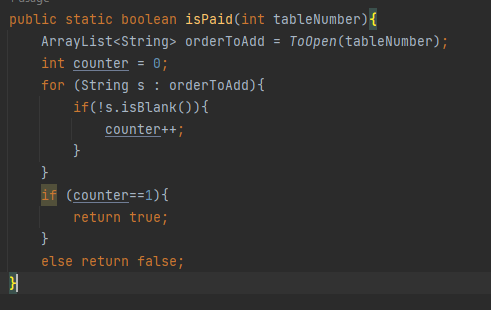


Used to delete a line from the order, by finding the index of the item that we want to delete.

### 4.8 Display()



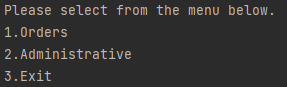
### 4.9 isPaid()



## 5. Main

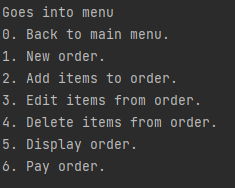
The flow of the program is simple.

The user is prompted with a simple 3 options menu.



### 5.1 Orders

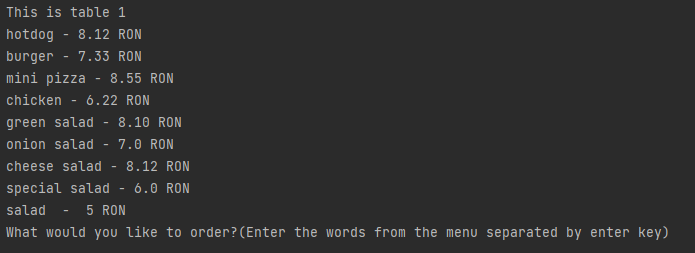
If the user selects “1” the next menu is displayed.



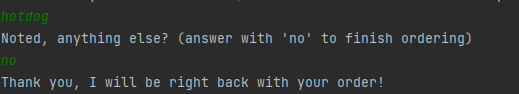
Here the user has 7 options.

#### 5.1.0 Choosing 0 simply returns the user to the first menu.

#### 5.1.1. Option 1 is used when we want to place a new order. Here the menu is also displayed to see the items available for ordering. And also where the system found an empty table for a new order.

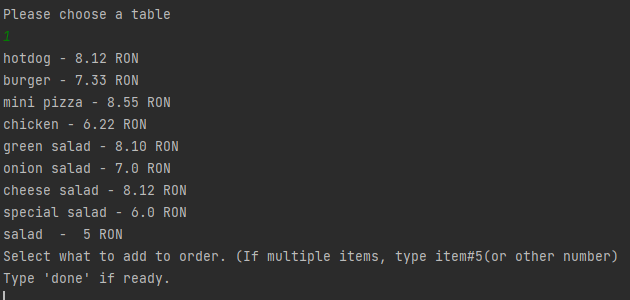


The users simply enter the items they want and if they want multiple items of the same type, the command is item#numberOfItems



After the users are done with the order, they type no and the menu goes back to the first part.

#### 5.1.2 Add items to order

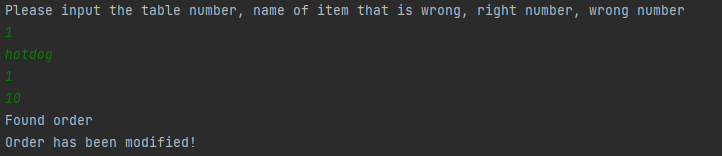


If the user wants to add more items to an existing order they use the Add Items to order option, with the same input as for a new order, but first they have to specify witch order to add to. Also the menu is displayed again to see what to add.

#### 5.1.3 Edit items from order



If the user ordered an incorrect amount of a certain item, they can modify the order here, all they have to do is input the table number that contains the mistake, what item has the mistake, what is the correct amount but also what is the wrong amount. After that, the system makes the changes.

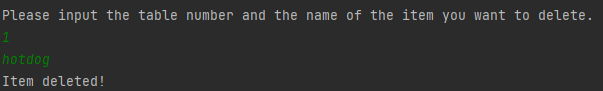


Here the user changes a typo from 10 hotdogs to 1.

#### 5.1.4 Delete items from order

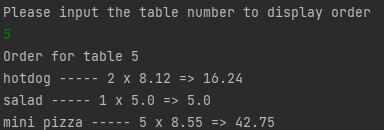


Here the user can choose the table and the item to delete from that specific table, by entering the table number and the item to be deleted.

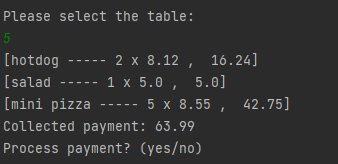


#### 5.1.5 Display Order

Simply displays the current order for the selected table.



#### 5.1.6 Payment



User is asked what table wants to pay its order. After the table is selected, the total for that table is calculated and the user is prompted with a confirmation for payment.

After selecting yes, the table is cleared of all items.

### 5.2 Administrative

In this menu, the program gathers all items from all orders and also how many times each item has been ordered. Then it saves the report in a file on disk, ordered.

### 5.3 Exit

Simple, the program exits.

## 6. References.

1. <https://github.com/cristianciurea/PPOO2022>

2. Java A Beginners Guide, 9th Edition (Schildt, Herbert)

3. Coding with John (Youtube Channel) <https://www.youtube.com/c/codingwithjohn>