Joseph Vetri

Lab #1

COM 210 S01

Lab Report

For the first lab, Cassandra and I were partners. Lab 1 consisted of 3 three basic java programs which build upon each other. Cassandra and I were able to complete problems 1-3 of the lab with minimal issues. Problems 1-3 were beneficial for jogging our memories and getting the java juices flowing. Problem 4, unlike the others, is a question that challenges our knowledge of COM210 thus far. Truthfully, having us compute time complexities helped me learn how to calculate more than the lecture (Missed part of it). I learn best when I see someone else do something, have the reasoning behind it explained, then attempt to do it. This lab fit that criteria.

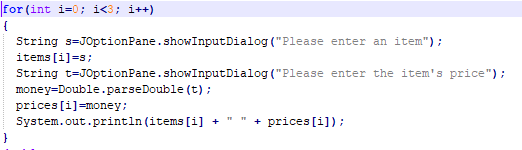
**Problem #1:**

Cassandra and I started thinking about how to handle problem 1 and had to decide between creating an object to represent food, or doing everything through the main method. We both decided that the main method approach would be more practical for this scale of a program. We both acknowledged that it may affect the time complexity of our algorithm, but I stand by the decision.

Our program starts out with two arrays set to a size of three. Array items represent the names of the grocery item, which is why it is type string. The second array prices, takes type double because it will store the price of the grocery item, and money should be able to be represented with cents as well as dollars.



The variable money is originally set to 0, I suppose we could’ve just declared it without setting a value. Later in a for loop money will be set to the amount the user types in, and that variable will then be stored in the prices array at index i.

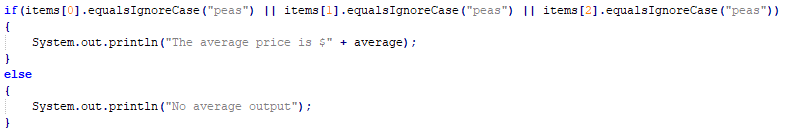


This for loop iterates 3 times, Using JOptionPane the user enters an Item. Since the item is type String we do not have to parse. Using s as the variable that holds the user input from JOptionPane, s is then passed to index i, which is 0 in the first iteration. The same scenario then happens for String t. This time we used a double, so variable t is parsed and money is set to the value of t. Money is then passed to index i of prices, which is 0. Writing this report I see a redundancy in money, and t. Items and prices are then printed at index 0/i with spacing for visual appeal. This same loop iterates twice more, with the only change being in the value of i.



Calculates the average by adding the value of all indices of prices and dividing by 3. The average is then printed.

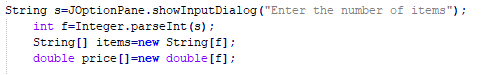
**Problem #2:**

For the second problem there wasn't much consideration of the best path to choose. We saw an if statement as the only option. Maybe a switch would have worked but I haven’t really got any experience with them.

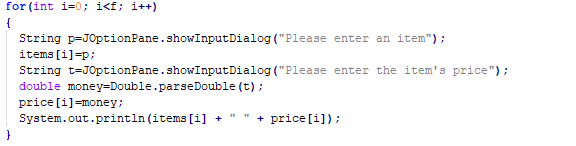
If array items at index 0,1,2 equals peas then calculate the average. We used the equalsIgnoreCase method to solve the case sensitive issue. Also looking back we were supposed to use the String.equals method (should've caught that). Else, print no average. This code is exactly the same as problem 1, except for the if statement.

**Problem #3:**

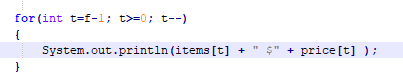
This code required some more consideration. I suggested to Cassandra to use a for loop iterating backward to solve the problem of the array being output in reverse order. Cassandra handled the issue of setting the array to the size of user input.



Using jOptionPane the user inputs the number of items. String s is then parsed into an int and int f is set to the value of s. The two arrays instead of being size 3 are now set to size f.



This for loop is an error by Cassandra and I. This loop should’ve been the one in reverse in my opinion. Cassandra handled all the coding, I said what we should do and she was the one coding, due to a miscommunication, our program prints the output 3 times, then the average if it contains peas, then the reverse output. This is the same for loop as problem 1 with slightly different naming. It is worth noting that in this loop i is set to less than f.



This loop iterates using t, which is f-1 (user input amount - 1). The -1 exists due to how an array at size 3 is size 3, but starts at 0, not 1. If the user typed 5 for f, the loop starts and prints index 4 of the array, de- incrementing by 1. Each iteration of the loop prints the items and price in reverse.

**Time Complexity**

Where **C1** is an assignment

Where **C2** is a request

Where **C3** is a print

Where **N** represents the amount of instances

**Problem 1:**

Equation: 3**c1** + **n**[2**c2** + 3**c1** + **c3**] + 2**c1** + **c3**

Simplified 5c1 + n[2c2 + 3c1 + c3] + c3

**Problem 2:**

Equation: 3**c1** + **n**[2**c2** + 3**c1** + **c3**] + 2**c1** + **c3**

Simplified 5c1 + n[2c2 + 3c1 + c3] + c3

**Problem 3:**

Equation: 3**c1** + **c2** + **n**[2**c2** + 3**c1** + **c3**] + 2**c1** + **c3** + **n**[**c3**]

Simplified: 5**c1** + **c2** + **n**[2**c2** + 3**c1** + **c3**] + **c3** + **n**[**c3**]

Cassandra was a very big help to wrap my head around time complexity. I feel like the way I understand it is flawed though. This equation we made has some flaws (I don't even know, it just seems weird). For instance there is no difference between 1 & 2, despite the if statement being there. I definitely feel better about them in general though.

According to our equation though, it would suggest that problem 1 & 2 have the same complexity and that problem 3 has the most complexity. Which is half true and half wrong, to me atleast. Obviously our largest problem would have the greatest time complexity, but problem 2 has more logic than problem 1, which HAS to make it more complex. Or at least in my mind. That is something I’d love to talk about in class if you read this and remember. I feel really good about this first lab, and am very happy with my partner Cassandra.