GROCERY STORE MANAGER
PROPOSAL

James Ticatic

Nicholas Board

CSC316: Data Structures for Computer Scientists

jrticati@ncsu.edu

ndboard@ncsu.edu

North Carolina State University

Department of Computer Science

January 20, 2017

Black Box Test Plan

For my black-box test cases, I will use the following test files:

customer.txt:

C12345678, Food Kitty, NC, 27695

C98765432, Ferris Teeter, SC, 24981

C55112233, Croger, VA, 23083

product.txt:

Wolf ice cream

Avent-Ferry granola

Pack chocolate milk

output.txt:

C12345678, Food Kitty, NC, 27695

C55112233, Croger, VA, 23083

C98765432, Ferris Teeter, SC, 24981

To start the program, run GroceryStoreManagerUI.java

Test ID	Description	Expected Results	Actual Results
testLoadInvalid Files (DT)	The files invalid_customer.txt and invalid_product.txt both do not exist. The user starts the software from the command line with the files invalid_customer.txt and invalid_product.txt.	The software prompts the user to retry and use valid file names.	
testGenerateSorted CustomerList (ECP)	The files customer.txt and product.txt both exist and follow the format shown above. The user starts the software from the command line with the files customer.txt and product.txt. The user chooses the menu option to generate a list of customers sorted in ascending order by customer ID.	A txt file is generated following the format output.txt shown above.	
testRetrieveProduct Information(BVA)	The files customer.txt and product.txt both exist and follow the format shown above. The user starts the software from the command line with the files customer.txt and product.txt. The user chooses to retrieve information and enters Brand: Wolf Product: ice cream	The program returns brand: Wolf product: ice cream Number of occurences: 1	
testRetrieveProduct NoBrandName(DT)	The files customer.txt and product.txt both exist and follow the format shown above. The user starts the software from the command line with the files customer.txt and product.txt.	The software reprompts the user to enter a value for brand.	

	The user chooses to retrieve information and does not enter Brand and enters Product: ice cream		
testRetrieveProduct NoDescription(DT)	The files customer.txt and product.txt both exist and follow the format shown above. The user starts the software from the command line with the files customer.txt and product.txt. The user chooses to retrieve information and enters Brand: Wolf and does not enter product	The software reprompts the user to enter a value for description.	

Data Structures

For this project I will use the following data structures:

> **Customer**: an array-based list that will resize when capacity is reached. An array-based list is a good option in terms of runtime efficiency because customers need to be added to the end which is constant run time for array-lists and sorted which is O(n) for array-lists when using bubble sorts which is about as good as it gets for sorting.
> **Product**: an array-based list that will resize when capacity is reached. An array-based list is a good option again because products need to be added to end which is constant run-time and looked up which is O(n) since we have to count the number of occurrences so we have to travel through the entire list every time we do a look up.

For the list for Customers, I will have to add Customers to the list and sort them based on customer ID. For the list of Products, I will have to add Products to the list and retrieve products contained in the inventory as well as the number of times the product appears in the file. Therefore, I will design my array-based list to use the following operations:

- addProduct: adds a product to the end of the list.
- addCustomer: adds a customer to the end of the list.
- sortCustomers: sorts list of customer in ascending order based on customer ID.
- lookUpProduct: retrieve the number of times the product appears in the file, if 0 is returned then the manager will produce a message that says the product is not in the file.

An alternate data structure with similar running time would be a singly-linked list that has a reference to the head and the tail. Having a reference to the tail will make adding to the end of the list constant run time and since sorting and looking up requires that you traverse the entire list anyways it will have the same runtimes as an array list would for these operations.

Operation	ArrayList	LinkedList with head & tail
addProduct	O(1)	O(1)
addCustomer	O(1)	O(1)
sortCustomers	O(n)	O(n)
lookUpProduct	O(n)	O(n)

Algorithm Design

Proposed algorithm

Algorithm sortCustomers(customers[])

Input: a list with n customers

Output: the list with customers sorted by ID

```
1   for i ← 0 to n-1 do
2       if title(n+1) comes before title (n) do
3           temporary ← n
4           title(n) ← title(n+1)
5           title (n+1) ← temporary
```

Algorithm lookUpProducts(products[])

Input: a list with n products

Input: a brand

Input: a description

Output: the number of times the product appears in the file

```
1   int counter = 0
2   for i ← 0 to n-1 do
3       if n(brand) ⇔ input(brand) and n(description) ⇔ input(description)
4           counter incremented
5   return counter
```

Algorithm Analysis

sortCustomers:

Line	# of operations	Description
1	1	Initialize i = 0
	n+1	Comparison of i < n-1
	n	i+1 on each loop
	n	i=i+1 assignment on each loop
2	1*n	Access title(n)
	1*(n+1)	Access title(n+1)
	n+1	Comparison title(n) and title(n+1)
3	1*n	Access title(n)
	1	Assign title(n) to temporary
4	1*(n+1)	Access title(n+1)
	1	Assign title(n+1) to title(n)
5	1	Access temporary
	1	Assign temporary to title(n+1)
Result	8n+9	

The estimated running time T(n) for the sortCustomers algorithm is: T(n) = 8n+9, which is O(n) as shown below using the limit test:

$$f(n) = 8n+9$$
$$g(n) = n$$

The limit as n goes to infinity of f(n)/g(n) will equal 8. The 9 is negligible and the n's will cancel. From the limit test, if c is between 0 and infinity, then f(n) is O(g(n)).
Since c= 8, T(n) = 8n+9 is O(n).

Line	# of operations	Description
1	1	Initialize counter = 0
2	1	Initialize i = 0
	n+1	Comparison of i < n-1
	n	i+1 on each loop
	n	i=i+1 assignment on each loop
3	1	Access brand input
	n	Access brand in array
	1	Access description input
	n	Access description in array
	n+1	Comparison of brands
	n+1	Comparison of descriptions
4	n	Increment counter
5	1	Return counter
Result	8n+6	

The estimated running time T(n) for the lookUpProduct algorithm is: T(n) = 8n+6, which is O(n) as shown below using the limit test:

$$f(n) = 8n+6$$
$$g(n) = n$$

The limit as n goes to infinity of f(n)/g(n) will equal 8. The 6 is negligible and the n's will cancel. From the limit test, if c is between 0 and infinity, then f(n) is O(g(n)).
Since c= 8, T(n) = 8n+6 is O(n).

Software Design

This UML follows the MVC design which has all the data handling separated from the user interface. This simplifies the program by placing all the business logic in different classes from the user interface. In addition, it follows the Singleton design so that there is only one GroceryStoreManager when the program is running. It is necessary to use Singleton so that multiple managers aren't accidentally created for some reason which can crash the program.