VNUHCM-UNIVERSITY OF SCIENCE

FACULTY OF INFORMATION TECHNOLOGY

CSC10003 – OBJECT-ORIENTED PROGRAMMING

OOP-Lab: Assignment 06

Lecturer
Mr. Nguyễn Lê Hoàng Dũng
Mr. Hồ Tuấn Thanh

Class
23CLC08

Student
23127226 – Nguyễn Đăng Minh

October 29th, 2024

Contents

1 Introduction

This report explains how the console application program works similar to the shopping cart in Shopee. The image below is the shopee cart.

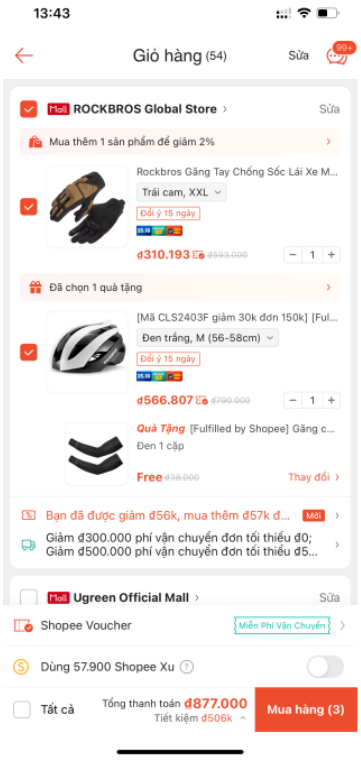

Figure 1: Shopee cart

My console application has some functions for user to interact the same way as the Shopee cart.

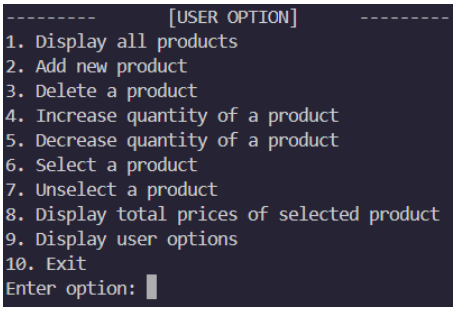

Figure 2: Menu of the program

2 How the program works

The program consists of four classes: Product, Node, LinkedList, and User.

2.1 Product

The Product class will contain all information about the product such as its name, shop, type, etc. Below is the UML of the Product class.

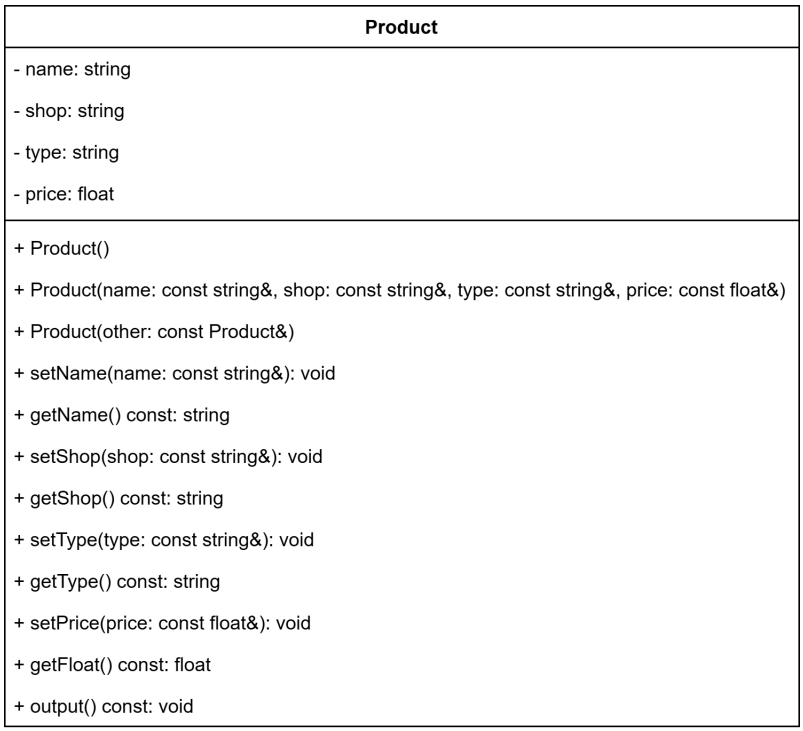

Figure 3: Product class diagram

The **output()** function is used to print the product's information.

2.2 Node

The Node class will contain a Product and a pointer to the next Node. Here I decide to use Linked-List structure to organize the product list.

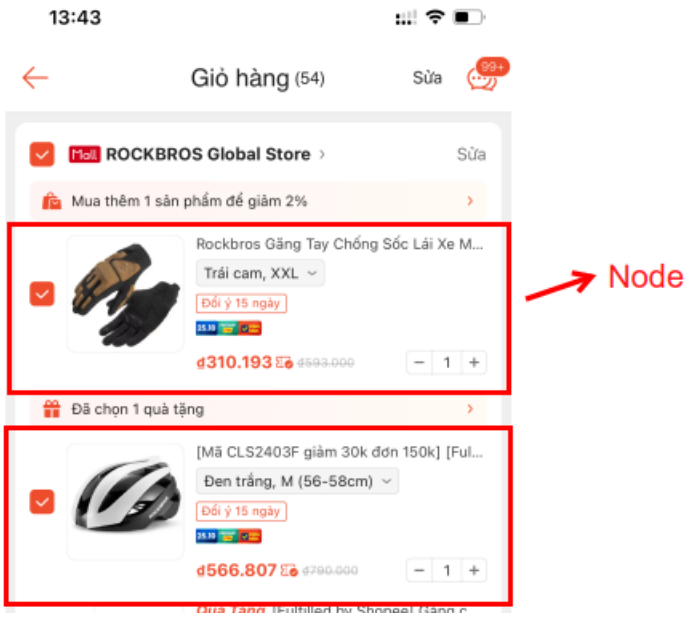

Figure 4: Linked List structure

Each block that contains a product, quantity, and selection state will be considered a Node in the Linked List. The Linked List will contain all the Node. The reason why I use Linked List is because it is easier and faster to delete an item. Since this is a small program so other operations such as traversal will not take so much time.

Node class will have the quantity, selection state, and the total price. Total price of a Node is the multiplication of product price and the quanity. Below is the UML class diagram of Node class.

Node
- product: Product - pNext: Node* - quanity: int - totalPrice: float - select: bool
+ Node() + Node(product: const Product&) + Node(product: const Product&, quanity: const int&) + ~Node() + setProduct(product: const Product&): void + getProduct() const: Product + setNext(pNext: Node*): void + getNext() const: Node* + setQuanity(quanity: const int&): void + getQuanity() const: int + adjustQuanity(change: const int&): void + getTotalPrice() const: float + setSelect(select: bool): void + getSelect() const: bool + output(id: const int&) const: void

Figure 5: Node class diagram

When user wants to update the quanity or selection state, the attributes in the Node class will change.

2.3 Linked List

The linked list will have a head and a tail of type Node* (pointer to Node). Its attributes also have num (amount of current element), totalPrice (total price of all nodes). In order to get the sum prices of only-selected nodes, there is a method **getSelectedPrice()** which only takes account into selected products.

Each Node in Linked List will have a temporary id (their current order). When delete a Node, the id of all Nodes after the deleted one will decrease one. The id purpose is for user to easily choose the product to adjust the quantity or delete it.

The Linked List class also has a **adjustQuantity(id: int, change: int)** to adjust the quantity of a Node at a specific id.

Below is the UML class diagram of Linked List.

LinkedList
- head: Node*
- tail: Node*
- num: int
- totalPrice: double
+ LinkedList()
+ ~LinkedList()
+ addNode(product: const Product&)
+ addNode(product: const Product&, quanity: const int&)
+ deleteNode(id: const int&): void
+ clear() : void
+ getNum() const: int
+ getTotalPrice() const: double
+ getSelectedPrice() const: double
+ adjustQuanity(id: const int&, change: const int&): void
+ adjustSelect(id: const int&, select: bool) const: void
+ output() const: void

Figure 6: LinkedList class diagram

Both LinkedList and Node has destructor to delete dynamic allocated memory.

2.4 User

The User class symbolizes the user interface, it contains the linked list and its operations based on the linked list.

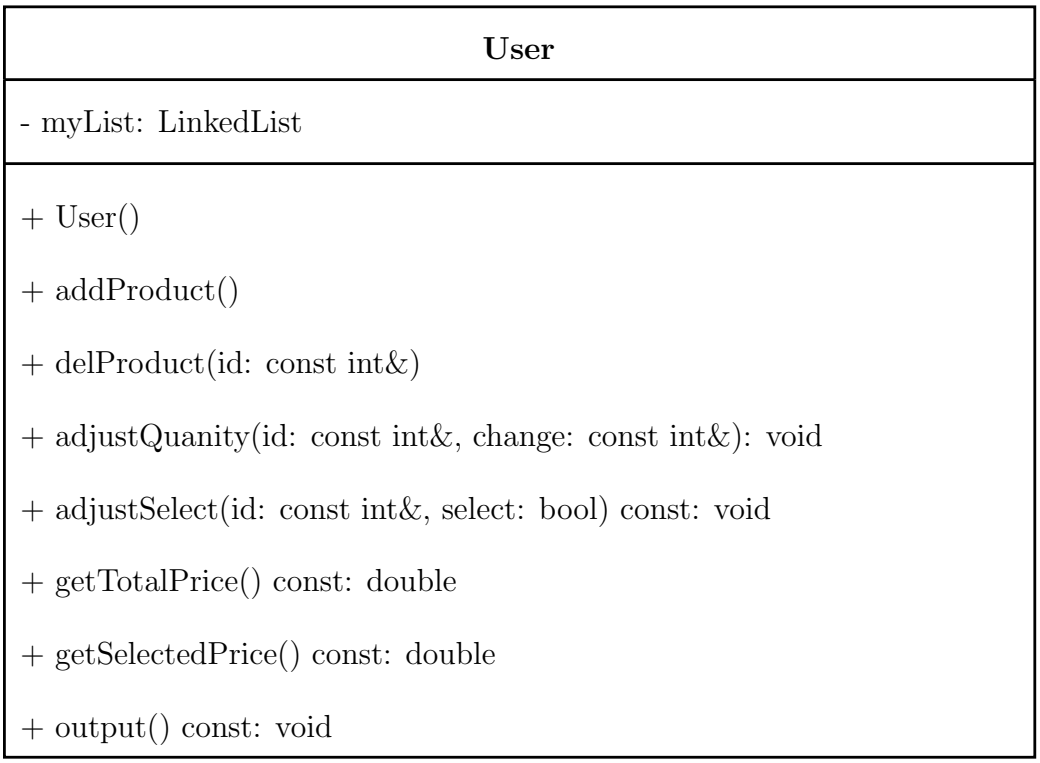

Figure 7: User class diagram

The User class has many functions that are as same as Linked List. It is for the user interaction, when user chooses to interact some function, the linked list will handle those. It is similar to separate abstraction and implementation, the user will not know the implementation below. They just need to know the interface (methods).

2.5 Demonstration of the program

The decrease-quanity, unselect function is as same as the reversed so I will not demonstrate them.

2.5.1 Menu

When open the program, you will receive a menu with ten options.

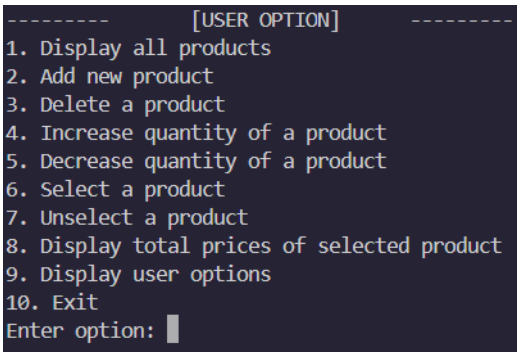

Figure 8: Menu

2.5.2 Add new product

Here is an example of how you can add a product and then display it to the screen.

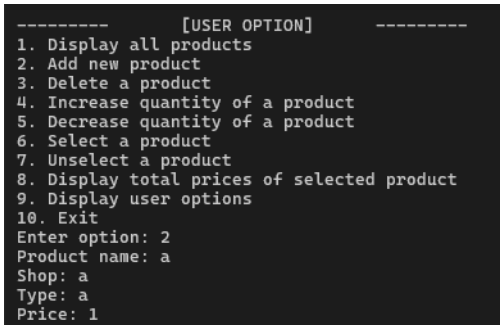

Figure 9: Add a product

Suppose I add two more products and then display it. The selection state of all products is initialize to be false (that is why price to pay equals zero). The quanity by default is also one.

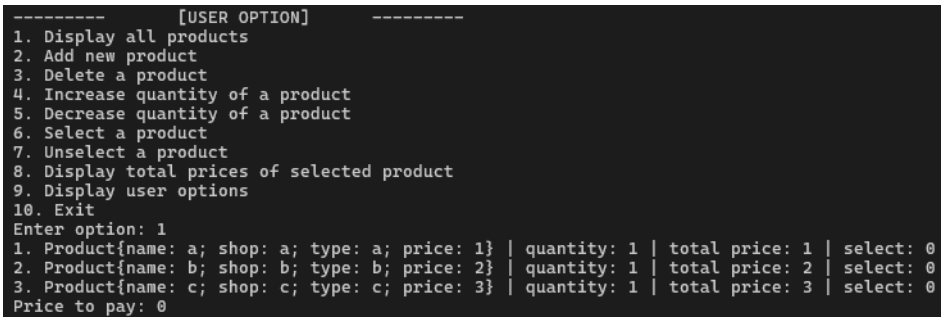

Figure 10: Display all products and their state

2.5.3 Select a product

To select a product, you need to type option 6 and the id of the product.

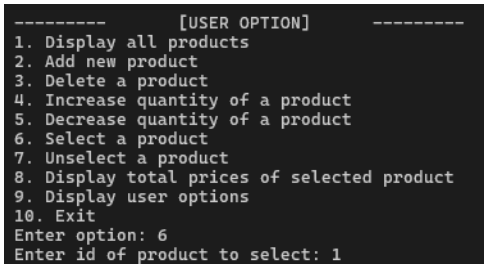

Figure 11: Select product with id 1 (name = a)

Now if I redisplay, the selection state of the first product (id = 1) is changed. Besides, the price to pay is also equal to the first product's price because it is selected.

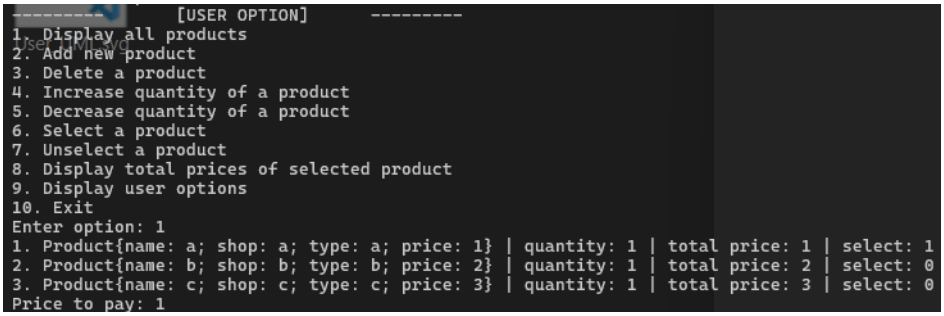

Figure 12: Selection state changed

2.5.4 Increase quanity

Type option 4 and product's id to increase the quanity of that product. Of course, if you type an invalid id (does not match any product), nothing will change.

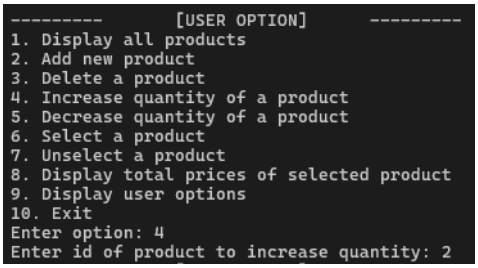

Figure 13: Increase quanity of product 2 (name = b)

Display to check:

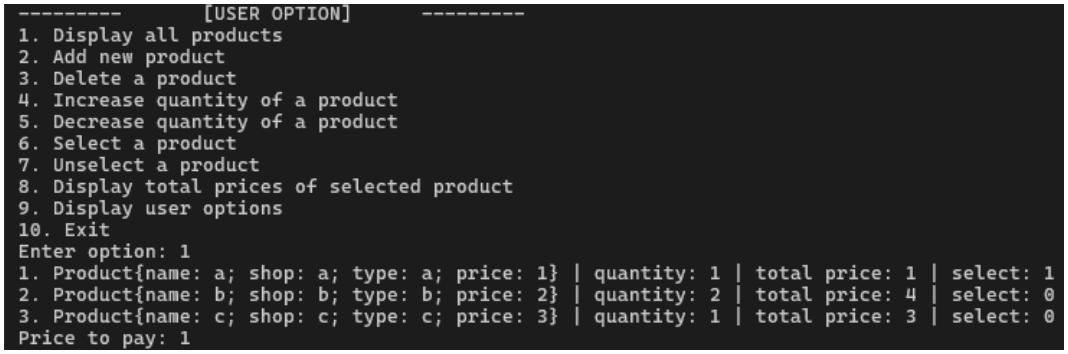

Figure 14: Quanity of product 2 is now two

Now if I select product 2, the price will be changed to 5.

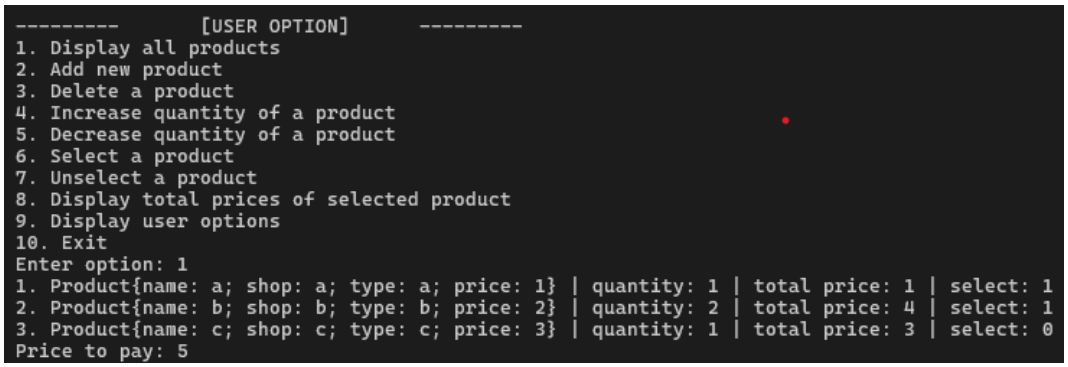

Figure 15: Total price now includes product 2

Price to pay has been update to 5.

2.5.5 Delete a product

Type option 3 and the product's id to delete that product. If you type an invalid id, nothing will change.

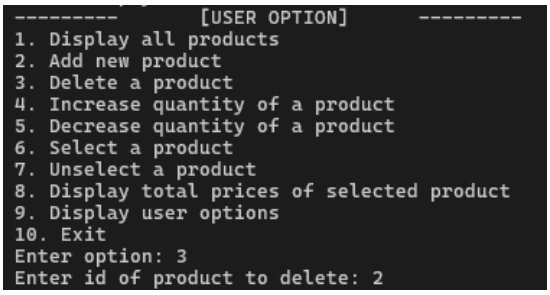

Figure 16: Delete second product

Display to check:

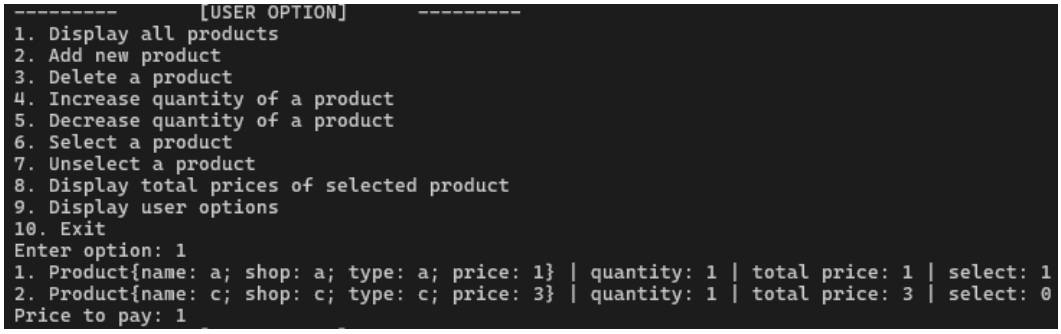

Figure 17: No more product 2 in list