Jason Goldfine-Middleton
932675265

CS 162 – Assignment 2 Reflection

Design Overview:

The goal of this assignment is to create a shopping list program that allows the user to add and remove list items as well as print the list to the console. Each item is required to have a name, a unit type, a quantity desired, and a unit price. The list display needs to contain all the user-added items with all four of these properties and the extended price (quantity * unit price). Additionally, the list display must show the total price of all the items. I am opting to allow the user to modify the quantity and unit price of each item in the list.

In order to achieve this task, I will enlist three classes: Item, List, and ListClient. The general purpose of each is as follows. Item will represent an individual item, containing a name, unit type, quantity, and unit price. List will represent a modifiable shopping list containing Item objects. ListClient will be called my the main() function and will repeatedly display a menu to the user and interpret the user's selections by adding, removing, or updating Items or displaying the List. If the user wishes to exit the program, he/she will do so through the ListClient. User input will be validated with the help of functions in the "utilities" namespace declared in the header file "utilities.hpp".

The Item class will contain the following fields:

- std::string item_name
- std::string unit
- int quantity
- double unit_price

The Item class will contain the following functions:

- Item()
- getters for each field
- double get_total_price()
- setters for the quantity and unit_price fields

The Item class will have two friend functions that overload the following operators:

- <<
- ==

The List class will contain the following fields:

- std::vector<Item> items
- double total_price

The List class will contain the following functions:

- List()

- double get_total_price()
- int length()
    - returns the number of different items in the list
- bool add_item(Item &it)
    - returns true if item is new and adds the item, otherwise returns false
- bool remove_item(int position)
    - returns true if the position is valid and removes the item at that position
- bool update_item_quantity(int position, int new_qty)
    - returns true if the position is valid and updates the item at that position
- bool update_item_price(int position, int new_price)
    - returns true if the position is valid and updates the item at that position
- [private] int position(Item &it)
    - returns -1 if an equivalent Item is not in the items vector or else returns its index

The ListClient class will contain the following field:

- List shopping_list

The ListClient class will contain the following functions (all void):

- print_list()
- display_menu()
- menu_selection()
    - displays the menu and does what the user requests
- do_users_bidding(int bid)
    - interprets the user's selection from the menu and selects the correct task to do
    - the user may exit the program if bid represents choice "exit"
- add_new_item()
    - adds a new Item to shopping_list
    - only adds if an equivalent Item is not already in list
    - user notified of success or failure
- remove_item()
    - removes an item from shopping_list
    - only removes if the user selects a valid item number
    - user notified of success or failure
- update_price()
    - updates the price of an item in shopping_list
    - only updates if the user selects a valid item number
    - user notified of success or failure
- update_quantity()
    - updates the quantity of an item in shopping_list

    - only updates if the user selects a valid item number
    - user notified of success or failure
- run()
    - loops menu_selection()

Looking at the relationships between the three classes, we can see that:

- ListClient HAS-A List
- ListClient USES-A Item
- List USES-A Item (in a way this could be considered a HAS-A relationship but since the field is a vector of Items, I am going with USES-A)

The main() function will create a ListClient object and call run() on it. Then the user will be able to operate on the List and manipulate Items through the ListClient interface. The ListClient never returns control to main() because the user can only exit the run() function via selecting to "exit" at the menu, which exits the program immediately.

Changes to the Initial Design:

Overall, there were not too many parts of the original design that had to be changed. One was how output was handled. Initially I planned to create print() functions for both the Item and the List classes. Later I realized that I could utilize operator overloading to output descriptions to a general std::ostream. This required creating an extra friend function for each of the two classes.

Another was how the List allowed Items to be removed or modified. In the original design, the mutator functions were passed in a reference to an Item, which was compared with the Item inside the items vector. Rather than forcing ListClient to produce new Item objects each time the user wanted to modify or remove an Item from the shopping list, I decided to let the user select a numbered item from the list. This number would be converted to the proper index of the items vector and passed into the remove_item(), update_item_quantity(), and update_item_price() functions.

I considered using pointers to Items in the original design but ultimately decided against it because creating new Item objects is fairly trivial and there is no need to reuse the same Item based on other design choices. Additionally, pointers leave a lot more room for error and require additional functions to be created in order to prevent memory leaks.

Jason Goldfine-Middleton
932675265

Test Plan and Results:

Testing Menu Selection			
Test Case	**Input Values**	**Expected Outcomes**	**Observed Outcomes**
Character entered	"a"	Print error and prompt again	As expected
String entered	"ascii"	Print error and prompt again	As expected
Input too low	0	Prompt again	As expected
Input too high	6	Prompt again	As expected
Select Print List	1	Display list and wait for user to press ENTER	As expected
Select Add Item	2	Display Add Item screen	As expected
Select Delete Item	3	Display Delete Item screen	As expected
Select Update Item Quantity	4	Display Update Item Quantity screen	As expected
Select Update Item Price	5	Display Update Item Price screen	As expected
Select Exit	6	Exit the program	As expected
Select multiple options	1, then ENTER, then 2, then ENTER, then 4	Display list and wait for user to press ENTER, display Add Item screen, let user enter info, wait for user to press ENTER, display Update Item Quantity screen	As expected

Jason Goldfine-Middleton
932675265

Testing Add Item			
Test Case	**Input Values**	**Expected Outcomes**	**Observed Outcomes**
Name contains spaces	"coffee beans"	No error message	As expected
Name contains no spaces	"soda"	No error message	As expected
Name is blank space	" "	No error message	As expected
Unit contains spaces	"6 pack"	No error message	As expected
Unit contains no spaces	"can"	No error message	As expected
Unit is blank space	" "	No error message	As expected
Quantity is a character	"a"	Error: invalid input, prompted again	As expected
Quantity is a character, then valid integer	"a", then 4	Error: invalid input, prompted again, no error	As expected
Quantity is a string	"test"	Error: invalid input, prompted again, no error	As expected
Quantity too low	-1	Error: valid range displayed, prompted again	As expected
Quantity is 0	0	No error	As expected
Quantity valid	5	No error	As expected
Quantity high	10000	No error	As expected
Price is negative		Error: valid range displayed, prompted again	As expected
Price is $0.00	0.0	No error	As expected
Price is positive	5.99	No error	As expected
Add first item to list	"beer", "can", 4, 1.29	Success displayed, displaying list shows item with correct values, including extended and total prices	As expected
Add second item to list	"eggs", "carton", 2, $4.05	Success displayed, displaying list shows item with correct values, including extended and total prices	As expected
Add item with	"beer",	Failure displayed,	As expected

same info as item in list, except for quantity	"can", 5, 1.29	displaying list shows no change in items	
Add item with same info as item in list, except for unit	"beer", "bottle", 4, 1.29	Success displayed, displaying list shows item with correct values, including extended and total prices	As expected
Add item with same info as item in list, except for price	"beer", "can", 4, 1.45	Success displayed, displaying list shows item with correct values, including extended and total prices	As expected

Testing Remove Item			
Test Case	**Input Values**	**Expected Outcomes**	**Observed Outcomes**
Nothing in list	None	Display nothing to remove	As expected
1 item in list, valid item selected	1	Display success, displaying list shows it is empty	As expected
1 item in list, invalid item selected	4	Display failure, displaying list shows no change	As expected
2 items in list, valid item selected	2	Display success, displaying list shows item has been removed and updated total price	As expected

Jason Goldfine-Middleton
932675265

Testing Update Item Quantity			
Test Case	**Input Values**	**Expected Outcomes**	**Observed Outcomes**
Nothing in list	None	Display nothing to modify	As expected
1 item in list, valid item selected	1	Display success, displaying list shows updated quantity, extended price, and total price	As expected
1 item in list, invalid item selected	5	Display failure, displaying list shows no change	As expected
2 items in list, valid item selected	1	Display success, displaying list shows updated quantity, extended price, and total price	As expected

Testing Update Item Price			
Test Case	**Input Values**	**Expected Outcomes**	**Observed Outcomes**
Nothing in list	None	Display nothing to modify	As expected
1 item in list, valid item selected	1	Display success, displaying list shows updated price, extended price, and total price	As expected
1 item in list, invalid item selected	16	Display failure, displaying list shows no change	As expected
2 items in list, valid item selected	2	Display success, displaying list shows updated price, extended price, and total price	As expected

Note that stubs were created for each function as the program was developed, with each being tested before moving on to the next. I began by coding the Item class, then the List class, and finally the ListClient class. Thanks to careful planning, I was able to robustly handle all user input.