**1.Design document**

**Introduction:**

When we go shopping in real life, we usually take a cart where we keep the items we like, and at times we pick multiple items of the same type. We can also discard items at or before checkout if you do not deem them necessary. This program basically simulates that same experience where you can add, remove, and modify your shopping to your needs, and finally, the program generates the total cost of all the items in your cart.

The program asks the user for inputs such as the name of the item, the price of the item, and the quantity. It also asks the user if they want to add more items and modify their cart with yes or no questions.

**Data Structures:**

In this project I used four classes 1) item. 2) ShoppingCart, 3) Bag. 4) BagInterface to implement the desired output. The BagInterface class was already given to us, and the shopping cart class is a derived class of the item class.

**Functions:**

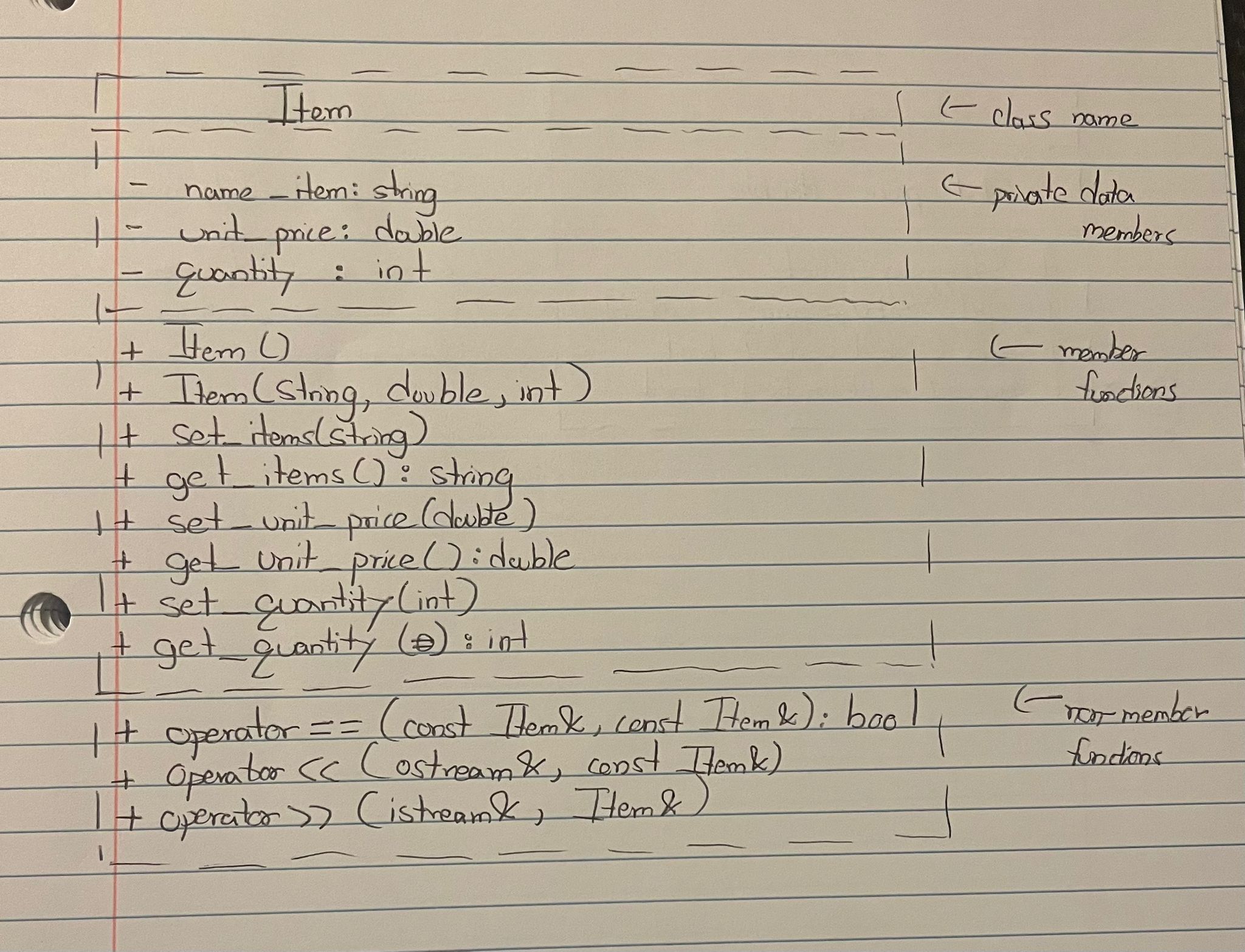
The program several functions:

* There are three constructors combined in both the item and the shopping cart class which allows us to initialize values of our data members.
* There are getter and setter functions for each private member in the item and shopping cart class which are name, price, quantity, and total price.
* There is a function which checks if two items are the same based on their name and price regardless of the quantity selected.
* There are functions for taking inputs and displaying outputs.
* There are functions to add and remove items from the cart
* And finally, a function that computes the total price of all the items in the cart.

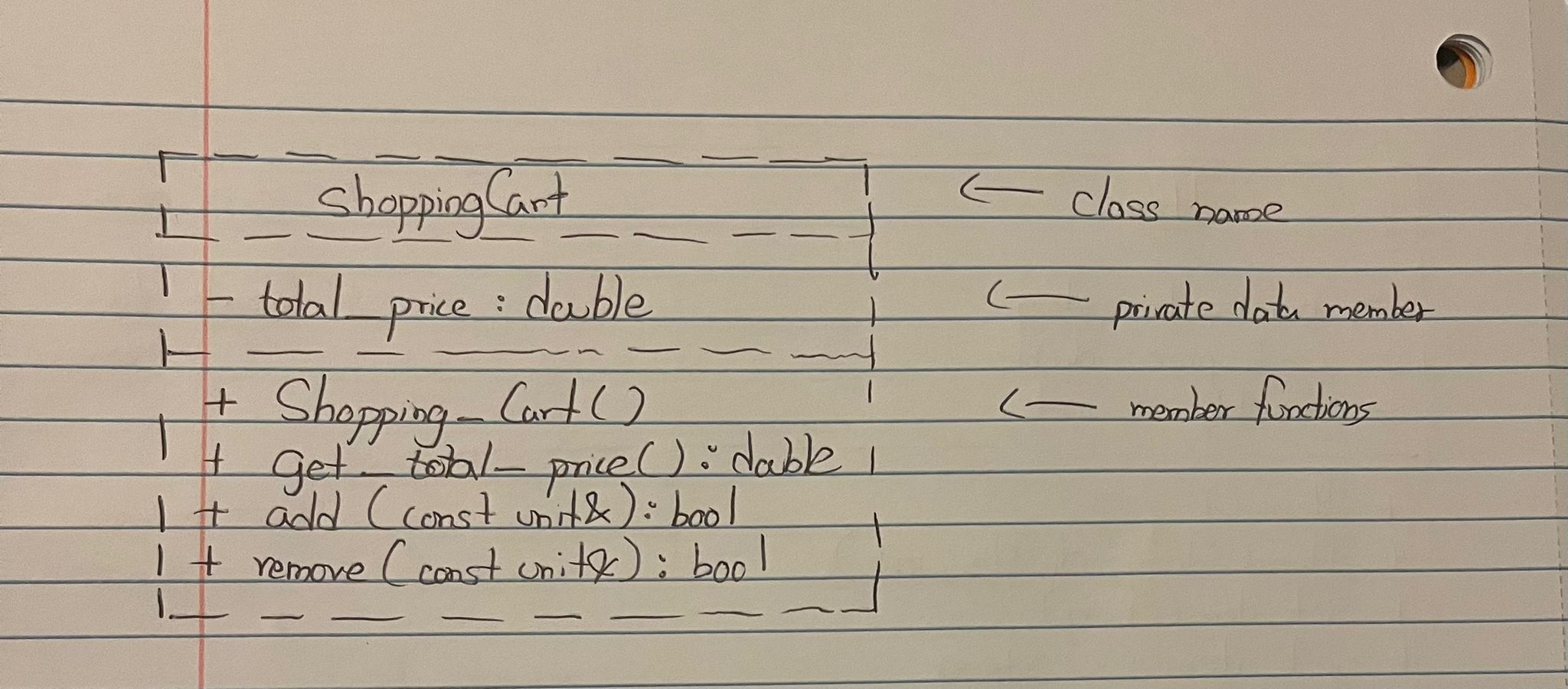
**The Main program:**

The main program asks the user to input the name, price, and quality of an item and then stores it as an object of the ShoppingCart class. With the use of loops, the program asks the user if they want to add more items and once the user is done the program displays all the items and the total price. The program then gives the user the option to add, remove, and change quantities of any of the items they picked and once they are fully satisfied with their choices it gives them an updated list of all the items in their cart and the program ends.

**UML diagram of item class**



**UML diagram of shopping cart class**



**2. Code list**

/\*\* item.h\*/

#ifndef ITEM\_H

#define ITEM\_H

#include <iostream>

#include <string>

using namespace std;

class Item

{

private:

/\*\* Name of the item \*/

string name\_item;

/\*\* Price of item \*/

double unit\_price;

/\*\* Quantity of item \*/

int quantity;

public:

/\*\*Constructor;

\*@pre none

\*@post intializes the value of item to default values\*/

Item();

/\*\*Constructor;

\*@param sting name of the item

\*@param double price of the item

\*@param int quantity of the item

\*@pre none

\*@post intializes the values of item to what the user defines\*/

Item(string, double, int);

/\*\*Member function\*/

/\*\* Sets the name\*/

/\*@param string the name of the item

/\*@pre none

/\*@post the name has been set to a particular value\*/

void set\_items(string);

/\*\*Member function\*/

/\*\*returns the name\*/

/\*@pre none

/\*@post the value returned is the name\*/

string get\_items() const;

/\*\*Member function\*/

/\*\* Sets the price\*/

/\*@param double the price of the item

/\*@pre none

/\*@post the price has been set to a particular value\*/

void set\_unit\_price(double);

/\*\*Member function\*/

/\*\*returns the price\*/

/\*@pre none

/\*@post the value returned is the price\*/

double get\_unit\_price() const;

/\*\*Member function\*/

/\*\* Sets the quantity\*/

/\*@param int the quantity of the item

/\*@pre none

/\*@post the quantity has been set to a particular value\*/

void set\_quantity(int);

/\*\*Member function\*/

/\*\*returns the quantity\*/

/\*@pre none

/\*@post the value returned is the quantity\*/

int get\_quantity() const;

};

/\*\*Compares two items\*/

/\*@param Item& one of the items\*/

/\*@param Item& the other item\*/

/\*@pre none

/\*@post compares two to see if they are equal\*/

bool operator ==(const Item&, const Item&);

ostream& operator <<(ostream&, const Item&);

/\*\* function for taking inputs\*/

/\*@param istream& takes user input

/\*@param Item& stores the input as item object\*/

/\*@pre none\*/

/\*@post takes input from user and stores it as item object\*/

istream& operator >>(istream&, Item&);

/\*\* function displaying outputs\*/

/\*@param ostream& output displays output

/\*@param Item& that displays item object\*/

/\*@pre none\*/

/\*@post function that takes and returns stream object\*/

ostream& operator <<(ostream&, const Item&);

bool operator ==(const Item&, const Item&);

#include "item.cpp"

#endif

/\*\* item.cpp \*/

/\*\*implementation file\*/

/\*\* file: item.cpp\*/

/\*\* Class IMPLEMENTED: item (See item.h for documentation.)\*/

#include "item.h"

Item::Item() {

name\_items = ""; //constructor that intializes default values

unit\_price = 0;

quantity = 0;

}

Item::Item(string d, double r, int c)

{

name\_item = d; // constructor that intializes user defined values

unit\_price = r;

quantity = c;

}

// Getter and Setter function implementations

void Item::set\_items(string d)

{

name\_item = d;

}

string Item::get\_items() const

{

return name\_item;

}

void Item::set\_unit\_price(double r)

{

unit\_price = r;

}

double Item::get\_unit\_price() const

{

return unit\_price;

}

void Item::set\_quantity(int r)

{

quantity = r;

}

int Item::get\_quantity() const

{

return quantity;

}

//checks if two items are equal

bool operator ==(const Item& item1, const Item& item2)

{

if (((item1.get\_items() == item2.get\_items()) and (item1.get\_unit\_price() == item2.get\_unit\_price())) == 1)

{

return true;

}

else

{

return false;

}

}

//function for taking input

istream& operator >>(istream& in, Item& i)

{

string d;

double r;

int c;

in >> d >> r >> c;

i.set\_items(d);

i.set\_unit\_price(r);

i.set\_quantity(c);

return in;

}

//function thats displays the output

ostream& operator <<(ostream& out, const Item& i)

{

out << i.get\_items() << " " << i.get\_unit\_price() << " " << i.get\_quantity();

return out;

}

/\*\* ShoppingCart.h\*/

#ifndef SHOPPING\_CART\_H

#define SHOPPING\_CART\_H

#include <iostream>

#include "Bag.h"

#include "item.h"

using namespace std;

template<typename unit>

class Shopping\_Cart : public Bag<unit>

{

private:

/\*\* Comined price of all items\*/

double total\_price;

public:

/\*\*Member function\*/

/\*\* Intializes the total price to 0\*/

/\*@pre none

/\*@post sets the total price to 0\*/

Shopping\_Cart();

/\*\*Member function\*/

/\*\*Returns the total price\*/

/\*@pre none

/\*@post Returns the total price\*/

double get\_total\_price() const;

/\*\*Member function\*/

/\*\*function for adding new item\*/

/\*@param thing& the item which will be added

/\*@pre none

/\*@post adds a new item to the list\*/

bool add(const unit&);

/\*\*function for removing item\*/

/\*@param unit& item to be removed

/\*@pre none

/\*@post removes a new item to the list\*/

bool remove(const unit&);

};

#include "shoppingCart.cpp"

#endif

/\*\* ShoppingCart.cpp\*/

/\*\*implementation file\*/

/\*\* file: ShoppingCart.cpp\*/

/\*\* Class IMPLEMENTED: ShoppingCart (See ShoppingCart.h for documentation.)\*/

#include "shoppingCart.h"

#include <iostream>

using namespace std;

template<typename unit>

Shopping\_Cart<unit>::Shopping\_Cart() : total\_price(0) //sets total\_price to zero

{

}

template<typename unit>

double Shopping\_Cart<unit>::get\_total\_price() const //returns total\_price

{

return total\_price;

}

template<typename unit>

bool Shopping\_Cart<unit>::add(const unit& stuff)

{

bool a = Bag<unit>::add(stuff); //adds an item

total\_price += stuff.get\_unit\_price() \* stuff.get\_quantity();

if (a == false)

{

total\_price -= stuff.get\_unit\_price();

}

return a;

}

template<typename unit>

bool Shopping\_Cart<unit>::remove(const unit& stuff) //removes an item

{

bool a = Bag<unit>::remove(stuff);

total\_price -= stuff.get\_unit\_price() \* stuff.get\_quantity();

return a;

}

/\*\*Bag.h\*/

// Created by Frank M. Carrano and Tim Henry.

// Copyright (c) 2013 \_Pearson Education\_. All rights reserved.

/\*\* ADT bag: Array-based implementation.

@file Bag.h \*/

#ifndef \_BAG

#define \_BAG

#include "BagInterface.h"

template<class ItemType>

class Bag : public BagInterface<ItemType>

{

private:

static const int DEFAULT\_BAG\_SIZE = 6;

ItemType items[DEFAULT\_BAG\_SIZE]; // array of bag items

int itemCount; // current count of bag items

int maxItems; // max capacity of the bag

// Returns either the index of the element in the array items that

// contains the given target or -1, if the array does not contain

// the target.

int getIndexOf(const ItemType& target) const;

public:

Bag();

int getCurrentSize() const;

bool isEmpty() const;

bool add(const ItemType& newEntry);

bool remove(const ItemType& anEntry);

void clear();

bool contains(const ItemType& anEntry) const;

int getFrequencyOf(const ItemType& anEntry) const;

vector<ItemType> toVector() const;

}; // end Bag

#include "Bag.cpp"

#endif

/\*\* Bag.cpp \*/

// Created by Frank M. Carrano and Tim Henry.

// Copyright (c) 2013 \_Pearson Education\_. All rights reserved.

/\*\* @file Bag.cpp (same as ArrayBag) \*/

#include "Bag.h"

#include <cstddef>

template<class ItemType>

Bag<ItemType>::Bag() : itemCount(0), maxItems(DEFAULT\_BAG\_SIZE)

{

} // end default constructor

template<class ItemType>

int Bag<ItemType>::getCurrentSize() const

{

return itemCount;

} // end getCurrentSize

template<class ItemType>

bool Bag<ItemType>::isEmpty() const

{

return itemCount == 0;

} // end isEmpty

template<class ItemType>

bool Bag<ItemType>::add(const ItemType& newEntry)

{

bool hasRoomToAdd = (itemCount < maxItems);

if (hasRoomToAdd)

{

items[itemCount] = newEntry;

itemCount++;

} // end if

return hasRoomToAdd;

} // end add

template<class ItemType>

bool Bag<ItemType>::remove(const ItemType& anEntry)

{

int locatedIndex = getIndexOf(anEntry);

bool canRemoveItem = !isEmpty() && (locatedIndex > -1);

if (canRemoveItem)

{

itemCount--;

items[locatedIndex] = items[itemCount];

} // end if

return canRemoveItem;

} // end remove

template<class ItemType>

void Bag<ItemType>::clear()

{

itemCount = 0;

} // end clear

template<class ItemType>

int Bag<ItemType>::getFrequencyOf(const ItemType& anEntry) const

{

int frequency = 0;

int searchIndex = 0;

while (searchIndex < itemCount)

{

if (items[searchIndex] == anEntry)

{

frequency++;

} // end if

searchIndex++;

} // end while

return frequency;

} // end getFrequencyOf

template<class ItemType>

bool Bag<ItemType>::contains(const ItemType& anEntry) const

{

return getIndexOf(anEntry) > -1;

} // end contains

/\* ALTERNATE 1

template<class ItemType>

bool Bag<ItemType>::contains(const ItemType& anEntry) const

{

return getFrequencyOf(anEntry) > 0;

} // end contains

\*/

/\* ALTERNATE 2

template<class ItemType>

bool Bag<ItemType>::contains(const ItemType& anEntry) const

{

bool found = false;

for (int i = 0; !found && (i < itemCount); i++)

{

if (anEntry == items[i])

{

found = true;

} // end if

} // end for

return found;

} // end contains

\*/

template<class ItemType>

vector<ItemType> Bag<ItemType>::toVector() const

{

vector<ItemType> bagContents;

for (int i = 0; i < itemCount; i++)

bagContents.push\_back(items[i]);

return bagContents;

} // end toVector

// private

template<class ItemType>

int Bag<ItemType>::getIndexOf(const ItemType& target) const

{

bool found = false;

int result = -1;

int searchIndex = 0;

// if the bag is empty, itemCount is zero, so loop is skipped

while (!found && (searchIndex < itemCount))

{

if (items[searchIndex] == target)

{

found = true;

result = searchIndex;

}

else

{

searchIndex++;

} // end if

} // end while

return result;

} // end getIndexOf

/\*\* BagInterface.h \*/

// Created by Frank M. Carrano and Tim Henry.

// Copyright (c) 2013 \_Pearson Education\_. All rights reserved.

/\*\* Listing 1-1.

@file BagInterface.h \*/

#ifndef \_BAG\_INTERFACE

#define \_BAG\_INTERFACE

#include <vector>

using namespace std;

template<class ItemType>

class BagInterface

{

public:

/\*\* Gets the current number of entries in this bag.

@return The integer number of entries currently in the bag. \*/

virtual int getCurrentSize() const = 0;

/\*\* Sees whether this bag is empty.

@return True if the bag is empty, or false if not. \*/

virtual bool isEmpty() const = 0;

/\*\* Adds a new entry to this bag.

@post If successful, newEntry is stored in the bag and

the count of items in the bag has increased by 1.

@param newEntry The object to be added as a new entry.

@return True if addition was successful, or false if not. \*/

virtual bool add(const ItemType& newEntry) = 0;

/\*\* Removes one occurrence of a given entry from this bag,

if possible.

@post If successful, anEntry has been removed from the bag

and the count of items in the bag has decreased by 1.

@param anEntry The entry to be removed.

@return True if removal was successful, or false if not. \*/

virtual bool remove(const ItemType& anEntry) = 0;

/\*\* Removes all entries from this bag.

@post Bag contains no items, and the count of items is 0. \*/

virtual void clear() = 0;

/\*\* Counts the number of times a given entry appears in bag.

@param anEntry The entry to be counted.

@return The number of times anEntry appears in the bag. \*/

virtual int getFrequencyOf(const ItemType& anEntry) const = 0;

/\*\* Tests whether this bag contains a given entry.

@param anEntry The entry to locate.

@return True if bag contains anEntry, or false otherwise. \*/

virtual bool contains(const ItemType& anEntry) const = 0;

/\*\* Empties and then f ills a given vector with all entries that

are in this bag.

@return A vector containing all the entries in the bag. \*/

virtual vector<ItemType> toVector() const = 0;

}; // end BagInterface

#endif

/\*\*project2.cpp\*/

/\*This program demonstrates a program using public inheritance of different to classes

to resemble a real-life shopping cart

Mohammad Aryan Khan

py9242yg

09/21/2023

Instructor:Jie Meichsner

\*/

#include <iostream>

#include <string>

#include <iomanip>

#include "item.h"

#include "shoppingCart.h"

#include "Bag.h"

using namespace std;

int main()

{

Item i, temp;

string choice;

Shopping\_Cart<Item> sc;

int max\_length, choices, new\_quantity;

cout << "Welcome to XXX SHOPPING CENTER." << endl;

cout << "Enter the item you selected in the following order:" << endl;

cout << "Name, Unit Price, quantity\n"

cout << "Name cannot contain any space, otherwise errors happen!)" << endl;

cin >> i;

sc.add(i);

cout << "Want to continue? y/n--> ";

cin >> choice;

bool not\_full = true;

while ((choice == "y") and (not\_full == true))

{

cout << endl << "--> ";

cin >> i;

not\_full = sc.add(i);

cout << "Want to continue? y/n--> ";

cin >> choice;

}

cout << endl << "Here is your order:" << endl;

cout << "-------------------------------------------------" << endl;

max\_length = 22;

cout << "Name";

cout << setw(max\_length) << "Unit\_price";

cout << setw(max\_length)<<"Quantity" << endl;

for ( int a = 0; a < sc.getCurrentSize(); a++)

{

cout << setw(16) << left << sc.toVector()[a].get\_items()

<< "$" <<setw(max\_length + 1) << left << fixed << setprecision(2) << sc.toVector()[a].get\_unit\_price()

<< sc.toVector()[a].get\_quantity() << endl;

}

cout << "The total charge is: $" << sc.get\_total\_price() << endl;

cout << "--------------------------------------------------" << endl;

cout << endl << "Want to modify your order? y/n--> ";

cin >> choice;

not\_full = true;

while((choice == "y") and (not\_full == true))

{

cout << "What do you want: Enter 1: add 2: remove 3: change quantity" << endl;

cout << "--> ";

cin >> choices;

switch(choices)

{

case 1:

{

cout << endl << "To add enter the item in the following order:" << endl;

cout << "Name, unit price, quantity" << endl;

cout << "--> ";

cin >> i;

sc.add(i);

cout << "The item has been added." << endl;

break;

}

case 2:

{

cout << endl << "Enter the item to remove in the following order:" << endl;

cout << "Name, unit price, quantity" << endl;

cout << "--> ";

cin >> i;

if (sc.contains(i) == false)

{

cout << "No such item in your shopping cart!" << endl;

}

else

{

sc.remove(i);

cout << "The item has been removed!" << endl;

}

break;

}

case 3:

{

cout << endl << "Enter the item to change as the following order:" << endl;

cout << "Name, unit price, quantity" << endl;

cout << "--> ";

cin >> i;

temp = i;

if(sc.contains(i) == false)

{

cout << "No such item exists." << endl;

break;

}

else

{

cout << "Enter a new quantity --> ";

cin >> new\_quantity;

while(new\_quantity <= 0)

{

cout << new\_quantity << " is not a valid input." << endl;

cout << "Enter a new quantity --> ";

cin >> new\_quantity;

}

}

sc.remove(i);

temp.set\_quantity(new\_quantity);

sc.add(temp);

cout << "The quantity has been modified." << endl;

break;

}

}

cout << endl << "Want to modify your order? y/n--> ";

cin >> choice;

}

cout << endl << "Here is your updated order:" << endl;

cout << "-------------------------------------------------" << endl;

cout << "You have ordered the following items:" << endl;

cout << "Name";

cout << setw(max\_length) << right << "Unit\_price";

cout << setw(max\_length) <<"Quantity" << endl;

for ( int a = 0; a < sc.getCurrentSize(); a++)

{

cout << setw(16) << left << sc.toVector()[a].get\_items()

<< "$" <<setw(max\_length + 1) << left << fixed << setprecision(2) << sc.toVector()[a].get\_unit\_price()

<< sc.toVector()[a].get\_quantity() << endl;

}

cout << "The total charge is $" << sc.get\_total\_price() << endl;

cout << "Thanks for shopping on XXX SHOPPING CENTER." << endl;

cout << "--------------------------------------------------" << endl;

return 0;

}

**3. User document**

The main program asks the user to input the name, price, and quality of an item and then stores it as an object of the ShoppingCart class. With the use of loops, the program asks the user if they want to add more items and once the user is done the program displays all the items and the total price. The program then gives the user the option to add, remove, and change quantities of any of the items they picked and once they are fully satisfied with their choices it gives them an updated list of all the items in their cart and the program ends.

The program’s name is project2.cpp. It is located at the following directory on CentOS:

/home/STCLOUDSTATE/py9242yg/CSCI301/project2

To compile the program simply enter:

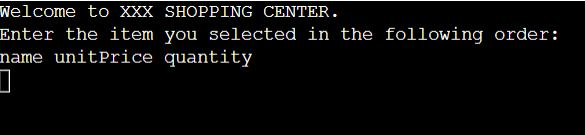
g++ project2.cpp

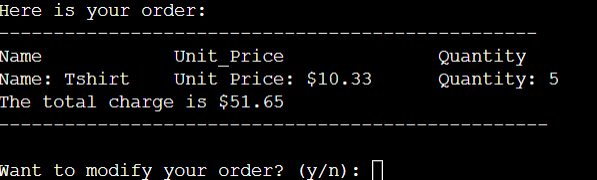
To run the program, enter:

./a.out

Then respond to the program’s prompts for inputs as specified. The program will continue to prompt the user for specific until the user decides to terminate the program.

For example,





**4. Test data plan**

* Valid input values: example of a valid user input would look like this:

Tshirt 12.99 4

* Boundary values: N/A
* Invalid input values: example of an invalid user input would look like this:

T shirt 12.99 4(with space between t and shirt) would show an error.

Other examples of invalid user input would include not following the order of input as instructed with name first, then price, and finally quantity.

**Summary**

In conclusion, in this project, we implemented a program that resembles a real-life shopping cart were the user can add, remove, and modify items (the quantity) and once they are satisfied the program displays the updated list of items in the shopping cart. We learned about virtual functions and how to use public inheritance.