SUPERMARKET INVENTORY SYSTEM

CO2203 OBJECT ORIENTED PROGRAMMING GROUP PROJECT

COURSE :	C02203 OBJECT ORIENTED PROGRAMMING
GROUP :	TEAM ERROR 404
YEAR :	2018/2019
DATE OF SUBMISSION:	11/07/2021

INTRODUCTION

This report provides information about implementing an autonomous inventory management system for Cool Mart which has managed the inventory and sales manually using spread sheets so far. The owner of Cool Mart decided to implement an autonomous inventory management system due to the gradual expansion of the business, hiring of more staff and the recent increase in demand for online sales due to pandemic.

The autonomous inventory management system was designed using C++ language and the UML class diagram of the design. There were some requirements which was concerned during designing process. This report briefly presents the designing of the system, UML class diagram, challenges faced while implementing the system, limitations and methods which can be used to further improvements.

REQUIREMENTS

Basically the client needs the system to be consist of three major sections.

Stock
The system should be responsible to the all details regarding the stocks in the inventory. The details of items should include the name of each item, how many of each item in the stock, retail price, discounts or promotions and the final price after applying discounts. There are main 10 categories of items in the system. They are Produce (fruits & vegetables), Meat & Seafood, Grains,
Bakery products, Frozen foods, Dairy products, Snacks and Sweets, Beverages, Health
& Beauty products. And also the brand of the product and whether they are local or imported products should be recorded. The unit which the quantity is measured also considered when designing this system.

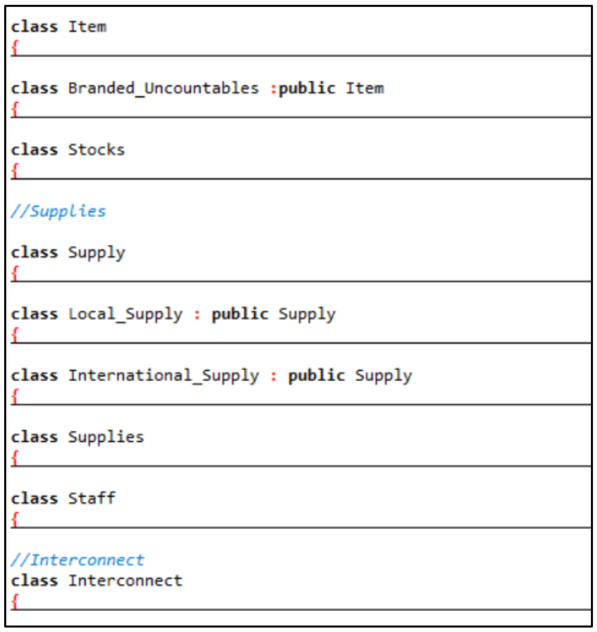
Staff
The staff of the supermarket manage the chain inventory system. Details such as full name and position of the all staff members must be recorded. The user name and the password should be requested from the person accessing the system to complete the access. According to the position, accessible paths are decided. The amount of control over the system will be decided by the position of the staff member. As an example, the only person who can add or remove a staff member is the owner.

Supply
Details of the supply chain brings stock into the inventory must be recorded including the details about the local and international supply sources. Details of the local supply should include the transport type, details of the items and transporting times, details of the imported supply include country and the shipment date. For international supplies data such as country of origin, the ship number, date it arrived at the local harbor and the date it was transported to the store should be recorded.

OOP CONCEPTS USED IN THE CODE

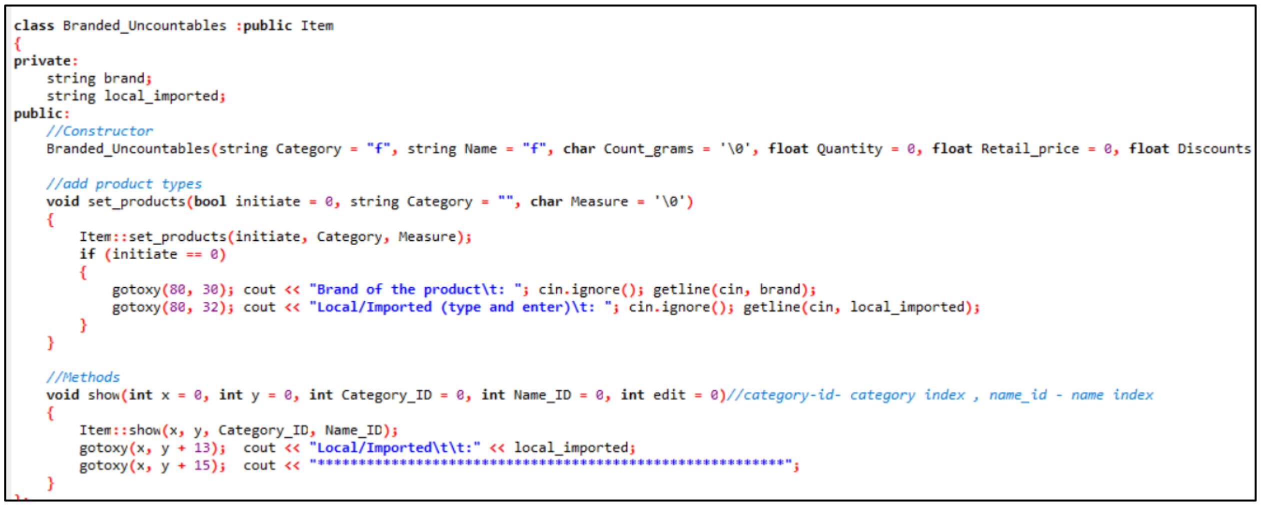
Encapsulation

Binding together the data and the functions which manipulates the data is known as the encapsulation. When coding the program, this concept is used in most cases.



```
class Item
{
class Branded_Uncountables :public Item
{
class Stocks
{
//Supplies
class Supply
{
class Local_Supply : public Supply
{
class International_Supply : public Supply
{
class Supplies
{
class Staff
{
//Interconnect
class Interconnect
{
```

Figure (01) : classes of the system



```
class Branded_Uncountables :public Item
{
private:
    string brand;
    string local_imported;
public:
    //Constructor
    Branded_Uncountables(string Category = "f", string Name = "f", char Count_grams = '\0', float Quantity = 0, float Retail_price = 0, float Discounts

    //add product types
    void set_products(bool initiate = 0, string Category = "", char Measure = '\0')
    {
        Item::set_products(initiate, Category, Measure);
        if (initiate == 0)
        {
            gotoxy(80, 30); cout << "Brand of the product\t: "; cin.ignore(); getline(cin, brand);
            gotoxy(80, 32); cout << "Local/Imported (type and enter)\t: "; cin.ignore(); getline(cin, local_imported);
        }
    }

    //Methods
    void show(int x = 0, int y = 0, int Category_ID = 0, int Name_ID = 0, int edit = 0)//category-id- category index , name_id - name index
    {
        Item::show(x, y, Category_ID, Name_ID);
        gotoxy(x, y + 13);  cout << "Local/Imported\t\t:" << local_imported;
        gotoxy(x, y + 15);  cout << "***********************************************************";
    }
```

Figure (02) : example for encapsulation

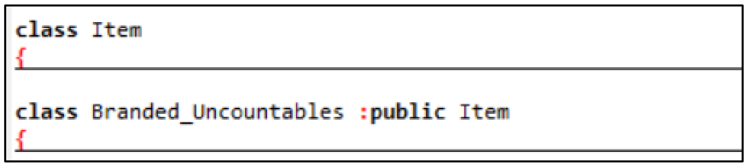
Figure 01 shows all the places where encapsulation concept is used. Figure 02 shows a detailed form of the class branded uncountable. In that the data and functions are bonded together under the class, which is a single unit.

Abstraction

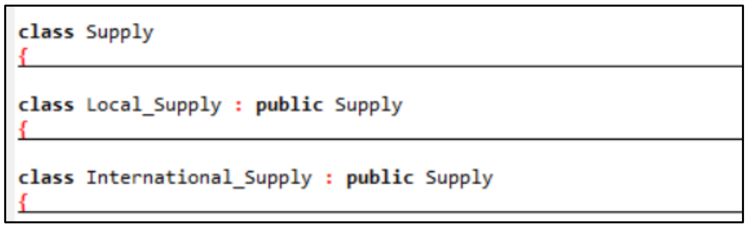
Abstraction can be known as providing only essential information about the data to outside word, hiding the background details or implementation. In a class some members are declared as private members and they are not accessible from the outside of the class. Other access specifiers are public and protected. Public members are accessible from anywhere of the program which protected members can be accessed by inside the class and for derived classes. The same example (figure 02) used for the encapsulation can be considered in this concept too. In that class, public and private access specifiers are used.

Inheritance

inheritance can be known as the procedure in which one class inherits the attributes and the methods of another class. The class whose properties are inherited is known as the parent class and the class which inherits the properties from the parent class is known as the child class.



```
class Item
{

class Branded_Uncountables :public Item
{
```

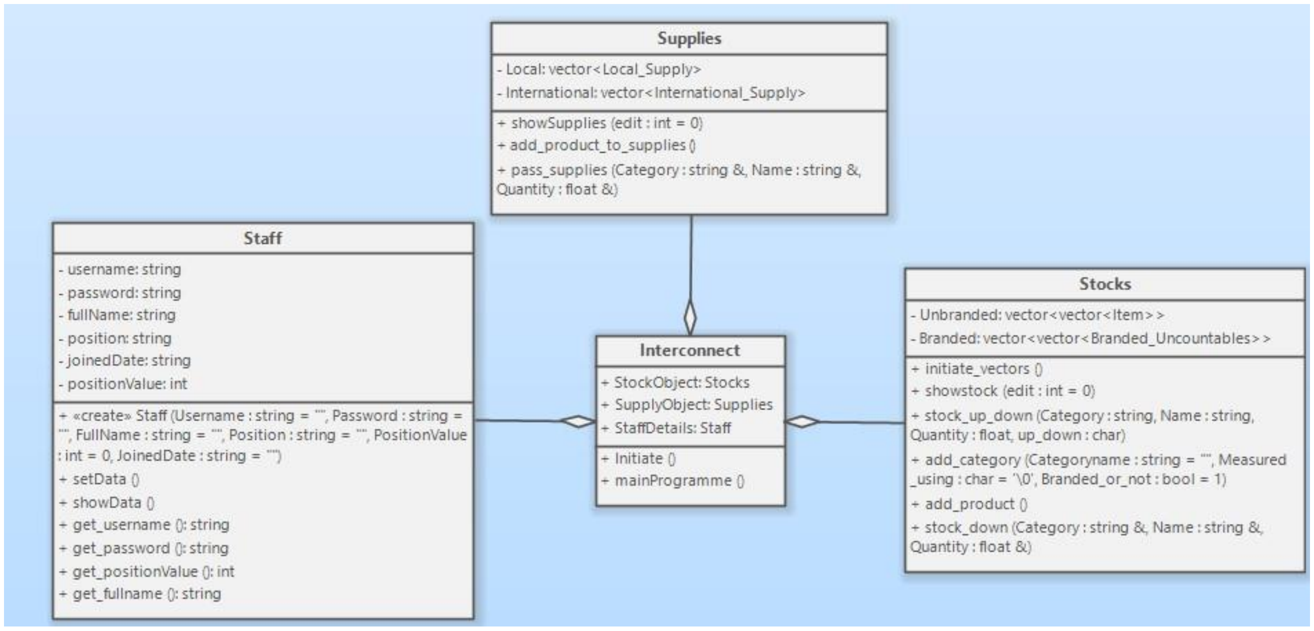


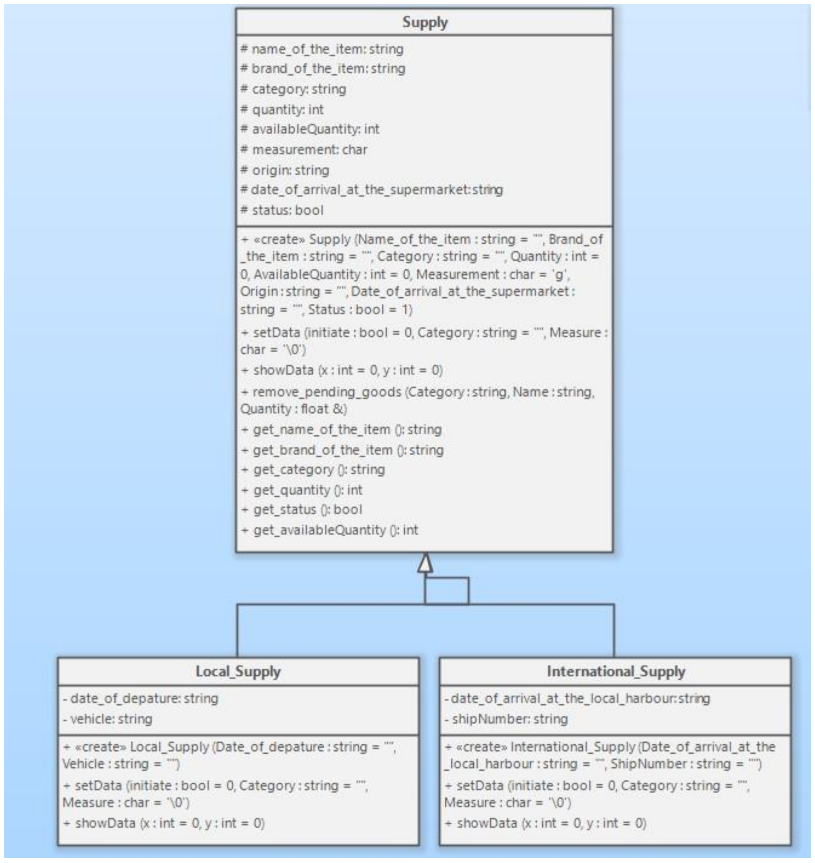
```
class Supply
{

class Local_Supply : public Supply
{

class International_Supply : public Supply
{
```

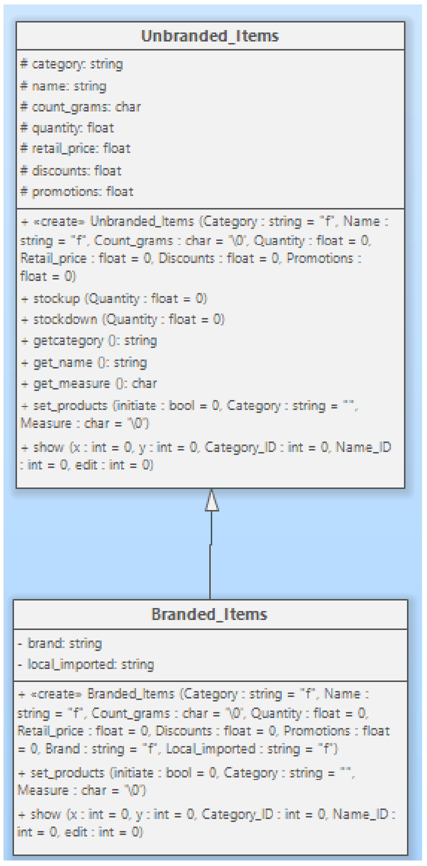
Figure (03) : examples for inheritance

In this code, inheritance concept is used in three cases. Branded uncountable are a child class/ sub class of the item class which is the parent class. When considering the supply class, it acts as a parent class to the childe classes, local supply and international supply. Both child classes contains the functions of parents class, therefore the length of the code was reduced.

UML CLASS DIAGRAM


Supplies
- Local: vector<Local_Supply>
- International: vector<International_Supply>
+ showSupplies (edit : int = 0)
+ add_product_to_supplies ()
+ pass_supplies (Category : string &, Name : string &, Quantity : float &)
Staff
- username: string
- password: string
- fullName: string
- position: string
- joinedDate: string
- positionValue: int
+ «create» Staff (Username : string = "", Password : string = "", FullName : string = "", Position : string = "", PositionValue : int = 0, JoinedDate : string = "")
+ setData ()
+ showData ()
+ get_username (): string
+ get_password (): string
+ get_positionValue (): int
+ get_fullname (): string
Interconnect
+ StockObject: Stocks
+ SupplyObject: Supplies
+ StaffDetails: Staff
+ Initiate ()
+ mainProgramme ()
Stocks
- Unbranded: vector<vector<Item>>
- Branded: vector<vector<Branded_Uncountables>>
+ initiate_vectors ()
+ showstock (edit : int = 0)
+ stock_up_down (Category : string, Name : string, Quantity : float, up_down : char)
+ add_category (Categoryname : string = "", Measured_using : char = '\0', Branded_or_not : bool = 1)
+ add_product ()
+ stock_down (Category : string &, Name : string &, Quantity : float &)

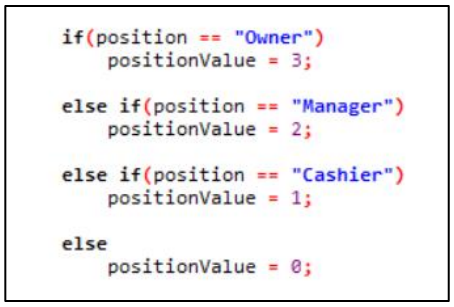

Supply
# name_of_the_item: string
# brand_of_the_item: string
# category: string
# quantity: int
# availableQuantity: int
# measurement: char
# origin: string
# date_of_arrival_at_the_supermarket:string
# status: bool
+ «create» Supply (Name_of_the_item : string = "", Brand_of_the_item : string = "", Category : string = "", Quantity : int = 0, AvailableQuantity : int = 0, Measurement : char = 'g', Origin : string = "", Date_of_arrival_at_the_supermarket : string = "", Status : bool = 1)
+ setData (initiate : bool = 0, Category : string = "", Measure : char = '\0')
+ showData (x : int = 0, y : int = 0)
+ remove_pending_goods (Category : string, Name : string, Quantity : float &)
+ get_name_of_the_item (): string
+ get_brand_of_the_item (): string
+ get_category (): string
+ get_quantity (): int
+ get_status (): bool
+ get_availableQuantity (): int
Local_Supply
- date_of_depature: string
- vehicle: string
+ «create» Local_Supply (Date_of_depature : string = "", Vehicle : string = "")
+ setData (initiate : bool = 0, Category : string = "", Measure : char = '\0')
+ showData (x : int = 0, y : int = 0)
International_Supply
- date_of_arrival_at_the_local_harbour: string
- shipNumber: string
+ «create» International_Supply (Date_of_arrival_at_the_local_harbour : string = "", ShipNumber : string = "")
+ setData (initiate : bool = 0, Category : string = "", Measure : char = '\0')
+ showData (x : int = 0, y : int = 0)


Unbranded_Items
# category: string
# name: string
# count_grams: char
# quantity: float
# retail_price: float
# discounts: float
# promotions: float
+ «create» Unbranded_Items (Category : string = "f", Name : string = "f", Count_grams : char = '\0', Quantity : float = 0, Retail_price : float = 0, Discounts : float = 0, Promotions : float = 0)
+ stockup (Quantity : float = 0)
+ stockdown (Quantity : float = 0)
+ getcategory (): string
+ get_name (): string
+ get_measure (): char
+ set_products (initiate : bool = 0, Category : string = "", Measure : char = '\0')
+ show (x : int = 0, y : int = 0, Category_ID : int = 0, Name_ID : int = 0, edit : int = 0)
Branded_Items
- brand: string
- local_imported: string
+ «create» Branded_Items (Category : string = "f", Name : string = "f", Count_grams : char = '\0', Quantity : float = 0, Retail_price : float = 0, Discounts : float = 0, Promotions : float = 0, Brand : string = "f", Local_imported : string = "f")
+ set_products (initiate : bool = 0, Category : string = "", Measure : char = '\0')
+ show (x : int = 0, y : int = 0, Category_ID : int = 0, Name_ID : int = 0, edit : int = 0)

CHALLENGES WHEN IMPLEMNTING THE SYSTEM

1. The main problem occurred during coding the system is the group members had to code the program separately due to the corona pandemic. Therefore it wasn't easy to continue the program under a one skeleton. To overcome this problem, we decided to divide the code under 4 sections which are stock, supply, staff and the interconnections. Functions and the data related to stock, supply and staff are grouped separately and treating them as separate boxes the coding process was carried on. Finally, theses these sections were connected using another class which is known as the interconnections.

2. Another challenge was to decide the accessible paths according to the position. As a solution for this a new variable was introduced as position value and an unique value was assigned depending on the position. Then by considering the position value and the user's selection, the related path can be accessed by the user if allowed.



```
if(position == "Owner")
    positionValue = 3;

else if(position == "Manager")
    positionValue = 2;

else if(position == "Cashier")
    positionValue = 1;

else
    positionValue = 0;
```

Figure (04) : assigning value to position

3. Another major challenge was managing the time. Due to the lack of time there are few faults in the submitted program. They are,
    - Transaction process doesn't proceed correctly.
    - Error handling of the code is not in very good level. The program is not case sensitive and the names / codes (item names, category names) have to enter as it is given.
    - Adding and removing process of staff is not completed.
    - Removing categories doesn't work properly.

LIMITATIONS

1. The major limitation of this system is that only one person can edit the data at one time. Though a new supplements are arrived, if the cashier is entering the transaction, the process of entering the new supplements has to be hold on.

2. Another limitation occurs if the market doesn’t has the generator facility to power up the system in power cuts. Therefore there should be some standby method to generate power to work continuously.

3. The worker should have some knowledge about the computer system. As an example, when considering floor worker, the lowest tier member of the staff may not have any knowledge about how to operate a computer. Therefor there should be a method to improve their knowledge about these systems.

FURTHER IMPROVEMENTS

1. As further improvements, the security of the system can be increased. frauds may occur since a it is not that impossible to steal a user name and a password. Therefore to improve this system advanced security methods such as fingerprint, voice recognition or face recognition can be used.

2. Another method to improve this system is the use of a user friendly interface with using shaped and colors. There for it would be easy to choose the wanted option and it will be more attractive than a black and white program. Further, vocals can be add to the system which presents how to select option or what is the option should be selected to perform a task.

3. To improve the security further, this system can be connected with the owners mobile number or any other owner’s personal device. When a suspicious access happens the owner can be automatically informed via sms or automatic call.

TEAM ERROR 404

Group members

19/ENG/027 : GURUGE A.G.M.T

19/ENG/033 : ILLANGARATHNA A.N.L

19/ENG/044 : JAYASINGHE J.A.D.S