Advance Software Design

Design and development of a Vending Machine using Object Oriented Modelling and Implementation.

Markos Galikas: w1895147
Ehtesham Hanifa: w1422297
Yusuf Mahamed: w1588964

University of Westminster

March 2023

Contents

List of Figures

1 Introduction

Vending machines are a common sight in public spaces, providing a quick and convenient way for people to purchase snacks, drinks and other items. However, designing and implementing a vending machine that can handle multiple products, accept payment and provide the correct change requires careful planning and execution. In this report, our group presents our project to design and implement a vending machine application using C# and object-oriented techniques. Our objectives include studying the high-level of software systems, documenting our design using graphical notations such as UML diagram and sequence diagrams and working collaboratively to produce a functional and well-designed application. We also aim to demonstrate our understanding of C# syntax and object-oriented design principles, and to explore the challenges and insights gained from working on software projects as a team.
Our intended audience for this project is both customers and administrators who will use the machine. The application is designed to allow customers to select products of their choice, insert the required amount and receive the product along with the correct change. The application also includes an error message if the machine is out of stock or if there's not enough change in the machine. administrators, who have login credentials, are able to increase stock and add change to the machine, and can display reports of stock level and amount of coins in the machine. By developing this vending machine application, we hope to gain practical experience in software design and implementation, as well as providing a useful and functional product for our audience.

2 Requirements Analysis

To ensure that we achieved our goals, we began by identifying the strengths and weaknesses of each member of the team and assigning roles that best suited their skills. This allowed us to work efficiently and effectively, ensuring that each team member contributed meaningfully to the project. The project manager played a crucial role in setting objectives and milestones for the team, and monitoring progress against these goals. They facilitated regular meetings to discuss updates and changes, ensuring that the project remained on track and that any issues were addressed promptly. The designer's task was to create a detailed UML and sequence diaphragm and work with the group to determine the design patterns to be used in our program. This required an in-depth understanding of the function and requirements of vending machines, as well as a critical eye for design and an ability to think creatively and outside the box. The developer's role was to code and test the application ,ensuring that it functioned as expected and that any bugs or errors were fixed promptly. Once we were able to determine each member's strength and weakness, we were able to assign them into their specific roles. We began by brainstorming and discussing the functions and requirements of a vending machine. We identified key features such as the ability to accept a variety of different coins, the ability to dispense products out of the machine and ensure that the correct products are delivered to the customer, providing information about the products available, the ability to provide accurate change to the customer when they insert more money than the cost of the product, the creation of an admin section in which an administrator can deposit change into the machine to ensure that it doesn't run out of coins, the ability to replenish stock levels for products, and finally the ability to generate reports that provide information about the amount of coins in the machine and the ability to change the price of products. After we had identified the essential functions of a vending machine, the next phase of our project was to translate these functions into a working program. Our group decided to implement the program as a console application using an object-oriented programming language, specifically C#. We face some limitations in terms of creating a more user-friendly interface, as the project was restricted to a console application without the option of a graphical user interface. Despite this challenge, we were determined to create a functioning program that met the requirements and objectives we had outlined earlier.

3 System Design

The architecture of our vending machine program will use an object-oriented approach, consisting of several classes that interact to perform the required functions. To determine the most suitable design pattern for our program, we researched three different options: the chain of responsibility, factory method, and memento method. The chain of responsibility pattern works by using a list of handler objects, each with limitations on the nature of requests they can handle. If an object cannot handle a request, it passes it on to the next object in the chain. At the end of the chain, there can be default or exceptional behaviour. (Bishop, J., 2007). The factory method pattern involves defining an interface for creating objects, allowing subclasses to decide which classes to instantiate. This pattern uses polymorphism to allow the code to work with any class that adheres to the interface defined by the factory method. (Sarcar, V., 2018). The memento pattern captures an object's internal state without violating the encapsulation. This pattern involves three main actors: the originator, memento, and caretaker(Lasater, C.G., 2006). After careful consideration, our group decided to use the chain of responsibility pattern. It is a suitable design pattern for handling the dispensing and management of coins in the vending machine, ensuring that all customers receive the correct change. In our program, the chain of responsibility works as follows; when a customer is owed 0.50p in change, the request dispense change is first passed to the object representing the 0.50p coin. If there are enough 0.50p coins in the machine to dispense the required amount, the 0.50p coin object handles the request , and the transaction is completed. If there are not enough 0.50p coins, the request is passed to the 0.20p coin object. If there are enough coins, the object will handle the request, and the transaction is completed. If there are not enough 0.20p coins, the request is passed to the 0.10p coin object, and so on until the transaction is complete. Using the chain of responsibility pattern ensures a flexible and extensible way to handle different types of coins and combinations of coins for dispensing change, making it the optimal choice for our vending machine program. The program consists of four classes: Interface.cs, CoinDispenser.cs, Enum.cs, and Snack.cs. The Interface.cs defines a set of methods that any class representing a vending machine must implement to interact with the vending machine's coin pool, prices, and change dispensing functionality. The Coin dispenser class is responsible for dispensing the minimum number of coins necessary to make change. It works through a chain of dispensers in descending order of denomination until the difference is fully dispensed, or an expectation is thrown indicating there is not enough change available. The Enum.cs defines a set of user types that can access the vending machine application. These types are User and Administrator , and the enum class allows for access & restriction based on the user type. This adds an extra layer of security into our application. The Snack.cs defines the characteristics of each snack that the vending machine dispenses. It includes information such as the name, price and quantity of each snack. This information is used to display the availability of snacks to the users. Overall, these four classes work together to create a functional and secure vending machine application that can dispense snacks and provide change in an efficient manner.

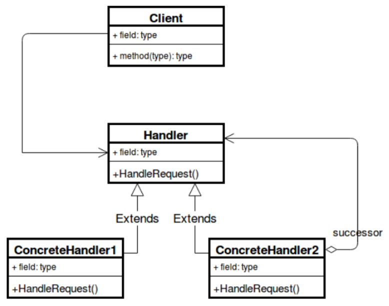


Figure 1: Example of Chain of responsibility

4 Implementation Details

Our implementation strategy as a group was to understand the classes and what functionality they would each have in our program. We began by creating a draft version which we called Vending machine. Essentially, in this Vending machine drafted we didn't implement our chain of responsibility as we wanted to get a basic understanding of the functionality of the vending machine before adding a chain of responsibility. In doing so, it allowed us to create our classes and allow each method to have appropriate actions needed to execute both User and Administrator needs. Our interface.cs included the following, a IVendingMachine Interface which defines a set of methods that any class representing a vending machine will implement in order to interact with the following methods.

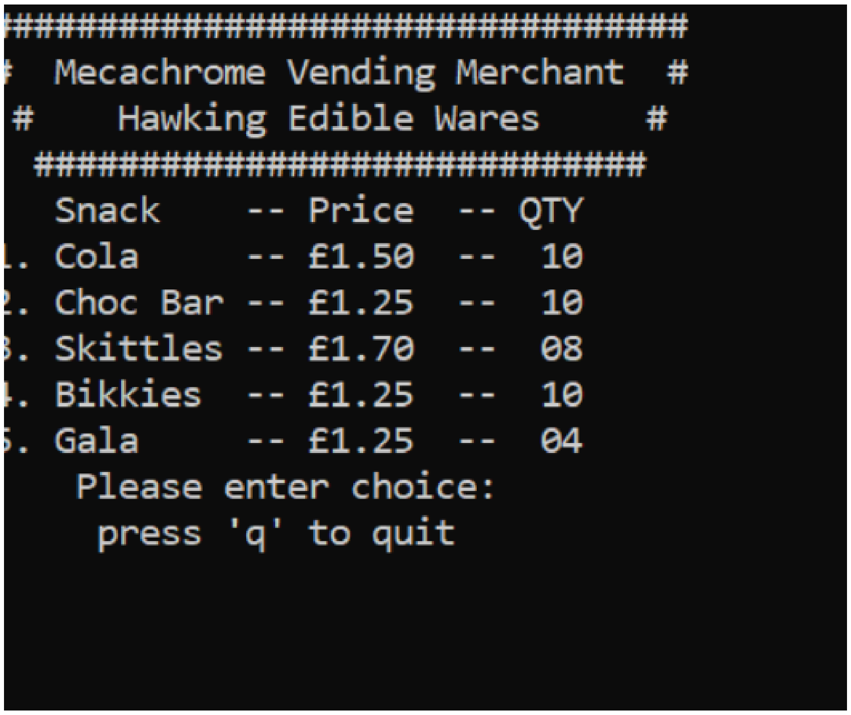
- Bool GiveChange (double difference, Dictionary<double,int> pool, int[] cashInserted); This method takes in the difference between the amount of money inserted and the price of selected snack, the current coin pool of vending machine and the cash inserted by the user, and returns a boolean value indicating whether or not change by working through the chain of dispensers in the descending order of denomination until the difference is fully dispensed or an exception is thrown is indicating there not enough change available.

- Void ChangePrice(List<Snack> snacks): We created a list of Snack objects and updated their prices in the vending machine.
- Void AddChange(Dictionary<double,int> pool: We created this method to take in a dictionary representing the coin pool of the vending machine and update it with the coins that are inserted by the user.
- Void GenerateReport(Dictionary<double,int> pool): A generate report method would take in the current coin pool of the vending machine and generate a report with the total amount of money in each denomination and the total money in the machine.

We created a Snack class for defining the characteristics of each snack that's in the vending machine. It has three properties 'label', 'quantity' and 'price', which we set through a constructor. Labels represent the name of the snack, quantity is the amount of snacks available, and price is the cost of each item. By implementing getter and setter methods, it allows the values to be retrieved and updated as needed. This ensures flexibility in managing our stock of snacks and pricing.
We created a User enumeration to identify the type of users accessing the application. This ensured security as it allowed users and administrators to only access what they need and nothing else. We added this feature to ensure that administrators have access to change prices of snacks and add coins to the vending machine. In comparison to users, who would have access to only purchasing. After many versions of our code, we were able to Implement our CoinDispenser class using the chain of responsibility design pattern.The abstract Coin dispenser class would define the common interface for all dispensers, while the concrete dispensers implement the dispense method based on their specific denomination. This allows for clear separation of responsibility and easy extensibility in case new denominations need to be added in the Future. SetNext method implemented by the group was used to link dispensers in a chain, so that if a dispenser can't fulfil a request, it passes it on to the next dispenser in the chain. This allows for requests to be handled by appropriate dispensers. Also, 'Dispense' ensures that each concrete dispenser class must implement this abstraction method. This method will take two arguments 'difference' to be dispensed and a 'pool' of available coins. The 'dispense' method returns a list of coins it will dispense to the user. The 'CoinDispencer' class, along with its concrete implementations, provides a flexible and extensible solution for dispensing coins in a vending machine or any other similar system. The use of chain of responsibility in our vending machine allows the vending machine to handle requests in a dynamic and efficient way, without the need for hard-coding each dispenser behaviour.

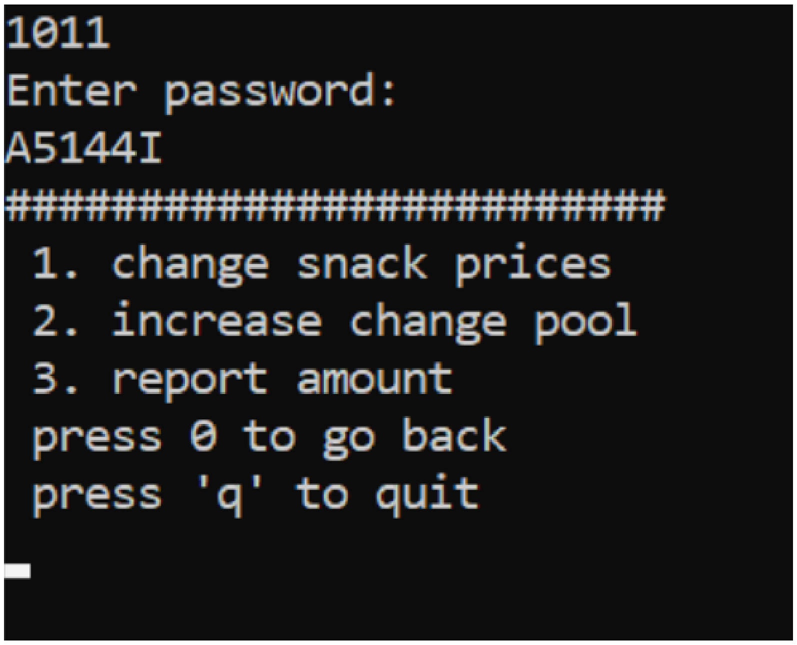
5 System Testing and Validation

In our system testing, as a group we have provided some images of our program for the User menu and for the Admin Menu. The rest is in the video format demonstrating the application.



```
################################
#  Mecachrome Vending Merchant  #
 #    Hawking Edible Wares     #
  ##############################
   Snack    -- Price  -- QTY
1. Cola     -- £1.50  --  10
2. Choc Bar -- £1.25  --  10
3. Skittles -- £1.70  --  08
4. Bikkies  -- £1.25  --  10
5. Gala     -- £1.25  --  04
    Please enter choice:
     press 'q' to quit
```

Figure 2: Customer Menu



```
1011
Enter password:
A5144I
############################
 1. change snack prices
 2. increase change pool
 3. report amount
 press 0 to go back
 press 'q' to quit

_
```

Figure 3: Admin Menu

6 Project Management

Date and Time: Wednesday 8th March 2023 17:30 – 19:00
In attendance: Ehtesham Hanifa, Yusuf Mahamed and Markos Galikas.
Absent: N/A
Location: Copland building room G1.06.
Agenda: First meeting together and discussing which role each member takes. Also setting targets for what needs to be achieved before our next meeting.
The meeting:
The meeting started of by reading the coursework description and deciding on which programming language to use. We opted for C#. We also decided on which design pattern to use, which we picked chain of responsibility. We needed to pick a role for each member and they will carry out their roles, as well as help each other when required. Ehtesham Hanifa is the project manager, Yusuf Mahamed is the designer and Markos Galikas is the developer. The objective to achieve by the next meeting is having started on the development of the vending machine. This includes having the menu of the vending machine ready, with the options of the snacks, the customer and admin being able to operate the vending machine as well as the UML class diagrams. The next meeting will be held inside the Cavendish library on the 15/03/2023.

Date and Time: Wednesday 15th March 2023 18:00 – 19:30

In attendance: Ehtesham Hanifa, Yusuf Mahamed and Markos Galikas.
Absent: N/A
Location: Cavendish Library level 4
Agenda: This is our second meeting. We will be discussing on what targets have been achieved and what we need to accomplish before the next meeting.
The meeting:
The meeting began by looking at what code has been written so far and what it looks like once we run the code. So far, we have the main menu for the vending machine, it states which snacks are available, how much of each snack is inside the vending machine, how much it costs and how much change will be given once the user inserts the coins. We also have an option for the admin, who will have control over how much snacks are inside the vending machine, how much coins are inside for giving out change and the price of the snacks. The objective to achieve by the next meeting is to implement the code for chain of responsibility for change dispensing, to be able to generate a report for the vending machine and to have a UML diagram ready. Also, the report will be started. The next meeting will be held inside Cavendish library on the 22/03/2023.

Date and Time: Wednesday 22th March 2023 17:30 – 19:00
In attendance: Ehtesham Hanifa, Yusuf Mahamed and Markos Galikas.
Absent: N/A
Location: Cavendish library level 4
Agenda: This is our third meeting together and we will be discussing on what code we have so far, as well as the UML diagram, the sequence diagram and how to make improvements.
The meeting:
The meeting started of by going through the updated version of the vending machine code and troubleshooting it to see if there's any errors. The code is working well and the report generating feature has been added onto the code too. This will generate a report that will display all the information regarding the snacks, such as quantity and prices, inside the vending machine. We have done debugging on the code and everything is working fine. A draft of the UML diagram and sequence diagram has been produced and we went through this as a group and discussed on what to add. We checked for any errors, which also included spelling errors, and we amended those. We have began writing up the report for the coursework. This includes the introduction, a systems design, implementation details, system testing and conclusion. The report will also include the code and the pseudo code for the vending machine. For the next meeting, we will meet on Tuesday 28th March and we will put everything together and start recording the vlog/group presentation.

Date and Time: Tuesday 28th March 2023 17:00 – 18:30
In attendance: Ehtesham Hanifa, Yusuf Mahamed and Markos Galikas.
Absent: N/A
Location: Cavendish Library level 4
Agenda: This is our final meeting. We will put everything together and submit

the assignment.
The meeting:
The main purpose of this meeting is to put the work together. We went through the code and we did some last minute debugging to make sure everything was alright. We then went through the report to make sure we had everything we needed for the report. This includes the introduction, a system's design, implementation details, system testing and validation, project management, conclusion, appendix and references. We have decided ***against*** using a pseudo code for the report and instead, we will screenshot the codes we have written and we will describe what that code is implementing. This will be included in the appendix. Each member of the group went through the report and we made sure there wasn't any spelling errors. Finally for the vlog, we screen recorded the program. The video recording shows the customer selecting the snack, putting the coins in, getting the snack, getting change back and the quantity of the snacks reducing by 1. The video also shows the admin menu and it shows how much money is in the vending machine and also having the ability to increase the prices. This demonstrates that the code we wrote works. A voice-over has been added onto the video explaining what is happening and a link to that video has been provided to the assessor to view. The work has now been put together, it is in a zipped file and has been uploaded onto blackboard for marking. This assignment has been submitted.

7 Conclusions

To conclude,

Overall as a group we were able to successfully achieve the objectives and milestones which we set out during our process of building a vending machine application. By applying the correct object-oriented techniques, we were able to implement the essential functionalities of a vending machine into a program. Our team overcame challenges such as the limitations of a console application, and implemented the chain of responsibility design pattern to ensure that customers receive the correct denominations. By working collaboratively and assigning roles to each individual in the group we were able to effectively and efficiently complete the projects. This project allowed us to gain practical experience in our advanced software design module and implementation, and explored the challenges and insights gained from working on software projects as a team. We also demonstrated our understanding of object-oriented design principles, the ability of creating graphical notations such as UML diagrams and sequence diagrams, and providing a useful and functional product for our intended audience. Overall, Our vending machine application was a success, and as a group we are satisfied with the work we accomplished.

8 Appendix

Video link: https://drive.google.com/file/d/13SOmsnu3OzOmnXrOPv-bJSonRLtt7URM/view?usp=share_link

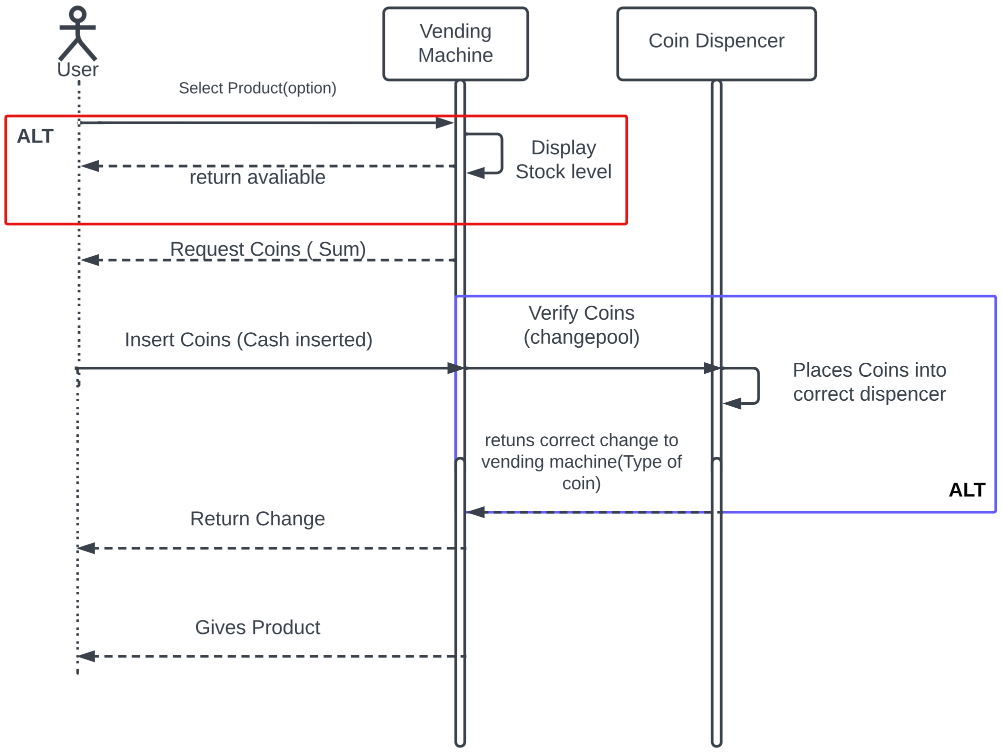

User
Vending Machine
Coin Dispencer
Select Product(option)
ALT
Display Stock level
return avaliable
Request Coins (Sum)
Verify Coins (changepool)
Insert Coins (Cash inserted)
Places Coins into correct dispencer
retuns correct change to vending machine(Type of coin)
ALT
Return Change
Gives Product

Figure 4: Customer Menu Sequence Diagram

Figure 5: Admin Menu Sequence Diagram

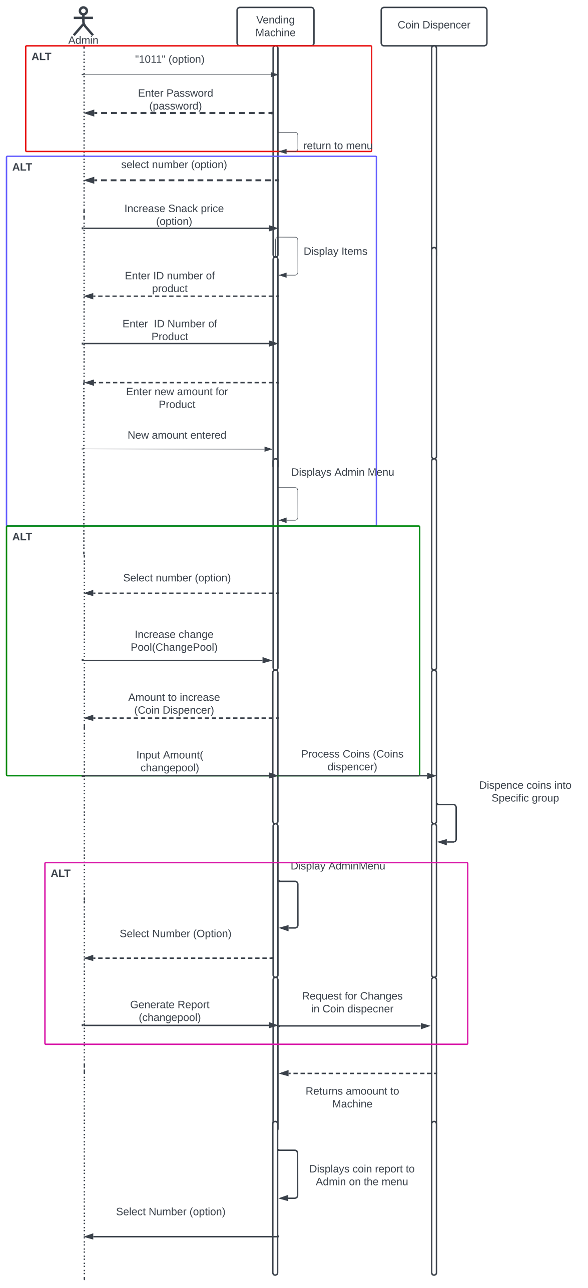
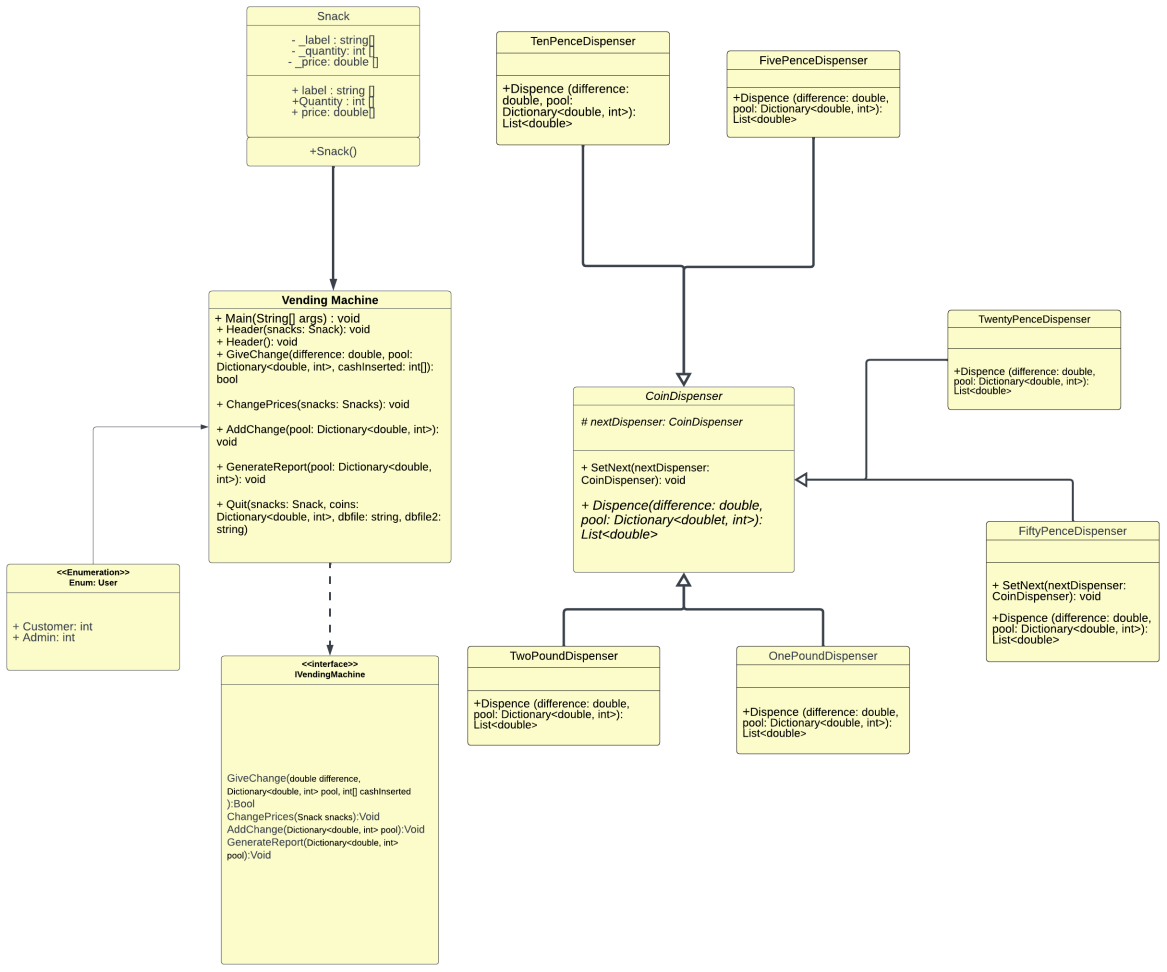

Admin
Vending Machine
Coin Dispencer
ALT
"1011" (option)
Enter Password (password)
return to menu
ALT
select number (option)
Increase Snack price (option)
Display Items
Enter ID number of product
Enter ID Number of Product
Enter new amount for Product
New amount entered
Displays Admin Menu
ALT
Select number (option)
Increase change Pool(ChangePool)
Amount to increase (Coin Dispencer)
Input Amount(changepool)
Process Coins (Coins dispencer)
Dispence coins into Specific group
ALT
Display AdminMenu
Select Number (Option)
Generate Report (changepool)
Request for Changes in Coin dispecner
Returns amoount to Machine
Displays coin report to Admin on the menu
Select Number (option)

Figure 6: UML Diagram



Snack
- _label : string[]
- _quantity: int []
- _price: double []
+ label : string []
+Quantity : int []
+ price: double[]
+Snack()
TenPenceDispenser
+Dispence (difference: double, pool: Dictionary<double, int>): List<double>
FivePenceDispenser
+Dispence (difference: double, pool: Dictionary<double, int>): List<double>
Vending Machine
+ Main(String[] args) : void
+ Header(snacks: Snack): void
+ Header(): void
+ GiveChange(difference: double, pool: Dictionary<double, int>, cashInserted: int[]): bool
+ ChangePrices(snacks: Snacks): void
+ AddChange(pool: Dictionary<double, int>): void
+ GenerateReport(pool: Dictionary<double, int>): void
+ Quit(snacks: Snack, coins: Dictionary<double, int>, dbfile: string, dbfile2: string)
TwentyPenceDispenser
+Dispence (difference: double, pool: Dictionary<double, int>): List<double>
CoinDispenser
# nextDispenser: CoinDispenser
+ SetNext(nextDispenser: CoinDispenser): void
+ Dispence(difference: double, pool: Dictionary<doublet, int>): List<double>
FiftyPenceDispenser
+ SetNext(nextDispenser: CoinDispenser): void
+Dispence (difference: double, pool: Dictionary<double, int>): List<double>
<<Enumeration>>
Enum: User
+ Customer: int
+ Admin: int
<<interface>>
IVendingMachine
GiveChange(double difference, Dictionary<double, int> pool, int[] cashInserted):Bool
ChangePrices(Snack snacks):Void
AddChange(Dictionary<double, int> pool):Void
GenerateReport(Dictionary<double, int> pool):Void
TwoPoundDispenser
+Dispence (difference: double, pool: Dictionary<double, int>): List<double>
OnePoundDispenser
+Dispence (difference: double, pool: Dictionary<double, int>): List<double>

9 References

1 Bishop, J., (2007). 'C# 3.0 Design Patterns: Use the Power of C# 3.0 to Solve Real-World Problems.' *O'Reilly Media, Inc.*

2 Sarcar, V. (2018). 'Java Design Patterns: A Hands-On Experience with Real-World Examples.' *Packt Publishing.*

3 Lasater, C.G., (2006). 'Design patterns.' *Jones & Bartlett Publishers.*