# Team Information

Team name: Cina

Members:

Katchenin Cindy Coulibaly – 2126537

Breanna de Forest – 2145494

# Project Description

Describe the application you are building and its features.

This application is a virtual vending machine. It allows a user to choose the quantity of an item that they would like to have in cart using sliders, an “Add” button, and a “Remove” button. Once the user is satisfied with what they have in their cart, they have the option to checkout. To checkout, a second window pops us, which allows the user to specify what payment method they would like to use. When that is decided and they pay for their items, they receive a receipt, it is saved to a file, and the user is given the option to open that file. The user can exit the payment screen, and everything will be reset so that the process can begin again.

The application both reads and writes to a couple files. On startup, the application reads from a stock data file to create all the product objects and get each products stock. This information is crucial for the application to run properly. The current file is a txt file, in which all the data is separated by commas. On each line, it starts with the product code, then the product name, then the product price, and ends with the product’s stock. At the end of every successful transaction, the application updates this stock data file so that it has the correct information. It also writes the receipt of the latest transaction to a transaction history file. The file is a simple txt file, and the newest receipts are found at the bottom of the file.

# Development Approach

Explain how did you prepare for the project. You can use the 5 steps of algorithmic thinking to you help build this section (you will need to elaborate on each step).

1. Understanding the problem.

When we first were assigned the project, we read through the instructions and talked between the two of us to choose which topic we wanted to do. From there we figured out the basics of what the input and output of a vending machine would be. For the vending machine to work, it needs products, and since we weren’t provided with any data, we would have to create sample data or some sort. When someone uses a vending machine, they choose what they want to buy, and then they receive that item and possibly a receipt. This means that while the application is running, the inputs would be the user’s choices, and the output would be the product they chose and a receipt.

, which can be found in the As for the application itself, we realized the user would have to be able to choose what items they want to add to their cart and they would have to indicate when they wanted to buy the items in their cart. Since the application is mimicking buying items, we knew the user would have to receive some sort of receipt, and that that receipt would probably need to be saved to an external file, which ended up being the Transactions.txt file in the DataFile directory.

1. Formulating the problem.

To get the products, we decided that the information would come from a file that stored the product information. The application would get the information from this file to set up the products that the user could then buy. This would be the StockData.txt file in the DataFiles directory.

While using the vending machine, the user would have to indicate exactly which item and how many of this item they want to add to their cart. They would also have to specify how they want to pay for the items they are buying. This means that the app would have to keep track of everything they wish to buy, it would have to calculate the cost of everything, and it would probably have to calculate change if the user ends up paying with cash.

Once a transaction has finished, the user probably expects to get a receipt for what they have bought, which means the application would have to create a receipt to give them. It might also want to save these receipts somewhere, like a Transactions.txt file in the DataFiles directory.

1. Developing the application \ algorithm.

Getting the products:

* To get the products, there needs to be a file that has all the information in it that the application can read and access.
* The information from the file has to be stored somewhere within the application, so there should be a Product class, so that the information can be saved as objects.

User adding/removing items from the cart:

* There needs to be a cart to store what the user wants to buy, so a ShoppingCart class and object has to be created.
* The user needs to know which items are available, so the items should be displayed on screen.
  + There should also be a visual element that indicates in an item isn’t available, like turning the background of the element gray.
* The user needs to be able to choose the quantity of an item, so there should be a slider somewhere on screen to allow them to choose a quantity.
  + It would also be very helpful to the user if they could see what the slider value is set as.
  + This also means there has to be some way to get the value of the slider.
* The user needs to be able to add an item to their cart, so it needs an add button.
  + This means there should be a method that actually adds the item to the cart when this button is pressed. It should also make sure that the quantity of the item the user is trying to buy is available in stock.
* The user should also be able to remove something from their cart, so a remove button could also be very helpful.
  + This means there should be a method to remove the item from the cart when the button is pressed. The logic would also have to make sure that the items that are being removed were already in the cart before the button was pressed.
* The user should also be able to see what items are in their cart, so that information should be somewhere on the screen.
  + This means there needs to be a way to access the element on the screen to continuously update the data whenever items are added and removed from the cart.
* The user also needs to be able to go to the checkout page to buy the items in their cart, so there should be a button for this.
  + Which means that there will need to be a method that opens the second window when this button is clicked.

Checkout:

* The user should be able to choose what payment type they would like to use, so there needs to be something on the window that allows them to make that choice.
  + If they choose cash, it also means they will have to somehow indicate what bills they are using to pay.
* There needs to be a way to for the user to finalize the transaction, so there should be a button they can click.
  + This means there needs to be a method that will make sure the payment type information is all valid and enough to cover the bill.
* Once the transaction has gone through:
  + A receipt needs to be created and given to the user in some way.
  + The receipt should be saved to an external file.
  + The original stock file data needs to be updated now that some products have lost some stock.

1. Implementing the application \algorithm.

A vending machine needs to have some sort of graphical interface so that the user can decide what they want to buy, and so that they can interact with the window’s elements to get the results they wish. From all the programming languages we know how to use, WPF seems like the best language to use to be able to make a GUI that is easy to understand and work with while getting the desired results. With WPF for the frontend, it was an obvious choice to go with C# for the backend portion of the project. It can implement all the elements needed to make a vending machine, and it is very easy to work with when paired with WPF.

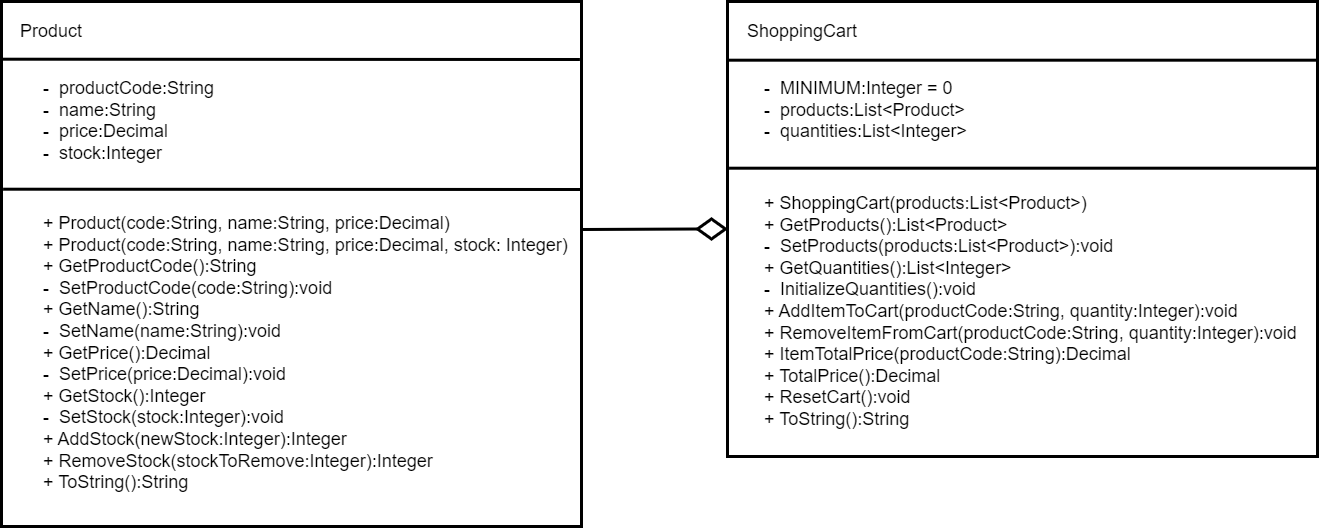
1. Testing.

To test the program, we thought it would be best to fully replicate the user’s experience. This means that to test the application, we ran the project as the user would and tested many different values, making sure that correct and good input would result in the proper responses, while incorrect inputs would result in error messages.

# OOP Design

Talk about the classes you need to create for the application and what is the purpose of each class. Include the UML class diagram in this section. The UML class diagram should include the relations between the created classes. Do not mention the WPF classes (Window, etc.)

This application has two classes: a Product class and a ShoppingCart class. This image is the UML diagram of these classes. A more detailed explanation of each class is provided below.



The Product class is the class that held the information for each item that could be bought from the vending machine. It has four private backing fields: the product code, the product name, the product’s price, and the product’s stock. The class has two constructors: one that takes the product code, name, and price, the other that takes the product code, name, price, and stock. The point of this is to be able to create a product even if the stock is unknown. All four backing fields have a public getter, so that their values can be accessed outside of the class, and private setters, so that their values cannot be changed when they shouldn’t change and so that it respects the rule of encapsulation. The only value that currently can change is the stock, so there are public methods to add and remove from the stock. To assure that the product is being represented, the ToString method is overwritten to return all the product’s information.

The ShoppingCart class is the class that represents what the user is planning on buying. It has three private backing fields: a constant “MINIMUM” that has a value of 0, a list of products, and a list of integer quantities. Both lists have a public getter so that they can be accessed outside of the class, but only the list of products has a private setter, which sets the list based on the list that is provided in the constructor when the ShoppingCart object is created. There is a private method that initializes the quantities list, but otherwise it does not have a setter. There are two methods that can add or remove items from the ShoppingCart, two methods that can get the total price of an item or the total price of the cart, a method to reset the cart and set all of the quantities back to the minimum amount, and finally the ToString method is overridden so that it returns every item in the cart and how much each of those items costs.

Since the ShoppingCart class has a list of Product items, it makes the relationship between the two classes aggregate, since the Products can be found within the ShoppingCart, but they can also exist on their own.

# Contributions

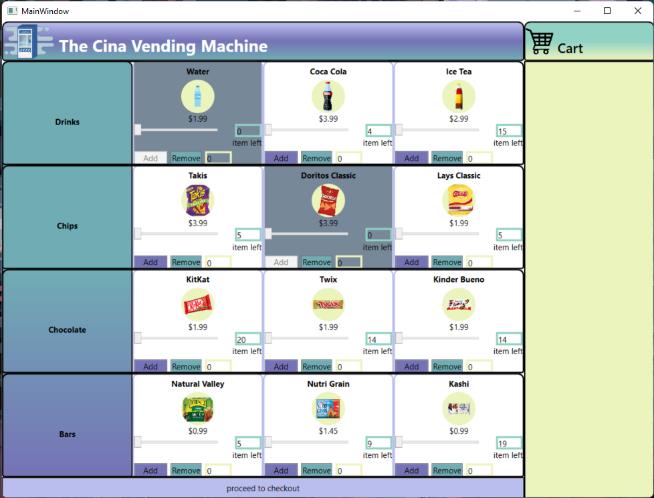
What did each team member do? How was the work in the project divided?

The work was divided into two halves: the frontend portion and the backend portion. After talking about which part is more interesting and which part we each thought we could do, it was decided that Cindy would do the frontend and Breanna would do the backend. This means that Cindy is the one who designed the vending machine, created how it looks, and made it look pretty, while Breanna is the one who created the classes, figured out the logic behind how the vending machine parts would work, and implemented the methods that are called when the user interacts with the vending machine.

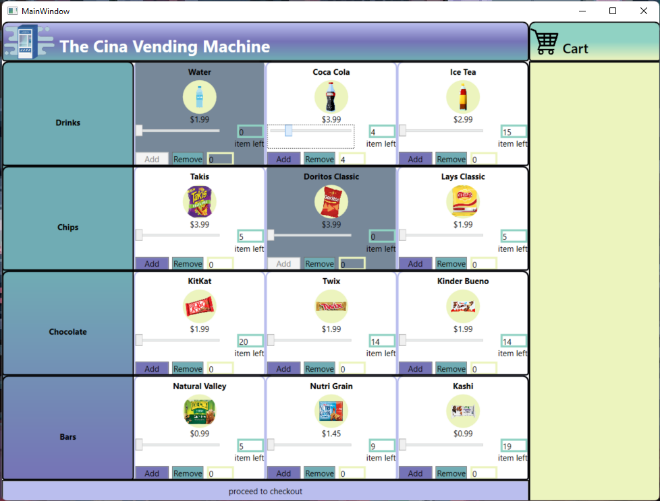
That being said, this is a group project, so if there was something either or us didn’t understand or know how to do, the other stepped up to help them with whatever they were stuck with.

# App Snapshots

This section includes snapshots of the final application showing different features. It could be a guideline to using the application. You may include snapshots of the app while being developed. Remember to add explanatory captions to the snapshots.



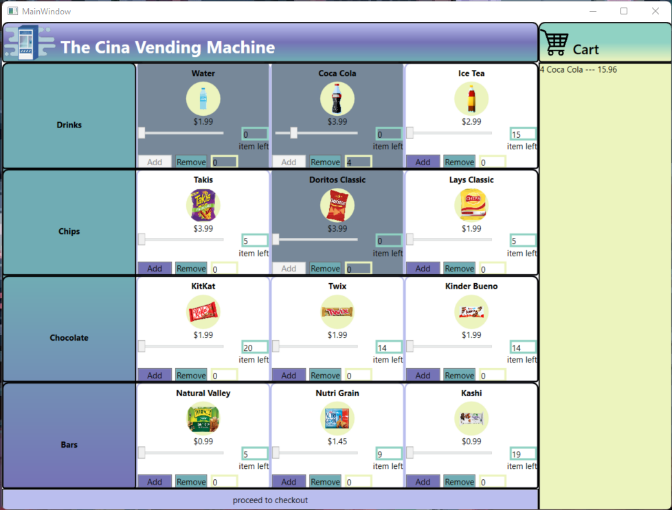
This is the first window the user sees. The items with a gray background represent the products that are out of stock and the items with a white background are the products that are in stock.



This snapshot is demonstrating how to choose the quantity of an item. As can be seen, the slider for Coca Cola has been moved to be selecting 4.



To add the quantity of an item to the cart, click the “Add”.

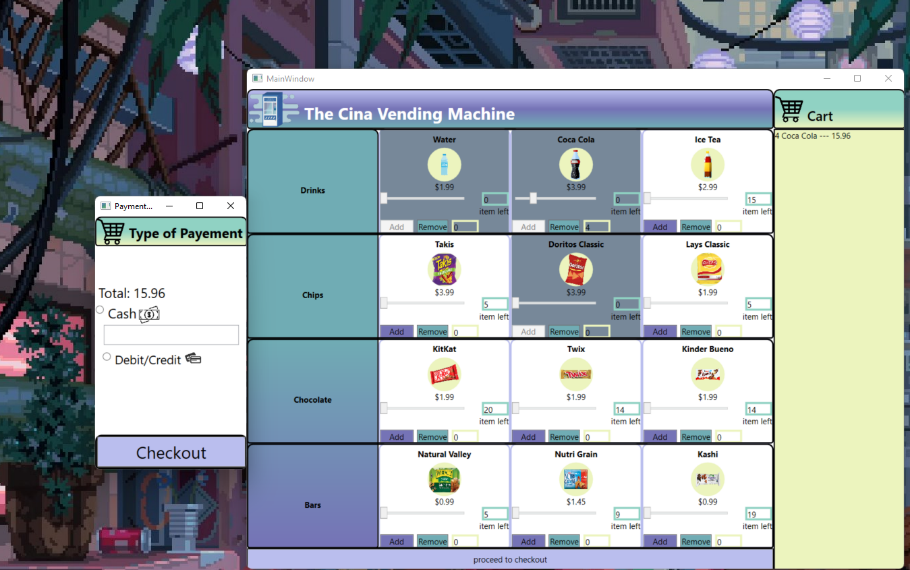


This snapshot is demonstrating how once the “Add” button is clicked, the quantity, product name, and price are added to the cart on the right.

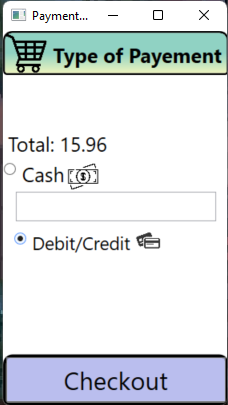
In this case, all of the Coca Cola stock is added to the cart, which means it has no stock left, so it is grayed out.



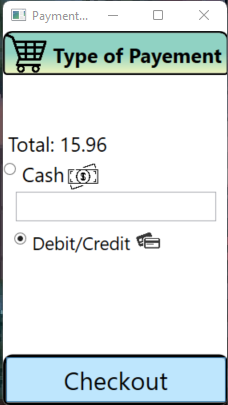
This is demonstrating that to checkout the items in the cart, the “proceed to checkout” button must be clicked.



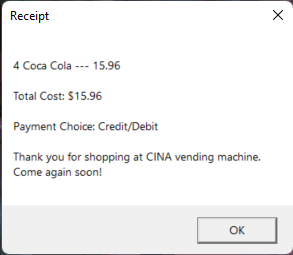
The window that pops up on the left is the checkout window.



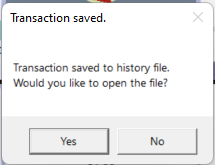
The user must choose their payment type. In this snapshot, the user is deciding to pay with their debit or credit card.



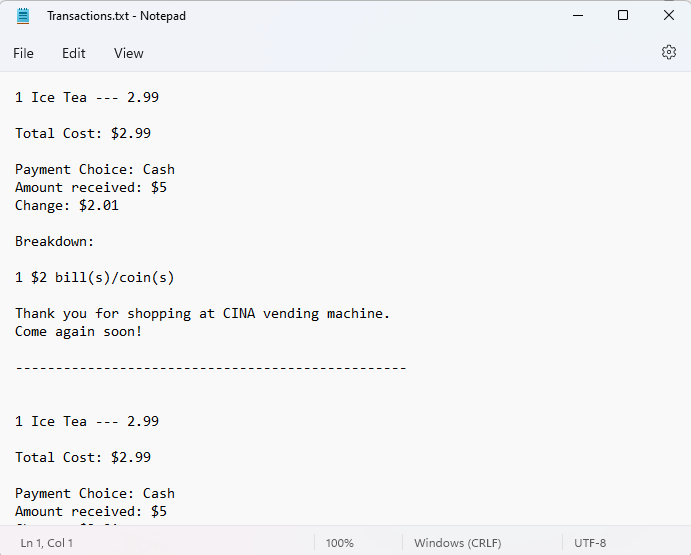
To actually checkout and receive the transaction receipt, click the “Checkout” button.



A receipt that resembles this will show up. Click the “OK” button to close it.



After getting the receipt, this message will show up letting the user know their receipt was saved to a transaction history file. They have the option of opening that file and can make their decision by clicking the “Yes” button or the “No” button.



If the user agreed to opening the file, a file like this one will open.

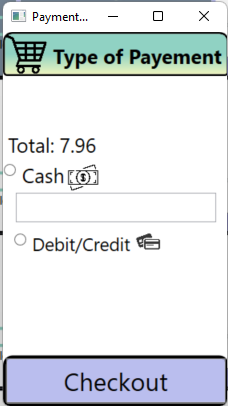
The following snapshots are demonstrating the same process except this time, the user chooses to pay with cash.



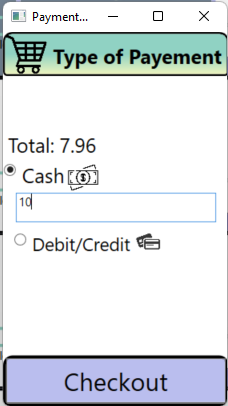
This snapshot demonstrates the slider being used to determine the quantity of an item. In this case it is for Lays Classics.



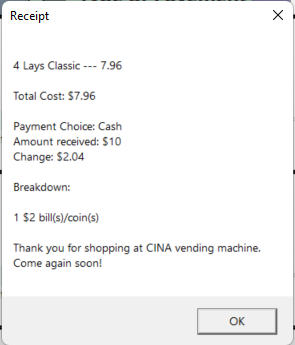
This demonstrates the user having added the chips to their cart, since the shopping cart now holds 4 bags of Lays Classic chips.



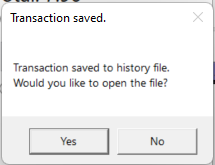
This is the screen that will show up after the user presses “proceed to checkout” on the main window.



This snapshot shows that the user has chosen to pay with cash, which means they have to enter the bill amount that they are using to pay.



This is the receipt the user receives after pressing on the “Checkout” button from the previous snapshot. One big difference between this receipt and the receipt the user receives when paying with debit or credit is that the user receives change, and therefore there is a breakdown of the change they will receive on the receipt.



This snapshot shows that the message notifying the user that their transaction has been saved pops up after they receive their receipt.

# Future Work

Discuss features or improvement that can be added to application.

After completing the application, there are a couple additions that could improve the application that would improve the overall experience of using the vending machine.

An improvement for the application would be to not allow the user to choose to add an amount of an item that is greater than the item’s stock. In this version of the application, the user can choose a quantity that is larger than the item’s stock, but they will be unable to add that quantity. Limiting a user’s selection ability to whatever the item’s stock is would give the user less false hope since they won’t have the option to choose something they cannot have.

Another improvement would be to replace the textbox that the user uses to indicate which bill they are putting in the machine to another system where they can enter multiple bills or coins. Currently, they are only able to enter one bill, but chances are a user will have coins on them that they might want to use instead.

One feature would be to allow the user to exit the checkout page and add or remove items from their cart before going to the checkout page once again. Currently, if the user wants to make changes to their cart once they have reached the checkout page, they are unable to.

Another feature that would be interesting to add would be the option to restock a product. Currently, the only way to restock a product is by changing the value in the stock file that is loaded on startup, but having that option while the application is already running would surely make it better. It would have to be set up in a way that only certain users would have the permissions to do this, which means that there would have to be some sort of identification or a passcode to enter before a button can be pressed, or something along these lines. This addition would definitely allow the application to run for longer periods of time.

# Teacher’s Comments from Demo:

These are the comments we noted down from the demo with the teacher.

The user experience was not good. There was no indication of the value of the sliders, an item’s stock, what is in the cart, which makes it extremely difficult to go through the application.

The slider value needs to be shown on the screen so that the user knows what quantity they are selecting.

The items in the shopping cart should be shown on screen so that the user knows what they have already added to their cart, since many people can easily forget what they have already added. This could be represented by a ListBox or a ListView on one of the sides of the window.

The start size of the window when the application first opens is quite awkward. It needs to open at the proper size, and also look nice when it is in full screen.

There should be images that show off the products instead of just having words.

Add a message box to indicate to the user that their receipt has been saved to a file. Additionally, could give the user the option to open the file their receipt was saved to.

Move the checkout information to a second window, and if any information needs to be passed between the windows, pass it using the constructor.

To avoid the program crashing, get the exception handling code from the CSConference app on GitHub. Can use that exact code as long as we add a comment mentioning where we got the code from.

For the snapshots, the application should be run without debugging so that the debugger options are not showing at the top of the windows.

Appendix A: Team Contract

This is an informal contract to ensure that all team members have a common understanding of what is expected in terms of work standards, communication, division or work, and conflict resolution.

# **Team Members (Name & ID)**

|  |  |  |
| --- | --- | --- |
|  | Name | Student ID |
| Member A: | Katchenin Cindy | 2126537 |
| Member B: | Breanna de Forest | 2145494 |

# **Strength & Weaknesses**

Within the context of this project, what are the strengths and weaknesses that each member brings to the team?

Member A: Strength: I’m punctual and easy to reach out. Weaknesses: I’m a person that gets distracted easily.

Member B: Strength: I’m a perfectionist so I always try to give my best work. Weaknesses: I procrastinate a lot.

# **Definition of “good enough” for this project**

What would the team collectively consider “good enough” of an achievement for the project?

Everything works and the UI looks decent.

*(One response for the whole team)*

Picked Topic

Vending Machine

Division of work

How will each member contribute to the project?

Member A: Front end

Member B: Back end

Frequency of communication

How often will the team be in touch and what tools will be used to communicate?

Every couple days. Maybe more when we get closer to the due date.

What is a reasonable delay to reply to messages? Is it the same for weekdays and weekends?

Response delays

A few hours. Less closer to the due date.

Receiving feedback

Each member must provide a sample sentence for how they would like to receive constructive feedback from their peers.

(If unsure, assume a hypothetical situation such as you have not completed your work in time or you have not replied to a message in a timely manner).

Member A: Hey Cindy, I have noticed that you haven’t done any of the work. I was wondering if you needed help with your part.

Member B: Hey Breanna. We agreed to have our parts done by time but you aren’t done yet. Has something happened? I’m not doing your work for you.

In case of conflict

If a team member fails to communicate as described in this contract or does not respond to constructive feedback, what measures should the other teammate take?

Spam message the other person across multiple messaging platforms, and if they still don’t respond, message the teacher and explain the situation.