1. **General Program Design**

I began the design process for this project by thinking about the ways in which each of my program’s component pieces—i.e., the vending machine itself, its user interface, and its inventories, both tangible and monetary—would need to be able to interact. In keeping with the project requirements, I only assigned a main method to my VendingMachineSimulator class, and I processed the majority of input/output commands from this main method, with occasional calls to other classes’ methods (some of which required user input) as needed. For the currency objects, I created a parent class called CurrencySystem, which I sub-class to create Currency, which is then sub-classed again to allow for the creation of PaperCurrency and MetalCurrency objects. The Currency System parent class allows for the Vending Machine’s bank to accept different national currencies and decline/accept bills or coins and give either bills or coins as change. With respect to the vending machine and its products, I created a VendingMachineProduct class and a VendingMachineBank class.

I read the product and currency inventories into my program from comma-delimited text files. Since both of these inventories are read in in this way, my program makes heavy use of arrays, if/else if loops, and a switch statement, with each case corresponding to user menu selections. When dividing the functionality among my classes, I endeavored to keep as much of the calculation and formatting within each class and out of my main program, so that I could call methods from each class from my main program to print formatted output (i.e. original and altered inventories; change, etc.) to the console. This did not work out exactly as cleanly as I might have hoped, but in the end, I focused on functionality and on ensuring that various use cases could be handled appropriately, and without the program breaking.

1. **Alternative Approaches Considered**

If I were to do this project again, I would focus on making a more appealing, intuitive user interface. For example, I originally planned to convert the vending machine inventory into a 2-dimensional array, with each item type’s placement in the array represented by a unique icon (i.e., all Coke bottles = “X” in this 2-D grid). This would allow for the implementation of a selection pad that more closely resembles real-life vending machines, in which users are prompted to select an item using alphanumeric coordinates. I rejected this approach because after reviewing the project specifications, it did not seem necessary; I do, however, feel it would be a helpful feature from the user’s perspective.

I also contemplated making the vending machine function more along the lines of an actual vending machine, in the sense that a typical vending machine forces you to put in money before you make a selection, and does not cancel your transaction (but may provide a coin return option) if you do not put in sufficient money. Rather, it will simply inform you that you have not put in enough money, and will wait for you (or, potentially, another user) to do so. This would have required reversing the steps outlined in the project requirements, and thus, I deemed such deviation to be potentially problematic.

1. **Lessons Learned**

This project helped to better understand the mechanics of a finite state machine. It was helpful for me to think about the vending machine as existing within a finite range of states, where transitions between states were caused by user interaction. It was also helpful for me to be able to associate programming concepts such as while loops with changes in the physical state of an actual machine prompted by various inputs the machine receives.

In general, were I to do this project again, I would begin with a more comprehensive diagram of each possible state of the vending machine that identifies each possible trigger that could move the machine from one state to the next. I would also try to come up with a different system for calculating and displaying change (as well as the altered currency inventory), as my initial approach to printing the bank inventory proved to be difficult to adapt in cases where the user wanted to put in more than one currency item as input and also needed to get change.

Additionally, I found in designing my classes and writing the methods for calculating product and currency inventories before and after transactions, that my VendingMachineProduct and VendingMachineBank classes had a lot in common and needed to be able to implement many of the same functions. Had I realized this earlier in the process, I could have reduced some of the redundancies in my code, and perhaps designed a parent class with common methods that both of these classes could have extended and customized as needed.