I have developed my Assignment 4 solution in order to meet the design goals for the vending machine. In particular, I have aimed to produce a solution that provides for easy extensibility to other forms of payment. Furthermore, I have attempted to design the system in such a way so that changing communication policies can be easily accounted for should this be required, and alternative hardware can also be easily included.

In essence, I have essentially split up the information that VendingMachine handles and created additional classes that handle the information, from which VendingMachine can interact with. With the addition of the CoinHandler and ProductHandler classes, the logic that VendingMachine now deals with is simplified. These two classes are now used to deal with the actions that the user can perform. For instance, after the user inserts a coin, CoinHandler will take note of the amount of money that was entered assuming that the coin was valid. When VendingMachine needs to use this information, it simply needs to call the appropriate method in CoinHandler.

After the user selects a pop, VendingMachine will use ProductHandler to retrieve the information about the selection, such as the kind of product and the cost of the product. This information can then be used against the information that CoinHandler provides. If there are sufficient funds to make the transaction, then the product is dispensed. Otherwise, we know that there are not sufficient funds to pay for the product, so the product is not dispensed.

In designing the system in this way, VendingMachine is no longer forced to deal with all of this information by itself. Clarity and understandability are therefore an advantage in this design, since VendingMachine need primarily deal with the logic, as opposed to handling the bulk of the data. In accordance with the design principles taught in class, this serves to promote divide and conquer (since we are separating the tasks to smaller more manageable tasks for each class), low coupling through information hiding, and a relatively simple and easy to understand design (since operations are grouped where one would likely expect them to occur).

This further provides added extensibility with regards to including other forms of payment. While it is not implemented, one could simply add extensibility features by adding methods that would deal with other forms of payment. For instance, if we were using a credit card as a form of payment, we could add a method cardPay that could check whether the credit card was maxed out. If not, then it can perform the transaction and then tell VendingMachine that the transaction was successful. The same could be said of including alternative hardware. There may need to be some changes as a result of different hardware, since there may be additional limitations in how transactions can be performed, the order in which the vending machine operates, limitations on payment and product choices, differences in its handling of change, etc. While it could be difficult to account for hardware changes, I believe that my solution would allow for these changes in an easier way, since the information is grouped in a logical way with regards to what each class does and what they do not. One can then make use of this information to account for how they will change the system. If one were presented with the old solution, it may be difficult to grasp where the relevant information is being held and operated on.

We note however that while designing the solution in this way, while it is the case that as a result, simplicity and added understandability results, this comes at the cost of additional flexibility since the information regarding coins and products are now handled elsewhere, separate from VendingMachine. That is, VendingMachine should now interact with the new classes to obtain information, instead of directly accessing it. In this case however, since we are aiming for ease of use and understanding, this appears to be a reasonable design pattern to adhere to.