Analysis of Vending Machine Simulator Program

Design Decisions and Reflections

by: Alina Grigorovitch

Introduction to Programming Using Java, section 82

Program Design and Analysis

Design and Implementation

My program was constructed with a main class, two separate class hierarchies for Snack and Currency, and several interfaces for performing actions on the data in instantiated classes. The functions of the main method were to: load the vending machine inventory, load the currency inside the machine, and serve as a user interface that prompted for choices and performed various transactions in response.

The class hierachies consisted of: a Snack superclass extended by two subclasses, Drink and SnackFood, that distinguished between the two major food groups found inside a vending machine; a Currency superclass extended by two subclasses, Coin and Bill, that distinguished between the two types of tender typically available. Originally the superclasses were abstract, but given the file I/O, it was easier to initialize an array of one type of class, and did not change the program's functionality to create objects of the subclasses.

The Snack superclass implemented the interface CostCalculator to find the cost of snack object given some additional circumstances, such as a user input quantity (which its method took as a parameter). The Currency superclass implemented the interfaces ManipulateCurrency, whose purpose was to perform additions, withdrawals, and state maintenance within the Currency object and keep track of the amounts of each currency when change was added and returned, and CurrencySymbols, which was a small interface that stored an association of currencies and their symbolic representations, useful for display purposes and easily customizable by editing the array values.

The main method began by instantiating object arrays for an inventory and a currency system and creating an object of BufferedReader class passed a new InputStream and the name of the file containing information. A for loop read each line and stored it in a temp String array that was split along the \t delimiter, then the constructor was created from data in the temp array. Next, it provided output and asked for user input, and looped through the options in a switch structure with a default setting to return an invalid input message. The first choice printed the display menu using the main class displayMenu method; the second and third choices printed the updated inventory and currency menus by calling toString and various accessory methods in the Currency and Snack classes. The bulk of the program rested in the last option, which performed the entire purchase and change retrieval exchange. First, it prompted the user for an item selection and a quantity selection. Because these were main method variables and not object variables, the total cost of the selection was done by the main method, but made use of the object method to access the selection's price. The user was then prompted to enter currency names until they chose to make a purchase, using a do..while loop. The program then matched the entered string with the name of the currency array to locate the object, obtain its value, and store it inside a temporary variable that incremented user additions (of ones, fives, quarters, etc) and another variable that stored the total value entered in the main method. If this passed the check of being greater than the cost of purchase, a change amount owed to the user was determined. A for(int) loop then ran through the currency array, ordered from greatest to least, to compare the change owed value to the values and quantities of each currency available, while tracking whether a currency object was removed. Finally, at the end of the transaction the object quantities were finalized and the temporary variables were reset with the appropriate reset methods.

Overall, my main goal was to create a program with maximum flexibility that would support future changes. For instance, I separated out tasks that were completely independent of the classes into interfaces for the purpose of being able to use something like CurrencySymbols in a variety of applications. ManipulateCurrency, too, can be used in any class that has to store added and subtracted amounts in "limbo", or that requires intermediate steps in adding two variables together, for instance in the event that a check needs to be performed. Though state maintenance could have been handled all in the main method, it would have made the code much more complicated, and hiding it inside an interface method went a long way in making the program more elegant. This could have also been a function of the classes, and the ManipulateCurrency interface was primarily an organizational tool.

Alternative and Additional Ideas

I had originally thought to create a more deeply nested class hierarchy and divide Drinks into Soda,Juice, while leaving Drinks non-abstract so that it could instantiate single object such as water without created a Water subclass. Although, now that I think about it, given the different kinds of waters there currently are, a Water class would be more future-oriented, and it would make sense after all to create a subclass for the sake of one object. For the scope of this project, however, that seemed unnecessary without giving any real structural benefit.

One element I would definitely add is a check in the main method after the user makes an inventory item selection to see whether item.quantity > 0, and if not, to return an error message. As always, error handling and invalid input is the area I would invest the most time in improving. Further, there was a small bug in how certain double values are handled, namely when that where there should have been a 0, there was a small remainder of around .009. I believe this is inherent to the way doubles are handled and next time would try to limit the number of digits to the right of the decimal place that can used for the monetary values, if that is possible. A final aspect I would like to improve upon for future programs is the initialization and input method for instantiating a objects. Rather than having a hardcoded number of objects in the object arrays for Snack and Currency, I would like the program to create objects of the specified class until the file read in is empty.

What I Learned

I learned a great deal from this assignment. During the course of putting the pieces together and trying to make them as elegant as possible, the way that the various parts of the language fit together as a whole clicked into place. It is hard to explain, but I think in the future the implementation of ideas will become more intuitive.

Specifically, by testing out various constructors in the main method, I got a good grasp on how to manipulate constructors and the use abstract reference variables for greater flexibility (for example, to make an array of different subclass objects). Also, I got a good feel for how to use objects to hide data manipulation, keep track of variables, and avoid incompatibility issues between data types. Most importantly perhaps, I got a better idea of how a program should be organized and what belonged to a class, what to an interface, and which actions required a method and which a method was unnecessary for.

References

1. “Java: The Complete Reference 9th Edition.” Schildt, Herbert. Oracle Press, 2014 McGraw Hill.
2. “Hear First Design Patterns.”Freeman, Eric; Freeman, Elisabeth; Bates, Bert; Sierra, Kathy. 2004 O'Reilly,