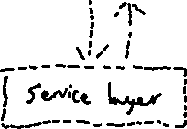
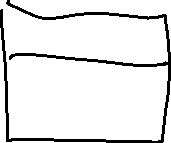
Features

* The program should **display all of the items and their respective prices** when the program starts, along with **an option to exit the program**.
  + Menu system
  + Display all items
* The **user must put in some amount of money before an item can be selected**.
  + Ask user to input money
  + **Then** ask user to select item
* Only **one item can be vended at a time**.
* If the user selects an item that costs more than the amount the user put into the vending machine, the program should **display a message indicating insufficient funds** and then **redisplay the amount the user had put into the machine**.
  + Check that inputted amount is less than price
    - If yes, output message and display amount again
      * (And ask for item to vend? Ask Eugene)
* If the user selects an item that costs equal to or less than the amount of money that the user put in the vending machine, the program should **display the change returned** to the user. Change must be displayed as the number of quarters, dimes, nickels, and pennies returned to the user.
  + If amount >= price, output change (use modulus exercise from earlier in course)
* Vending machine **items must be stored in a file**. Inventory for the vending machine must be read from this file when the program starts and must be written out to this file just before the program exits. The program must track the following properties for each item:
  + LoadFromFile at start, SaveToFile at end
  1. Item name
  2. Item cost
  3. Number of items in inventory
* When an item is vended, the program must update the inventory level appropriately. If the machine runs out of an item, it should no longer be available as an option to the user. However, the items that have an inventory level of zero must still be read from and written to the inventory file and stored in memory.
  + Item class with inventory attribute, when zero cannot vend

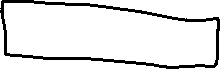
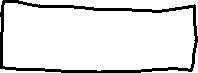
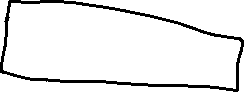
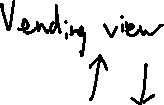
Guidelines

* You should take considerable time to design this application before you even think about writing code. Follow the Service Layer and DAO interface design approaches shown in the write-ups and videos.
* This application must follow the MVC pattern used for all previous labs (App class, Controller, View, Service Layer, DAO) – this includes the use of constructor-based dependency injection.
* You must have **a full set of unit tests** for your DAO and Service Layer components.
* You must use BigDecimal for all monetary calculations where applicable.
* You must include at least one lambda function in the solution.
* You must use **application-specific exceptions** and your application must fail gracefully under all conditions (i.e. no displaying a stack trace when an exception is thrown). At a minimum you should have the following application-specific exceptions thrown by your Service Layer:
  + One that is thrown when the user tries to purchase an item but doesn't deposit enough money (i.e. InsufficientFundsException)
  + One that is thrown when the user tries to purchase an item that has zero inventory (i.e. NoItemInventoryException)
* Use enums to represent the values of different coins.
* Include an Audit DAO to log events and the time they occurred.

Initial MVC for vending machine (28/06/2022)



1. View.DisplayMenu()
2. GetUserInput() (Money in, and item requested)
3. VendItem(Item requested) returns Transaction DTO
4. If transaction is successful, deincrement stock and output change (call view.displayChange(double), converts to currency denominations)
5. If not, “**display a message indicating insufficient funds** and then **redisplay the amount the user had put into the machine**“ again ask Eugene if we got straight to asking for a new item to vend or ask for more change
6. Head back to the start



Diagram

Description automatically generatedFlowchart: Class Diagrams: