Mini Project - 3

Welcome to this Mini Project.

Now you work for a software development company that has been hired to develop the software of the vending machines in your local hospitals and schools.

- Your job is to customize the behavior of the existing generic VendingMachine class using method inheritance and overriding.

- You need to create two subclasses: HospitalVendingMachine and SchoolVendingMachine that inherit from VendingMachine (this class is already defined in the VendingMachine.py file that you need to download for this project).

Both

- These subclasses must have a custom greeting message for the user, so you must override the sales_menu method in these two subclasses. You must make them print the corresponding message (shown below) before calling the sales_menu method of the VendingMachine superclass.

The custom messages that must be printed before calling sales_menu() in the subclasses are:

- For HospitalVendingMachine:

print("\n========== Welcome to our Hospital Vending Machine ========== \nWe hope you are feeling better today!\n")

- For SchoolVendingMachine:

print("\n========== Welcome to our School Vending Machine ========== \nWe hope you have a great day full of learning!")

Note: \n indicates that a new line will be printed.

- Add their corresponding __init__() to make them inherit the attributes of VendingMachine. (optional).

- Add a class attribute snack_prices in these two subclasses to replace the value defined in the superclass.

You can customize the prices (values) to your liking but please maintain the structure of the original dictionary.

- Override the method find_snack_price in both subclasses to make them use the class attribute that corresponds to the subclass.

HospitalVendingMachine

- In HospitalVendingMachine, add a method print_days_until_maintenance that prints a string with the number of days remaining until the machine needs maintenance (Tip: it's an instance attribute of the superclass).

SchoolVendingMachine

- In SchoolVendingMachine, add a class attribute student_debt (a dictionary with students' names as the keys and their corresponding debt with the vending machine as values).

- Then, add a method print_student_debt that takes the name of the student as an argument and prints the value of his/her debt.

Important: the VendingMachine class has many methods defined, but you already know how to work with every element in the file. You have worked very hard during the course and you are now capable of understanding this program.

I suggest taking some time to read through the code and familiarize yourself with this class. This is intended to help you practice working with existing code and reading code written by other developers. I included descriptive comments for each one of the methods, but please do not hesitate to ask if you have any questions.

This is the code in VendingMachine.py (please download the .py file below).

```
class VendingMachine:

    total_revenue = 0 # Total revenue of all vending machines in the system

    snack_prices = {"candy": 2.00, "soda": 1.50, "chips": 3.00, "cookies": 3.50}

    # Instance attributes
    def __init__(self, inventory, serial, days_until_maintenance):
        self.inventory = inventory # dictionary with {<snack>: <amount>} as key-value pairs.
```

```
Possible snacks: candy, soda, chips, cookies. Keys written in lowercase.
        self.revenue = 0           # Initially, when an instance of the vending machine is
created, the revenue is 0 and it's updated with each sale.
        self.serial = serial
        self.days_until_maintenance = days_until_maintenance

    # Method that displays an interactive menu to process a sale.
    # Displays the options, gets user input to select the snack, and calls
    # another method to process the sale.
    def sales_menu(self):

        # The user has the option to buy several types of snacks
        # so the program is repeated if the user indicates that he/she
        # would like to buy another snack
        while True:

            greetings = "\nWelcome! I have:\n"
            request = "\nPlease enter the number of the item: "

            # Print a welcome message with the snacks available
            print(greetings)

            i = 1
            for snack in self.inventory:
                print("(" + str(i) + ") " + snack.capitalize())
                i += 1

            # Get the user input (option selected)
            cust_input = int(input(request))

            # Repeat if the input doesn't meet the requirements
            while cust_input <= 0 or cust_input > len(self.inventory):
                print("Please enter a number from 1 to", len(self.inventory))
                # Get the user input (option selected)
                cust_input = int(input(request))

            # Display appropriate message
            self.process_sale(list(self.inventory.keys())[cust_input - 1].lower())
            answer = int(input("\nWould you like to buy another snack?\nEnter 1 for YES and 0 for
NO: "))

            # If the customer does not wish to buy another snack
            if not answer:
                break


    # Method that processes the sale by asking the user how many snacks of that type
    # he/she would like to buy and calls another method to opdate the inventory
    def process_sale(self, option): # option must be in lowercase

        print("\nYou selected: %s" % option.capitalize())

        if self.inventory[option] > 0:

            # Display current snack inventory and product
            print("Great! I currently have %d %s in my inventory\n" % (self.inventory[option],
option))
```

```
            # Ask for the number of snacks
            num_items = int(input("How many %s would you like to buy?\n" % option))

            # Handle cases where user might enter a negative number or zero
            while num_items <= 0:
                print("Please enter a positive integer")
                num_items = int(input("\nHow many %s would you like to buy?\n" % option))

            # Update inventory if there are enough snacks available
            if num_items <= self.inventory[option]:
                self.remove_from_inventory(option, num_items)

                # Update the machine's revenue
                total = self.update_revenue(option, num_items)

                print("That would be: $ " + str(total))

                # Display a message confirming the purchase and current inventory
                print("\nThank you for your purchase!")
                print("Now I have %d %s and my revenue is $%d" % (self.inventory[option], option,
self.revenue))

            else:
                print("I don't have so many %s. Sorry! :(" % option)

        else:
            print("I don't have any more %s. Sorry! :(" % option)


    # Method that updates the vending machine's (instance) inventory by
    # decrementing the availability of the snack chosen.
    def remove_from_inventory(self, option, num_items):
        self.inventory[option] -= num_items

    # Update the revenue of the instance of VendingMachine
    # and update the class attribute total revenue.
    def update_revenue(self, option, num_items):
        # Find price of the snack
        price = self.find_snack_price(option)

        # Update Instance and class
        self.revenue += num_items * price
        VendingMachine.total_revenue += num_items * price

        return num_items * price

    # Find the price of the snack selected in the class attribute
    # and return it
    def find_snack_price(self, snack):
        return VendingMachine.snack_prices[snack]

    # Method that prints a message with the total revenue of the instance of VendingMachine
    def display_revenue(self):
        print("The total revenue of this vending machine is:", self.revenue)


# Subclasses
```

```
# Subclasses
# ============================================================================================

class HospitalVendingMachine(VendingMachine):

    # Complete the class



class SchoolVendingMachine(VendingMachine):

    # Complete the class


# Instances
```