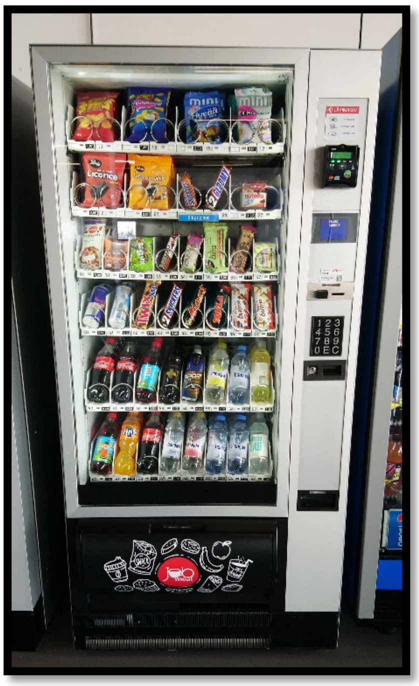
Assignment 1 – Vending Machine

Object Oriented Programming

University of Southern Denmark

Due date: 21st Oct. 2022, 23:59



This assignment considers a simple vending machine that is able to dispense snack products and soft drinks after receiving a payment. The machine consists of three sub-components; a compartment for storing snacks, a compartment for storing soft drinks and a compartment for storing cash. The snacks and soft drinks compartments store snack and soft drink objects, respectively.

General Requirements

The assignment requires you to define six classes, and these must be part of your solution. For each class, I have added an overall description, along with a set of specific requirements. The tasks stated in the requirements sections must be fulfilled in the solution for the assignment to be accepted. Each class gets more detailed descriptions of fields, constructors and methods that should be part of these classes. However, the descriptions for the fields/constructors/methods, particularly the hints, should be seen more as recommendations, and you can deviate from them if you come up with other solutions. Furthermore, if you want to add additional features to represent a smarter/more realistic vending machine, feel free to do so.

One particular requirement for all classes:

Every field variable you define within a class must use the private modifier. Get and Set methods must be used to access or change the values of these variables.

As part of the assignment, you must document your code. This can either be done in a separate document or, alternatively, you can add the documentation directly as comments in the code. The documentation must describe:

- The overall logic of each class (a few lines)
- The overall logic and purpose of each constructor in a class (a few lines)
- The overall logic of each method in a class (a few lines)
- The property that each field variable represents (one line)

If you make any particular design choices, you must document those as well, along with the reason for applying them. Also, if you identify any limitations to your defined logic, you should state those as well.

You are free to choose the names of the classes, methods and variables in your code. However, they should be sensible and follow the style provided in the Lecture 2 presentation and: https://www.bluej.org/objects-first/styleguideplain.html

I recommend that you use BlueJ for the assignment. However, if you prefer another IDE, you can use that instead.

Also, please note that some of the requirements and suggested hints/solutions in this document do not necessarily to the best solution in terms of design. However, they are proposed to make you use most of the concepts that have been covered in lectures up to now.

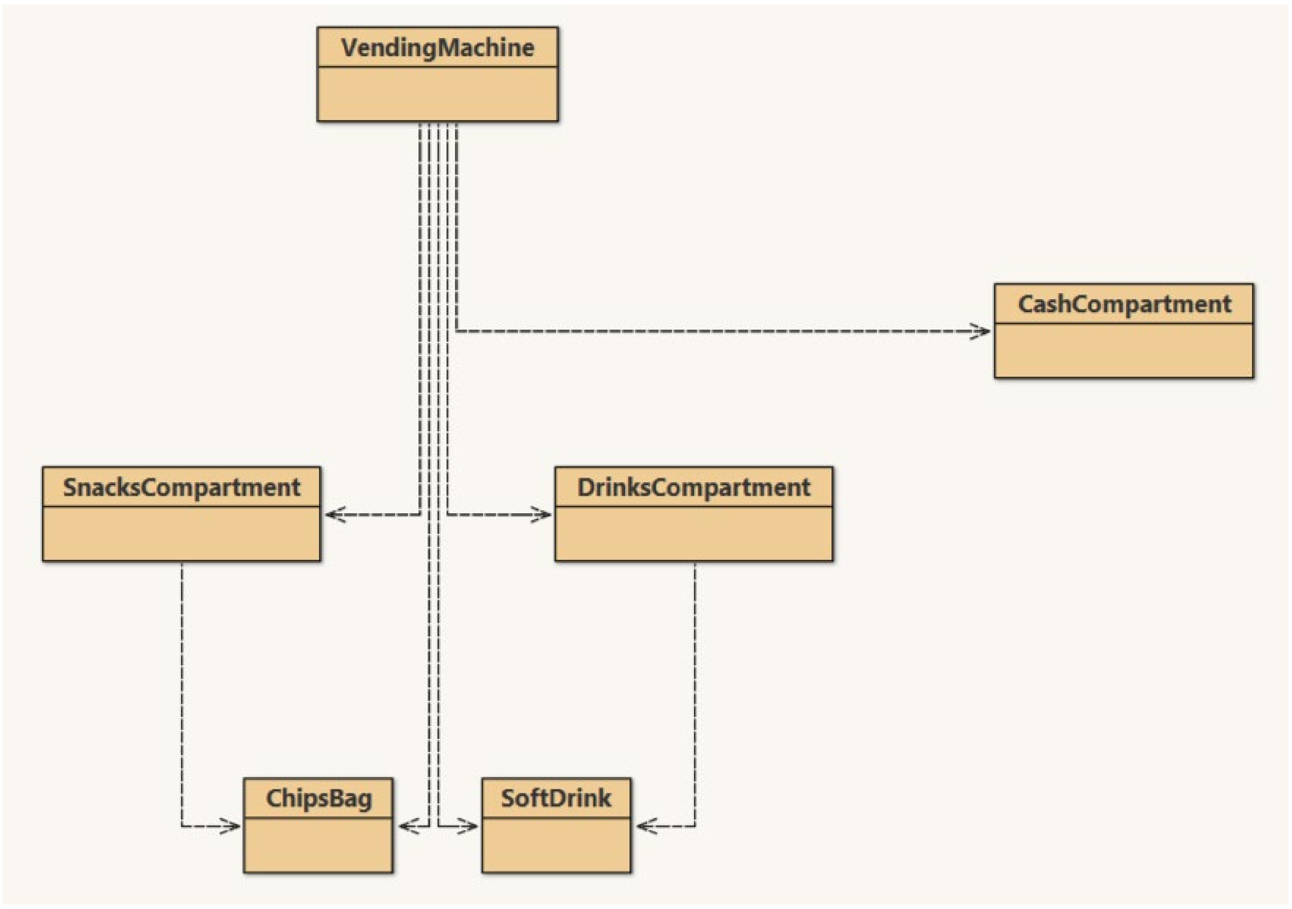
Classes

As the first step, the following set of new and empty classes must be defined:

- Vending machine
- Snacks compartment
- Soft drinks compartment
- Snack
- Soft drink
- Cash compartment

The requirements and recommended content for each of these classes are covered in the following subsections. Note that a bottom-up approach is used, i.e., defining the classes whose objects form sub-parts of objects of other classes.

Also note, if using BlueJ and not adding additional classes, you should have a class diagram similar to the following when you are finished (again, you can use your own class names instead):



1. Soft Drink/Snack

The soft drink and snack classes must be used to create objects representing single units of soft drinks and snacks that are stored in their respective compartments in the vending machine and can be dispensed when purchased.

Requirements

Soft drink and snack objects should be very simple and should not be able to do any actions on their own. Their only distinct property is their product name, e.g., "Coca-Cola Zero" or "Snickers", which must be retrievable using a Get method. ✓

The two classes must be identical (except for their names). It is therefore recommended that you name the fields and methods generically and then create a copy of the first class and change its class name to make the other class. ✓

Fields

The two classes should include the following field variable:

F1. Product name of soft drink/snack ✓

Constructors

The two classes should include a constructor that performs the following task:

C1. Takes a name as a parameter and sets the product name of the created object to that value ✓

Methods

The two classes should include a method that performs the following task:

M1.Returns the value of the name parameter (i.e., a Get method) ✓

2. Soft Drinks Compartment

The soft drinks compartment class must be used to create objects that represent compartments in vending machines that are specifically used to store and dispense soft drinks. They have a limited number of slots to store soft drink objects, and these slots can be emptied (upon purchases) and restocked.

Requirements

The drinks compartment class must fulfil the following list of requirements:

- **Contain a limited number of slots that can store soft drink objects**
- **Count the amount of currently stored soft drinks with a specific product name and return this number**
- **Count the total amount of stored soft drinks and return this number**
- **Return a requested amount of soft drink objects with a specific product name from its stocks. These must be removed from a corresponding number of slots. If there are not enough soft drinks to cover the request, either the action must be cancelled, or all the remaining soft drinks must be returned and removed from their slots. In both cases, a corresponding message must be printed to the console.**
- **Restock empty slots with a provided number of soft drinks with a specific product name. If there are not enough slots left, either the action should be cancelled or all remaining empty slots should be restocked. In both cases, a corresponding message must be printed to the console.**

Important: The slots for storing drinks must be represented by an array (not a list) that stores references to soft drinks for filled slots and null values for empty slots. The length of this array must correspond to the capacity of the compartment.

Fields

The drinks compartment class should include the following fields:

F1. Capacity, i.e., the total number of soft drink objects that can be stored in the compartment
F2. Listing of soft drinks stored in the compartment (must be an array)

Constructors

The drinks compartment class should include constructors that perform the following tasks:

C1. Takes no parameters, sets the capacity at a default value and initialises the soft drink listing field with an array length corresponding to the capacity value, all entries set to null
C2. Does the same as the constructor above but takes a parameter which is used to set the capacity

Methods

The drinks compartment class should include methods that perform the following tasks:

M1. Counts the total number of stored soft drink objects in the compartment.
Hint: This can be done by iterating through the field variable array containing the references to the stored soft drinks and incrementally counting the occurrences of non-null values.

M2. Counts the number of soft drinks with a specific product name that is passed as a parameter.
Hints: Do the same as in method M1 but, additionally, check for soft drinks whose product name value matches the passed parameter value. Access the value of each soft drink by using the `.` (dot) character preceded by the array entry name and followed by the Get method for the product name in the soft drink object, e.g., `drinksArray[i].getProductName()`.

List of soft Drinks

Check whether the non-null condition is fulfilled before checking the product name, as you might otherwise get a null-pointer error if there are empty slots in the array! Also, remember to use the `equals` operator when comparing product names instead of == !

break command does not work within if statement, but not needed since else is only used when if is falsed. #class: SoftDrinksCompartment #method: refill

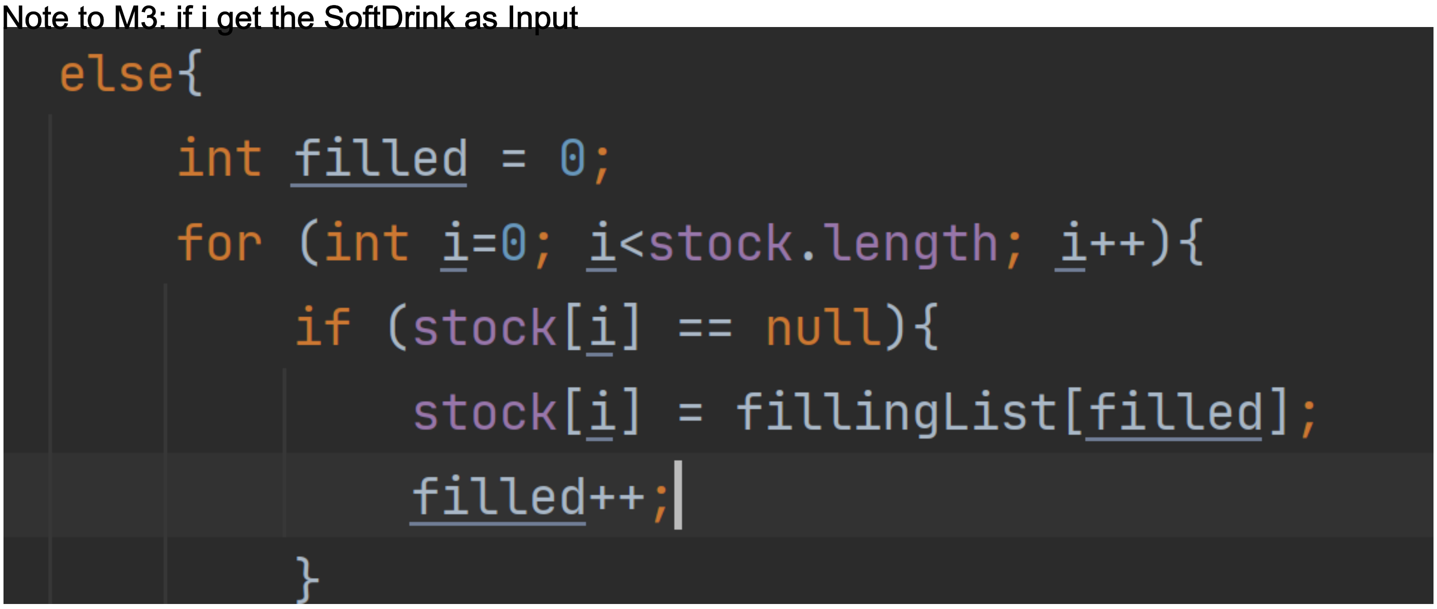
M3. Takes a ~~number of soft drinks~~ with a specific product name as parameters and restocks the empty slots by creating the stated number of new soft drink objects and storing their references in empty slots in the soft drinks listing field array. Consider the order of empty slots that are filled, and consider actions to be taken if there are not enough slots to store all the new soft drinks.
Hints: Keep the logic simple and iterate through the array and fill null value entries with new soft drinks (use the `new` keyword with the soft drink constructor to do this) as they are encountered during the iteration. Keep track of the number of soft drinks added using a local variable, and if it reaches the number passed as an input parameter, stop the iteration with the break keyword. If the iteration stops before this number has been reached, print a message to the console with the number of soft drinks that were added to the compartment and the number of drinks that were discarded due to insufficient free slots.
If you instead want to cancel the restock action altogether when there is insufficient space for the provided number of soft drink objects, compare this number with the number of free slots (use the method from M2 for this, and subtract the returned value from the capacity field value to get the free slots number). Make this comparison as the first action in the method, and if it is true, print a message to the console and use the `break` keyword to stop the execution of the remaining statements in the method.

M4. Takes a requested number of soft drinks with a specific product name as input parameters and returns (dispenses) a corresponding number of the soft drink objects stored in the soft drinks listing field array. These objects must subsequently be removed from the array by replacing them with `null`. As there might be multiple soft drink objects returned by the method, the return value should be an array of soft drinks.
Hints: Start by checking whether there are enough soft drinks in the compartment to cover the ordered amount. Use the method from M2 for that. Decide whether you want the action to be interrupted (by using the `break` keyword) or dispense all the remaining soft drink objects.
Then create a new array to store the soft drinks to be dispensed, the output array. The length must match the requested amount (or the remaining number of drink objects if this is smaller). Also, declare and initialise a local variable to keep track of the current position in the output array.
Then, use an iteration similar to the one in method B to check the presence and product name of all soft drink objects in the stored drinks listing field array. If there is a match, assign the soft drink in the field array entry to the current output array entry (remember that only reference to the soft drink object is copied to the output) and increment the local position variable by one. Then remove the soft drink from the field array by replacing the drink object reference with null.
If the position variable value equals the number of drinks to be dispensed, stop the method execution by returning the output array using the `return` keyword. Otherwise, return the array after the iteration has finished.

M5. Takes a soft drink product name as an input parameter and returns (dispenses) exactly one soft drink (if possible). It must have the same method name and output type (an array containing objects of the soft drink type) so that overloading is employed.
Hint: Simply call method M4 with the value 1 for the first input parameter and return the output from this call. No further work is necessary for this method.

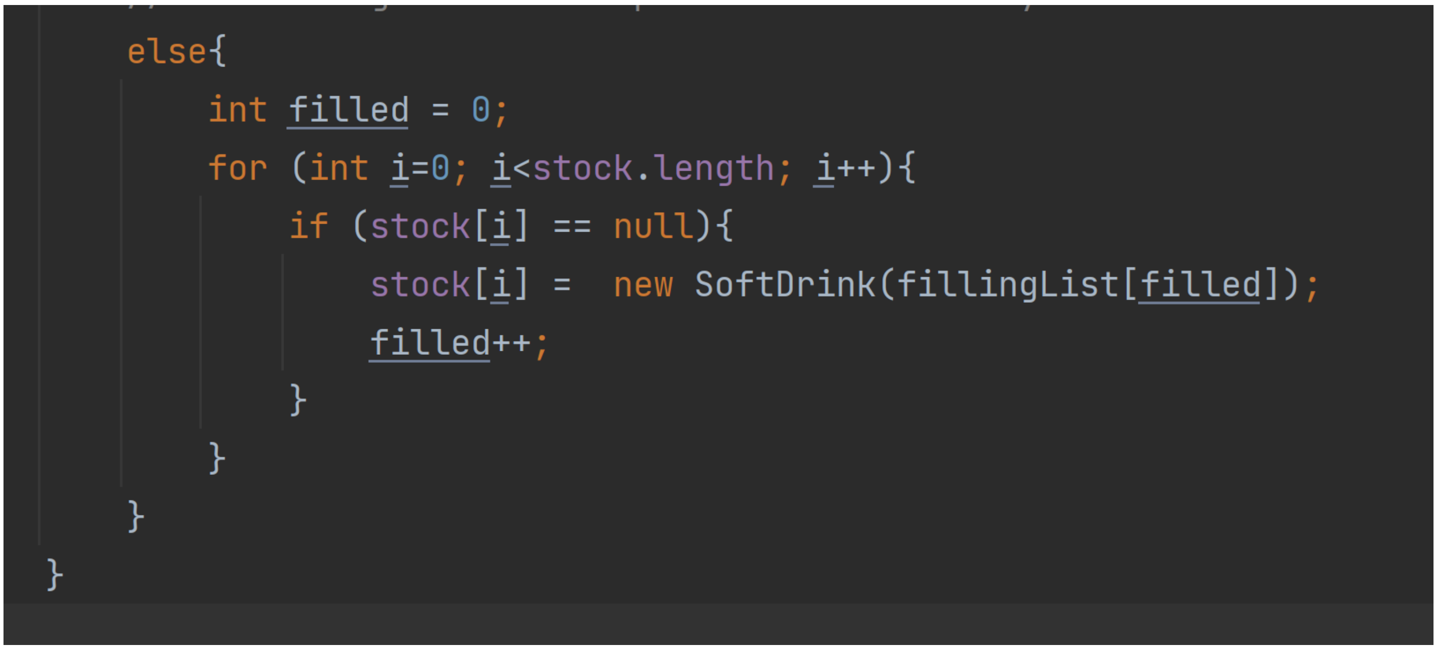
M3: I think i dont get a list of soft drinks as input,
i only get a list of names of soft drinks
I implemented it like we are getting only the names not the soft drink, ask the prof what he wants. If we get the finished soft drink, than it makes no sense to use the new keyword because of the showed code work below here.

Note to M3: if i get the SoftDrink as Input



```
else{
    int filled = 0;
    for (int i=0; i<stock.length; i++){
        if (stock[i] == null){
            stock[i] = fillingList[filled];
            filled++;
        }
```

M3 if i get the name of the Softdrink as an input:



```
    else{
        int filled = 0;
        for (int i=0; i<stock.length; i++){
            if (stock[i] == null){
                stock[i] =  new SoftDrink(fillingList[filled]);
                filled++;
            }
        }
    }
}
```

3. Snacks Compartment

The snacks compartment class must be used to create objects that represent compartments in vending machines that are specifically used to store and dispense snacks. They have a limited number of slots to store snack objects, and these slots can be emptied (upon purchases) and restocked.

Requirements

The requirements for the snacks compartment class are exactly the same as for the soft drinks class. This class should therefore be a copy of the soft drinks compartment class where the names of the variables, methods and constructors have been changed accordingly (use search and replace for this). Also, the default compartment capacity for the compartment can be changed to differ between the two classes.

[illegible]

4. Cash Compartment

The cash compartment class must be used to create objects that represent compartments in vending machines that are used to store received cash and keep track of payments from individual purchases.

Requirements

The cash compartment class must fulfil the following list of requirements:

- **Store the total amount of cash received through drinks/snacks purchases**
- **Store a list of payments that have been received for each purchase**
- **Provide information on the current cash balance in the vending machine**
- **Provide information on the amount of cash received for a specific purchase**

Important: The information on purchase payments must be stored in an ArrayList object, not an Array. For this purpose, add the `import java.util.*;` statement at the very beginning of the class file.

Fields

The cash compartment class should include the following fields:

F1. Cash balance, i.e., the total amount of cash received from purchases

F2. Individual purchase payments list, i.e., list of cash amounts received from purchases in order of when they were performed.
Hint: This list should be initialised per default, i.e., assign an empty ArrayList object containing the desired object type at the declaration statement. Also, keep it simple and store double values in the list.

Constructors

No particular constructors are needed for the cash compartment class. The default one is sufficient.

Methods

The cash compartment class should include methods that perform the following tasks:

M1. Returns the current amount of cash contained.
Hint: This is a simple Get method.

M2. Receives an input amount of cash, adds it to the cash balance and adds an entry with the payment to the list of received payments from purchases.
Hint: This is almost a simple Set method. Increment the cash balance variable and use the `.add()` method for ArrayList objects to add the payment information.

M3. Receives the index number of a purchase whose information must be retrieved and returns the corresponding payment from the payment list.
Hints: This is a specific version of a Get method. Use the `.get()` method for ArrayList objects to retrieve the payment information.
It might also be a good idea to use a conditional statement to compare the input parameter value with the number of entries in the list (use `.size()`) to prevent nullpointer errors when a non-existent list entry is requested. Print a message to the console and return a value of 0.0 in this case.

5. Vending Machine

The vending machine class must be used to create objects that represent vending machines which accept purchase requests for a number of products with a certain product name and dispenses those according to a received payment and the available number of products in the compartments it contains. It returns the excess amount of cash paid upon purchase. Furthermore, the machine accepts restocking of products to its contained compartments.

The vending machine contains one soft drink compartment, one snacks compartment and one cash compartment as subcomponents. It is hence the top-level class in the hierarchy of classes defined (if the Main is disregarded) for this assignment and is the only one accessible to customers who want to buy products or employees who want to restock them.

Requirements

The vending machine class must fulfil the following list of requirements:

- **Contain references to objects of the three different compartment classes, one for each.**
- **Contain price information for snacks and soft drinks, respectively. For simplicity, a single price should be applied to all product names of each product type.**
- **Accept restock requests for a particular product type and forward the request to the compartment containing that product type.**
- **Accept a purchase request for a number of products with a specific product name, along with a cash payment. It must then return a number of products, subject to the limitations of insufficient payment and/or the number of products in stock.**

Note that the vending machine must be responsible for the purchase process itself, but not the processes that are carried out by the compartment objects. Instead, it should call the relevant methods at those to make them carry out their individual responsibilities, e.g., managing stocks of soft drinks.

Fields

The vending machine class should include the following field variables:

F1. Soft drinks compartment object reference. An object must be created and assigned as part of the declaration, as the vending machine will not work as intended if missing this component.
Hint: Consider which constructor for the drinks compartment to call: either the one with the default capacity or the one where you set it. Remember the `new` keyword.
F2. Snacks compartment object reference. Similar to F1.
F3. Cash compartment object reference. Similar to F1 and F2.
F4. Price for purchasing snacks
F5. Price for purchasing soft drinks

Constructors

The vending machine should include a constructor that performs the following task:

C1. Takes a parameter for the snack price and another for the soft drink price and assigns them to the two respective field variables.
Hint: Remember to use the `this` reference keyword if using the same names for parameters and field variables.

Methods

The vending machine class should include methods that perform the following tasks:

M1. Receives a request for a number of soft drinks with a specific product name along with an amount of cash payment as input parameters. Then checks if the purchased amount is limited by too little cash and/or the number of products in stock and print a corresponding message to the console if this is the case. Finally, retrieves the required (and possibly limited) number of objects from the soft drinks compartment, adds the cash paid for the products to the cash compartment, prints a message with the refunded cash (i.e. the remainder after purchase) to the console and returns the purchased products in an array.
Hints: Start by declaring two local variables to denote the limits from insufficient payment and stock, respectively, and then set them to the number of products. Then calculate the limit from payment (use `Math.floor()` to round down and `(int)` to convert from `double` to `int`), print a message to the console and assign the limited value to the respective local variable. Do the same for the stock limit (here, you can call the appropriate method at the soft drinks compartment to get the number of products available).
Then determine the minimum among the requested amount of product and the two local variables using `Math.min()` (you have to use the method twice, as it only takes to parameters).
Finally, insert the cash for the determined number of purchased soft drinks by calling the appropriate method at the cash compartment object, and print the remaining amount to the console. Then get the products dispensed from the soft drinks compartment and return them directly as the outputs of this method.
Note: If using the above approach, ensure that the compartment returns the remaining available soft drink objects when stock is limited instead of cancelling the action. If the latter is the case, you should adjust this method accordingly. Otherwise, vending machine steals the cash, and the customer will pay for nothing. 😉

M2. Same as M1, but for the purchase of snacks.
Hint: Copy M1 and change the method and variable names and method calls accordingly.

M3. Receives a number of soft drinks and a product name as parameters and restocks the soft drinks compartment accordingly.
Hint: Call the appropriate method at the stored soft drinks compartment object and pass along the respective parameters. The compartment object must take care of this task.

M4. Receives a number of snacks and a product name as parameters and restocks the snacks compartment accordingly.
Hint: Same as M2.