Candy Machine

A common place to buy candy is from a candy machine. A new candy machine is bought for the school, but it is not working properly. The machine sells candies, chips, gum and cookies. You have been asked to write a program for this candy machine so that it can be put into operation.
The program should do the following:

1. Show the customer the different products sold by the candy machine
2. Let the customer make the selection
3. Show the customer the cost of the item selected
4. Accept money from the customer
5. Release the item

A candy machine has two main components: a built-in cash register and several dispensers to hold and release the products. Therefore, we need to define a class to implement the cash register, a class to implement the dispenser, and a class to implement the candy machine.

Below is the UML diagram of the class `CashRegister` and class `Dispenser`.

```
CashRegister
--------------------------
-cashonHand: int
--------------------------
+CashRegister()
+CashRegister(int)
+currentBalance():int
+acceptAmount(int):void
```

```
Dispenser
--------------------------
-numberOfItems: int
-cost: int
--------------------------
+Dispenser()
+Dispenser(int, int)
+getCount():int
+getProductCost():int
+makeSale():void
```

Cash Register
Let us first discuss the properties of a cash register. The register has some cash on hand (cashonHand), it accepts the amount from the customer, and if the amount entered is more than the cost of the item, then, it returns the change. The cash register should also be able to show the candy machine's owner the amount of money in the register at any given time.

Definition of the methods

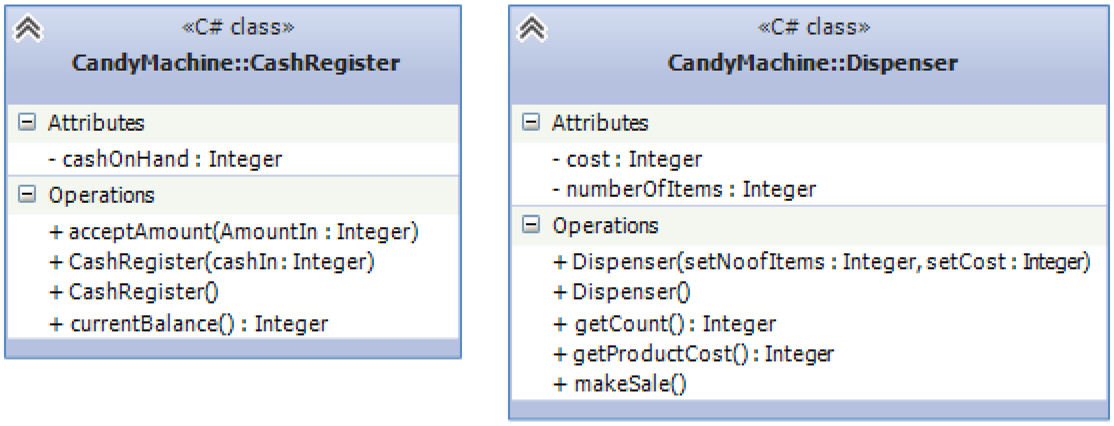
1. The default constructor sets the value of the instance variable `cashOnHand` to 500 cents.
2. The constructor with a parameter sets the value of the instance variable to the value specified by the user. The value is passed as a parameter to the constructor. Note that the constructor checks for valid values of the parameter. If the value of the parameter is less than 0, the value assigned to the instance variable is 500.
3. The method `currentBalance` shows the current amount in the cash register.
4. The method `acceptAmount` accepts the amount entered by the customer. It updates the cash in the register by adding the amount entered by the customer to the previous amount in the cash register.

Dispenser
The dispenser releases the selected item, if it is not empty. It should show the number of items in the dispenser and the cost of the item.

Definition of the methods:

1. The default constructor assigns the default values to the instance variables. The default value of both instance variables is 50.
2. The constructor with parameter checks for valid values of the parameters. If these values are less than 0, the default values are assigned to the instance variables. The default value of each instance variable is 50.
3. The method `getCount` returns the number of items of a particular product. Because the number of items currently in the dispenser is stored in the instance variable `numberOfItems`, it returns the value of the instance variable `numberOfItems`.
4. The method `getProductCost` returns the cost of a product. Because the cost of a product is stored in the instance variable `cost`, it returns the value of instance variable `cost`.
5. When a product is sold, the number of items in that dispenser is reduced by 1. Therefore, the method `makeSale` reduces the number of items in the dispenser by 1. That is, it decrements the value of the instance variable `numberOfItems` by 1.

The figure below show the expected UML diagram of the two classes described above.


«C# class»
CandyMachine::CashRegister
Attributes
- cashOnHand : Integer
Operations
+ acceptAmount(AmountIn : Integer)
+ CashRegister(cashIn : Integer)
+ CashRegister()
+ currentBalance() : Integer
«C# class»
CandyMachine::Dispenser
Attributes
- cost : Integer
- numberOfItems : Integer
Operations
+ Dispenser(setNoofItems : Integer, setCost : Integer)
+ Dispenser()
+ getCount() : Integer
+ getProductCost() : Integer
+ makeSale()

Main Program
When the program executes, it must do the following.

1. Show the different products sold by the candy machine
2. Show how to select a particular product
3. Show how to terminate the program

Furthermore, these instructions must be displayed after processing each selection (except exiting the program), so that the user need not to remember what to do if he or she want to buy two or more items. Once the user makes the appropriate selection, the candy machine must act accordingly. If the user opts to buy a product and if that products is available, the candy machine should show the cost of the product and ask the user to deposit the money. If the money deposited is at least the cost of the item, the candy machine should sell the item and display an appropriate message.

Method `showSelection`
This method displays the necessary information to help the user select and buy a product. Essentially, it contains the following output statements. Let's assume that the candy machine sells four types of products.

```
**** Welcome to Rogie's Candy Shop ****
To select an item enter
1 for Candy
2 for Chips
3 for Gum
4 for Cookies
0 to View Balance
9 to Exit
Enter you choice:_
```

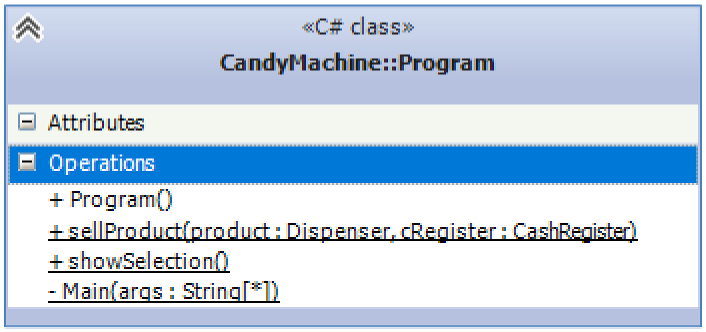
Next we describe the method `sellProduct`

Method `sellProduct`
This method attempts to sell the products selected by the customer. Therefore, it must have access to the dispenser holding the product. The first thing that this method does is check whether the dispenser holding the product is empty. If the dispenser is empty, the method informs the customer that this product is sold out.

If the dispenser is nonempty, it tells the user to deposit the necessary amount to buy the product. If the user does not deposit enough money to buy the product, the method `sellProduct` tells the user how much additional money must be deposited. If the money deposited by the user is sufficient, it accepts the money and sells the product. Selling the product means to decrement the number the number of items in the dispense by 1,and update the money in the cash register by adding the cost of the product. If the user entered money more than the cost of the product, the program will return the change and only capture the original cost of the product.

From this discussion, it is clear that the method sellProduct must have access to the dispenser holding the product as well as the cash register. Therefore, this method has two parameters: one corresponding to the dispenser, and the other corresponding to the cash register.

The figure below show the expected UML diagram of the Main program described above.


«C# class»
CandyMachine::Program

Attributes

Operations

+ Program()
+ sellProduct(product : Dispenser, cRegister : CashRegister)
+ showSelection()
- Main(args : String[*])