Coding Assignment 3

Building a CokeMachine

Objective

Create a class called CokeMachine and a `Code3_xxxxxxxxxx.java` program to instantiate your own CokeMachine and exercise its functionality.

Submission

You should submit 2 files in a zip file named `Code3_xxxxxxxxxx.zip`

`Code3_xxxxxxxxxx.java`

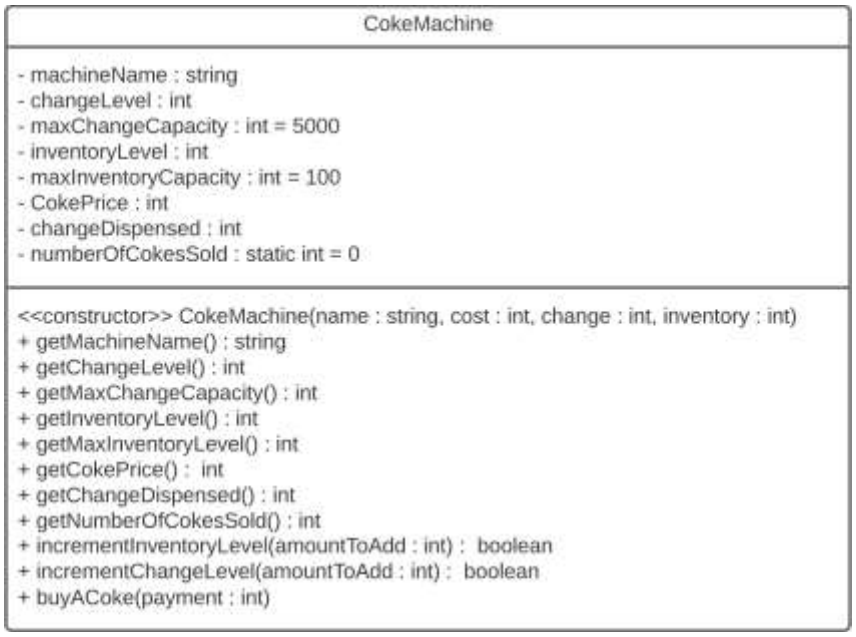
`CokeMachine.java`

Assumptions

1. For now, the CokeMachine only dispenses Coke and no other types of items.
2. Payments are entered in pennies - $0.50 is entered as 50, $1.00 is entered as 100.
3. Any displays of change in `Code3_xxxxxxxxxx.java` will be in dollars and cents.
4. Any payment entered cannot be used as part of the dispensed change. For example, if a Coke costs 30 cents and the user enters 1234 ($12.34) for payment, then the machine will only dispense a Coke if is has enough change for 1204.
5. When restocking the machine, the entire restock request has to fit. If the entire quantity does not fit, then the restock request is rejected.
6. When adding change to the machine, the entire amount has to fit. If the entire amount of change does not fit, then the change addition is rejected.
7. Your machine only dispenses 1 Coke at a time.

Class Diagram

Use the following Class Diagram to start creating your CokeMachine class. Please note that exact names (spellings and case) **must** be used.



```
CokeMachine
- machineName : string
- changeLevel : int
- maxChangeCapacity : int = 5000
- inventoryLevel : int
- maxInventoryCapacity : int = 100
- CokePrice : int
- changeDispensed : int
- numberOfCokesSold : static int = 0
<<constructor>> CokeMachine(name : string, cost : int, change : int, inventory : int)
+ getMachineName() : string
+ getChangeLevel() : int
+ getMaxChangeCapacity() : int
+ getInventoryLevel() : int
+ getMaxInventoryLevel() : int
+ getCokePrice() :  int
+ getChangeDispensed() : int
+ getNumberOfCokesSold() : int
+ incrementInventoryLevel(amountToAdd : int) :  boolean
+ incrementChangeLevel(amountToAdd : int) :  boolean
+ buyACoke(payment : int)
```

Code3_xxxxxxxxxx.java

Part 1

Create a menu that will call the methods in your Coke Machine. `CokeMenu()` is a method in the interface to display the menu, take the menu choice and validate it. It should include a try-catch to handle non integer entries and continue to ask for valid input (just like PencilMenu).

```
0.    Walk away
1.    Buy a Coke
2.    Restock Machine
3.    Add change
4.    Display Machine Info
```

Part 2

When you instantiate your CokeMachine object, your constructor will assign

Your machine's name

The price of a single Coke

The amount of change in the machine

The number of items (inventory level) in the machine

The constructor I used (that you will see in the example output) is

```
CokeMachine MyCokeMachine("CSE 1325 Coke Machine", 50, 500, 100);
```

The parameters are Machine Name, Coke cost, change level and inventory level.

The max capacities of the machine (inventory and change) are set in the initializers of the instance variables. These values are given in the class diagram.

Part 3

Any printing to the screen should take place in the `Code3_xxxxxxxxxx.java` (referred to as the interface for the rest of this document) file. Your `CokeMachine` class should not ask for anything or print anything. The class diagram is set up such that all methods return the information needed to print messages to the screen.

Part 4

`displayMoney()` is a method in the interface; therefore, should ONLY be called from the interface (never from your class – we discussed the why to this when we talked about converting PencilMachine to an object).

CokeMachine.java

Part 1

`CokeMachine.java` should contain all of the methods shown in the class diagram. Fill in the details of how these methods work by observing the program running.

Part 2

Additional Information about the using of `action` in `buyACoke()`

`buyACoke()` also returns the enum ACTION value that shows what happened while executing the code in `buyACoke()`.

For example, if a Coke costs $0.50 and a payment of 30 cents is entered, then `buyACoke()` will fail but your interface needs to know so it can print a message to inform the user what went wrong – your interface needs to know what action to take. `buyACoke()` returns the enum value to your interface so that it can print a message.

For example,

`buyACoke()`

if payment is insufficient, then `INSUFFICIENTFUNDS` will be returned to the interface which will then print a message about insufficient funds

See sample output for what the interface should print for each return value.

Suggested values for the `enum` (`enum` should be made public in your `CokeMachine` class).

```
DISPENSECHANGE, INSUFFICIENTCHANGE, INSUFFICIENTFUNDS, EXACTPAYMENT,
NOINVENTORY
```

Part 3

The case statement for menu option 4 in the interface will contain **ONLY** 2 lines.

```
System.out.println(MyCokeMachine);

break;
```

When the object is given to `println()` like this, the default `toString()` method will try to print it. You must override the default `toString()` with your own version of `toString()` in your `CokeMachine` class. You are required to use `String.format()` to build your output string. Your overloaded `toString()` method must only contain one line of code

```
return String.format(………………………);
```