Project 4: Coffee Machine

CS 1410

Background

In this project you will implement an object-oriented design that simulates a vending machine that dispenses different types of (old-fashioned) coffee as well as chicken bouillon. Since this is our first project using an object-oriented design, it will be a console app to keep things as simple as possible.

The valid user operations for this "machine" are **insert** <coin>, **select** <product>, **cancel**, and **quit**. The **cancel** command returns any coins that have been inserted before a product was selected and dispensed. Here is a sample execution of the console program you will build (user input is in **bold** type):

```
$ python coffee.py


_____________________________________
    PRODUCT LIST: all 35 cents, except bouillon (25 cents)
    1=black, 2=white, 3=sweet, 4=white & sweet, 5=bouillon"
    Sample commands: insert 25, select 1.
>>> Your command: select 1
Sorry. Not enough money deposited.


_____________________________________
    PRODUCT LIST: all 35 cents, except bouillon (25 cents)
    1=black, 2=white, 3=sweet, 4=white & sweet, 5=bouillon"
    Sample commands: insert 25, select 1.
>>> Your command: insert 1
INVALID AMOUNT >>>
We only take half-dollars, quarters, dimes, and nickels.


_____________________________________
    PRODUCT LIST: all 35 cents, except bouillon (25 cents)
    1=black, 2=white, 3=sweet, 4=white & sweet, 5=bouillon"
    Sample commands: insert 25, select 1.
>>> Your command: insert 35
INVALID AMOUNT >>>
We only take half-dollars, quarters, dimes, and nickels.


_____________________________________
    PRODUCT LIST: all 35 cents, except bouillon (25 cents)
    1=black, 2=white, 3=sweet, 4=white & sweet, 5=bouillon"
    Sample commands: insert 25, select 1.
>>> Your command: Hi!
Invalid command.


_____________________________________
    PRODUCT LIST: all 35 cents, except bouillon (25 cents)
    1=black, 2=white, 3=sweet, 4=white & sweet, 5=bouillon"
    Sample commands: insert 25, select 1.
>>> Your command: insert 25
Depositing 25 cents. You have 25 cents credit.


_____________________________________
    PRODUCT LIST: all 35 cents, except bouillon (25 cents)
```

```
    1=black, 2=white, 3=sweet, 4=white & sweet, 5=bouillon"
    Sample commands: insert 25, select 1.
>>> Your command: insert 10
Depositing 10 cents. You have 35 cents credit.

_____________________________________
    PRODUCT LIST: all 35 cents, except bouillon (25 cents)
    1=black, 2=white, 3=sweet, 4=white & sweet, 5=bouillon"
    Sample commands: insert 25, select 1.
>>> Your command: insert 5
Depositing 5 cents. You have 40 cents credit.

_____________________________________
    PRODUCT LIST: all 35 cents, except bouillon (25 cents)
    1=black, 2=white, 3=sweet, 4=white & sweet, 5=bouillon"
    Sample commands: insert 25, select 1.
>>> Your command: select 1
Making black:
     Dispensing cup
     Dispensing coffee
     Dispensing water
Returning 5 cents.

_____________________________________
    PRODUCT LIST: all 35 cents, except bouillon (25 cents)
    1=black, 2=white, 3=sweet, 4=white & sweet, 5=bouillon"
    Sample commands: insert 25, select 1.
>>> Your command: insert 10
Depositing 10 cents. You have 10 cents credit.

_____________________________________
    PRODUCT LIST: all 35 cents, except bouillon (25 cents)
    1=black, 2=white, 3=sweet, 4=white & sweet, 5=bouillon"
    Sample commands: insert 25, select 1.
>>> Your command: cancel
Returning 10 cents.

_____________________________________
    PRODUCT LIST: all 35 cents, except bouillon (25 cents)
    1=black, 2=white, 3=sweet, 4=white & sweet, 5=bouillon"
    Sample commands: insert 25, select 1.
>>> Your command: insert 25
Depositing 25 cents. You have 25 cents credit.

_____________________________________
    PRODUCT LIST: all 35 cents, except bouillon (25 cents)
    1=black, 2=white, 3=sweet, 4=white & sweet, 5=bouillon"
    Sample commands: insert 25, select 1.
>>> Your command: insert 10
Depositing 10 cents. You have 35 cents credit.

_____________________________________
    PRODUCT LIST: all 35 cents, except bouillon (25 cents)
    1=black, 2=white, 3=sweet, 4=white & sweet, 5=bouillon"
    Sample commands: insert 25, select 1.
>>> Your command: select 2
Making white:
     Dispensing cup
```

```
      Dispensing coffee
      Dispensing creamer
      Dispensing water


_____________________________________
    PRODUCT LIST: all 35 cents, except bouillon (25 cents)
    1=black, 2=white, 3=sweet, 4=white & sweet, 5=bouillon"
    Sample commands: insert 25, select 1.
>>> Your command: insert 5
Depositing 5 cents. You have 5 cents credit.


_____________________________________
    PRODUCT LIST: all 35 cents, except bouillon (25 cents)
    1=black, 2=white, 3=sweet, 4=white & sweet, 5=bouillon"
    Sample commands: insert 25, select 1.
>>> Your command: insert 5
Depositing 5 cents. You have 10 cents credit.


_____________________________________
    PRODUCT LIST: all 35 cents, except bouillon (25 cents)
    1=black, 2=white, 3=sweet, 4=white & sweet, 5=bouillon"
    Sample commands: insert 25, select 1.
>>> Your command: insert 25
Depositing 25 cents. You have 35 cents credit.


_____________________________________
    PRODUCT LIST: all 35 cents, except bouillon (25 cents)
    1=black, 2=white, 3=sweet, 4=white & sweet, 5=bouillon"
    Sample commands: insert 25, select 1.
>>> Your command: select 3
Making sweet:
      Dispensing cup
      Dispensing coffee
      Dispensing sugar
      Dispensing water


_____________________________________
    PRODUCT LIST: all 35 cents, except bouillon (25 cents)
    1=black, 2=white, 3=sweet, 4=white & sweet, 5=bouillon"
    Sample commands: insert 25, select 1.
>>> Your command: insert 25
Depositing 25 cents. You have 25 cents credit.


_____________________________________
    PRODUCT LIST: all 35 cents, except bouillon (25 cents)
    1=black, 2=white, 3=sweet, 4=white & sweet, 5=bouillon"
    Sample commands: insert 25, select 1.
>>> Your command: insert 10
Depositing 10 cents. You have 35 cents credit.


_____________________________________
    PRODUCT LIST: all 35 cents, except bouillon (25 cents)
    1=black, 2=white, 3=sweet, 4=white & sweet, 5=bouillon"
    Sample commands: insert 25, select 1.
>>> Your command: select 4
Making white & sweet:
      Dispensing cup
      Dispensing coffee
```

```
      Dispensing sugar
      Dispensing creamer
      Dispensing water


_____________________________________
    PRODUCT LIST: all 35 cents, except bouillon (25 cents)
    1=black, 2=white, 3=sweet, 4=white & sweet, 5=bouillon"
    Sample commands: insert 25, select 1.
>>> Your command: insert 10
Depositing 10 cents. You have 10 cents credit.


_____________________________________
    PRODUCT LIST: all 35 cents, except bouillon (25 cents)
    1=black, 2=white, 3=sweet, 4=white & sweet, 5=bouillon"
    Sample commands: insert 25, select 1.
>>> Your command: insert 10
Depositing 10 cents. You have 20 cents credit.


_____________________________________
    PRODUCT LIST: all 35 cents, except bouillon (25 cents)
    1=black, 2=white, 3=sweet, 4=white & sweet, 5=bouillon"
    Sample commands: insert 25, select 1.
>>> Your command: insert 10
Depositing 10 cents. You have 30 cents credit.


_____________________________________
    PRODUCT LIST: all 35 cents, except bouillon (25 cents)
    1=black, 2=white, 3=sweet, 4=white & sweet, 5=bouillon"
    Sample commands: insert 25, select 1.
>>> Your command: select 5
Making bouillon:
      Dispensing cup
      Dispensing bouillonPowder
      Dispensing water
Returning 5 cents.


_____________________________________
    PRODUCT LIST: all 35 cents, except bouillon (25 cents)
    1=black, 2=white, 3=sweet, 4=white & sweet, 5=bouillon"
    Sample commands: insert 25, select 1.
>>> Your command: quit
Total cash received: $1.65
```

Coins are inserted one at a time. Only half-dollar coins, quarters, dimes, and nickels are accepted. Enforce this. Print an error message if an invalid amount was entered.

The following CRC cards reflect the needed classes and how they interact in simulating the coffee machine.

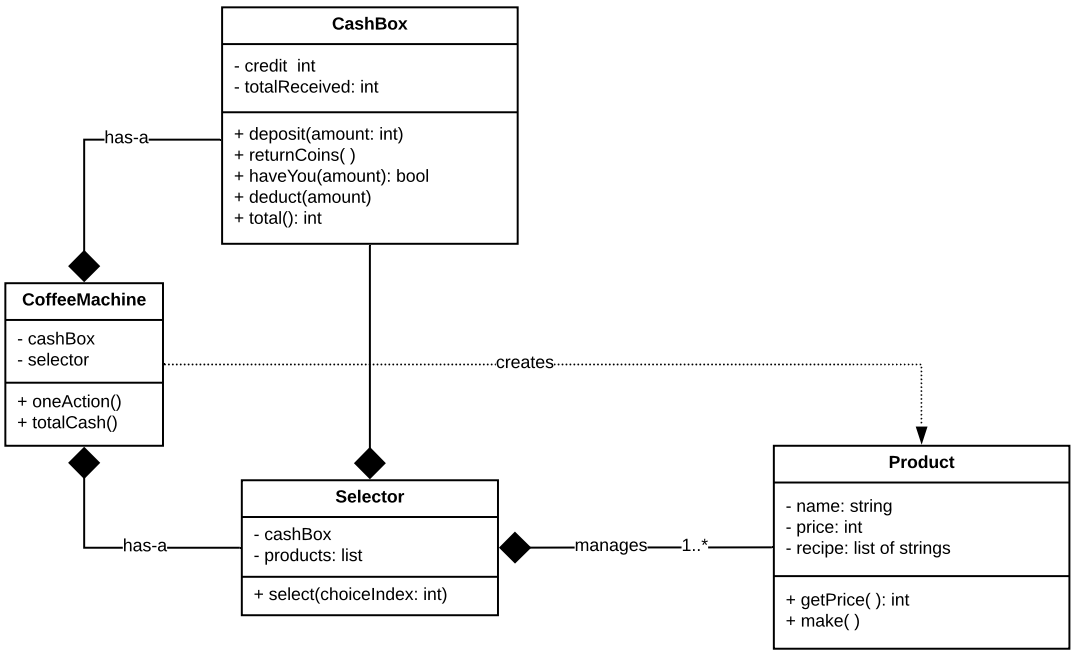
CoffeeMachine	
• Abstraction of the outer machine, holding all the parts. • Responsible for constructing machine, capturing external input.	• CashBox • Selector

CashBox	
• Abstraction of a cashbox/change maker on a real machine. • Responsible for accepting and tracking coins, making change.	

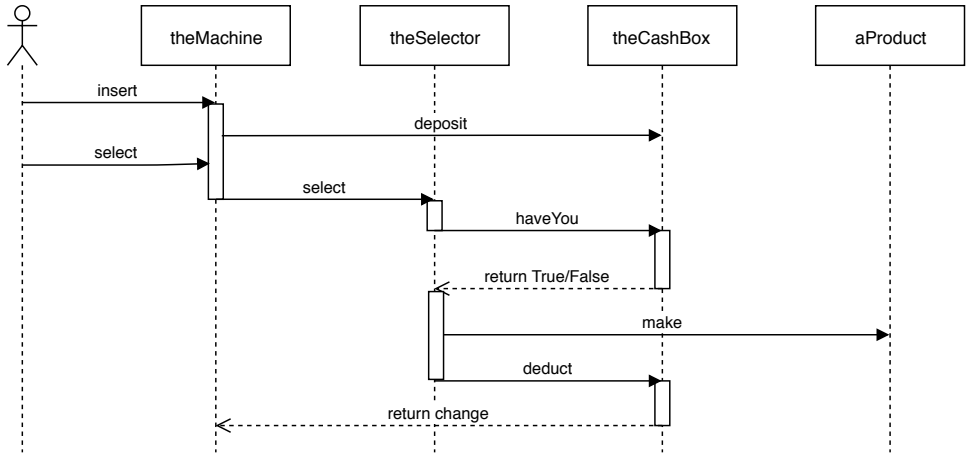
Selector	
• Abstraction of the internal control mechanism. • Knows products & selection, coordinates payment and drink making.	• CashBox • Product

Product	
• Abstraction of the drink. • Responsible for knowing its price and recipe. • Dispenses the drink.	

A UML class diagram appears below.



Here is a UML sequence diagram that depicts a typical transaction.



The main program is simply:

```
def main():
    m = CoffeeMachine()
    while m.one_action():
        pass
    total = m.totalCash()
    print(f"Total cash: ${total/100:.2f}")
```

The `one_action` method returns `True` unless `quit` was entered, in which case it returns `False`, terminating the program. An action is one of: **insert** <amount>, **select** <number>, **cancel**, or **quit**. Valid insertion amounts are any combination of fifty-cent pieces, quarters, dimes, and nickels. If the amount is invalid, print an error message (*Note*: pennies are not accepted).

As you can see in the execution trace above, there are **five** possible **selections**: 1=black, 2=white, 3=sweet, 4=white & sweet, 5=bouillon. "White" means add a shot of creamer, and "sweet" means add a shot of sugar. These are recorded among the five corresponding recipes, as shown in the corresponding selections in the execution trace above. A `Product` holds the selection name (e.g., "black"), and a list of Ingredients in the proper order. Ingredients are just one of the following **strings**: "cup", "coffee", "sugar", "creamer", "water".

We will assume that there is an **unlimited supply** of ingredients.

Requirements

Implements all classes above as illustrated. The `CoffeeMachine.one_action` method prints the instructions and awaits user input (normally we would use a GUI app for this). The user initiates a transaction by inserting money, at which point `one_action` calls `CashBox.deposit` to credit the user with the amount(s) inserted pending a completed transaction or a `cancel` command and returns `True`.

The `cancel` command returns to the user any credited coins pending in the `CashBox`. Keep track of the accumulated amount that is in the `CashBox` from completed transactions (returned by `CashBox.total`).

The `select` command causes `one_action` to invoke the selector's `select` method, passing the number representing the user's selection. The selector determines the `Product` from the given index and then

- asks the `CashBox` if there is enough money pending to cover the cost of the `Product`. An error is printed if there isn't enough.
- calls `Product.make`, which displays the output for dispensing the product
- calls `CashBox.deduct`, which accepts the coins, returns any change left over, and keeps proceeds from product sales

Implementation Notes

Have the `CoffeeMachine` constructor:

1. create the cashbox
2. create all the products
3. create the selector, passing it the cashbox and list of products

Your main function then calls `CoffeeMachine.one_action` in a loop as shown earlier in this document.

Think this through before coding. It is usually better to implement and test the simplest classes first, i.e., the classes with the fewest dependencies (start with `Product`).

FAQs

Q. Why does the coffee machine create everything?
A. It is the interface with the user and contains everything. But after creating the products and passing them to the selector, it has nothing more to do with the products. The `one_action` method in `CoffeeMachine` collaborates with the cashbox or selector as needed.

Q. Why do we have a selector?
A. To coordinate the validation and dispensing of a user's selection. We could have the coffee machine do it directly, but it is good design to use a separate abstraction for a complex operation. The selector 1) verifies that there is enough money entered for the selected product (by asking the product for its price and then asking the cashbox if enough money has been inserted), 2) directs the product to dispense itself, and 3) tells the cashbox that it can keep the money and dispense change due. The selector doesn't need to know a product's ingredients because the product knows its own recipe. "A place for everything and everything in its place." The selector is an internal mechanism that the user doesn't know about, but is necessary for efficient, modular operation.

Q. I'm a little confused about the cashbox. It seems like it has to keep track of 2 separate amounts.
A. It does indeed. It needs to know how much money has been inserted for the pending transaction; it may have to refund it, so it can't stash it away yet. Only after the transaction is complete can it keep the money.

Q. What does **one_action** do?
A. It performs "one action", duh. An action is either **insert**, **select**, **cancel**, or **quit**. After validating user input, it performs one of those actions or prints an error message. If the user enters "quit", it returns False so the loop in **main** will terminate.