

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.

Here is a sample execution of the console program (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
INPUT ERROR >>>
We only take half-dollars, quarters, dimes, and nickels.
Coin(s) returned.
```

```
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
INPUT ERROR >>>
We only take half-dollars, quarters, dimes, and nickels.
Coin(s) returned.
```

```
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 whiteSweet:
 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: \$1.65

Only half-dollar coins, quarters, dimes, and nickels are accepted. Enforce this.

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

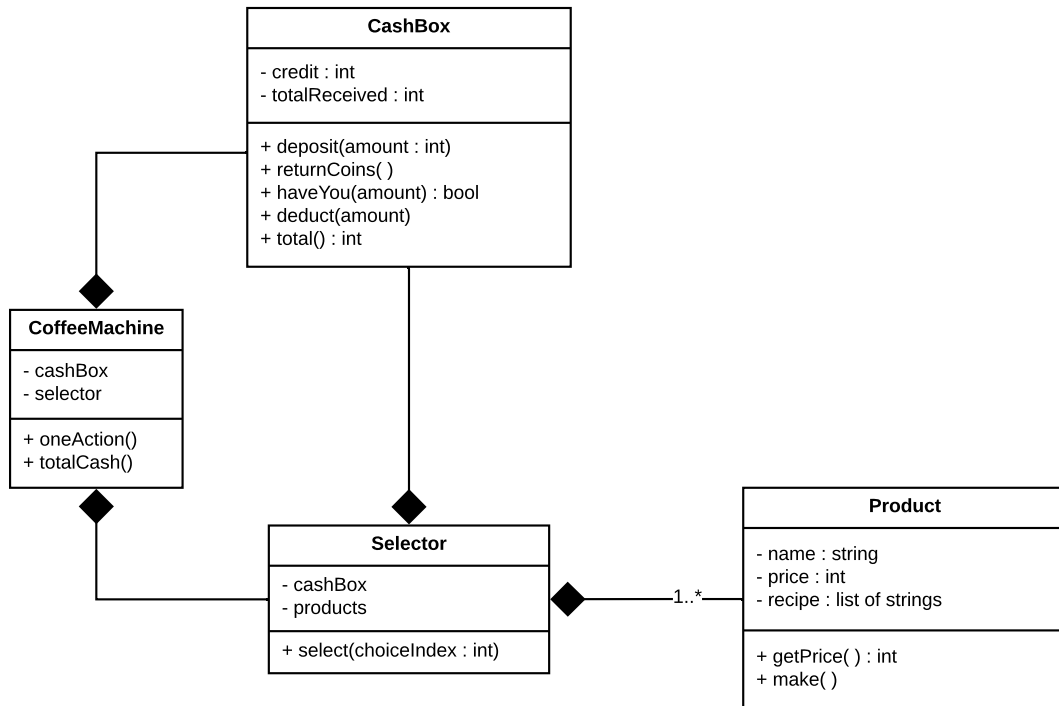
CoffeeMachine	
<ul style="list-style-type: none"> • Abstraction of the outer machine, holding all the parts. • Responsible for constructing machine, capturing external input. 	<ul style="list-style-type: none"> • CashBox • Selector

CashBox	
<ul style="list-style-type: none"> • Abstraction of a change maker or cashbox on a real machine. • Responsible for making change, accepting coins. 	

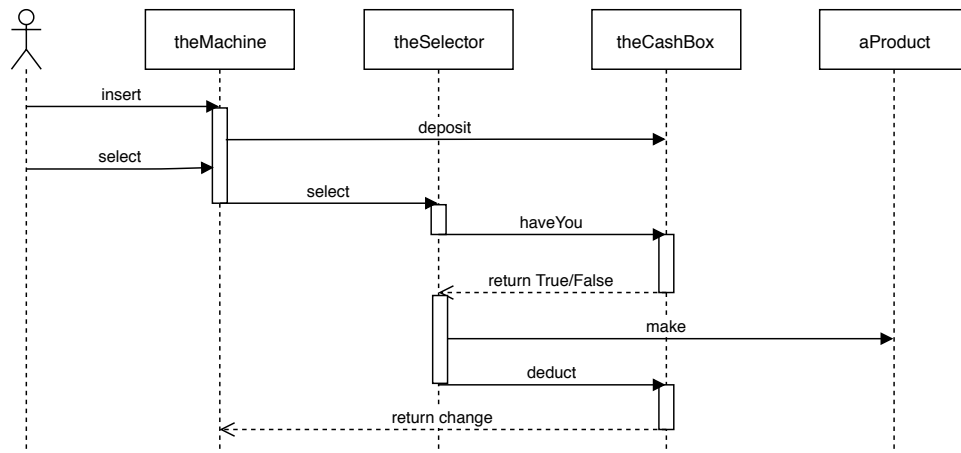
Selector	
<ul style="list-style-type: none"> • Abstraction of the internal selector and controller. • Knows products & selection, coordinates payment and drink making. 	<ul style="list-style-type: none"> • CashBox • Product

Product	
<ul style="list-style-type: none"> • Abstraction of the drink. • Responsible for knowing its price and recipe. • Dispenses the drink. 	

The 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.oneAction():
        pass
    total = m.totalCash()
    print(f"Total cash: ${total/100:.2f}")
```

The `oneAction` 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, return the coins to the user and print an error message (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 `oneAction` 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 `oneAction` 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 currently waiting to be used 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 `oneAction` 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 initialize the products, cashbox, and selector. It passes the list of `Product` objects to the `Selector` constructor. Your main function then calls `CoffeeMachine.oneAction` 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.