Description

Let's redesign our program and write a class that represents the coffee machine. The class should have a method that takes a string as input. Every time the user inputs a line (a string) to the console, the program invokes this method with one argument: the line that user input to the console. This system simulates pretty accurately how real-world electronic devices work. External components (like buttons on the coffee machine or tapping on the screen) generate events that pass into the single interface of the program.

The class should not use system input at all; it only should handle input that comes to it via this method and its string argument.

The first problem that comes to mind: how to write that method in a way that it represents all that coffee machine can do? If the user inputs a single number, how can the method determine what that number is: a variant of coffee chosen by the user or the number of the disposable cups that a special worker added into the coffee machine?

The right solution to this problem is to store the current state of the machine. The coffee machine has several states it can be in. For example, the state could be "choosing an action" or "choosing a variant of coffee". Every time the user inputs something and a program passes that line to the method, the program determines how to interpret this line using the information about the current state. After processing this line, the state of the coffee machine can be changed or can stay the same. The most efficient way of handling states is using an enum with all predefined states.

Remember, that:

- For the espresso, the coffee machine needs 250 ml of water and 16 g of coffee beans. It costs $4.
- For the latte, the coffee machine needs 350 ml of water, 75 ml of milk, and 20 g of coffee beans. It costs $7.
- And for the cappuccino, the coffee machine needs 200 ml of water, 100 ml of milk, and 12 g of coffee. It costs $6.

Instruction

Refactor the program. Make it so that you can communicate with the coffee machine through a single method.

Example

Your coffee machine should have the the same initial resources as in the example (400 ml of water, 540 ml of milk, 120 g of coffee beans, 9 disposable cups, $550 in cash).
The symbol `>` represents the user input. Notice that it's not the part of the input.

```
Write action (buy, fill, take, remaining, exit):
> remaining

The coffee machine has:
400 of water
540 of milk
120 of coffee beans
9 of disposable cups
$550 of money

Write action (buy, fill, take, remaining, exit):
> buy

What do you want to buy? 1 - espresso, 2 - latte, 3 - cappuccino, back - to main menu:
> 2
I have enough resources, making you a coffee!

Write action (buy, fill, take, remaining, exit):
> remaining

The coffee machine has:
50 of water
465 of milk
100 of coffee beans
8 of disposable cups
$557 of money

Write action (buy, fill, take, remaining, exit):
> buy

What do you want to buy? 1 - espresso, 2 - latte, 3 - cappuccino, back - to main menu:
> 2
Sorry, not enough water!

Write action (buy, fill, take, remaining, exit):
> fill

Write how many ml of water do you want to add:
> 1000
Write how many ml of milk do you want to add:
> 0
Write how many grams of coffee beans do you want to add:
> 0
Write how many disposable cups of coffee do you want to add:
> 0

Write action (buy, fill, take, remaining, exit):
> remaining

The coffee machine has:
1050 of water
465 of milk
100 of coffee beans
8 of disposable cups
$557 of money

Write action (buy, fill, take, remaining, exit):
> buy

What do you want to buy? 1 - espresso, 2 - latte, 3 - cappuccino, back - to main menu:
> 2
I have enough resources, making you a coffee!

Write action (buy, fill, take, remaining, exit):
> remaining

The coffee machine has:
700 of water
390 of milk
```

```
80 of coffee beans
7 of disposable cups
$564 of money

Write action (buy, fill, take, remaining, exit):
> take

I gave you $564

Write action (buy, fill, take, remaining, exit):
> remaining

The coffee machine has:
700 of water
390 of milk
80 of coffee beans
7 of disposable cups
$0 of money

Write action (buy, fill, take, remaining, exit):
> exit
```