An OO Design Example

Adapted From:

Coffee Machine Design Problem

By Alistair Cockburn

http://alistair.cockburn.us/SearchTitles?for=coffee machine
The Coffee Machine Problem

We are contractors who just won a bid to design a custom coffee vending machine for the employees of Acme Fijet Works to use on their lunch and coffee breaks.

Arnold, the owner of Acme Fijet Works, like the common software designer, eschews standard solutions. He wants his own, custom design. He is, however, a cheapskate. Arnold tells us he wants a simple machine.

All he wants is a machine that serves coffee for 35 cents, with or without sugar and creamer. That’s all. He expects us to be able to put this little machine together quickly and for little cost.

We get together and decide there will be:

- Cashbox
  - Coin slot
  - Coin return
  - Coin return lever
- Front Panel
  - Black
  - With Cream
  - Black with Sugar
  - With Cream and Sugar
- Mixer
- Array of Dispensers
  - Cup dispenser
  - Coffee powder dispenser
  - Sugar dispenser
  - Creamer dispenser
  - Water dispenser

Now, design the program that runs the machine using objects. Start with the classes above, what are their responsibilities, how do they collaborate to deliver the simple service?
The Coffee Machine Problem

Our design needs to meet the following scenarios:

1. Kim puts in a quarter and a dime and then selects a coffee.
2. Kim puts two quarters in and then selects a coffee.
3. Kim puts in a quarter, then pushes the coin return lever.
4. Kim puts in two quarters, then walks away from the machine and forgets to come back.
5. Kim buys two coffees, white with sugar. The sugar dispenser runs out of sugar after the first.

A Typical First Coffee Machine Design

<table>
<thead>
<tr>
<th>Component</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cash Box</td>
<td>Knows amount of money put in.</td>
</tr>
<tr>
<td></td>
<td>Gives change.</td>
</tr>
<tr>
<td></td>
<td>Knows the price of coffee.</td>
</tr>
<tr>
<td></td>
<td>Turns the Front Panel on and off.</td>
</tr>
<tr>
<td>Front Panel</td>
<td>Captures the selection</td>
</tr>
<tr>
<td></td>
<td>Knows what to mix in each.</td>
</tr>
<tr>
<td></td>
<td>Tells the mixer what to mix.</td>
</tr>
<tr>
<td>Mixer</td>
<td>Instructs the dispensers to dispense some amount of each product.</td>
</tr>
<tr>
<td>Dispensers</td>
<td>Knows how to dispense a fixed amount.</td>
</tr>
<tr>
<td></td>
<td>Knows when it is empty.</td>
</tr>
<tr>
<td></td>
<td>(cup, coffee powder, sugar, creamer, water)</td>
</tr>
</tbody>
</table>
Object Diagram for the Design

Sequence Diagram for the Design
Redesigning the Coffee Machine

Arnold has another bright idea.

After five machines are installed and have been operating for a while, Arnold comes along and says, "I would like to add chicken soup, at a price of 25 cents, to every machine. Change the software."

We modify the hardware to include one more button for chicken soup and one more dispenser for the soup mix.

How do we change the software design?

Modified Coffee Machine Design

- **Cash Box**: Knows amount of money put in.
  - Gives change.
  - Knows the price of coffee.
  - Turns the Front Panel on and off.
  - Answers how much money has been put in.

- **Front Panel**: Captures the selection
  - Knows the price and recipe for each selection.
  - Asks the Cash Box how much money was put in.
  - Tells the mixer what to mix.

- **Mixer**: Instructs the dispensers to dispense some amount of each product.

- **Dispensers**: Knows how to dispense a fixed amount.
  - Knows when it is empty.
  - (cup, coffee powder, sugar, creamer, water, soup mix)
Updated Object Diagram for the Design

Updated Sequence Diagram
Changing Software Requirements

- Arnold (our boss) changed the requirements of our application after we wrote our program.
  - This is typically what happens.
  - In fact, your career depends on it!
  - Users expect that a simple change for them is a simple change in the software.

- Knowing that requirements will change is an incentive to create software that can be flexibly enhanced.

- A good OO design can aid in flexibility by creating small cohesive objects that encapsulate the data where we need it and abstract the details.

Arnold is baaack!

In a few weeks, Arnold returns with a brilliant idea. He has heard that some companies use their employee badges to directly debit the cost of coffee purchases from their employees' paychecks. Arnold's employees already have badges, so he thinks this should be a simple change to the system. And he hates to be behind the curve.

He modifies the hardware to include a badge reader and network connection to the payroll system.

*How do we change our design?*
Modified Coffee Machine Design

- **Cash Box**
  - Knows amount of money put in.
  - Gives change.
  - Accepts cash or charge.
  - Turns the Front Panel on and off.
  - Answers how much money has been put in.
  - Answers whether a given amount of credit is available.

- **Front Panel**
  - Captures the selection.
  - Knows the price and recipe for each selection.
  - Asks the Cash Box how much money was put in.
  - Asks the Cash Box if sufficient credit is available.
  - Tells the mixer what to mix.

- **Mixer**
  - Instructs the dispensers to dispense some amount of each product.

- **Dispensers**
  - Knows how to dispense a fixed amount.
  - Knows when it is empty.
  - (cup, coffee powder, sugar, creamer, water, soup mix)

Comments on redesign

- Our initial design broke the rules of cohesiveness.

- The FrontPanel class was doing too much.
  - It knew the price and was validating the costs.

- In this redesign we had to correct that and move the knowledge of the prices to the cash box.
  - The FrontPanel did not need to know how much money was put in, but only that enough money had been put in.
  - The CashBox is now managing all of the details about payments.
  - This also allows for only one object to interact with the payroll system.
Updated Object Diagram for the Design

Updated Sequence Diagram for the Design
Not Again....

People are starting to buy lattes instead of his coffees. Arnold wants the machine modified just slightly, so that he can create a “drink of the week”. He wants to be able to add new drinks and change prices anytime, to match his competition. He wants to be able to add espresso, cappuccino, hot chocolate, latte, choco-latte, steamed milk — in short, anything he can mix together.

*How do we change our design?*

---

### Modified Coffee Machine Design

**Cash Box**
- Knows amount of money put in.
- Gives change.
- Accepts cash or charge.
- Turns the Front Panel on and off.
- Answers whether a given amount of credit is available.

**Product**
- Knows its own price and recipe.

**Recipe**
- Knows the quantity and sequence of ingredients.

**Product Selector**
- Captures the selection.
- Knows products.
- Asks the Cash Box if sufficient credit is available.
- Manages Cash Box and mixer to validate and produce the drink.

**Mixer**
- Instructs the dispensers to dispense some amount of each product.

**Dispenser Register**
- Knows the dispensers

**Dispensers**
- Knows how to dispense a fixed amount.
- Knows when it is empty.
Comments on redesign

- Our initial design broke the rules of cohesiveness.
- The FrontPanel class was still doing too much.
- In this redesign we removed the knowledge of the recipes.

Updated Object Diagram for the Design
Updated Sequence Diagram for the Design

Sample Interaction for Final Version

Notes on Final Design

- Allows for change:
  - Can accommodate new products easily.
  - Products can be modified easily for price and recipe.
  - The product object is passed around the other classes, carrying the required information with it.
  - Each object has a cohesive, small set of responsibilities.