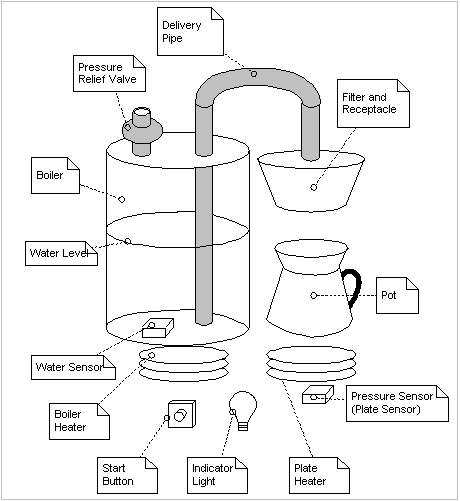
**The Coffee Maker**

**Introduction**

The Coffee Maker problem is a very popular exercise (and exam question) in the Design Patterns, and OOP community. I believe this is originated from Robert Martin's "*First Principles of OOD and UML" course. The problem is also described in Martin's book "Designing Object Oriented C++ Applications using the Booch Methodology" as well as in his book “Agile Principles, Patterns, and Practices”. The Extreme Programming web site is also working through this exercise, but using XP techniques. Many interviewers are using this coffee maker problem (or something similar like a vending machine) for their interview test.*

## Problem Summary

(From Designing Object Oriented C++ Applications using the Booch Methodology by Robert Martin).

**

#### Figure 1 -- Coffee Maker Schematic

The Mark IV special makes up to 12 cups of coffee at a time. The user places a filter in the filter holder, fills the filter with coffee grounds, and slides the filter holder into its receptacle. The user then pours up to 12 cups of water into the water strainer and presses the "Brew" button. The water is heated until boiling. The pressure of the evolving steam forces the water to be sprayed over the coffee grounds, and coffee drips through the filter into the pot. The selectable coffee strength (strong, medium, light) is controlled by the force of the spray (not in the diagram) as follow: strong coffee: light slow spray, medium coffee: medium spray, and light coffee: strong fast spray. The pot is kept warm for extended periods by a warmer plate, which only turns on if there is coffee in the pot. If the pot is removed from the warmer plate while coffee is sprayed over the grounds, the flow of water is stopped, so that brewed coffee does not spill on the warmer plate. The following hardware needs to be monitored or controlled.

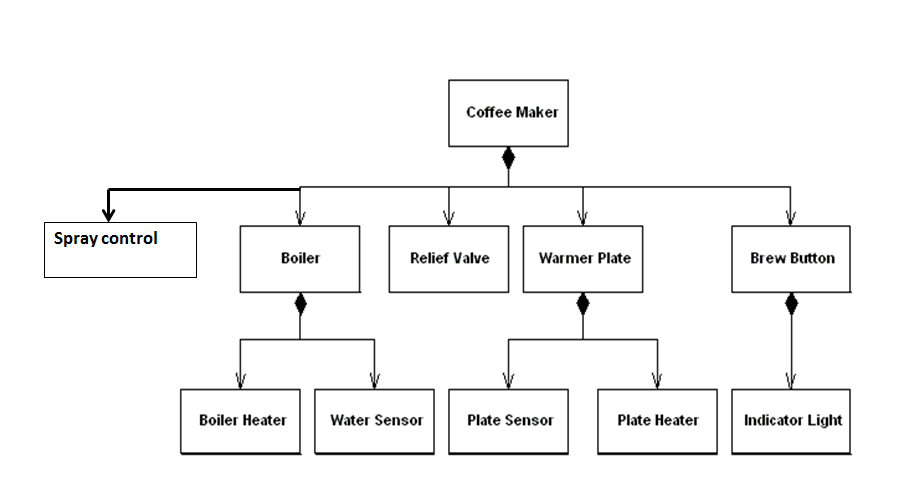
* *The heating element for the boiler. It can be turned on or off.*
* *The heating element for the warmer plate. It can be turned on or off.*
* *The sensor for the warmer plate. It has three states:* ***warmerEmpty****,* ***potEmpty****, and* ***potNotEmpty****.*
* *A sensor for the boiler, which determines if there is water present or not. It has two states:*

#### boilerEmpty or boilerNotEmtpy.

* *The brew button. This momentary button starts the brewing cycle. It has an indicator that lights up when the brewing cycle is over and the coffee is ready.*
* *A pressure-relief valve that opens to reduce the pressure in the boiler. The drop in pressure stops the flow of water to the filter. It can be opened or closed.*

## Understanding the Problem

The following simple UML diagram capturing the relationships of the various physical elements of the problem. Note that this is a description of the physical relationships between the coffee maker components and may have little to do with the final software configuration. Your program most likely DO NOT have the following “UML” diagram display.



#### Figure 2 - Physical Component Identification

Requirements:

Need a piece of software to operate the Coffee maker

**Use case 1: Brew Coffee**

The user (Coffee Lady) needs to

* + Put empty pot on warmer.
  + Fill boiler with water.
  + Put filter and coffee grounds into filter holder
  + load it into the receptacle.
  + select the coffee strength (strong, medium\*, light); medium is default.
  + press “Brew Button”

In order to brew coffee, the system (coffee maker) needs to have the different hardware properties

* Closes relief valve to allow water to flow to receptacle
* turns on boiler heater.
* Adjusts the spray force setting according to the selected coffee strength.
* When boiler is empty, turns off boiler heater.
* The indicator light is also turned on.

**Use case 2: Pour Coffee – add extras as needed**

Basic black coffee (strong, medium, or light = $1.10; plus mocha = add $0.90; plus whip = add $1.25.

You need to design your software to handle new additional extras (condiments) as they become available with minimal changes.

The Coffee lady wants to pour a cup of coffee from the pot and add on the extras. She also wants to keep track of the total cost of her coffee with the extra additions. When she removes the coffee pot from the warmer, the maker needs to turn off the warm, turn the light off if it was on.

Precondition: The Coffee Maker is not Brewing. A new condiment (cinnamon) has arrived, but our coffee lady wants to select mocha and cinnamon but no whip.

* Coffee lady: Lifts pot from warmer.
* Coffee maker: Detects the pot has been lifted
* Coffee maker: turns off the plate warmer.
* Coffee maker: If the indicator light is on, then turn it off
  + Coffer lady: replaces pot onto the warmer
  + Coffee maker: if there is coffee remaining in the pot, restart the plate warmer
  + Coffee Lady adds the extras
  + Coffee Lady receives the cumulative updates on the cost.

**Use case 3: Pour Interrupt Brewing:**

When the coffee pot is removed from the warmer before the coffee brewing is done, the coffee maker should turn off the boiler, open the relief valve (to stop the water from entering the filter). When the pot is returned, the maker should automatically restart the brewing process.

Precondition: The coffee maker is NOT brewing.

The user (Coffee Lady) and the Coffee Maker should need to

* Coffee lady: Lifts pot from warmer.
* Coffee maker: Detects the pot has been lifted
* Coffee maker: turns off the boiler
* Coffee maker: opens the (pressure) relief valve
* Coffee maker: If the indicator light is on, then turn it off
  + Coffer lady: replaces pot onto the warmer
  + Coffee maker: turns on the boiler
  + Coffee maker: close the relief valve
  + Coffee maker: continue the brew cycle

There are many possible design patterns available for the above problem. Choose them wisely.

**Expected deliverables:**

I would expect you to rely on using your learned Java design patterns and OO programming skills with minimal use of “if-then-else” and/or “Switch” statements.

1. (90%) Your software printout should show the sequences of the above three use cases for:
2. The coffee lady morning routine
   * + Brews a pot of black medium coffee
     + Interrupts the coffee brewing about ¾ thru
     + Makes herself a coffee with whip only and writes down the cost
     + Rushes to work
3. In the evening, she invited a friend over for coffee.
   * + Brews a pot of light coffee
     + Interrupts the coffee brewing about ¼ thru
     + Makes herself a coffee with cinnamon after returning the pot back to the brewing.
     + Records the cost (coffee with cinnamon)
     + After the brewing is done, she makes her friend a coffee with mocha.
     + Records the cost (coffee with mocha)
4. (10%) UML diagram to illustrate your software design.

**Project Design Details:**

**Use Case 1:**  
When I first looked into Use Case 1, I had come up with a basic OO Design that hadn’t really used any Design patterns. I then thought, considering we are doing a step by step process with individual components I could try and implement a Composite Design Pattern, that will interface each the Relief Valve, Boiler, and Indicator. After trying this I quickly realized, that I needed to simulate the water flowing from the boiler to the receptacle to the coffee pot. So, I created this DeliveryPipeControl class that would delegate the water movement into the receptacle and bring the coffee out into the pot. Which it did exactly what I need it to do. However, it started to feel a little dirty the way I have done it, as the CoffeeMaker contains these Components aswell as the DeliverPipeControl Object. Creating a lot of relationships when looking at the UML Diagram. But, when looking at other Composite Examples, it looks like it is alright. One other thing to mention is that I’ve had to create a updateComponent method in the BrewCycleComposite class as If I update Pot in the coffeebrewer object, I found it wasn’t updating the Pot in the DeliveryPipeControl Component.  
  
**Use Case 2:**

For the second use case. I remember talking about using the decorator design pattern. This is so we can wrap these Condiment Decorators around the base object of CoffeeBeverage, which allows us to have cumulative updates to know the cost of the base coffee + all the Condiments, as well as you will have a list of what is in the current coffee. Considering we want to split CoffeeBeverage into multiple different strengths off coffee, we turn CoffeeBeverage into an abstract class, and get each specific Strength class to extend it. Making each of these a Base Object for the Decorator pattern. I do use a Coffee Cup that has a reference variable of CoffeeBeverage, and use a setter for Wrapping the Condiments on the Base. But, it works the same as a regular Decorator pattern.

When Talking about lifting the Coffee Pot from the Warmer, I have created a PlateSensorState interface, that will automatically update the Warmer’s heating element to on or off depending on the specific state. Also, created an OnOffContol interface for other objects with similar needs.

**Use Case 3:**  
For adding interruptions when lifting the coffee pot during brewing. I knew I had to create some sort of listener for when the WarmerPlate’s PlateSensorState turns into WarmerEmptyState during the brewing cycle. So, with my current setup with the BrewCycleComposite I realized I either need to create some sort of Adapter to simulate this event or do something manually. Again, at this point I was not sure if using the Composite pattern was the best route to go. But, I wanted to still try and see if I could get it to work. I realized that it didn’t really seem too possible with my current setup, so I created a second placePot method in coffeemaker that will get the DeliveryPipeControl to automatically stop after a set amount of cycles. Then the user Lifts the pot, which does turn off the warmer and the indicator light, controlled by the plate sensor. And, if the user decides to place the pot back on and start brewing again. It will start where it left off. Getting the rest of the water from the boiler into the receptacle, then the pot.

Extra notes:

There are a couple areas where I’ve just used Interfaces and Polymorphism to solve the problems:  
SprayControl and WarmerPlate. I believe it is possible to use the state pattern on the warmer plate area, however, I chose to leave it as is.  
  
A and B take functionality from Use Cases, therefore not big reason to explain them.

**Jordan Emslie**