# Design a Simulator for the Coffee Maker

Design is accomplished in three ways on an Extreme Programming (XP) project. There are CRC cards, refactoring, and pair programming. CRC cards can be considered a strategic level of design, pair programming is at the tactical level and refactoring serves both.

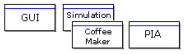
For this problem I would feel better with the team working together. So let's get out the cards!

Let's start out with our hardware's interface. We know we have a programmable interface adapter (PIA). We can set out a card to represent one object that will be our interface to the PIA.


Now let's add our coffee maker code as if it were a single object. It may or may not be, but right now we are designing the simulator. Let's put it right next to the PIA because it will interface with it.


Next we need to add the simulation object. This will be the object that loops simulating time and makes the PIA react as if it was being powered by real hardware. I am thinking that the simulation will hold onto the coffee maker code. I put the card slightly under the coffee maker code. The simulation will signal the coffee maker when ever the PIA has changed. The coffee maker can then react to the change.


The GUI will complete our design. It interacts with the simulation showing the state of our coffee maker and accepting user input like the brew button. But the team is not sure about this design. The simulation interacts with the PIA and needs to know the internals of the coffee maker code, which also interacts with the PIA. Wouldn't it be better if the PIA itself was the simulation? This cuts down on several interfaces.

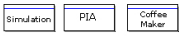



Let's just start over. A good thing about designing with CRC cards is that we can sweep the desk clean as many times as we want and we have not wasted large amounts of time on creating diagrams for each alternative design.

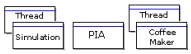
We are back to just the PIA. This is the one thing we must have.


To this we add the coffee maker object(s). This interfaces with the PIA. Now at this point we could just say that the PIA runs the simulation, but after some discussion the team doesn't like that. We want the PIA interface to be simple and generic, a closer representation of the hardware. Adding the simulation portion to it does not achieve that.


So let's put our simulation object back into play. But instead of the simulation owning the coffee maker, let's say that the simulation only interfaces with the PIA. The simulation has no internal knowledge of how the coffee maker works. We all like this better. It will even be a better simulation.




In order to make this work we will have to have separate threads for the coffee maker and the simulation. This adds a level of complexity, but removes about two levels of complexity in the exchange. We are reducing net complexity because we do not have to provide one coffee maker interface for the simulator and some other interface that will run the coffee maker on real hardware. On the other hand Java Threads are just not that complex. And as a bonus we can test the coffee maker exactly as it would run on real hardware. We agree this is better even though it is multithreaded.




Last, we add our GUI to interface to the simulation. This seems like a good design to start out with. Remember, we can change our minds when ever things become difficult to implement. We rely on this strength of XP so that we don't have to design out every detail of every class in advance. Now we need think of a system metaphor to fit this design.