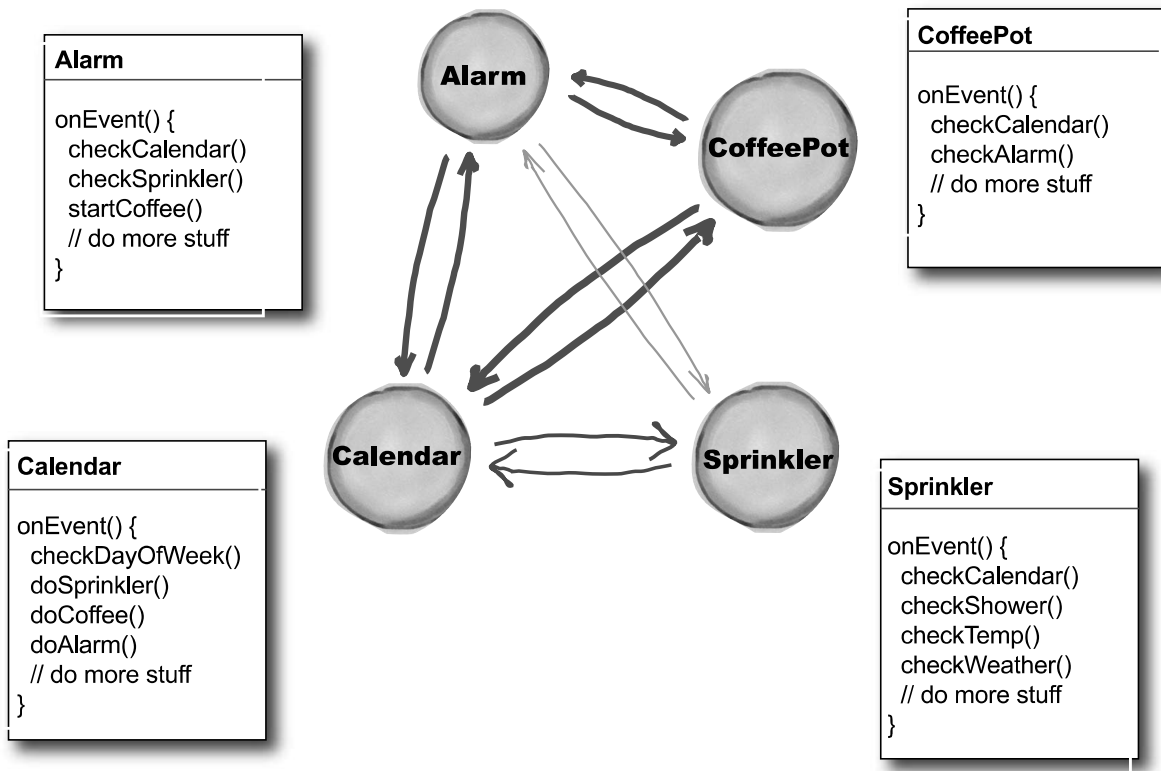


Mediator

Use the Mediator Pattern to centralize complex communications and control between related objects.

A scenario

Bob has a Java-enabled auto-house, thanks to the good folks at HouseOfTheFuture. All of his appliances are designed to make his life easier. When Bob stops hitting the snooze button, his alarm clock tells the coffee maker to start brewing. Even though life is good for Bob, he and other clients are always asking for lots of new features: No coffee on the weekends... Turn off the sprinkler 15 minutes before a shower is scheduled... Set the alarm early on trash days...



HouseOfTheFuture's dilemma

It's getting really hard to keep track of which rules reside in which objects, and how the various objects should relate to each other.

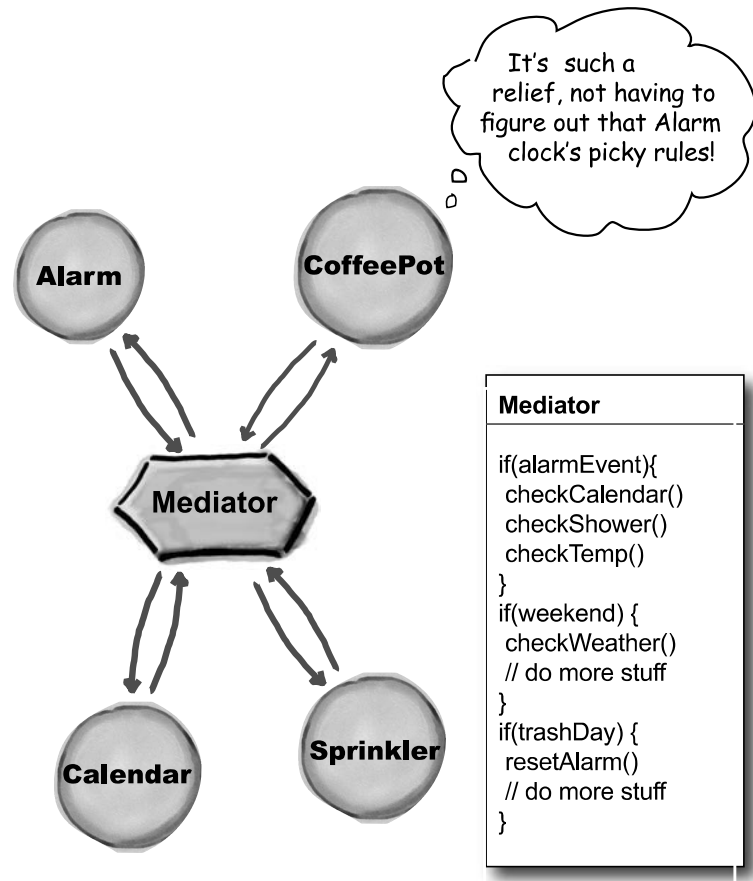
Mediator in action...

With a Mediator added to the system, all of the appliance objects can be greatly simplified:

- They tell the Mediator when their state changes.
- They respond to requests from the Mediator.

Before adding the Mediator, all of the appliance objects needed to know about each other... they were all tightly coupled. With the Mediator in place, the appliance objects are all *completely decoupled* from each other.

The Mediator contains all of the control logic for the entire system. When an existing appliance needs a new rule, or a new appliance is added to the system, you'll know that all of the necessary logic will be added to the Mediator.



Mediator Benefits

- Increases the reusability of the objects supported by the Mediator by decoupling them from the system.
- Simplifies maintenance of the system by centralizing control logic.
- Simplifies and reduces the variety of messages sent between objects in the system.

Mediator Uses and Drawbacks

- The Mediator is commonly used to coordinate related GUI components.
- A drawback of the Mediator pattern is that without proper design, the Mediator object itself can become overly complex.