The Mediator Pattern

The **Mediator Pattern** define an object that encapsulates how a set of objects interact.

Toni Sellarès
Universitat de Girona

The Mediator Pattern: Motivation

- As more classes are developed in a program the problem of communication between these classes may become more complex.
- This makes the program harder to maintain since any change may affect code in several other classes.
- Use the Mediator Pattern to centralize complex communications and control between related objects.
- Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.

The Mediator Pattern: Definition

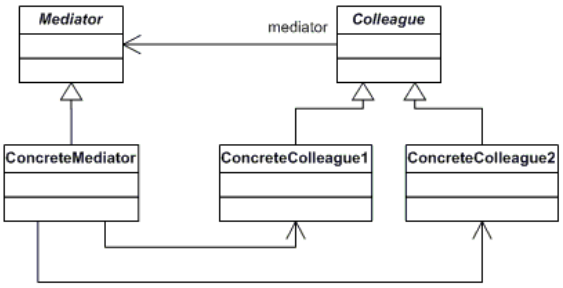
The Mediator Pattern define an object that encapsulates how a set of objects interact.

Participants

Mediator: defines an interface for communicating with Colleague objects.

ConcreteMediator:

- implements cooperative behavior by coordinating Colleague objects.
- knows and maintains its colleagues.



Colleague classes:

- each Colleague class knows its Mediator object.
- each colleague communicates with its mediator whenever it would have otherwise communicated with another colleague.

The Mediator Pattern: Structural Code

```
/**
 * Defines an interface for communicating with Colleague objects.
 */
public interface Mediator {
}


/**
 * Implements cooperative behavior by coordinating
 * Colleague objects. Knows and maintains its colleagues.
 */
public class ConcreteMediator implements Mediator {
        private ConcreteColleague1 colleague1 = new ConcreteColleague1();
        private ConcreteColleague2 colleague2 = new ConcreteColleague2();
}
```

```
/**
 * Each Colleague class knows its Mediator object. Each Colleague
 * communicates with its mediator whenever it would have otherwise
 * communicated with another colleague.
 */
public class Colleague {
    private Mediator mediator;
    public Mediator getMediator() {
                return mediator;
    }
    public void setMediator( Mediator mediator ) {
                this.mediator = mediator;
    }
}
/**
 * Each Colleague class knows its Mediator object. Each Colleague
 * communicates with its mediator whenever it would have otherwise
 * communicated with another colleague.
 */
public class ConcreteColleague1 { }
```

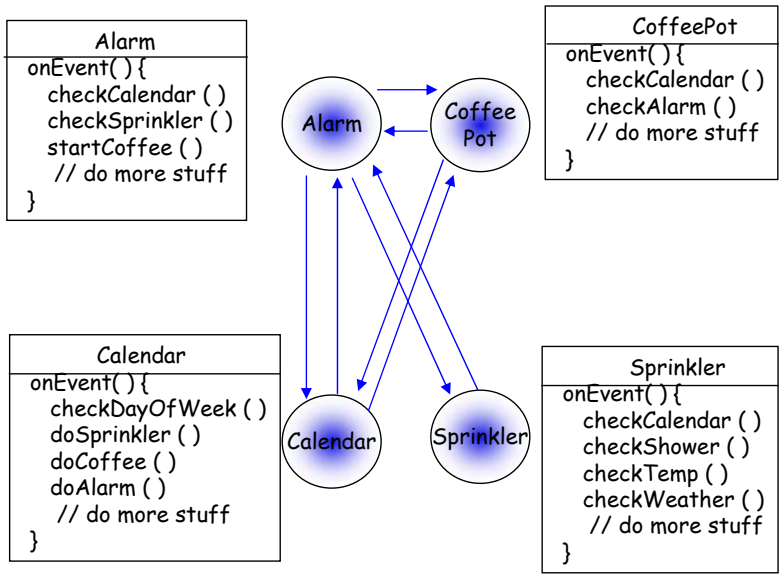
The Mediator Pattern: Example

Scenario:

- Bob has a Java-enabled house. 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.
- Life is good!!….but Bob wants more:
  - No coffee on weekends (he's crazy!)
  - Turn off the sprinkler 15 minutes before a shower is scheduled
  - Set the alarm early on trash days

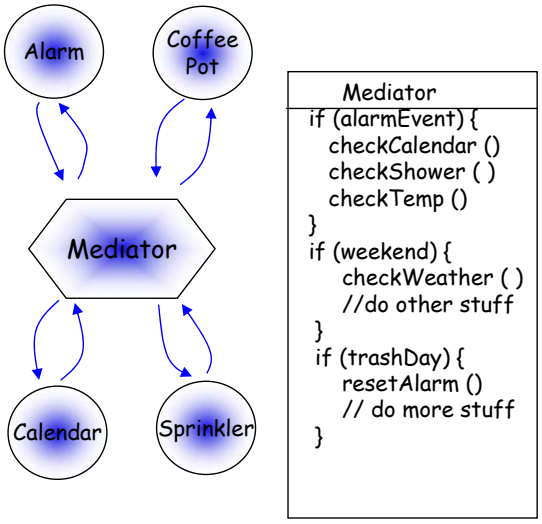
Dilemma:

- Extremely hard to keep track of all the rules and how the objects relate to each other.



Mediator in action…..

- With a Mediator added to the system, all of the appliances can be greatly simplified:
  - They tell the mediator when their state changes.
  - They respond to requests from the mediator.
- Before the Mediator all appliances needed to know about each other (Tightly coupled).
- After the mediator appliances need to know about only the Mediator: completely decoupled!
- Mediator contains the control logic for the entire system:
  - New rule: add it to the mediator!



Benefits, Uses and Drawbacks

Benefits:

- Promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.
- Increases the reusability of the objects supported by the Mediator by decoupling them from the system.
- Simplifies maintenance of the system by decentralizing control logic.
- Simplifies and reduces the variety of messages sent between objects in the system.

Uses:

- The mediator is commonly used to coordinate related GUI components.

Drawbacks:

- A drawback of the Mediator pattern is that without proper design, the Mediator object itself can become overly complex.