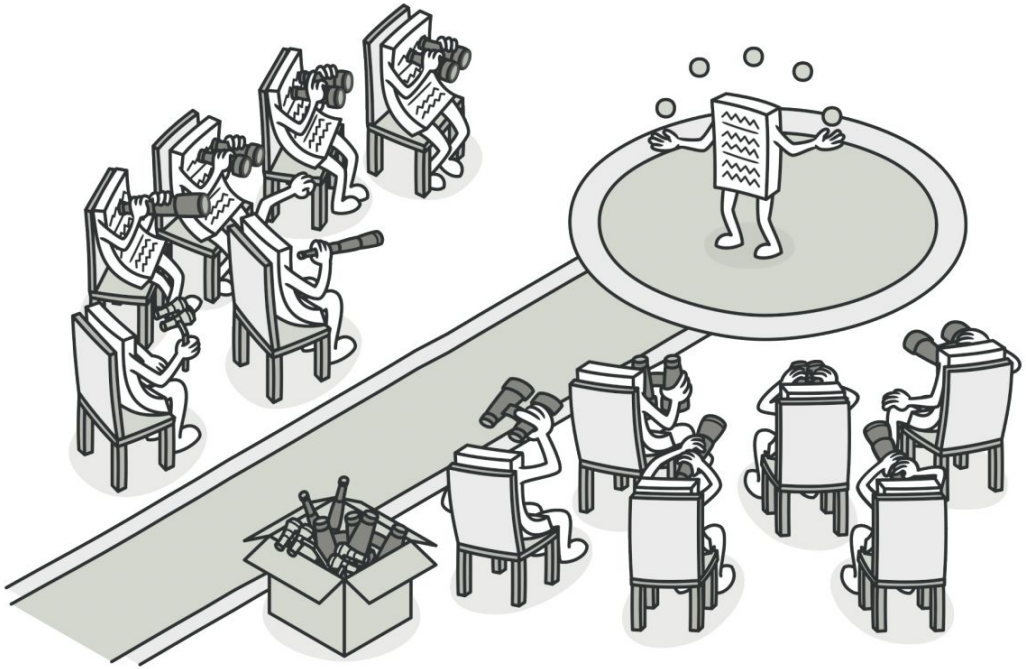
Design Patterns

Observer

Observer AKA Event-Subscriber AKA listener



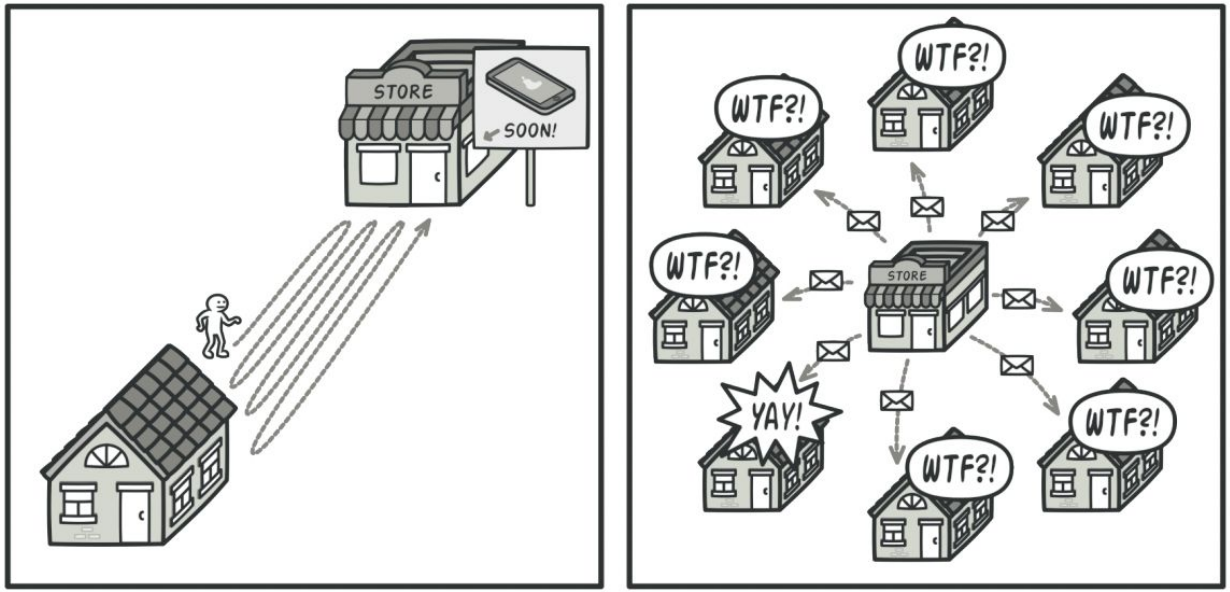
Observer: Problem

- Imagine that you have two types of objects: a Customer and a Store .
- The customer is very interested in a particular brand of product (say, it's a new model of the iPhone) which should become available in the store very soon.
- The customer could visit the store every day and check product availability. But while the product is still en route, most of these trips would be pointless.

Observer: Problem

- On the other hand, the store could send tons of emails (which might be considered spam) to all customers each time a new product becomes available.
- This would save some customers from endless trips to the store.
    - At the same time, it'd upset other customers who aren't interested in new products.
- Either the customer wastes time checking product availability or the store wastes resources notifying the wrong customers.

Observer: Problem



Visiting the store vs. sending spam

Observer: Real World Analogy

- If you subscribe to a newspaper or magazine, you no longer need to go to the store to check if the next issue is available.
- Instead, the publisher sends new issues directly to your mailbox right after publication or even in advance.
- The publisher maintains a list of subscribers and knows which magazines they're interested in. Subscribers can leave the list at any time when they wish to stop the publisher sending new magazine issues to them.

Observer: Real World Analogy

Magazine and newspaper subscriptions.

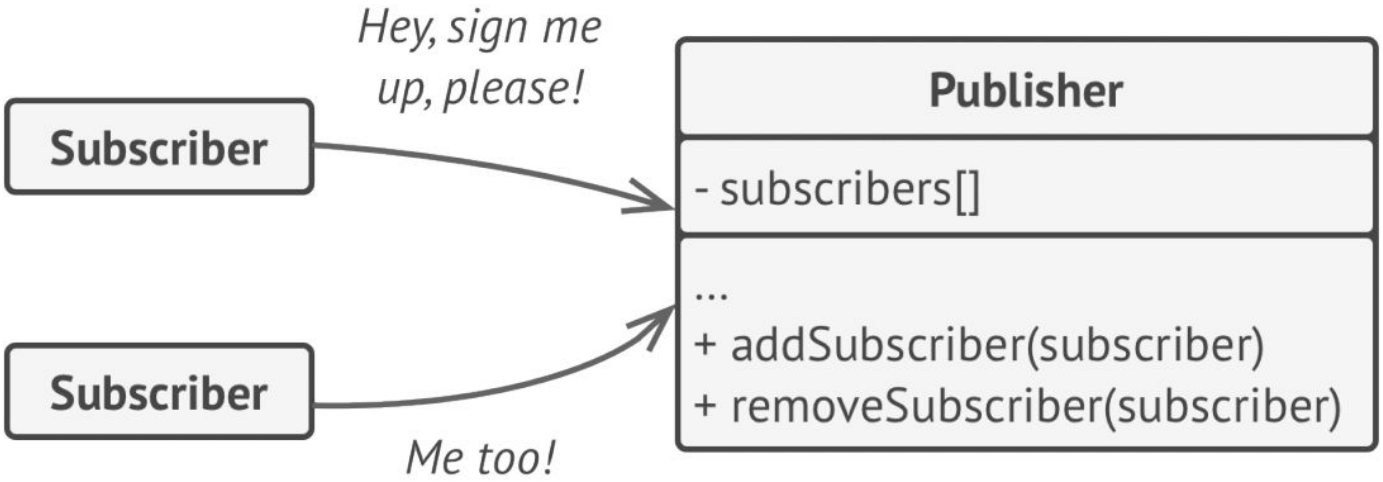
Observer: Solution

- The object that has some interesting state is often called Subject, but since it's also going to notify other objects about the changes to its state, we'll call it **publisher**.
- All other objects that want to track changes to the publisher's state are called **subscribers**.
- The Observer pattern suggests that you add a subscription mechanism to the publisher class
- So individual objects can subscribe to or unsubscribe from a stream of events coming from that publisher.

Observer: Solution

- Isn’t as complicated as it sounds. In reality, this mechanism consists of:
  - an array field for storing a list of references to subscriber objects
  - several public methods which allow adding subscribers to and removing them from that list.

Observer: Solution



A subscription mechanism lets individual objects subscribe to event notifications.

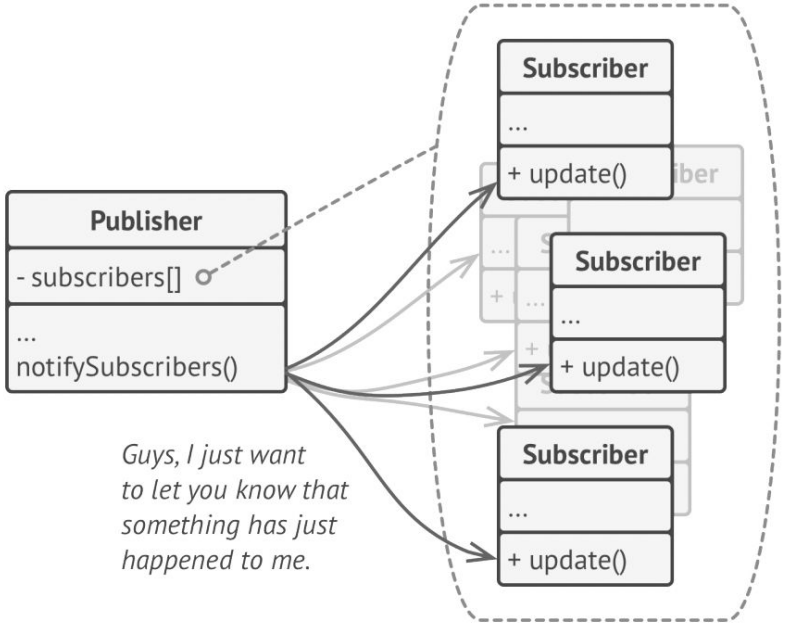
Observer: Solution

- Now, whenever an important event happens to the publisher, it goes over its subscribers and calls the specific notification method on their objects.
- Real apps might have dozens of different subscriber classes that are interested in tracking events of the same publisher class.
- You wouldn’t want to couple the publisher to all of those classes. Besides, you might not even know about some of them beforehand if your publisher class is supposed to be used by other people.
- What do we do in this case?

Observer: Solution

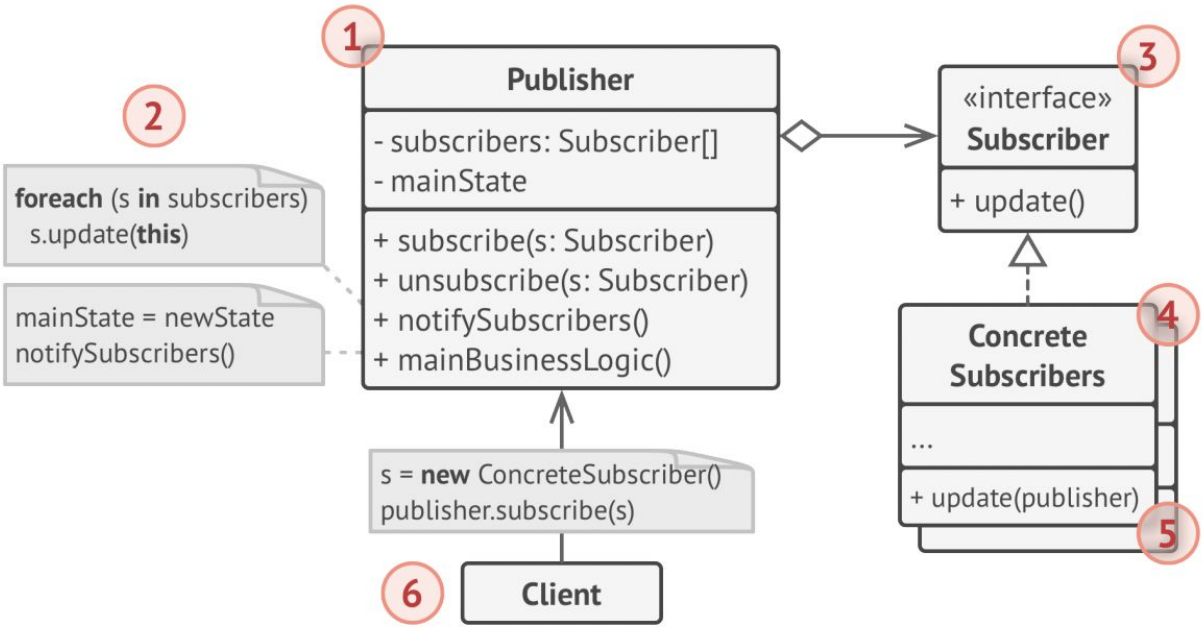
- All subscribers implement the same interface and that the publisher communicates with them only via that interface.
- This interface should declare the notification method along with a set of parameters that the publisher can use to pass some contextual data along with the notification.

Observer: Solution



Publisher notifies subscribers by calling the specific notification method on their objects.

Observer: Structure



Observer: Structure

- The Publisher issues events of interest to other objects. These events occur when the publisher changes its state or executes some behaviors. Publishers contain a subscription infrastructure that lets new subscribers join and current subscribers leave the list.

Observer: Structure

- The Publisher issues events of interest to other objects. These events occur when the publisher changes its state or executes some behaviors. Publishers contain a subscription infrastructure that lets new subscribers join and current subscribers leave the list.
- When a new event happens, the publisher goes over the subscription list and calls the notification method declared in the subscriber interface on each subscriber object.

Observer: Structure

- The Publisher issues events of interest to other objects. These events occur when the publisher changes its state or executes some behaviors. Publishers contain a subscription infrastructure that lets new subscribers join and current subscribers leave the list.
- When a new event happens, the publisher goes over the subscription list and calls the notification method declared in the subscriber interface on each subscriber object.
- The Subscriber interface declares the notification interface. In most cases, it consists of a single update method. The method may have several parameters that let the publisher pass some event details along with the update.

Observer: Structure

- The Publisher issues events of interest to other objects. These events occur when the publisher changes its state or executes some behaviors. Publishers contain a subscription infrastructure that lets new subscribers join and current subscribers leave the list.
- When a new event happens, the publisher goes over the subscription list and calls the notification method declared in the subscriber interface on each subscriber object.
- The Subscriber interface declares the notification interface. In most cases, it consists of a single update method. The method may have several parameters that let the publisher pass some event details along with the update.
- Concrete Subscribers perform some actions in response to notifications issued by the publisher. All of these classes must implement the same interface so the publisher isn't coupled to concrete classes.

Observer: Structure

- The Publisher issues events of interest to other objects. These events occur when the publisher changes its state or executes some behaviors. Publishers contain a subscription infrastructure that lets new subscribers join and current subscribers leave the list.
- When a new event happens, the publisher goes over the subscription list and calls the notification method declared in the subscriber interface on each subscriber object.
- The Subscriber interface declares the notification interface. In most cases, it consists of a single update method. The method may have several parameters that let the publisher pass some event details along with the update.
- Concrete Subscribers perform some actions in response to notifications issued by the publisher. All of these classes must implement the same interface so the publisher isn't coupled to concrete classes.
- The Client creates publisher and subscriber objects separately and then registers subscribers for publisher updates.

Observer: Pros and Cons

- Open/Closed Principle. You can introduce new subscriber classes without having to change the publisher's code (and vice versa if there's a publisher interface).
- You can establish relations between objects at runtime.
- Subscribers are notified in random order.

Observer: Pros and Cons

Homework:

Read and understand the observer.py class in Github.