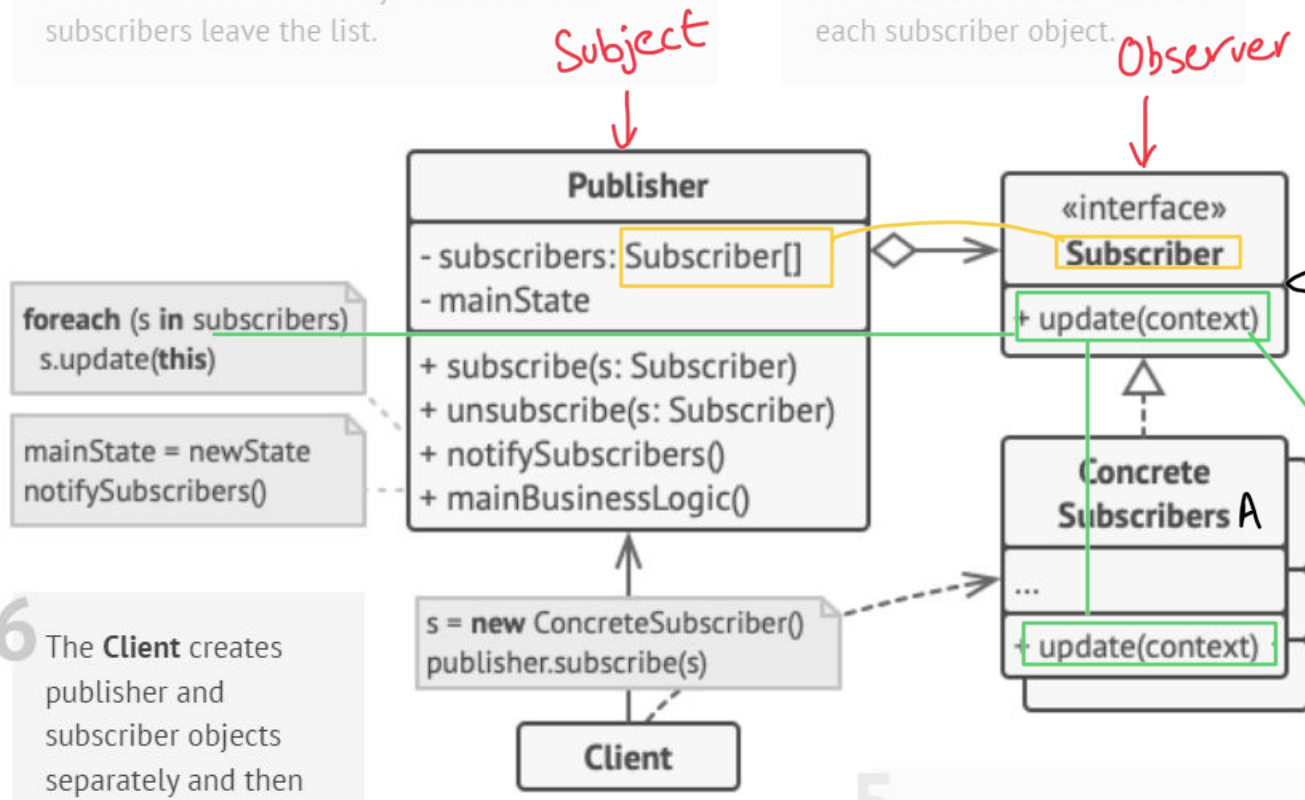
# Structure

1 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.

2 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.




3 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.

4 **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.

5 Usually, subscribers need some contextual information to handle the update correctly. For this reason, publishers often pass some context data as arguments of the notification method. The publisher can pass itself as an argument, letting subscriber fetch any required data directly.

6 The **Client** creates publisher and subscriber objects separately and then registers subscribers for publisher updates.

# Pros and Cons

***Pros:***

- *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.

**Cons:**

- Subscribers are notified in random order.