Observer Pattern 101

Agenda

- What general problems observer pattern solve ?
- What are design patterns ?
- Observer pattern & its analogy
- Code example
- Pros & common variants
- Resources

Pune Developers Community

Meetup

Search

Start a new group | Log in | Sign up

Pune Developer's Community

Pune, India

12,589 members · Public group

Organized by **Suyog** and **6 others**

Share:

About | Events | Members | Photos | Discussions | More

Join this group

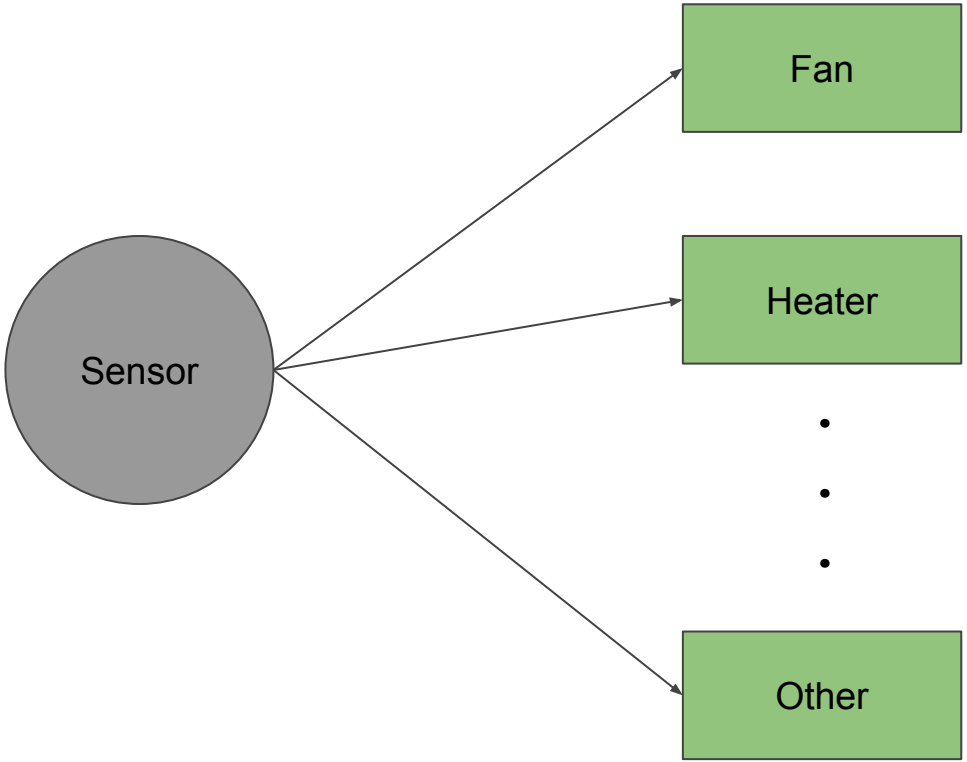
•••

Problem Statement

Develop a pseudo application for a smart home in which there is centralised **Sensor** which continuously measures temperature. As soon as temperature changes multiple home **Devices** (e.g. Fan, Heater etc) shall get notified.

Our application should be extensible since in future many more device can use sensor measurements and also existing devices can stop using sensor measurements.

Problem Statement



Similar Problems

- Given a toggle button whenever user changes its state, some other parts of application logic must get executed.
- In an MVC pattern based application, whenever model changes the view must get updated.
- In events based application, on trigger of any event many other parts of application should get notified.

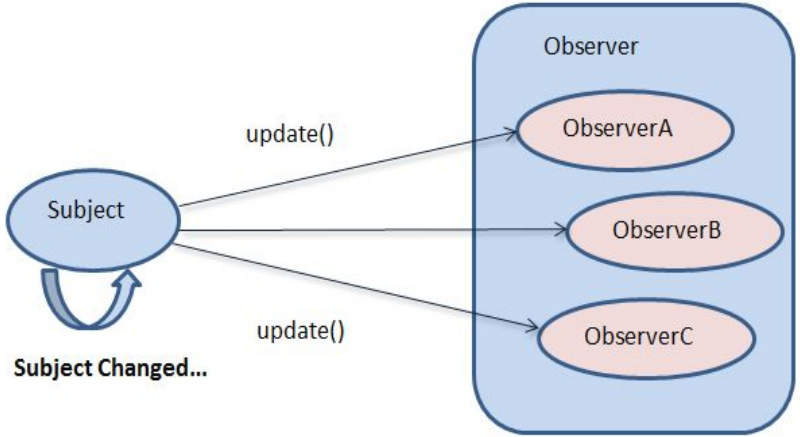
In Summary, when a state of one object changes multiple other objects get informed.

Design Patterns

Design Patterns are reusable solutions to commonly occurring problems in software design. The design patterns are not a specific piece of code, but a general concept or a best practises for solving a similar problems.

Observer Pattern

The observer pattern is a behavioural design pattern in which an object, called the **subject**, maintains a list of its dependents, called **observers**, and notifies them automatically of any state changes, usually by calling one of their methods.



Analogy

Observer pattern is similar to how magazine subscription works:

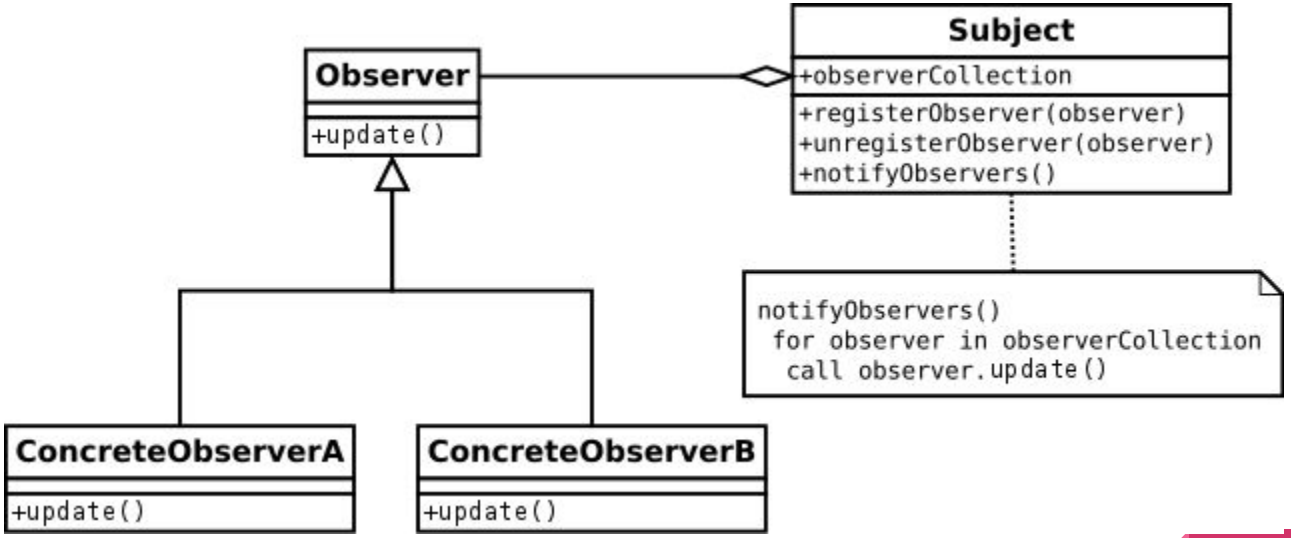
- You subscribe to a particular publisher, and every time there’s a new edition it gets delivered to you. As long as you remain a subscriber, you get latest magazine.
- You unsubscribe when you don’t want magazine anymore, and they stop being delivered.
- People keeps on subscribing and unsubscribing to magazine, and publisher always delivers new edition of magazine to subscribed people.

Code Example

```typescript
interface Subject {
    registerObserver(o: Observer);
    unregisterObserver(o: Observer);
    notifyObservers();
}
interface Observer {
    notify(temperature: number);
}
class Sensor implements Subject {
    private temperature: number;
    private observers: Observer[] = [];
    registerObserver(o: Observer) {
        this.observers.push(o);
    }
    unregisterObserver(o: Observer) {
        let index = this.observers.indexOf(o);
        this.observers.splice(index, 1);
    }
    notifyObservers() {
        for (let observer of this.observers) {
            observer.notify(this.temperature);
        }
    }
    setTemperature(temp: number) {
        console.log('Sensor: New Temperature measured : ', temp);
        this.temperature = temp;
        this.notifyObservers();
    }
}
```

```
31  class Fan implements Observer {
32      subject: Subject;
33
34      constructor(sensor: Subject) {
35          this.subject = sensor;
36          sensor.registerObserver(this);
37      }
38
39      notify(temperature: number) {
40          console.log('FAN: Got new temperature values');
41          if (temperature > 30) {
42              console.log('FAN : Its quite hot, turning myself ON');
43          } else {
44              console.log('FAN: Its cool, turning myself OFF');
45          }
46      }
47  }
48
49  // Driver code
50  let sensor = new Sensor();
51  let fan = new Fan(sensor);
52  sensor.setTemperature(35);
53  sensor.setTemperature(18);
54  /*
55   Output
56  Sensor: New Temperature measured :  35
57  FAN: Got new temperature values
58  FAN : Its quite hot, turning myself ON
59  Sensor: New Temperature measured :  18
60  FAN: Got new temperature values
61  FAN: Its cool, turning myself OFF
```

UML Diagram


Observer
+update()
Subject
+observerCollection
+registerObserver(observer)
+unregisterObserver(observer)
+notifyObservers()
notifyObservers()
for observer in observerCollection
call observer.update()
ConcreteObserverA
+update()
ConcreteObserverB
+update()

Pros of Observer Pattern

Observer pattern establishes a **loosely coupled** link between subject and observers.

- Subject is unaware of concrete class of observers and vice-versa.
- We can add new observer at any time. Likewise we can remove observers at any time.
- We never need to modify subject class to add a new type of observer.
- We can use either subject or observer independent of each other.

Common Variants

- Many Subject to Many Observers: An observer may observe multiple subjects at a time.

- Observer triggers the update: If a state of Subject changes too frequent, it's better to trigger the update from observer.

- Push & Pull Method: If the data passed by Subject is too big, it’s better to just inform about the state change to observer. And than let observer pull a subset of data that it needs.

Suggested Resources

If you like to further explore world of design patterns here are some suggested resources you can start with:

- https://www.oodesign.com/
- https://refactoring.guru/design-patterns
- Head First Design Patterns Book

For any query/comment/feedback you can mail me at *mohitkh7@gmail.com*

Thank You !