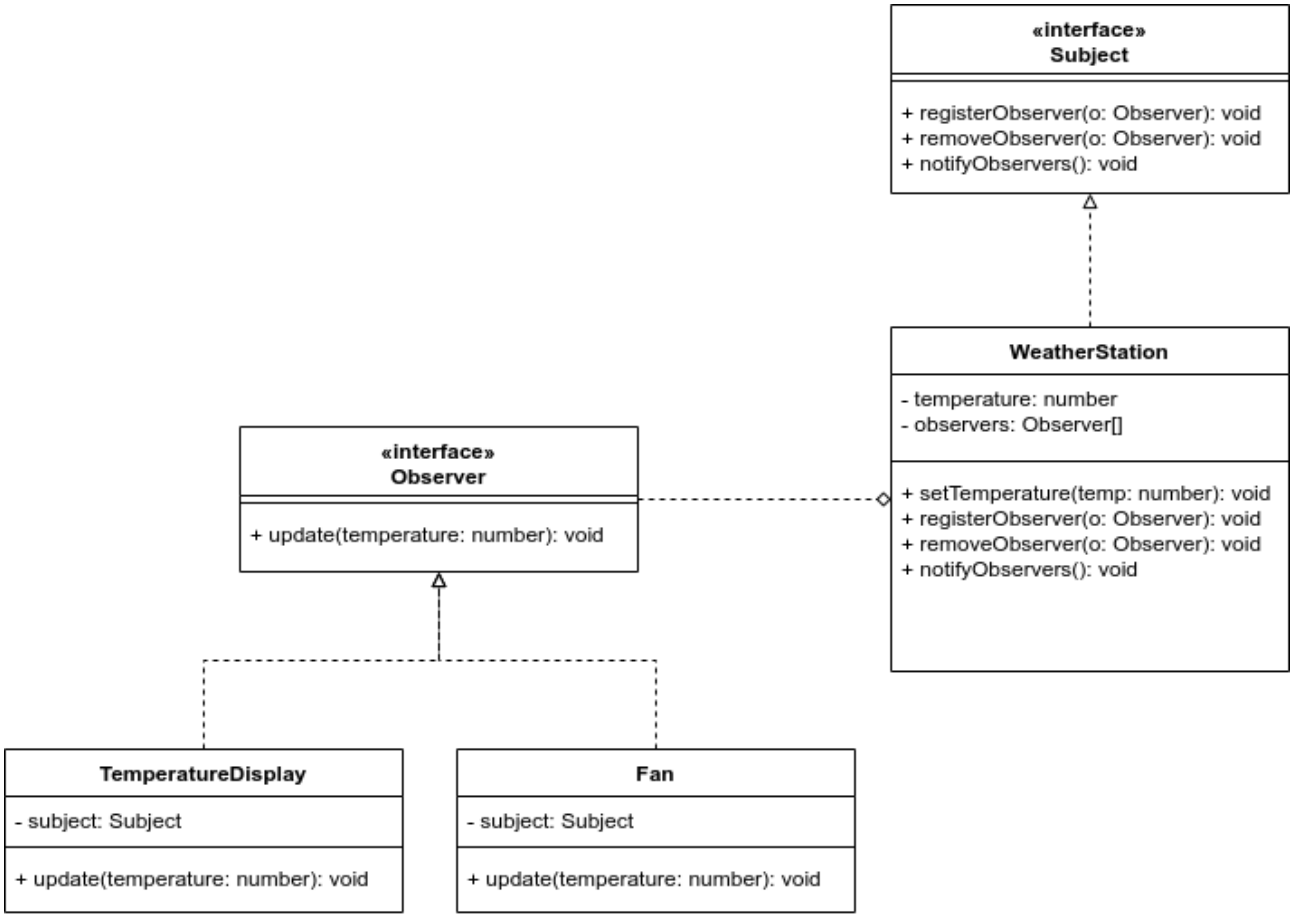
Design Patterns
Behavioral
Observer



```
    interface Subject {
        registerObserver(o: Observer): void;
        removeObserver(o: Observer): void;
        notifyObservers(): void;
    }

    interface Observer {
        update(temperature: number): void;
    }

    class WeatherStation implements Subject {
        private temperature!: number;
        private observers: Observer[] = [];
        setTemperature(temp: number): void {
            console.log(`WeatherStation: new temperature measurement: ${temp}`);
            this.temperature = temp;
            this.notifyObservers();
        }
        registerObserver(o: Observer): void {
            if (this.observers.includes(o)) throw new Error("Observer already registered");
            this.observers.push(o);
        }
        removeObserver(o: Observer): void {
            let obsItem = this.observers.indexOf(o);
            if (obsItem < 0) throw new Error("Observer not registered");
            this.observers.splice(obsItem, 1);
        }
        notifyObservers(): void {
            for (const observer of this.observers)
                observer.update(this.temperature);
        }
    }

    class TemperatureDisplay implements Observer {
        private subject!: Subject;

        constructor(weatherStation: Subject) {
            this.subject = weatherStation;
            weatherStation.registerObserver(this);
        }

        update(temperature: number): void {
            console.log('TemperatureDisplay: I need to update my display.')
            //Logic
        }
    }

    class Fan implements Observer {
        private subject!: Subject;

        constructor(weatherStation: Subject) {
            this.subject = weatherStation;
            weatherStation.registerObserver(this);
        }

        update(temperature: number): void {
            if (temperature > 25) {
                console.log('Fan: Its hot here, turning myself on');
                //Logic
            } else {
                console.log('Fan: Its nice and cool, turning myself off');
                //Logic
            }
        }
    }

    let weatherStation = new WeatherStation();
    let weatherDisplay = new TemperatureDisplay(weatherStation);
    let fan = new Fan(weatherStation);
    weatherStation.setTemperature(20);
    weatherStation.setTemperature(30);

// OUTPUT:
// WeatherStation: new temperature measurement: 20
// TemperatureDisplay: I need to update my display.
// Fan: Its nice and cool, turning myself off
// WeatherStation: new temperature measurement: 30
// TemperatureDisplay: I need to update my display.
// Fan: Its hot here, turning myself on
```