Observer pattern

It's behavioral design pattern where an object maintains a list of its dependents.

The dependents call observers.

It's providing one-to-many relationship that when one object changes state its update automatically and notifies all dependencies.

Observer pattern has some principles:

- Observable (Subject):
  1. Subscribe – add object that depends on the state of the subject
  2. Unsubscribe – remove object from the state.
  3. Update – change state (occur when observer triggered).
- Observer – object that depends on the subject and provide an update that the subject calls when it notifies observers.
- Loose coupling – subjects and observers communicate with well-defined interfaces. That mean there's no coupling between the subject and it's observers.
- Dynamic subscription – can added and removed observers dynamically on run time.
- Event-driven architecture
- Broadcasting updates – dispatch the notifies for all the registered observers on the subject (real example: Socket.io)
- Consistent interface
- Maintaining state consistency – ensure that the state of the observers is stay consistent with the state of the subject.

In nodejs, the EventEmitter in way to implement the Observer pattern, providing a convenient mechanism for objects to emit events and have listeners responds to this events.

In Angular, there is feature called two-way data biding that allows synchronization of data between a component and its view, like between component data and the HTML file. The implementation of this feature is through the Observer pattern (rxjs library):

Service:

```
import { Injectable } from '@angular/core'
import { Subject } from 'rxjs'

@Injectable({
  providedIn: 'root'
})
export class StateService {
  private stateSubject = new Subject<string>()

  state$ = this.stateSubject.asObservable()

  updateState(newState: string): void {
    this.stateSubject.next(newState)
  }
}
```

Component:

```
import { Component, OnInit } from '@angular/core'
import { StateService } from './state.service'

@Component({
  selector: 'app-root',
  template: `
    <div>
      <p>Current State: {{ currentState }}</p>
      <button (click)="updateState()">Update State</button>
    </div>
  `,
})
export class AppComponent implements OnInit {
  currentState: string = ''

  constructor(private stateService: StateService) {}

  ngOnInit() {
    // Subscribe to changes in the state
    this.stateService.state$.subscribe((newState) => {
```

```
      this.currentState = newState
    })
  }


ngOnDestroy() {
    this.subscription.unsubscribe()
  }

  updateState() {
    const newState = `New State ${Math.random()}`
    this.stateService.updateState(newState)
  }
}
```