Novita syahwa tri hapsari

2311104007

A. Berikan salah satu contoh kondisi dimana design pattern “Observer” dapat digunakan
   Jawaban:
   Observer Pattern cocok digunakan ketika terdapat satu objek (subject) yang perubahannya harus secara otomatis memberi tahu dan memperbarui banyak objek lain (observers)
   Contoh kasus: Dalam aplikasi cuaca, terdapat satu pusat data cuaca (Subject) yang memberikan informasi suhu, kelembaban, dan tekanan udara. Setiap kali data ini diperbarui, tampilan seperti aplikasi web, notifikasi HP, atau papan LED (Observers) juga harus ikut diperbarui secara otomatis.
B. Berikan penjelasan singkat mengenai langkah-langkah dalam mengimplementasikan design pattern
   Jawab :

   - **Buat antarmuka Observer**

   Mendefinisikan metode `update()` yang akan dipanggil oleh Subject ketika ada perubahan.

   - **Buat antarmuka Subject**

   Mendefinisikan metode untuk:

   - Mendaftarkan observer (`attach()`),
   - Menghapus observer (`detach()`), dan
   - Memberi tahu semua observer (`notify()`).

   - **Implementasikan kelas konkret untuk Subject**

   Kelas ini menyimpan data utama dan daftar observer. Ketika data berubah, kelas ini memanggil `notify()` untuk memperbarui semua observer.

   - **Implementasikan kelas konkret untuk Observer**

   Kelas ini mengimplementasikan metode `update()` yang berisi logika untuk menyesuaikan diri dengan perubahan dari Subject.

- **Hubungkan observer dengan subject**

  Observer didaftarkan ke subject menggunakan `attach()` agar bisa menerima pembaruan secara otomatis.

C. Berikan kelebihan dan kekurangan dari design pattern “Observer”

Kelebihan:

- Observer Pattern memungkinkan penambahan objek-objek pengamat (observer) tanpa harus mengubah kode pada objek utama (subject), sehingga mempermudah pengembangan sistem.
- Objek subject tidak perlu mengetahui detail implementasi observer. Hal ini membuat hubungan antar komponen menjadi longgar (loosely coupled) dan lebih mudah dikelola.
- Ketika terjadi perubahan pada subject, seluruh observer yang terdaftar akan diperbarui secara otomatis tanpa intervensi manual.
- Pattern ini sangat sesuai digunakan pada sistem yang datanya sering berubah dan perlu disebarluaskan ke banyak bagian secara real-time.

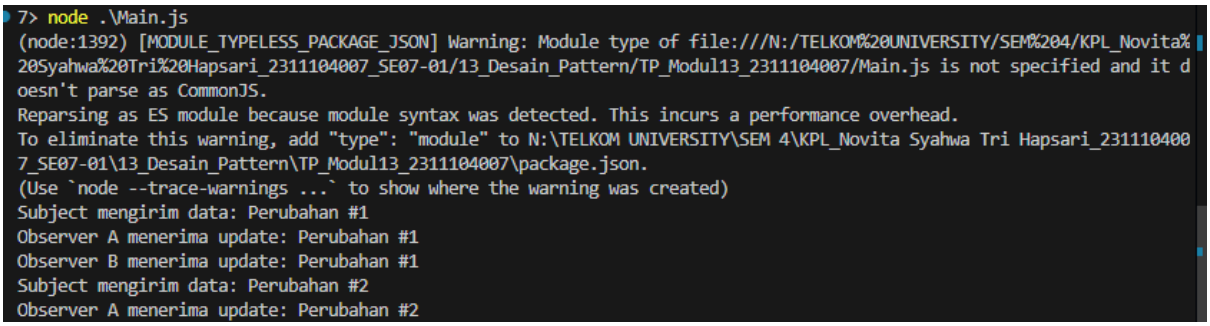
Kekurangan:

- Dengan banyaknya observer yang terhubung ke satu subject, pelacakan proses dan debugging menjadi lebih kompleks.
- Jika jumlah observer sangat banyak, proses pemanggilan metode update() pada masing-masing observer dapat membebani kinerja sistem.
- Ketergantungan antar observer bisa menimbulkan masalah
- Kesulitan dalam pengujian

Code :

```
    export class ConcreteObserver {
    constructor(name) {
        this.name = name;
    }

    update(data) {
        console.log(`${this.name} menerima update: ${data}`);
    }
    }
    export class Subject {
    constructor() {
        this.observers = [];
    }

    subscribe(observer) {
        this.observers.push(observer);
    }

    unsubscribe(observer) {
        this.observers = this.observers.filter(obs => obs !== observer);
    }

    notify(data) {
        this.observers.forEach(observer => observer.update(data));
    }
    }
// Baris ini mengimpor dua class, yaitu Subject (pengirim notifikasi) dan ConcreteObserver
(penerima notifikasi),
// agar bisa digunakan di file utama.
import { Subject } from './Subject.js';
import { ConcreteObserver } from './Observer.js';
//Membuat instance subject yang akan menjadi pusat notifikasi untuk para observer.
const subject = new Subject();
//Membuat dua observer (pengamat) bernama “Observer A” dan “Observer B”.
// Nama ini akan digunakan saat mencetak update.
const observer1 = new ConcreteObserver("Observer A");
const observer2 = new ConcreteObserver("Observer B");
//Mendaftarkan kedua observer ke dalam subject agar mereka menerima notifikasi saat ada
perubahan.
subject.subscribe(observer1);
subject.subscribe(observer2);
//Menampilkan pesan bahwa subject akan mengirim update, kemudian notify() akan memanggil
method update() pada semua observer
//  yang terdaftar, dengan data "Perubahan #1".
console.log("Subject mengirim data: Perubahan #1");
subject.notify("Perubahan #1");
//Menghapus Observer B dari daftar observer, jadi setelah ini dia tidak akan menerima
notifikasi lagi.
subject.unsubscribe(observer2);
//Sekali lagi subject mengirim update, tapi kali ini hanya Observer A yang menerima karena
Observer B sudah dihapus sebelumnya.
console.log("Subject mengirim data: Perubahan #2");
subject.notify("Perubahan #2");
```

hasil running :



Penjelasan code:

Dalam penerapan Observer Design Pattern dalam JavaScript. Di sini terdapat dua class utama: Subject dan ConcreteObserver. Subject berperan sebagai pusat data yang dapat memberitahu (men-notify) para observer ketika ada perubahan. Sementara itu, ConcreteObserver adalah objek yang ingin mengetahui perubahan dari Subject. Setiap ConcreteObserver punya nama dan memiliki method update() untuk menerima data baru dari Subject.

Dalam program utama (Main.js), pertama-tama dibuat objek subject, lalu dua observer (observer1 dan observer2) dibuat dan didaftarkan ke dalam subject dengan subscribe(). Ketika subject.notify("Perubahan #1") dipanggil, kedua observer akan menerima pesan tersebut. Setelah itu, observer2 dihapus dari daftar pengamat dengan unsubscribe(). Ketika notify("Perubahan #2") dipanggil lagi, hanya observer1 yang akan menerima pesan karena observer2 sudah tidak berlangganan lagi. Pola ini sangat berguna untuk membangun sistem di mana objek-objek perlu bereaksi secara otomatis terhadap perubahan dari objek lain.