2311104035
Reza Irawan

Tugas Pendahuluan
Modul 13

1. Penjelasan Design Pattern Observer

   A. Contoh Kondisi Penggunaan Observer

   Observer pattern cocok digunakan pada sistem **notifikasi**. Misalnya, dalam aplikasi cuaca, setiap kali data cuaca berubah, semua komponen yang menampilkan informasi (seperti suhu, kelembapan, dll) harus diperbarui secara otomatis. Observer memudahkan banyak *subscriber* untuk mengikuti perubahan tanpa mengganggu logika utama.

   B. Langkah-langkah Implementasi Observer

   1. Buat class **Subject** yang menyimpan daftar observer dan menyediakan method untuk menambah, menghapus, dan memberi tahu observer.
   2. Buat class **Observer** yang memiliki method update() untuk menerima notifikasi.
   3. Saat state Subject berubah, ia akan memanggil method update() milik semua observer yang terdaftar.

   C. Kelebihan dan Kekurangan Observer

   Kelebihan:

   1. Memisahkan logika antara objek utama (subject) dan pengamat (observer).
   2. Mempermudah penambahan pengamat baru tanpa mengubah kode subject.
   3. Cocok untuk event-driven systems.

   Kekurangan:

   1. Sulit dilacak saat jumlah observer banyak (debugging).
   2. Performa menurun jika terlalu banyak observer aktif.
   3. Ketergantungan tersembunyi antara subject dan observer jika tidak didokumentasikan dengan baik.

2. ObserverPattern.js

```
class Subject {
  constructor() {
    this.observers = [];
    this.state = 0;
  }

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

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

  notify() {
    this.observers.forEach(observer => observer.update(this));
  }

  setState(newState) {
    this.state = newState;
    this.notify();
  }

  getState() {
    return this.state;
  }
}

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

  update(subject) {
    console.log(`${this.name} menerima update: State sekarang = ${subject.getState()}`);
  }
}

const subject = new Subject();

const observer1 = new ConcreteObserver("Observer 1");
const observer2 = new ConcreteObserver("Observer 2");

subject.attach(observer1);
subject.attach(observer2);

console.log("Set state ke 1");
subject.setState(1);

console.log("Set state ke 2");
subject.setState(2);

subject.detach(observer1);

console.log("Set state ke 3 (Observer 1 sudah dilepas)");
subject.setState(3);
```

Kode di atas menerapkan pola desain Observer dalam JavaScript, di mana objek `Subject` menyimpan daftar observer (pengamat) yang akan diberi tahu setiap kali ada perubahan state. Kelas `ConcreteObserver` mewakili pengamat yang merespons perubahan dengan mencetak status terbaru. Dua pengamat ditambahkan ke `Subject`, lalu state-nya diubah beberapa kali. Setelah `observer1` dilepas, hanya `observer2` yang menerima pembaruan berikutnya.

Hasil running :

```
Set state ke 1
Observer 1 menerima update: State sekarang = 1
Observer 2 menerima update: State sekarang = 1
Set state ke 2
Observer 1 menerima update: State sekarang = 2
Observer 2 menerima update: State sekarang = 2
Set state ke 3 (Observer 1 sudah dilepas)
Observer 2 menerima update: State sekarang = 3
```