

## 1. Penjelasan Design Pattern

### A. Berikan salah satu contoh kondisi dimana design pattern “Observer” dapat digunakan

Cocok saat terdapat satu object (subject) yang perubahannya harus otomatis diberitahukan ke objek lain (observer).

### B. Berikan penjelasan singkat mengenai langkah-langkah dalam mengimplementasikan design pattern “Observer”

- Membuat interface atau class observer
- Membuat class subject
- Mengimplementasikan subject dengan cara menyimpan daftar observer dan memanggil update pada masing-masing observer saat terjadi perubahan
- Mengimplementasikan observer pada saat menerima notifikasi dari subject dan melakukan aksi
- Menghubungkan subject dan observer

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

- Kelebihan
  - Skalabilitas
  - Mendukung event-driven programming
  - Keterkaitan longgar atau loose coupling, yaitu subject tidak perlu tau detail dari observer, mereka hanya mengikuti kontrak (interface)
- Kekurangan
  - Debugging yang lebih kompleks
  - Kemungkinan kebocoran memori
  - Urutan notifikasi yang tidak terjamin

## 2. Penjelasan Observer

Kode ini mengimplementasikan pola desain Observer dalam JavaScript, di mana kelas `PusatData` bertindak sebagai subject yang menyimpan data dan daftar observer, serta memberikan notifikasi kepada semua observer ketika datanya berubah. Observer direpresentasikan oleh kelas `Pengamat` yang memiliki method `update()` untuk menerima perubahan. Dalam fungsi utama, dua observer didaftarkan ke `PusatData`, lalu ketika data diubah dengan `setData()`, semua observer diberi tahu. Setelah salah satu observer dilepas dengan `detach()`, hanya observer yang tersisa yang menerima notifikasi saat data diperbarui lagi.

```
JS PusatDataSingleton.js U JS PusatDataObserver.js U X
13_Design_Pattern > TP_2311104013 > JS PusatDataObserver.js > ...
1  class PusatData {
2      constructor() {
3          this.observers = [];
4          this.data = "";
5      }
6
7      attach(observer) {
8          this.observers.push(observer);
9      }
10
11     detach(observer) {
12         this.observers = this.observers.filter((obs) => obs !== observer);
13     }
14
15     notify() {
16         for (const observer of this.observers) {
17             observer.update(this);
18         }
19     }
20
21     setData(newData) {
22         console.log(`PusatData: Mengubah data menjadi '${newData}'`);
23         this.data = newData;
24         this.notify();
25     }
26
27     getData() {
28         return this.data;
29     }
30 }
31
32 class Pengamat {
33     constructor(nama) {
34         this.nama = nama;
35     }
36
37     update(subject) {
38         console.log(`${this.nama} menerima update: data baru adalah '${subject.getData()}'`);
39     }
40 }
41
42 const pusat = new PusatData();
43
44 const pengamat1 = new Pengamat("Observer 1");
45 const pengamat2 = new Pengamat("Observer 2");
46
47 pusat.attach(pengamat1);
48 pusat.attach(pengamat2);
49
50 pusat.setData("Data 1");
51
52 pusat.detach(pengamat2);
53
54 pusat.setData("Data 2");
55
```

### 3. Output

```
1104013> node .\PusatDataObserver.js
● PusatData: Mengubah data menjadi 'Data 1'
  Observer 1 menerima update: data baru adalah 'Data 1'
  Observer 2 menerima update: data baru adalah 'Data 1'
  PusatData: Mengubah data menjadi 'Data 2'
  Observer 1 menerima update: data baru adalah 'Data 2'
○ PS E:\KULIAH\Semester 4\Konstruksi Perangkat Lunak\Praktikum\
1104013> 
```