

## Penjelasan Observer Design Pattern

### 1. Contoh Kondisi Penggunaan

Salah satu contoh penggunaan Observer Pattern adalah pada aplikasi notifikasi cuaca. Misalnya, ada sebuah objek pusat pengamat cuaca (WeatherStation) yang mendeteksi perubahan suhu, kelembapan, dan tekanan udara. Beberapa objek lain seperti tampilan suhu (TemperatureDisplay), tampilan kelembapan (HumidityDisplay), dan sistem peringatan (WeatherAlertSystem) ingin mengetahui setiap kali data cuaca berubah. Dengan Observer Pattern, semua objek ini bisa berlangganan ke WeatherStation dan akan diberi tahu secara otomatis saat data cuaca berubah, tanpa WeatherStation harus mengetahui detail tiap objek tampilan.

### 2. Langkah-langkah Implementasi

- a. Pisahkan fungsionalitas utama (publisher) dari bagian lain yang akan menjadi subscriber.
- b. Buat antarmuka (interface) untuk subscriber, misalnya dengan metode `update(data)`, yang akan dipanggil saat ada perubahan.
- c. Buat antarmuka untuk publisher yang mendefinisikan metode `subscribe(listener)`, `unsubscribe(listener)`, dan `notify(data)`.
- d. Implementasikan logika langganan dalam kelas publisher, biasanya dengan menyimpan daftar subscriber dan memanggil `update()` mereka saat event terjadi.
- e. Buat kelas subscriber konkret yang mengimplementasikan antarmuka dan menyediakan aksi spesifik saat menerima notifikasi.
- f. Di sisi klien, buat instance subscriber dan daftarkan ke publisher.

### 3. Kelebihan & Kekurangan

#### a. Kelebihan

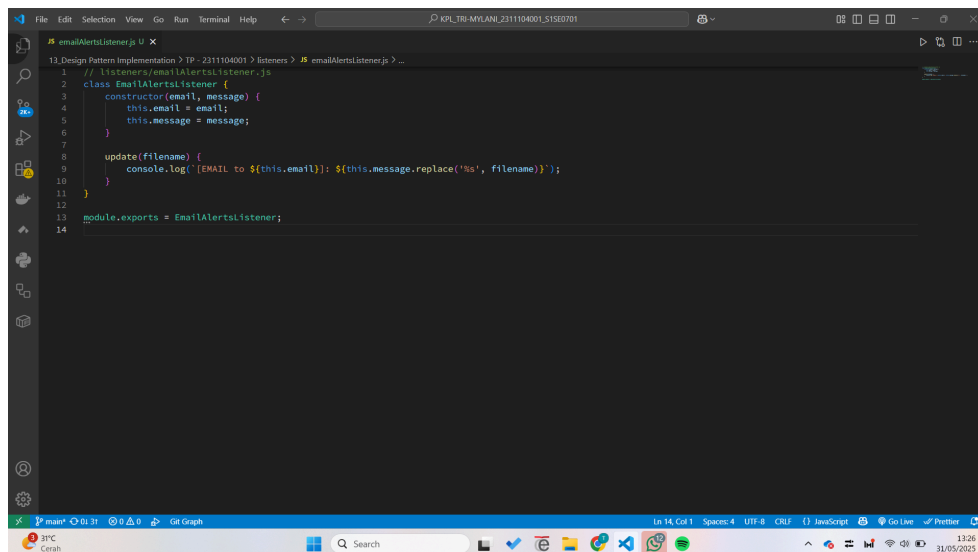
- Mengikuti prinsip Open/Closed: Publisher dan subscriber bisa dikembangkan tanpa mengubah kode satu sama lain.
- Fleksibel dan Dinamis: Objek subscriber bisa bergabung atau keluar kapan saja saat runtime.
- Decoupling: Publisher tidak perlu mengetahui detail dari subscriber; cukup tahu bahwa mereka implementasi dari antarmuka yang sama.

- Responsif terhadap event: Memudahkan sistem untuk merespons berbagai event secara modular.

b. Kekurangan

- Sulit dilacak: Saat jumlah subscriber banyak, sulit mengetahui siapa yang bereaksi terhadap event tertentu.
- Ketergantungan tidak terlihat: Hubungan antar objek tidak eksplisit, sehingga bisa menyulitkan debugging.
- Risiko memory leaks: Jika subscriber tidak dihapus dari daftar saat tidak lagi digunakan, bisa terjadi memory leak.
- Urutan notifikasi tidak terjamin: Tidak ada jaminan urutan saat subscriber menerima notifikasi, yang bisa menimbulkan masalah dalam logika aplikasi tertentu.

### emailAlertsListener.js



```

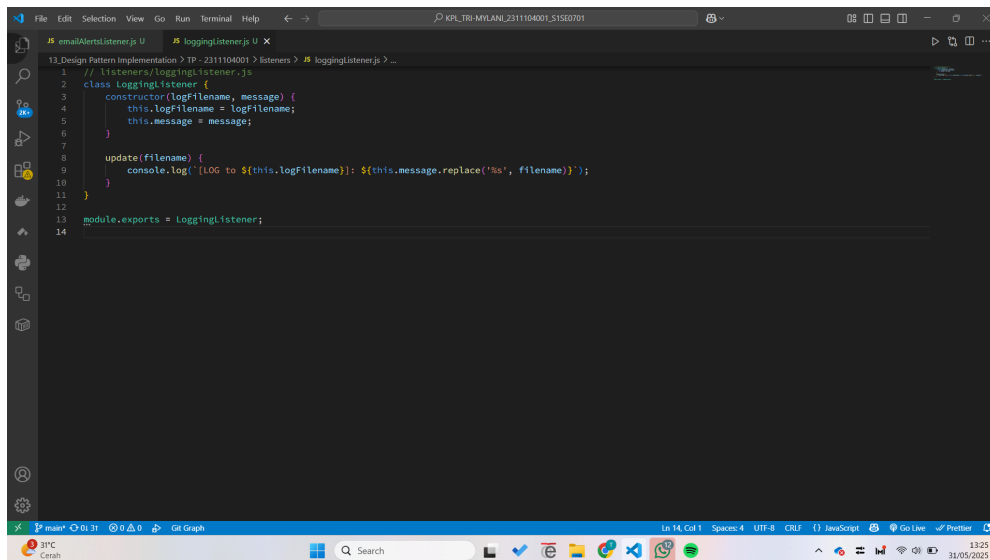
1 // listeners/emailAlertsListener.js
2 class EmailAlertsListener {
3   constructor(email, message) {
4     this.email = email;
5     this.message = message;
6   }
7
8   update(filename) {
9     console.log(`EMAIL to ${this.email}: ${this.message.replace('%s', filename)}`);
10  }
11 }
12
13 module.exports = EmailAlertsListener;
14

```

### Penjelasan :

Kode ini mendefinisikan kelas EmailAlertsListener yang bertugas sebagai pendengar event untuk mengirim notifikasi email (simulasi). Ketika event terjadi, metode update(filename) dipanggil dengan nama file sebagai parameter. Di dalamnya, pesan email yang sudah disiapkan akan dicetak ke konsol dengan menggantikan %s dengan nama file terkait. Meskipun tidak benar-benar mengirim email, ini mensimulasikan pemberitahuan email saat event tertentu, seperti penyimpanan file, dipicu.

## loggingListener.js



```

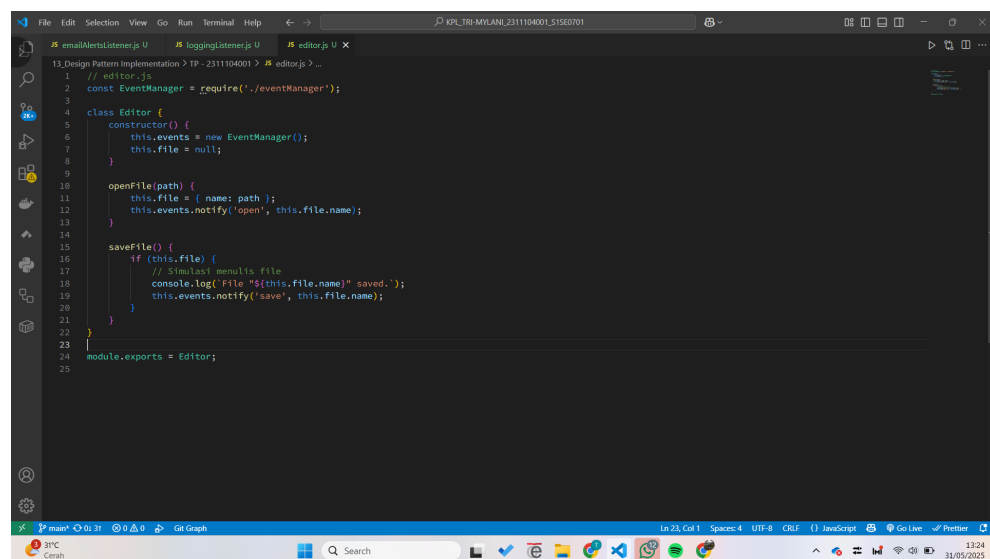
1 // loggingListener.js
2 class LoggingListener {
3   constructor(logFilename, message) {
4     this.logFilename = logFilename;
5     this.message = message;
6   }
7
8   update(filename) {
9     console.log('[LOG to $this.logFilename]: $this.message.replace('%s', filename)');
10  }
11 }
12
13 module.exports = LoggingListener;
14

```

Penjelasan :

Kode di atas mendefinisikan kelas LoggingListener yang berfungsi sebagai pendengar event untuk mencatat aktivitas ke sebuah file log (simulasi). Saat event terjadi, metode update(filename) dipanggil dengan nama file sebagai parameter. Di dalamnya, pesan log yang sudah diatur sebelumnya akan ditampilkan ke konsol dengan format yang menggantikan %s dengan nama file yang terkait. Meskipun tidak benar-benar menulis ke file, ini mensimulasikan pencatatan log setiap kali event dipicu.

## editor.js



```

1 // editor.js
2 const EventManager = require('./EventManager');
3
4 class Editor {
5   constructor() {
6     this.events = new EventManager();
7     this.file = null;
8   }
9
10  openFile(path) {
11    this.file = { name: path };
12    this.events.notify('open', this.file.name);
13  }
14
15  saveFile() {
16    if (this.file) {
17      // Simulasi menulis file
18      console.log('File $this.file.name saved. ');
19      this.events.notify('save', this.file.name);
20    }
21  }
22 }
23
24 module.exports = Editor;
25

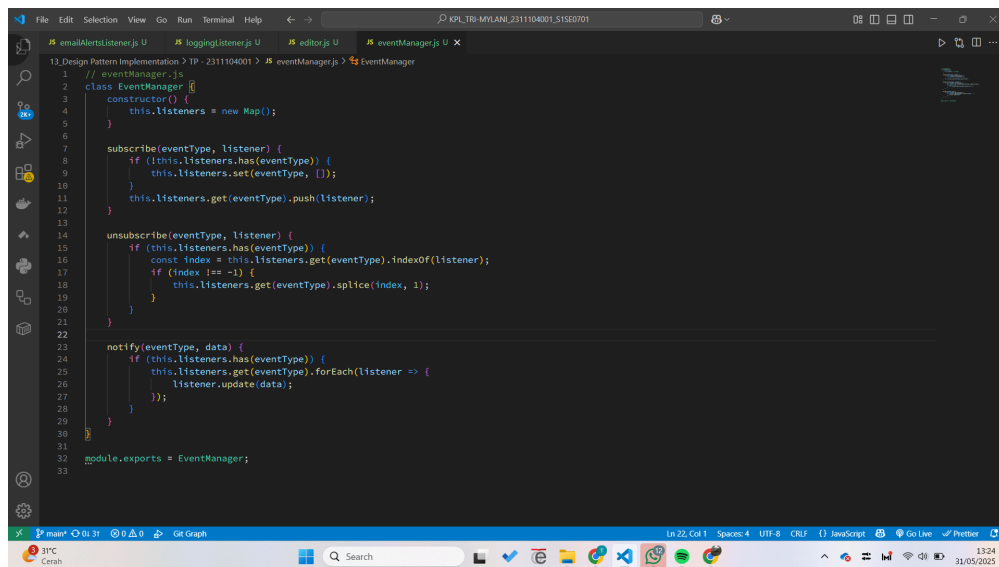
```

Penjelasan :

Kode editor.js ini mendefinisikan kelas Editor yang menggunakan EventManager untuk

mengelola event. Saat sebuah file dibuka lewat metode `openFile(path)`, objek Editor menyimpan nama file dan memberi tahu semua listener yang berlangganan event 'open' dengan mengirimkan nama file tersebut. Saat metode `saveFile()` dipanggil, jika ada file yang dibuka, akan ditampilkan pesan bahwa file telah disimpan, lalu event 'save' diberitahukan ke semua listener dengan nama file. Jadi, kelas ini memungkinkan proses membuka dan menyimpan file sekaligus memicu event yang bisa direspons oleh listener lain secara terpisah.

### eventManager.js



```

1 // eventManager.js
2 class EventManager {
3   constructor() {
4     this.listeners = new Map();
5   }
6
7   subscribe(eventType, listener) {
8     if (!this.listeners.has(eventType)) {
9       this.listeners.set(eventType, []);
10    }
11    this.listeners.get(eventType).push(listener);
12  }
13
14  unsubscribe(eventType, listener) {
15    if (this.listeners.has(eventType)) {
16      const index = this.listeners.get(eventType).indexOf(listener);
17      if (index !== -1) {
18        this.listeners.get(eventType).splice(index, 1);
19      }
20    }
21  }
22
23  notify(eventType, data) {
24    if (this.listeners.has(eventType)) {
25      this.listeners.get(eventType).forEach(listener => {
26        listener.update(data);
27      });
28    }
29  }
30
31  module.exports = EventManager;
32
33

```

### Penjelasan :

Kode di atas adalah implementasi kelas `EventManager` yang berfungsi mengelola sistem event dan listener. `EventManager` menyimpan daftar listener untuk setiap jenis event dalam sebuah `Map`. Dengan metode `subscribe`, listener baru bisa didaftarkan ke event tertentu, sedangkan `unsubscribe` digunakan untuk menghapus listener dari event tersebut. Saat sebuah event terjadi, metode `notify` akan memanggil metode `update` pada semua listener yang terdaftar untuk event itu, sekaligus mengirim data terkait event tersebut. Ini memungkinkan sistem untuk memberitahu banyak listener secara otomatis ketika suatu event terjadi.

### index.js

```

1 // index.js
2 const Editor = require('./editor');
3 const LoggingListener = require('./listeners/loggingListener');
4 const EmailAlertsListener = require('./listeners/emailAlertsListener');
5
6 const editor = new Editor();
7
8 const logger = new LoggingListener(
9   'log.txt',
10  'Someone has opened the file: %s'
11);
12 editor.events.subscribe('open', logger);
13
14 const emailAlerts = new EmailAlertsListener(
15   'admin@example.com',
16   'Someone has changed the file: %s'
17);
18 editor.events.subscribe('save', emailAlerts);
19
20 // Simulasi
21 editor.openFile('artikel.txt');
22 editor.saveFile();
23

```

### Penjelasan :

Kode di atas merupakan contoh penggunaan pola event listener pada sebuah editor sederhana. Di sini, objek Editor memiliki event yang bisa didaftarkan ke berbagai listener atau pendengar. LoggingListener bertugas mencatat ke file log ketika file dibuka, sedangkan EmailAlertsListener mengirim email pemberitahuan saat file disimpan. Ketika editor.openFile('artikel.txt') dipanggil, event open akan memicu LoggingListener untuk mencatat aktivitas membuka file. Begitu juga saat editor.saveFile() dipanggil, event save memicu EmailAlertsListener untuk mengirim notifikasi email. Dengan begitu, sistem ini bisa merespons berbagai kejadian secara terpisah dan terorganisir.

### Outputnya

```

PS D:\KULIAH\515E0701\SEM4\PRAKTIKUM KONSTRUKSI PERANGKAT LUNAK\KPL_TRI-MYLANI_2311104001_515E0701> node "d:\KULIAH\515E0701\SEM4\PRAKTIKUM KONSTRUKSI PERANGKAT LUNAK\KPL_TRI-MYLANI_2311104001_515E0701\12_Design Pattern Implementation\TP - 2311104001\tempCodeRunnerFile.js"
[LOG to log.txt]: Someone has opened the file: artikel.txt
File "artikel.txt" saved.
[Email to admin@example.com]: Someone has changed the file: artikel.txt
PS D:\KULIAH\515E0701\SEM4\PRAKTIKUM KONSTRUKSI PERANGKAT LUNAK\KPL_TRI-MYLANI_2311104001_515E0701>

```