

Link Github :

https://github.com/MartryatusSofia/KPL_Martryatus-Sofia_2311104003_SE0701/tree/master/13_Design_Pattern/Jurnal_modul13

1. Berikan salah dua contoh kondisi dimana design pattern “Singleton” dapat digunakan.

Jawab :

1. Koneksi ke Database

Jika aplikasi hanya memerlukan satu koneksi global ke database, maka pattern Singleton cocok untuk memastikan hanya satu instance dari objek koneksi yang dibuat.

2. Logger (Pencatat Log Aplikasi)

Logger sering digunakan di berbagai bagian aplikasi. Dengan Singleton, semua bagian aplikasi dapat menggunakan satu instance logger yang sama untuk konsistensi.

1. Berikan penjelasan singkat mengenai langkah-langkah dalam mengimplementasikan design pattern “Singleton”.

Jawab :

1. Buat class dengan konstruktor private, agar tidak bisa diinstansiasi dari luar class.

2. Buat atribut statis (misalnya `_instance`) di dalam class untuk menyimpan satu-satunya instance dari class tersebut.

3. Buat method statis (misalnya `getInstance()`) untuk mengakses instance. Jika instance belum ada, method ini akan membuatnya; jika sudah ada, akan mengembalikan instance yang sama.

2. Berikan tiga kelebihan dan kekurangan dari design pattern “Singleton”.

menerima dan menampilkan pesan bahwa mereka menerima update tersebut.

Jawab :

Kelebihan:

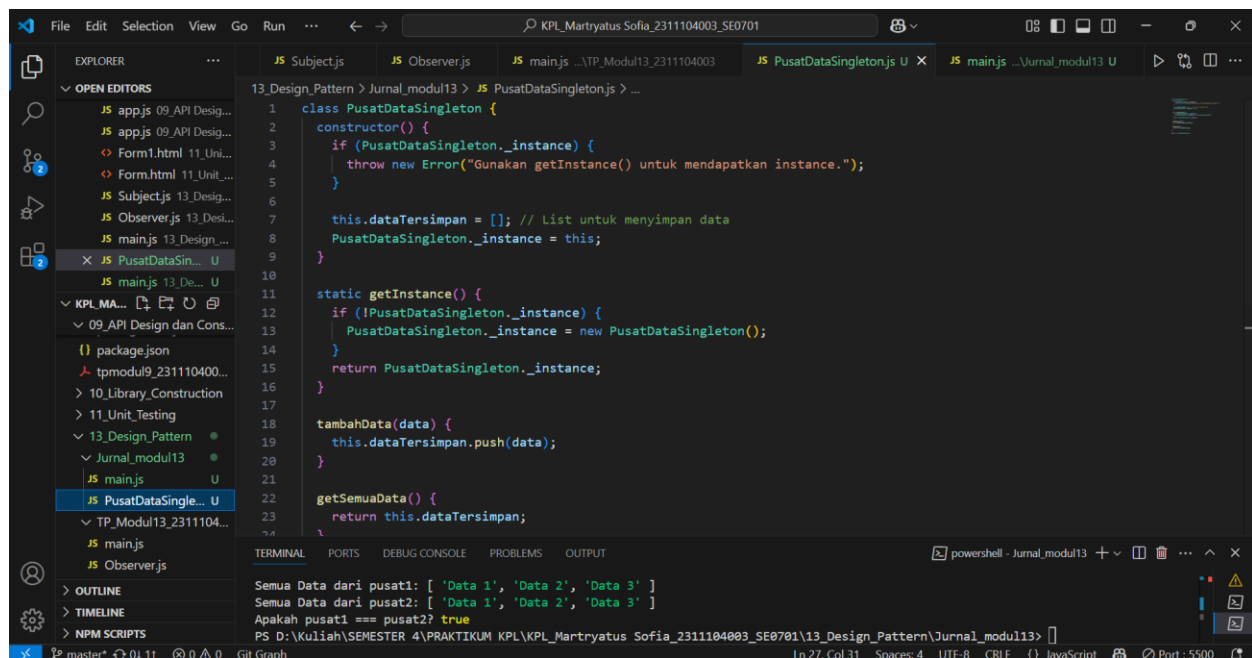
1. Kontrol Akses Terpusat – Semua akses ke resource tertentu dikelola oleh satu instance saja.
2. Mengurangi Penggunaan Memori – Tidak perlu membuat banyak objek serupa.

3. Mudah Diterapkan dan Dipanggil – Bisa langsung dipanggil dengan cara statis tanpa harus membuat objek baru.

Kekurangan:

1. Sulit untuk Unit Testing – Karena instance tunggal bersifat global dan bisa mempengaruhi state antar tes.
2. Melanggar Prinsip Single Responsibility – Class bisa menjadi terlalu kompleks karena mengelola instance sendiri.
3. Kurang Fleksibel dalam Multithreading – Di lingkungan paralel, Singleton butuh pengamanan ekstra untuk menghindari duplikasi.

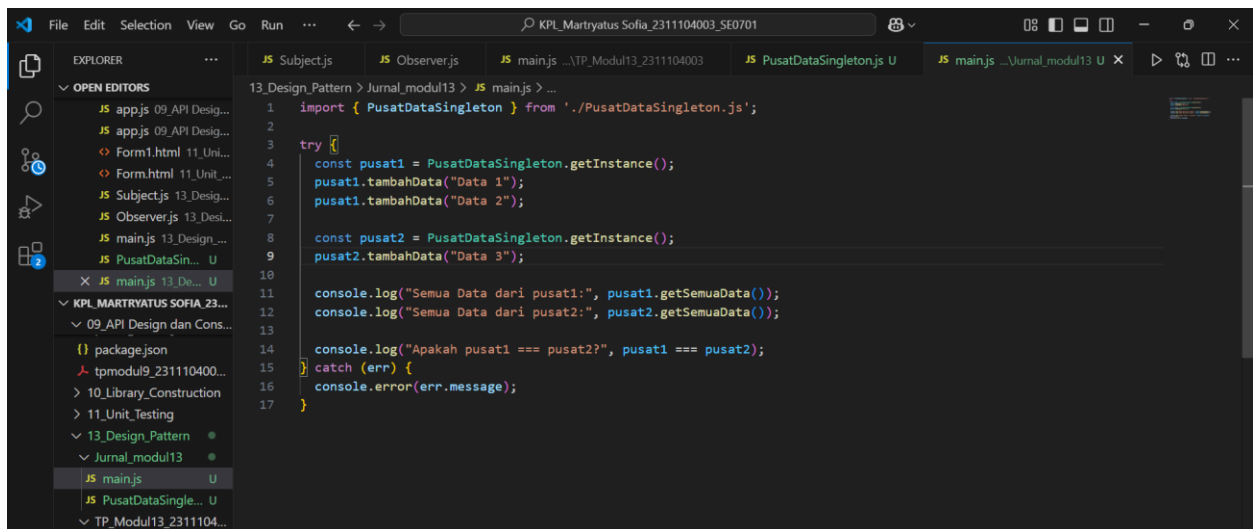
Code Program :



```
1 class PusatDataSingleton {
2   constructor() {
3     if (PusatDataSingleton._instance) {
4       throw new Error("Gunakan getInstance() untuk mendapatkan instance.");
5     }
6
7     this.dataTersimpan = []; // List untuk menyimpan data
8     PusatDataSingleton._instance = this;
9   }
10
11   static getInstance() {
12     if (!PusatDataSingleton._instance) {
13       PusatDataSingleton._instance = new PusatDataSingleton();
14     }
15     return PusatDataSingleton._instance;
16   }
17
18   tambahData(data) {
19     this.dataTersimpan.push(data);
20   }
21
22   getSemuaData() {
23     return this.dataTersimpan;
24   }
25 }
```

Terminal Output:

```
Semua Data dari pusat1: [ 'Data 1', 'Data 2', 'Data 3' ]
Semua Data dari pusat2: [ 'Data 1', 'Data 2', 'Data 3' ]
Apakah pusat1 === pusat2? true
PS D:\Kuliah\SEMESTER 4\PRAKTIKUM KPL\KPL_Martryatus Sofia_2311104003_SE0701\13_Design_Pattern\Jurnal_modul13>
```



```
import { PusatDataSingleton } from './PusatDataSingleton.js';
```

Import class Singleton PusatDataSingleton dari file eksternal agar bisa digunakan di file ini.

```
try {
  const pusat1 = PusatDataSingleton.getInstance();
```

Memanggil method statis getInstance() untuk mendapatkan instance pertama dari Singleton. Jika belum ada instance sebelumnya, maka objek baru akan dibuat.

```
pusat1.tambahData("Data 1");
pusat1.tambahData("Data 2");
```

Menambahkan dua data ke pusat1. Karena ini Singleton, data ini disimpan di satu-satunya instance yang tersedia.

```
const pusat2 = PusatDataSingleton.getInstance();
pusat2.tambahData("Data 3");
```

Mendapatkan lagi instance PusatDataSingleton. Namun karena Singleton, yang dikembalikan adalah instance yang sama dengan pusat1. Maka data "Data 3" juga ditambahkan ke instance yang sama.

```
console.log("Semua Data dari pusat1:", pusat1.getSemuaData());  
console.log("Semua Data dari pusat2:", pusat2.getSemuaData());
```

Menampilkan semua data yang telah ditambahkan dari kedua variabel (pusat1 dan pusat2). Hasilnya akan sama, karena keduanya menunjuk ke instance yang sama.

```
console.log("Apakah pusat1 === pusat2?", pusat1 === pusat2);
```

Mengecek apakah pusat1 dan pusat2 benar-benar merujuk pada objek yang sama di memori. Hasilnya pasti true.

```
} catch (err) {  
    console.error(err.message);  
}
```

Penanganan error jika terjadi kesalahan saat membuat/mengakses instance.

Hasil Running

```
PS D:\Kuliah\SEMESTER 4\PRAKTIKUM KPL\KPL_Martryatus Sofia_2311104003_SE0701\13_Design_Pattern\Jurnal_modul13> node main.js  
PS D:\Kuliah\SEMESTER 4\PRAKTIKUM KPL\KPL_Martryatus Sofia_2311104003_SE0701> cd .\13_Design_Pattern\  
PS D:\Kuliah\SEMESTER 4\PRAKTIKUM KPL\KPL_Martryatus Sofia_2311104003_SE0701\13_Design_Pattern> cd .\Jurnal_modul13\  
PS D:\Kuliah\SEMESTER 4\PRAKTIKUM KPL\KPL_Martryatus Sofia_2311104003_SE0701\13_Design_Pattern\Jurnal_modul13> node main.js  
PS D:\Kuliah\SEMESTER 4\PRAKTIKUM KPL\KPL_Martryatus Sofia_2311104003_SE0701\13_Design_Pattern\Jurnal_modul13> node main.js  
Semua Data dari pusat1: [ 'Data 1', 'Data 2', 'Data 3' ]  
Semua Data dari pusat2: [ 'Data 1', 'Data 2', 'Data 3' ]  
Apakah pusat1 === pusat2? true  
PS D:\Kuliah\SEMESTER 4\PRAKTIKUM KPL\KPL_Martryatus Sofia_2311104003_SE0701\13_Design_Pattern\Jurnal_modul13> 
```