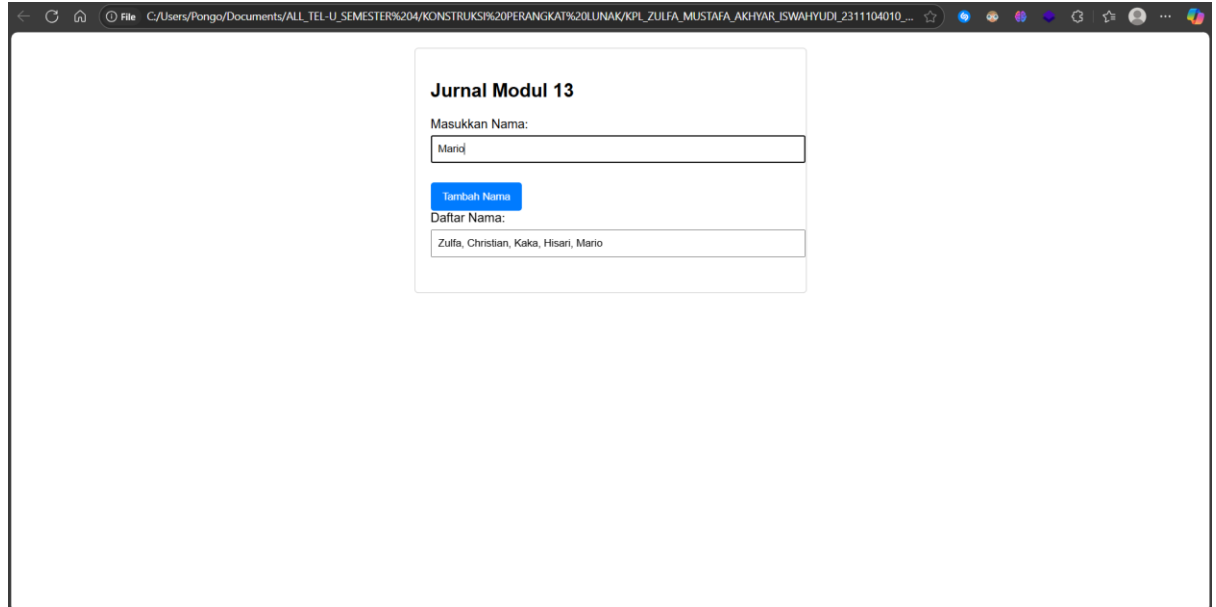


- **Dua contoh kondisi dimana design pattern "Singleton" dapat digunakan:**
  1. Ketika kita memerlukan hanya satu instance dari suatu kelas dan ingin memastikan bahwa kelas tersebut memiliki satu dan hanya satu instance.
  2. Ketika akses ke instance kelas harus terkontrol melalui satu titik akses tunggal.
- **Langkah-langkah dalam mengimplementasikan design pattern "Singleton":**
  1. Buatlah kelas dengan konstruktor yang menginisialisasi atribut-atribut yang diperlukan.
  2. Buatlah atribut statis `_instance` yang akan menyimpan instance tunggal dari kelas tersebut.
  3. Buatlah metode statis `getInstance()` yang akan mengembalikan instance dari kelas. Jika `_instance` belum dibuat, metode ini akan membuat instance baru.
  4. Batasi akses ke konstruktor dengan menjadikannya privat sehingga tidak dapat diakses dari luar kelas.
- **Tiga kelebihan dan kekurangan dari design pattern "Singleton":**
  - Kelebihan:
    1. Mengontrol akses ke instance tunggal dari suatu kelas.
    2. Memastikan bahwa hanya satu instance yang ada dan menyediakan akses global ke instance tersebut.
  - Kekurangan:
    1. Bisa menyebabkan masalah dalam pengujian karena instance tunggal dapat menyimpan state antar tes.
    2. Dapat menyulitkan dalam pengembangan jika tidak diatur dengan baik.

## Kasus Implementasi:

### 1.) Program Input nama dalam satu section

Program diatur menggunakan script.js yang dimana class atau method dijalankan berdasarkan satu objek yang berjalan secara global. Artinya penampungan data beserta tampilan datanya bersifat kolektif seperti contoh berikut:



The screenshot shows a web browser window with a file path in the address bar. The main content is a form titled "Jurnal Modul 13". Inside the form, there is a label "Masukkan Nama:" above a text input field. The input field contains the text "Mario". Below the input field is a blue button labeled "Tambah Nama". Underneath the button is a label "Daftar Nama:" followed by a text area that lists the names "Zulfa, Christian, Kaka, Hisari, Mario".

## TP

Observer Pattern cocok digunakan dalam sistem **notifikasi**. Misalnya, dalam aplikasi cuaca: ketika data cuaca berubah, maka semua tampilan (widget, aplikasi, notifikasi) yang tergantung pada data tersebut harus diperbarui secara otomatis.

### Langkah-langkah implementasi Observer:

1. Buat **Subject** (yang diamati) yang menyimpan daftar observer.
2. Subject menyediakan method `attach()`, `detach()`, dan `notify()`.
3. Buat **Observer** (yang mengamati), yaitu objek yang akan diberi tahu saat ada perubahan di subject.
4. Saat subject berubah, ia akan memanggil `notify()` dan semua observer akan dipanggil.

### C. Kelebihan dan Kekurangan:

#### Kelebihan:

- Memisahkan subject dan observer → meningkatkan modularitas.
- Mempermudah sistem untuk bertambah fitur notifikasi tanpa mengubah subject.

#### Kekurangan:

- Bisa menjadi **kompleks** jika terlalu banyak observer.
- Tidak menjamin urutan notifikasi.
- Sulit di-debug jika terjadi **chain reaction** dari notifikasi.

Contoh Implementasi:

Kode yang diimplementasikan menggunakan bahasa JavaScript ini menerapkan pola perancangan Observer Pattern, yaitu sebuah pola yang memungkinkan suatu objek (disebut **subject**) untuk memberi tahu sekumpulan objek lain (disebut **observers**) apabila terjadi perubahan pada dirinya. Dalam konteks kode ini, objek yang berperan sebagai **subject** adalah WeatherStation, sementara TemperatureDisplay dan TemperatureLogger merupakan **observers** yang akan merespons setiap kali suhu berubah.

Pertama-tama, WeatherStation menyimpan daftar observer melalui array observers. Setiap observer bisa ditambahkan menggunakan method attach() dan dihapus melalui method detach(). Ketika suhu diperbarui dengan setTemperature(), maka method notify() akan dipanggil untuk memberitahu semua observer bahwa terjadi perubahan suhu. Masing-masing observer memiliki method update() yang akan dipanggil secara otomatis oleh WeatherStation saat terjadi perubahan.

Dalam simulasi program, dibuat instance dari WeatherStation, lalu dua observer (TemperatureDisplay dan TemperatureLogger) ditambahkan ke dalamnya. Ketika suhu diubah menjadi 25°C dan kemudian 30°C, kedua observer secara otomatis mencetak pesan ke konsol: satu untuk menampilkan suhu saat ini, dan satu lagi untuk mencatatnya sebagai log. Proses ini menggambarkan bagaimana perubahan dalam subject dapat dipantau dan direspons secara otomatis oleh banyak observer tanpa saling bergantung secara langsung.

Implementasi ini menunjukkan kekuatan Observer Pattern dalam menjaga pemisahan tanggung jawab dan meningkatkan modularitas kode, yang sangat berguna dalam sistem yang dinamis dan interaktif seperti notifikasi, UI, atau sistem pemantauan.

```
▼ KONSTRUKSI PERANGKAT LUNAK
> .qodo
▼ KPL_ZULFA_MUSTAF...
> 01_Pengantar_Praktikum...
> 02_Pengenalan_IDE_dan_...
> 03_HTML
> 04_OOP
> 05_Generics
> 06_DBC
> 07_Parsing
> 08_Runtime_Configuratio...
> 09_API Design dan Const...
> 10_Library
> 11
> 12
▼ 13
  ▼ Jurnal
    </> index.html U
    JS script.js U
  ▼ TP
    JS observer.js U
  .gitignore
  .hintc

KPL_ZULFA_MUSTAFA_AKHYAR_ISWAHYUDI_2311104010_SE0701 > 13 > TP > JS observer.js > ...
1 // Subject
2 class WeatherStation {
3   constructor() {
4     this.observers = [];
5     this.temperature = 0;
6   }
7
8   attach(observer) {
9     this.observers.push(observer);
10  }
11
12  detach(observer) {
13    this.observers = this.observers.filter(obs => obs !== observer);
14  }
15
16  notify() {
17    this.observers.forEach(observer => observer.update(this.temperature));
18  }
19
20  setTemperature(temp) {
21    console.log(`Suhu berubah menjadi ${temp}°C`);
22    this.temperature = temp;
23    this.notify();
24  }
25 }
26
27 // Observer
28 class TemperatureDisplay {
29   update(temp) {
30     console.log(`[Display] Suhu saat ini: ${temp}°C`);
31   }
32 }
33
34 class TemperatureLogger {
35   update(temp) {
36     console.log(`[Logger] Mencatat suhu: ${temp}°C`);
37   }
38 }
39
40 // Main method simulasi
41 const station = new WeatherStation();
42 const display = new TemperatureDisplay();
43 const logger = new TemperatureLogger();
44
45 // Menambahkan observer
46 station.attach(display);
47 station.attach(logger);
48
49 // Mengubah suhu
50 station.setTemperature(25);
51 station.setTemperature(30);
52
```

```
PS C:\Users\Pongo\Documents\ALL_TEL-U_SEMESTER 4\KONSTRUKSI PERANGKAT LUNAK\KPL_ZULFA_MUSTAFA_AKHYAR_ISWAHYUDI_2311104010_SE0701\13\TP> node observer
Suhu berubah menjadi 25°C
[Display] Suhu saat ini: 25°C
[Logger] Mencatat suhu: 25°C
Suhu berubah menjadi 30°C
[Display] Suhu saat ini: 30°C
PS C:\Users\Pongo\Documents\ALL_TEL-U_SEMESTER 4\KONSTRUKSI PERANGKAT LUNAK\KPL_ZULFA_MUSTAFA_AKHYAR_ISWAHYUDI_2311104010_SE0701\13\TP> node observer
Suhu berubah menjadi 25°C
[Display] Suhu saat ini: 25°C
[Logger] Mencatat suhu: 25°C
Suhu berubah menjadi 30°C
[Display] Suhu saat ini: 25°C
[Logger] Mencatat suhu: 25°C
Suhu berubah menjadi 30°C
[Display] Suhu saat ini: 30°C
Suhu berubah menjadi 30°C
[Display] Suhu saat ini: 30°C
[Logger] Mencatat suhu: 30°C
```