Observer Design Pattern  
Observer Design Pattern termasuk dalam kategori *Behavioral Design Pattern* dan berfungsi untuk menciptakan sistem di mana satu objek utama (disebut *Subject*) bisa memberi tahu objek-objek lainnya (*Observers*) ketika terjadi perubahan pada dirinya.

Pola ini juga sering disebut sebagai pola *Publisher-Subscriber*. Dalam konteks ini, *Subject* berperan sebagai “penerbit” informasi, sedangkan *Observer* sebagai “pelanggan” yang menerima informasi tersebut. Ketika ada perubahan pada *Subject*, semua *Observer* yang telah berlangganan akan langsung mendapatkan notifikasi dan dapat merespons perubahan itu tanpa *Subject* perlu tahu detail dari masing-masing *Observer*.

Penggunaan Observer Pattern :

1. **Event Handling di Antarmuka Pengguna (UI)**  
   Contoh paling umum adalah saat pengguna menekan tombol di aplikasi. Tombol tersebut akan memicu *event listener* yang kemudian menjalankan fungsi tertentu. Dengan Observer Pattern, logika tampilan bisa dipisah dari logika aksi yang terjadi, membuat kode lebih terstruktur.
2. **Sistem Realtime atau Live Update**  
   Misalnya dalam aplikasi dashboard atau sistem pemantauan data real-time. Ketika ada data baru yang masuk atau berubah di pusat, secara otomatis semua tampilan yang terhubung (observer) akan memperbarui diri tanpa harus disambung satu per satu secara manual.

**Kelebihan:**

1. **Kopling Longgar (Loose Coupling)**  
   Observer dan Subject tidak saling tergantung secara langsung. Subject hanya tahu bahwa ada yang perlu diberi tahu, tapi tidak peduli bagaimana cara observer memproses perubahan tersebut.
2. **Modular dan Mudah Dikembangkan**  
   Observer bisa ditambah atau dihapus kapan saja tanpa perlu mengubah isi dari kelas Subject. Cocok untuk aplikasi yang akan berkembang terus.
3. **Responsif terhadap Perubahan**  
   Perubahan data langsung bisa dirasakan oleh bagian lain dari aplikasi. Sangat ideal untuk sistem real-time atau yang butuh update dinamis.
4. **Mendukung Prinsip Open/Closed**  
   Kelas Subject tidak perlu diubah setiap kali ingin menambahkan perilaku baru; cukup dengan menambah observer baru saja.

**Kekurangan:**

1. **Sulit Dilacak**  
   Karena proses notifikasi berjalan secara otomatis, cukup sulit untuk melacak siapa saja yang mendapat notifikasi saat debugging.
2. **Beban Tambahan (Overhead)**  
   Jika jumlah observer sangat banyak, proses memberi tahu semua observer bisa menjadi berat dan mempengaruhi performa.
3. **Hubungan Antar Objek Tidak Terlihat Jelas**  
   Karena subject tidak tahu secara eksplisit siapa saja observer-nya, kadang sulit memahami aliran data di sistem.
4. **Potensi Memory Leak**  
   Jika lupa menghapus observer yang tidak dibutuhkan lagi, objek tersebut akan tetap menerima notifikasi dan bisa membuat penggunaan memori tidak efisien.

ObserverPattern.js



