2311104025
Eduardo Bagus Prima Julian
SE0701

Subject.js

```
class Subject {
  constructor() {
    this._observers = [];
  }

  /**
   * Menambahkan observer ke daftar pengamat.
   * @param {Observer} observer
   */
  attach(observer) {
    this._observers.push(observer);
  }

  /**
   * Menghapus observer dari daftar pengamat.
   * @param {Observer} observer
   */
  detach(observer) {
    this._observers = this._observers.filter(obs => obs !== observer);
  }

  /**
   * Memberi notifikasi ke semua observer yang terdaftar.
   * @param {any} data Data yang dikirim ke observer
   */
  notify(data) {
    this._observers.forEach(observer => {
      observer.update(data);
    });
  }
}

module.exports = Subject;
```

Penjelasan:

Kode kelas Subject ini mengimplementasikan pola desain Observer di mana sebuah objek Subject menyimpan daftar pengamat (_observers) yang akan diberi notifikasi ketika terjadi perubahan data. Metode attach digunakan untuk menambahkan observer baru ke dalam daftar, sedangkan detach menghapus observer yang sudah tidak ingin menerima pembaruan. Metode notify kemudian memanggil fungsi update pada setiap observer yang terdaftar, mengirimkan data terbaru sebagai parameter, sehingga observer bisa merespon perubahan tersebut secara real-time. Pendekatan ini memungkinkan pemisahan tanggung jawab antara sumber data dan objek yang memantau perubahan, sehingga kode lebih modular dan mudah dikembangkan.

Observer.js

```
JS Observer.js > Observer
1   class Observer {
2     /**
3      * Membuat instance Observer dengan nama tertentu.
4      * @param {string} name Nama observer
5      */
6     constructor(name) {
7       this.name = name;
8     }
9
10    /**
11     * Fungsi callback saat menerima update.
12     * @param {any} data Data yang diterima dari subject
13     */
14    update(data) {
15      console.log(`${this.name} menerima update: ${data}`);
16    }
17  }
18
19  module.exports = Observer;
```

Penjelasan:

Kelas Observer merepresentasikan objek pengamat yang menerima informasi atau notifikasi dari objek Subject. Setiap instance Observer memiliki properti name sebagai identitasnya, yang diinisialisasi melalui konstruktor. Metode update berfungsi sebagai callback yang dipanggil oleh Subject saat ada perubahan atau data baru yang ingin disampaikan, dan dalam implementasi ini, metode tersebut hanya menampilkan pesan ke konsol dengan menyertakan nama observer dan data yang diterima. Dengan struktur ini, setiap observer bisa merespons perubahan sesuai kebutuhan saat diberi tahu oleh subject, mendukung pola desain Observer yang memisahkan sumber data dan objek yang memantau perubahan.

Main.js

```
JS main.js > ...
1   const Subject = require('./Subject');
2   const Observer = require('./Observer');
3
4   // Membuat instance Subject (objek yang diamati)
5   const dataCenter = new Subject();
6
7   // Membuat beberapa observer (pengamat)
8   const observerA = new Observer("Observer A");
9   const observerB = new Observer("Observer B");
10  const observerC = new Observer("Observer C");
11
12  // Menambahkan observer ke subject
13  dataCenter.attach(observerA);
14  dataCenter.attach(observerB);
15  dataCenter.attach(observerC);
16
17  // Mengirim notifikasi ke semua observer
18  dataCenter.notify("Data telah diperbarui!");
19
20  // Menghapus salah satu observer
21  dataCenter.detach(observerB);
22
23  // Mengirim notifikasi kedua kali
24  dataCenter.notify("Update kedua, Observer B sudah tidak menerima.");
```

Penjelasan:

Kode di atas menunjukkan penggunaan pola desain Observer dalam praktek. Pertama, dibuat sebuah instance Subject bernama dataCenter yang berfungsi sebagai sumber datayang diamati. Selanjutnya, tiga instance Observer dibuat dengan nama berbeda:observerA, observerB, dan observerC. Ketiga observer tersebut kemudian didaftarkan (attach) ke data Center, sehingga mereka akan menerima notifikasi setiap kali dataCenter mengirimkan update melalui metode notify. Saat notify dipanggil dengan pesan "Data telah diperbarui!", ketiga observer menerima dan menampilkan pesan tersebut. Kemudian, observerB dilepas (detach) dari dataCenter, sehingga pada notifikasi kedua yang dikirim dengan pesan berbeda, hanya observerA dan observerC yang menerima pemberitahuan, sedangkan observerB tidak lagi mendapatkan update. Ini memperlihatkan fleksibilitas pola Observer dalam menambah dan menghapus pengamat secara dinamis.

Output:

```
PS D:\KPL\KPL_Eduardo Bagus Prima Julian_2311104025_SE0701\14_Clean_Code\TP> node main.js
Observer A menerima update: Data telah diperbarui!
Observer B menerima update: Data telah diperbarui!
Observer C menerima update: Data telah diperbarui!
Observer A menerima update: Update kedua, Observer B sudah tidak menerima.
Observer C menerima update: Update kedua, Observer B sudah tidak menerima.
```

Perbedaan dengan sebelumnya:

Kelas Subject berfungsi sebagai objek yang mengelola daftar observer yang terdaftar untuk menerima pembaruan. Dengan metode attach, observer dapat didaftarkan, dan dengan detach, observer dapat dihapus dari daftar. Ketika ada data baru yang perlu disampaikan, metode notify akan memanggil metode update pada setiap observer yang terdaftar, sehingga semua observer menerima notifikasi secara real-time. Pada sisi lain, kelas Observer merepresentasikan objek pengamat yang memiliki identitas melalui properti name. Metode update pada observer berfungsi sebagai callback yang mengeksekusi aksi ketika menerima notifikasi, dalam contoh ini hanya menampilkan pesan ke konsol. Pada implementasi utama (main.js), sebuah instance Subject bernama dataCenter dibuat, lalu beberapa observer (observerA, observerB, dan observerC) dibuat dan didaftarkan ke dataCenter. Ketika dataCenter.notify() dipanggil, semua observer menerima pesan. Jika salah satu observer dihapus menggunakan detach, maka observer tersebut tidak akan lagi menerima notifikasi berikutnya. Pendekatan ini mengikuti pola desain Observer yang memungkinkan pemisahan antara sumber data dan objek pengamat, serta mendukung komunikasi yang fleksibel dan dinamis antar objek.