Jurnal Modul 13

1. **MENJELASKAN DESIGN PATTERN SINGLETON**
   - **A. Berikan salah dua contoh kondisi dimana design pattern "Singleton" dapat digunakan.**
     a. Pengelolaan Koneksi ke Database
        i. Dalam sebuah aplikasi, biasanya hanya diperlukan satu objek koneksi database yang digunakan bersama oleh berbagai bagian aplikasi untuk menghindari duplikasi koneksi yang berlebihan dan memastikan konsistensi data.

     b. Logger Aplikasi
        i. Logger bertugas mencatat log aktivitas sistem. Biasanya aplikasi hanya membutuhkan satu instance logger yang digunakan bersama di seluruh bagian aplikasi, agar semua catatan log tersimpan dalam format dan tempat yang konsisten.
   - **B. Berikan penjelasan singkat mengenai langkah-langkah dalam** mengimplementasikan design pattern "Singleton".
     a. Membuat Konstruktor Kelas Private
        i. Agar objek tidak bisa dibuat dari luar kelas.

     b. Membuat Static Method untuk Mengakses Instance
        i. Method ini bertugas untuk memeriksa apakah instance sudah ada. Kalau belum, dia akan membuatnya; kalau sudah, cukup mengembalikan instance yang ada.

     c. Menyimpan Instance dalam Properti Static
        i. Properti ini menyimpan instance tunggal dari kelas tersebut.

     d. Contoh sederhana dalam pseudocode:

```
class Singleton {
  private static Singleton instance;

  private Singleton() { }

  public static Singleton getInstance() {
    if (instance == null) {
      instance = new Singleton();
    }
    return instance;
  }
}
```

C. Berikan tiga kelebihan dan kekurangan dari design pattern "Singleton".

Kelebihan:

1. **Mengontrol Akses ke Instance Tunggal**
    - Memastikan hanya satu instance dari kelas yang dibuat, sehingga bisa mengontrol penggunaan resource penting.
2. **Global Access Point**
    - Instance bisa diakses dari mana saja dalam program melalui method static.
3. **Mendukung Inisialisasi Secara Lazy**
    - Objek baru dibuat hanya saat dibutuhkan, bukan saat aplikasi pertama dijalankan.

Kekurangan:

1. **Menyulitkan Pengujian Unit**
    - Karena bersifat global dan state-nya bisa berubah-ubah antar test case, menyulitkan pengujian terisolasi.
2. **Membatasi Kemampuan Ekstensi**
    - Karena konstruktor private, kelas ini tidak bisa di-subclass, membatasi fleksibilitas kode.
3. **Potensi Masalah dalam Aplikasi Multithread**
    - Jika tidak ditangani dengan hati-hati (misalnya tanpa sinkronisasi), dua thread bisa saja membuat instance bersamaan.

2. Implementasi Jurnal

```
PS C:\DEVELOPMENT\Java Script> &
workspaceStorage\e8342bcbeb4af108a
Daftar Data:
1. Tsaqif Hisyam
2. Christian Felix
3. Zulfa Mustafa
4. Asisten: Kaka

Daftar Data:
1. Tsaqif Hisyam
2. Christian Felix
3. Zulfa Mustafa

Jumlah data di data1: 3
Jumlah data di data2: 3
PS C:\DEVELOPMENT\Java Script>
```

Di atas adalah hasil output dari percobaan Singleton yang terdiri dari class Class PusatDataSingleton menggunakan design pattern Singleton dengan properti _instance untuk memastikan hanya satu objek yang dibuat. Atribut DataTersimpan berupa list string digunakan untuk menyimpan data, sementara konstruktor bersifat private dan hanya bisa diakses melalui method GetDataSingleton(). Data dapat ditambahkan menggunakan AddSebuahData(), dihapus dengan HapusSebuahData(), ditampilkan melalui PrintSemuaData(), dan diambil seluruhnya lewat GetSemuaData(). Karena menerapkan Singleton, variabel data1 dan data2 dalam program utama sebenarnya mengacu pada objek yang sama, sehingga perubahan data melalui salah satunya akan langsung terlihat di yang lain.

Tugas Pendahuluan Modul 13

1. **MENJELASKAN SALAH SATU DESIGN PATTERN**
   A. **Berikan salah satu contoh kondisi dimana design pattern "Observer" dapat digunakan**

      Salah satu contoh penerapan Observer adalah dalam aplikasi notifikasi cuaca. Ketika data cuaca (suhu, kelembaban, tekanan udara) diperbarui oleh pusat pengendali, berbagai komponen seperti tampilan display, aplikasi mobile, atau papan pengumuman otomatis akan menerima notifikasi pembaruan data secara langsung tanpa harus terus-menerus mengecek secara manual.

   B. **Berikan penjelasan singkat mengenai langkah-langkah dalam mengimplementasikan design pattern "Observer"**

      1. **Buat Interface Observer**
         - Interface ini berisi method `update()` yang akan dipanggil saat data di Subject berubah.
      2. **Buat Interface Subject**
         - Berisi method untuk `attach()`, `detach()`, dan `notifyObservers()` kepada semua Observer yang terdaftar.
      3. **Implementasikan Concrete Subject**
         - Class ini menyimpan state/data yang diamati dan daftar Observer, serta mengatur method untuk menambah/menghapus observer dan memberi notifikasi saat state berubah.
      4. **Implementasikan Concrete Observer**
         - Class ini berisi implementasi dari `update()` dan menentukan apa yang dilakukan saat menerima notifikasi perubahan dari Subject.
      5. **Buat Program Utama**
         - Menghubungkan observer dengan subject, mengubah state subject, dan mengamati bagaimana observer otomatis mendapat notifikasi.

   C. **Berikan kelebihan dan kekurangan dari design pattern "Observer"**

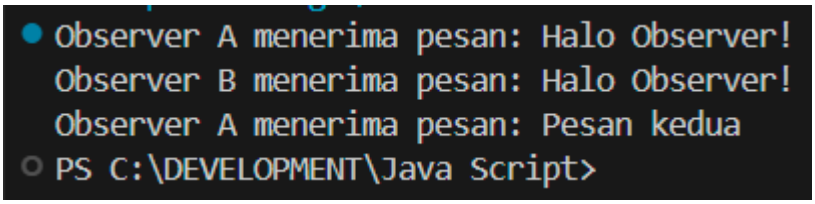
      Kelebihan:

      1. **Loose Coupling (Keterkaitan Longgar)**
         - Subject dan Observer tidak perlu saling mengetahui implementasi detail masing-masing.
      2. **Mendukung Sistem Dinamis**
         - Observer bisa ditambahkan atau dihapus kapan saja tanpa mengubah kode subject.
      3. **Efisien dalam Penyebaran Notifikasi**
         - Perubahan data langsung diberitahukan ke semua observer yang terdaftar.

      Kekurangan:

1. **Sulit Ditelusuri Jika Observer Banyak**
    - Saat jumlah observer bertambah, bisa menyulitkan debugging dan pemantauan alur program.
2. **Potensi Masalah Performansi**
    - Jika notifikasi harus dikirim ke banyak observer sekaligus, bisa berdampak pada performa.
3. **Resiko Memory Leak**
    - Jika observer tidak dilepas/didetach dengan benar, bisa menyebabkan memory leak.

2. **Implementasi Tugas Pendahuluan**



Di atas adalah output dari implementasi Observer.
Penjelasan Tiap Baris di Main(ProgramObserver.java)

a. **Baris 1: Membuat objek ConcreteSubject** untuk menyimpan state/message yang akan diamati. **Baris 2-4**: **Membuat dua observer (observer1, observer2)** lalu mendaftarkannya ke subject menggunakan attach().
b. **Baris 5: Mengubah state message subject** via setMessage(). Ini akan memanggil notifyObservers() dan setiap observer yang terdaftar akan menerima notifikasi lewat method update().
c. **Baris 6: Menghapus observer2 dari daftar observer** menggunakan detach().
d. **Baris 7: Mengubah pesan lagi**, sekarang hanya observer1 yang akan menerima pesan karena observer2 sudah dihapus.