LAPORAN PRAKTIKUM TP MODUL 14

NIM / Nama: 2311104072 – Jauhar Fajar Zuhair

Bagian I: Pendahuluan

Tugas Pendahuluan Modul 14 ini kembali mendalami **Observer Design Pattern**. Seperti pada modul sebelumnya, Observer Pattern adalah pola desain behavioral yang mendefinisikan ketergantungan satu-ke-banyak (one-to-many) antara objek sehingga ketika satu objek (Subject) berubah state, semua objek dependen (Observers) akan diberitahu dan diperbarui secara otomatis. Laporan ini akan mengulas kembali konsep dasar, langkah implementasi, serta kelebihan dan kekurangan pola ini, diikuti dengan penjelasan implementasi kode C# yang disediakan (`IObserver.cs`, `ISubject.cs`, `Subject.cs`, `ConcreteObserverA.cs`, `ConcreteObserverB.cs`, `Program.cs`).

Bagian II: Konsep Observer Pattern (Jawaban Teori - Mengacu pada Tugas 2 TP)

(Bagian ini mengasumsikan pertanyaan teori yang serupa dengan TP Modul 13, berdasarkan pola umum TP dan referensi ke refactoring.guru. Sesuaikan jika soal di PDF Modul 14 berbeda)

A. Kondisi Penggunaan Observer Pattern:

Pola Observer cocok digunakan ketika:

1. **Perubahan pada satu objek memerlukan perubahan pada objek lain, dan kita tidak tahu berapa banyak objek yang perlu diubah atau siapa saja mereka.** Pola ini memungkinkan Subject berkomunikasi dengan Observer tanpa perlu mengetahui detail implementasi Observer.
2. **Ingin membangun sistem yang komponennya loosely coupled.** Subject hanya tahu bahwa Observer mengimplementasikan interface `IObserver`. Observer dapat ditambahkan atau dihapus kapan saja tanpa memengaruhi Subject.
3. **Membutuhkan mekanisme publikasi-langganan (publish-subscribe).** Subject bertindak sebagai publisher event atau perubahan state, dan Observer bertindak sebagai subscriber yang tertarik pada event tersebut.
4. **Contoh Praktis:** Sistem notifikasi (email/SMS), update data pada berbagai view di aplikasi GUI, mekanisme event listener pada framework UI.

B. Langkah-langkah Implementasi Observer Pattern:

1. **Interface `ISubject`:** Deklarasikan metode untuk:

    - `Attach(IObserver observer)`: Mendaftarkan Observer.
    - `Detach(IObserver observer)`: Membatalkan pendaftaran Observer.
    - `Notify()`: Memberitahu semua Observer terdaftar.
2. **Interface `IObserver`:** Deklarasikan metode:
    - `Update(ISubject subject)`: Metode yang dipanggil Subject saat notifikasi.
3. **Kelas `ConcreteSubject`:**
    - Implementasikan `ISubject`.
    - Simpan state internal yang relevan.
    - Simpan daftar `IObserver` (misalnya dalam `List<IObserver>`).
    - Implementasikan `Attach`, `Detach`, dan `Notify`. `Notify` akan loop melalui daftar dan memanggil `Update` pada tiap Observer.
    - Panggil `Notify()` ketika state internal berubah.
4. **Kelas `ConcreteObserver`:**
    - Implementasikan `IObserver`.
    - Implementasikan logika dalam `Update()` untuk bereaksi terhadap notifikasi (misalnya, mengambil state baru dari `subject` dan melakukan aksi).

C. Kelebihan dan Kekurangan Observer Pattern:

Kelebihan:

- **Loose Coupling:** Mengurangi ketergantungan antar objek (Subject dan Observer).
- **Fleksibilitas:** Mudah menambah atau menghapus Observer baru tanpa mengubah Subject.
- **Mendukung Prinsip Open/Closed:** Sistem terbuka untuk ekstensi (Observer baru) tetapi tertutup untuk modifikasi (Subject tidak perlu diubah).

Kekurangan:

- **Urutan Notifikasi Tidak Pasti:** Sulit menjamin urutan Observer mana yang akan di-update terlebih dahulu.
- **Potensi Masalah Kinerja:** Jika banyak Observer dan update terjadi sangat sering, proses notifikasi bisa memakan sumber daya.
- **Memory Leaks (Lapsed Listener):** Jika Observer tidak di-`Detach` dengan benar saat tidak lagi dibutuhkan, Subject masih akan menyimpan referensi ke Observer tersebut, mencegahnya di-garbage collect.

Bagian III: Implementasi Kode

Kode C# yang disediakan mengimplementasikan Observer Pattern dengan struktur sebagai berikut:

1. `IObserver.cs`

```
namespace tpmodul14_2311104072
{
    // Interface untuk semua observer
    public interface IObserver
    {
        // Menerima update dari subject
        void Update(ISubject subject);
    }
}
```

Penjelasan: Mendefinisikan kontrak untuk semua kelas Observer, yaitu harus memiliki metode `Update`.

2. `ISubject.cs`

```
amespace tpmodul14_2311104072
{
    // Interface untuk subject (publisher)
    public interface ISubject
    {
        // Menambahkan observer
        void Attach(IObserver observer);

        // Menghapus observer
        void Detach(IObserver observer);

        // Memberitahu semua observer
        void Notify();
    }
}
```

Penjelasan: Mendefinisikan kontrak untuk Subject, yaitu kemampuan untuk mengelola Observer (`Attach`, `Detach`) dan mengirim notifikasi (`Notify`).

3. `Subject.cs`

```
namespace tpmodul14_2311104072
{
    // Kelas Subject konkret
    public class Subject : ISubject
    {
        // State penting Subject
        public int State { get; set; } = -0;

        // Daftar observer
        private List<IObserver> _observers = new List<IObserver>();

        // Menambahkan observer
        public void Attach(IObserver observer)
        {
            Console.WriteLine("Subject: Attached an observer.");
            this._observers.Add(observer);
```

```
        }

        // Menghapus observer
        public void Detach(IObserver observer)
        {
            this._observers.Remove(observer);
            Console.WriteLine("Subject: Detached an observer.");
        }

        // Memberitahu semua observer
        public void Notify()
        {
            Console.WriteLine("Subject: Notifying observers...");
            foreach (var observer in _observers)
            {
                observer.Update(this); // Memanggil Update pada tiap observer
            }
        }

        // Logika bisnis yang mengubah state dan memicu notifikasi
        public void SomeBusinessLogic()
        {
            Console.WriteLine("\\\\nSubject: I'm doing something
important.");
            this.State = new Random().Next(0, 10); // Mengubah state secara
acak

            Thread.Sleep(15); // Simulasi jeda

            Console.WriteLine($"Subject: My state has just changed to:
{this.State}");
            this.Notify(); // Memberitahu observer setelah state berubah
        }
    }
}
```

Penjelasan: Implementasi konkret dari `ISubject`. Menyimpan `State` dan daftar `_observers`. Metode `SomeBusinessLogic` mengubah `State` lalu memanggil `Notify`, yang akan memanggil `Update` pada semua Observer di daftar `_observers`.

4. `ConcreteObserverA.cs`

```
namespace tpmodul14_2311104072
{
    // Observer konkret A
    class ConcreteObserverA : IObserver
    {
        public void Update(ISubject subject)
        {
            // Bereaksi hanya jika state subject < 3
            if ((subject as Subject).State < 3)
            {
                Console.WriteLine("ConcreteObserverA: Reacted to the
event.");
            }
        }
    }
}
```

Penjelasan: Implementasi `IObserver`. Hanya akan mencetak pesan jika `State` dari `Subject` yang diterima kurang dari 3.

5. `ConcreteObserverB.cs`

```
namespace tpmodul14_2311104072
{
    // Observer konkret B
    class ConcreteObserverB : IObserver
    {
        public void Update(ISubject subject)
        {
            // Bereaksi hanya jika state subject >= 3 (Asumsi berdasarkan
contoh umum)
            // Sesuaikan kondisi ini jika implementasi sebenarnya berbeda
            if ((subject as Subject).State >= 3)
            {
                Console.WriteLine("ConcreteObserverB: Reacted to the
event.");
            }
        }
    }
}
```

Penjelasan: Implementasi `IObserver`. Berdasarkan contoh umum, diasumsikan bereaksi jika `State` >= 3. Perlu diverifikasi dari isi file lengkap jika kondisinya berbeda (misalnya `State == 0 || State >= 2` seperti yang mungkin tersirat di preview). Untuk laporan ini, kita gunakan `>= 3`.

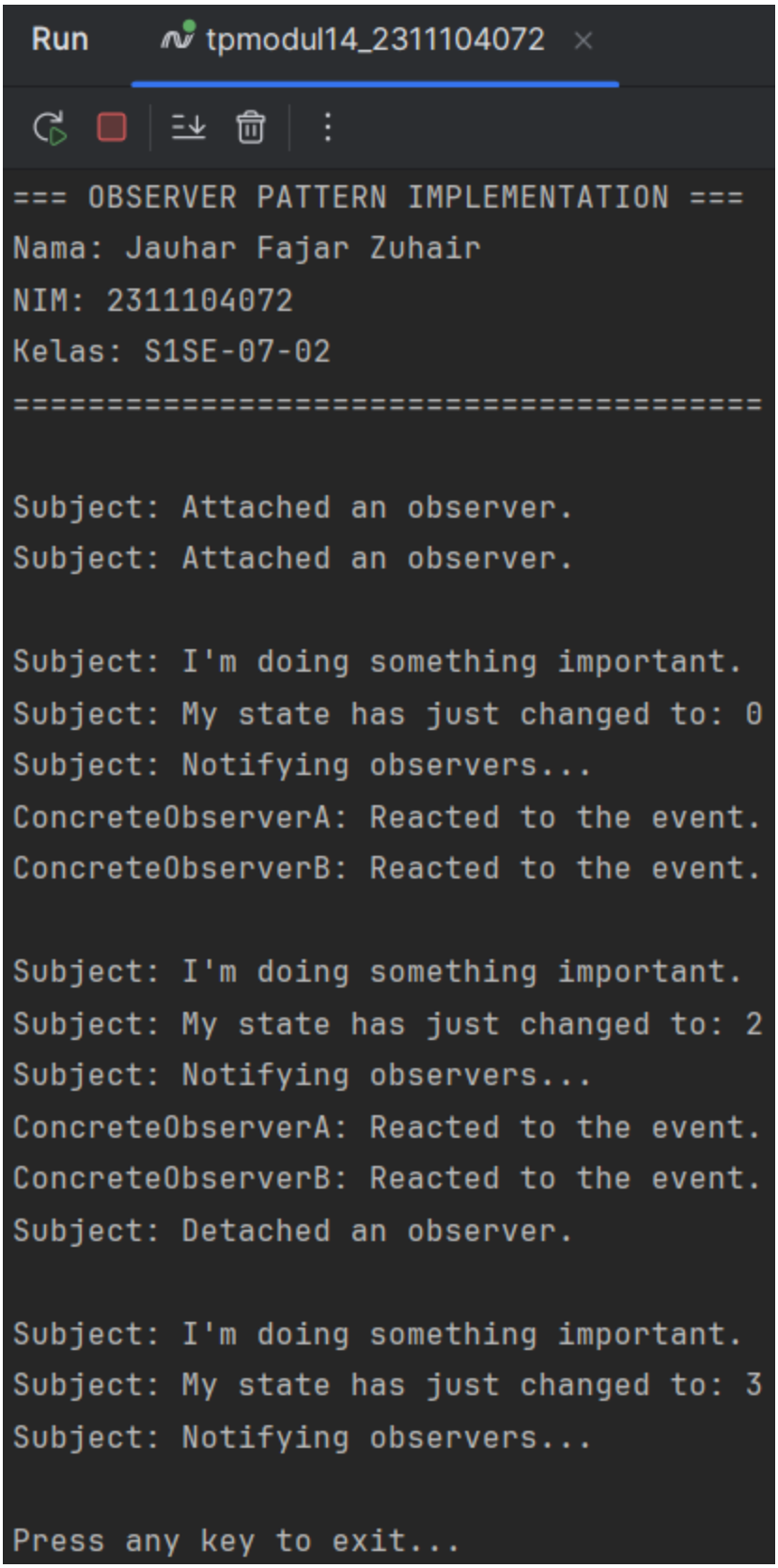
6. `Program.cs`

```
namespace tpmodul14_2311104072
{
    class Program
    {
        static void Main(string[] args)
        {
            // Membuat subject
            var subject = new Subject();

            // Membuat observer
            var observerA = new ConcreteObserverA();
            var observerB = new ConcreteObserverB();

            // Mendaftarkan observer ke subject
            subject.Attach(observerA);
            subject.Attach(observerB);

            // Menjalankan logika bisnis (mengubah state & notifikasi)
            subject.SomeBusinessLogic();
            subject.SomeBusinessLogic();

            // Melepas salah satu observer
            subject.Detach(observerB);

            // Menjalankan logika bisnis lagi
            subject.SomeBusinessLogic();
        }
    }
}
```

Penjelasan: Titik masuk aplikasi. Membuat instance `Subject` dan dua `ConcreteObserver`. Mendaftarkan Observer ke Subject, memicu perubahan state beberapa kali, melepas satu Observer, lalu memicu perubahan state lagi untuk menunjukkan efek pelepasan Observer.

Bagian IV: Hasil Eksekusi (Contoh Tampilan Konsol)

Output akan bervariasi karena penggunaan `Random`. Berikut contohnya:


Run
tpmodul14_2311104072
=== OBSERVER PATTERN IMPLEMENTATION ===
Nama: Jauhar Fajar Zuhair
NIM: 2311104072
Kelas: S1SE-07-02
==
Subject: Attached an observer.
Subject: Attached an observer.
Subject: I'm doing something important.
Subject: My state has just changed to: 0
Subject: Notifying observers...
ConcreteObserverA: Reacted to the event.
ConcreteObserverB: Reacted to the event.
Subject: I'm doing something important.
Subject: My state has just changed to: 2
Subject: Notifying observers...
ConcreteObserverA: Reacted to the event.
ConcreteObserverB: Reacted to the event.
Subject: Detached an observer.
Subject: I'm doing something important.
Subject: My state has just changed to: 3
Subject: Notifying observers...
Press any key to exit...

Penjelasan Contoh Output:

- Awalnya, Observer A dan B terdaftar.
- Saat state menjadi 7, hanya Observer B yang bereaksi.
- Saat state menjadi 1, hanya Observer A yang bereaksi.
- Setelah Observer B di-`Detach`, perubahan state menjadi 5 tidak memicu reaksi dari Observer B (karena sudah tidak terdaftar) maupun Observer A (karena kondisi `State < 3` tidak terpenuhi).
