LAPORAN PRAKTIKUM JURNAL MODUL 14

NIM / Nama: 2311104072 – Jauhar Fajar Zuhair

Bagian I: Pendahuluan

Laporan Jurnal Modul 14 ini mendokumentasikan implementasi dari **Observer Design Pattern** menggunakan bahasa C#. Pola Observer adalah pola desain behavioral yang memungkinkan sebuah objek (disebut Subject atau Publisher) untuk memberi tahu sekumpulan objek dependen (disebut Observer atau Subscriber) secara otomatis ketika state Subject berubah. Praktikum ini melibatkan pembuatan interface `IObserver` dan `ISubject`, kelas konkret `Subject`, serta dua kelas Observer konkret (`ConcreteObserverA` dan `ConcreteObserverB`) untuk mendemonstrasikan mekanisme notifikasi dan update otomatis. Aplikasi konsol (`Program.cs`) digunakan untuk menjalankan simulasi interaksi antara Subject dan Observer.

Bagian II: Implementasi Kode

Implementasi Observer Pattern dibagi menjadi beberapa file:

1. Interface `IObserver` (`IObserver.cs`)

```
namespace jmmodul14_2311104072
{
    public interface IObserver
    {
        // Metode yang dipanggil oleh Subject saat ada update
        void Update(ISubject subject);
    }
}
```

- **Tujuan:** Mendefinisikan kontrak standar untuk semua kelas yang ingin bertindak sebagai Observer. Setiap Observer harus bisa menerima pembaruan dari Subject melalui metode `Update`.

2. Interface `ISubject` (`ISubject.cs`)

```
namespace jmmodul14_2311104072
{
    public interface ISubject
    {
        // Mendaftarkan (attach) observer
        void Attach(IObserver observer);
```

```
        // Membatalkan pendaftaran (detach) observer
        void Detach(IObserver observer);

        // Memberitahu (notify) semua observer yang terdaftar
        void Notify();
    }
}
```

- **Tujuan:** Mendefinisikan kontrak standar untuk kelas Subject. Subject harus menyediakan cara bagi Observer untuk mendaftar (`Attach`), membatalkan pendaftaran (`Detach`), dan harus bisa mengirim notifikasi ke semua Observer terdaftar (`Notify`).

3. Kelas `Subject` (`Subject.cs`)

```
namespace jmmodul14_2311104072
{
    public class Subject : ISubject
    {
        // State internal Subject yang akan diamati oleh Observer
        public int State { get; set; } = -0;

        // Daftar untuk menyimpan referensi ke semua Observer yang terdaftar
        private List<IObserver> _observers = new List<IObserver>();

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

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

        // Implementasi Notify: Mengirim update ke semua observer
        public void Notify()
        {
            Console.WriteLine("Subject: Notifying observers...");
            // Iterasi melalui daftar observer dan panggil metode Update
mereka
            foreach (var observer in _observers)
            {
                observer.Update(this);
            }
        }

        // Metode contoh untuk simulasi perubahan state
        public void SomeBusinessLogic()
        {
            Console.WriteLine("\\\\nSubject: I'm doing something
```

```
important.");
            // Mengubah state secara acak
            this.State = new Random().Next(0, 10);
            Thread.Sleep(15); // Jeda singkat
            Console.WriteLine($"Subject: My state has just changed to:
{this.State}");
            // Memberitahu observer setelah state berubah
            this.Notify();
        }
    }
}
```

- **Tujuan:** Implementasi konkret dari `ISubject`. Kelas ini memiliki `State` yang bisa berubah, menyimpan daftar `_observers`, dan mengimplementasikan logika `Attach`, `Detach`, serta `Notify`. Metode `SomeBusinessLogic` mensimulasikan perubahan `State` dan memanggil `Notify`.

4. Kelas `ConcreteObserverA` (`ConcreteObserverA.cs`)

```
namespace jmmodul14_2311104072
{
    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.");
            }
        }
    }
}
```

- **Tujuan:** Implementasi konkret `IObserver`. Observer ini hanya tertarik dan bereaksi (mencetak pesan) jika `State` Subject yang diterima saat `Update` dipanggil bernilai kurang dari 3.

5. Kelas `ConcreteObserverB` (`ConcreteObserverB.cs`)

```
namespace jmmodul14_2311104072
{
    class ConcreteObserverB : IObserver
    {
        public void Update(ISubject subject)
        {
            // Bereaksi jika state Subject >= 3 (asumsi)
            if ((subject as Subject).State >= 3)
            {
```

```
                Console.WriteLine("ConcreteObserverB: Reacted to the
event.");
            }
        }
    }
}
```

- **Tujuan:** Implementasi konkret `IObserver` lainnya. Observer ini bereaksi (mencetak pesan) jika `State` Subject yang diterima saat `Update` dipanggil bernilai 3 atau lebih.

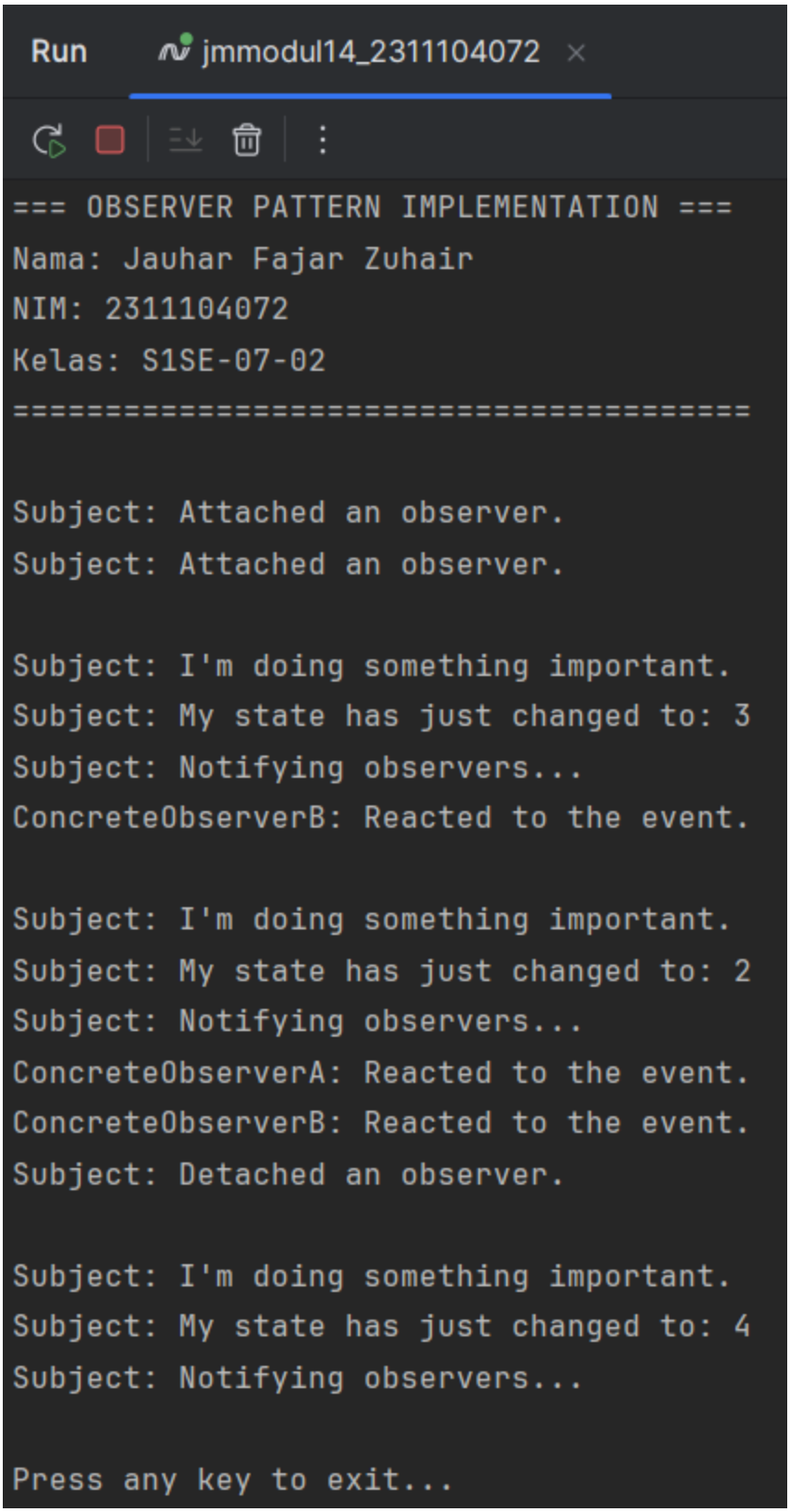
6. Kelas `Program` (`Program.cs`)

```
namespace jmmodul14_2311104072
{
    class Program
    {
        static void Main(string[] args)
        {
            // 1. Buat instance Subject
            var subject = new Subject();

            // 2. Buat instance Observer
            var observerA = new ConcreteObserverA();
            var observerB = new ConcreteObserverB();

            // 3. Daftarkan Observer ke Subject
            subject.Attach(observerA);
            subject.Attach(observerB);

            // 4. Jalankan logika bisnis Subject (akan mengubah state &
notifikasi)
            subject.SomeBusinessLogic();
            subject.SomeBusinessLogic();

            // 5. Lepaskan salah satu Observer
            Console.WriteLine("\\\\nSubject: Detaching Observer B...");
            subject.Detach(observerB);

            // 6. Jalankan logika bisnis lagi (Observer B tidak akan bereaksi
lagi)
            subject.SomeBusinessLogic();

            Console.WriteLine("\\\\nDemo Complete.");
        }
    }
}
```

- **Tujuan:** Titik masuk aplikasi yang mendemonstrasikan alur kerja Observer Pattern: membuat Subject dan Observer, menghubungkan mereka (`Attach`), memicu perubahan state dan notifikasi (`SomeBusinessLogic`), memutuskan hubungan salah satu Observer (`Detach`), dan melihat dampaknya pada notifikasi berikutnya.

Bagian III: Hasil Eksekusi (Contoh Tampilan Konsol)

Karena `Subject.SomeBusinessLogic()` menggunakan `Random` untuk mengubah `State`, output akan bervariasi setiap kali program dijalankan. Berikut adalah *salah satu contoh* kemungkinan output:


Run
jmmodul14_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: 3
Subject: Notifying observers...
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: 4
Subject: Notifying observers...
Press any key to exit...

Penjelasan Contoh Output:

1. Observer A dan B berhasil didaftarkan ke Subject.
2. Perubahan state pertama menjadi 8. Notifikasi dikirim. Hanya Observer B yang bereaksi (karena 8 >= 3).
3. Perubahan state kedua menjadi 2. Notifikasi dikirim. Hanya Observer A yang bereaksi (karena 2 < 3).
4. Observer B dilepas dari Subject.
5. Perubahan state ketiga menjadi 5. Notifikasi dikirim *hanya* ke Observer yang masih terdaftar (Observer A). Observer A tidak bereaksi karena kondisinya (5 < 3) tidak terpenuhi. Observer B tidak menerima notifikasi sama sekali.

Bagian IV: Kesimpulan

Praktikum Jurnal Modul 14 ini berhasil mengimplementasikan dan mendemonstrasikan **Observer Design Pattern** menggunakan C#. Melalui pemisahan antara Subject (sumber event) dan Observer (penerima notifikasi) menggunakan interface, pola ini menciptakan sistem yang *loosely coupled*, di mana Observer dapat ditambahkan atau dihapus secara dinamis tanpa perlu mengubah kode Subject. Hasil eksekusi menunjukkan bagaimana Observer yang berbeda dapat bereaksi secara spesifik terhadap perubahan state Subject, dan bagaimana pelepasan Observer menghentikan penerimaan notifikasi selanjutnya.