

Nama : Muhammad Rizaldy Akbar

Kelas : SE063

NIM : 2211104065

MENJELASKAN SALAH SATU DESIGN PATTERN

1. Buka halaman web <https://refactoring.guru/design-patterns/catalog> kemudian baca design pattern

dengan nama “Observer”, dan jawab pertanyaan berikut ini (dalam Bahasa Indonesia):

A. Berikan salah satu contoh kondisi dimana design pattern “Observer” dapat digunakan

Jawab :

Design pattern Observer dapat digunakan pada sistem pemberitahuan ketersediaan produk di toko online. Misalnya, pelanggan dapat berlangganan pada produk tertentu dan akan secara otomatis menerima notifikasi ketika produk tersebut tersedia di toko. Dengan cara ini, pelanggan tidak perlu mengecek toko berulang kali, dan toko hanya mengirim notifikasi kepada pelanggan yang benar-benar tertarik.

B. Berikan penjelasan singkat mengenai langkah-langkah dalam mengimplementasikan design pattern “Observer”

Jawab :

Langkah-langkah mengimplementasikan design pattern Observer adalah sebagai berikut:

1. Membuat antarmuka Observer yang mendefinisikan metode update() sebagai metode notifikasi.
2. Membuat antarmuka Subject (penerbit) yang memiliki metode untuk mendaftarkan (attach), menghapus (detach), dan memberi notifikasi (notify) kepada observer.
3. Subject menyimpan daftar observer yang berlangganan dan memanggil metode update() pada masing-masing observer saat terjadi perubahan status.
4. Observer mengimplementasikan metode update() untuk melakukan aksi sebagai respons terhadap perubahan.
5. Pada waktu runtime, klien membuat objek Subject dan Observer, lalu menghubungkan observer ke subject sesuai kebutuhan.

C. Berikan kelebihan dan kekurangan dari design pattern “Observer”

Jawab :

Kelebihan design pattern Observer adalah:

- Memungkinkan hubungan satu-ke-banyak antar objek secara dinamis saat runtime tanpa mengubah kode penerbit.
- Memudahkan penambahan observer baru tanpa memodifikasi subject (prinsip terbuka/tertutup).
- Memudahkan sinkronisasi dan komunikasi antar objek yang saling bergantung.

Kekurangannya adalah:

- Urutan notifikasi ke observer bisa tidak terprediksi, sehingga membuat debugging sulit.
- Bila banyak observer terdaftar, bisa menurunkan performa aplikasi.
- Risiko memory leak jika observer tidak dilepas dari daftar langganan saat tidak digunakan.

IMPLEMENTASI DAN PEMAHAMAN DESIGN PATTERN OBSERVER

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace tpmodul13_2211104065
{
    // Subject (yang akan diamati)
    public class Subject
    {
        private List<IObserver> observers = new List<IObserver>();

        // Menambahkan Observer
        public void Attach(IObserver observer)
        {
            observers.Add(observer);
        }

        // Menghapus Observer
        public void Detach(IObserver observer)
        {
            observers.Remove(observer);
        }

        // Memberitahukan Observer tentang perubahan
        public void Notify()
        {
            foreach (var observer in observers)
            {
                observer.Update();
            }
        }
    }

    // Observer (pengamat)
    public interface IObserver
    {
        void Update();
    }

    // ConcreteObserver (pengamat konkret)
    public class ConcreteObserver : IObserver
    {
        private string _name;
        private Subject _subject;
    }
}
```

```

public ConcreteObserver(string name, Subject subject)
{
    _name = name;
    _subject = subject;
    _subject.Attach(this);
}

public void Update()
{
    Console.WriteLine($"Observer {_name} has been notified.");
}
}

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

        // Membuat Observer
        var observer1 = new ConcreteObserver("Observer 1", subject);
        var observer2 = new ConcreteObserver("Observer 2", subject);

        // Memberitahukan Observer tentang perubahan pada Subject
        Console.WriteLine("Notifying observers...");
        subject.Notify();

        // Menghapus observer
        subject.Detach(observer1);

        // Memberitahukan Observer yang tersisa
        Console.WriteLine("Notifying observers after detaching Observer 1...");
        subject.Notify();
    }
}

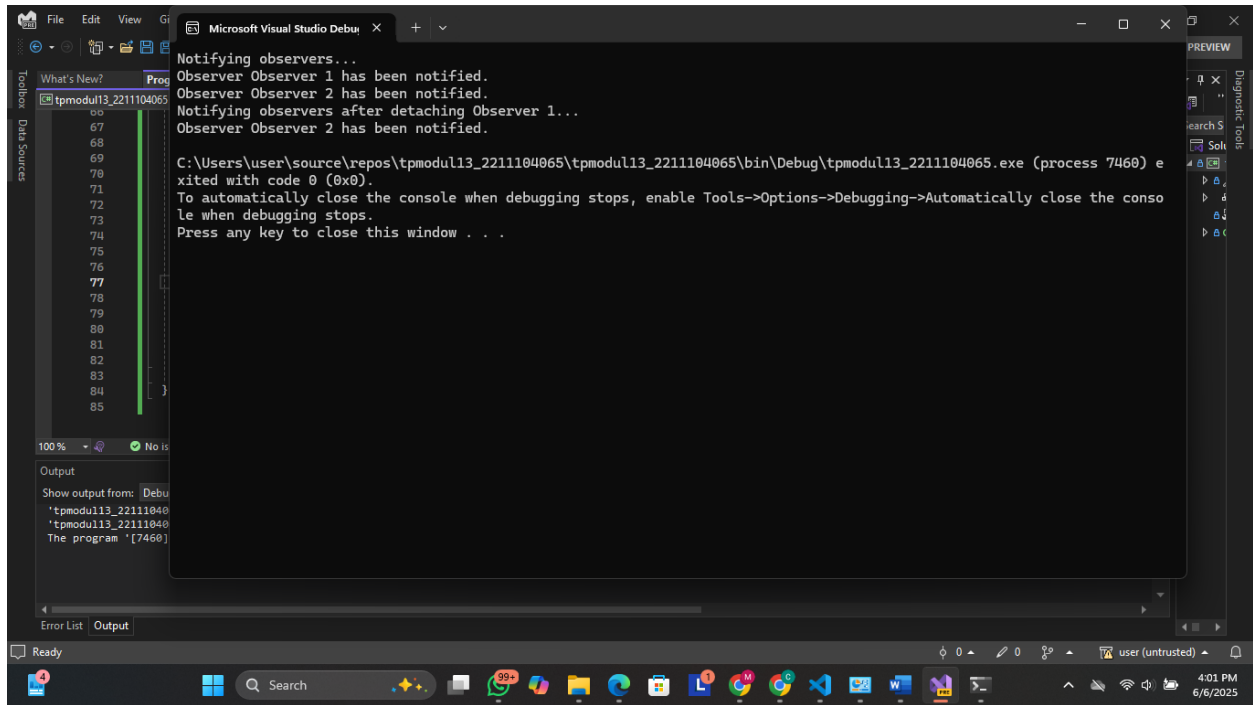
```

Penjelasan kode

- Subject menyimpan daftar observers dan menyediakan method Attach, Detach, dan Notify.
- IObserver adalah antarmuka yang wajib diimplementasikan oleh semua observer.
- ConcreteObserver adalah implementasi nyata dari observer yang mendaftar ke Subject saat dibuat, dan mencetak pesan saat menerima notifikasi.
- Di Main, dua observer dibuat dan didaftarkan ke Subject. Ketika Notify() dipanggil, kedua observer diberi tahu.
- Setelah observer1 dihapus, hanya observer2 yang menerima notifikasi selanjutnya.

Kesimpulan: Pola ini berguna ketika banyak objek perlu merespons perubahan pada satu objek pusat secara otomatis dan sinkron

Output



The screenshot shows the Microsoft Visual Studio Debug Console interface. The main window displays the following output:

```
Notifying observers...
Observer Observer 1 has been notified.
Observer Observer 2 has been notified.
Notifying observers after detaching Observer 1...
Observer Observer 2 has been notified.

C:\Users\user\source\repos\tpmodul13_2211104065\tpmodul13_2211104065\bin\Debug\tpmodul13_2211104065.exe (process 7460) e
xited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

On the left, the Source Code window shows the following code snippet:

```
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
```

At the bottom, the Output window shows the following information:

```
Show output from: Debug
'tpmodul13_22111040
'tpmodul13_22111040
The program '[7460]
```

The Windows taskbar at the bottom shows the system clock as 4:01 PM on 6/6/2025.

