Nama : Muhammad Fariz Nur Hidayat

Kelas : SE063

NIM : 2211104069

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:

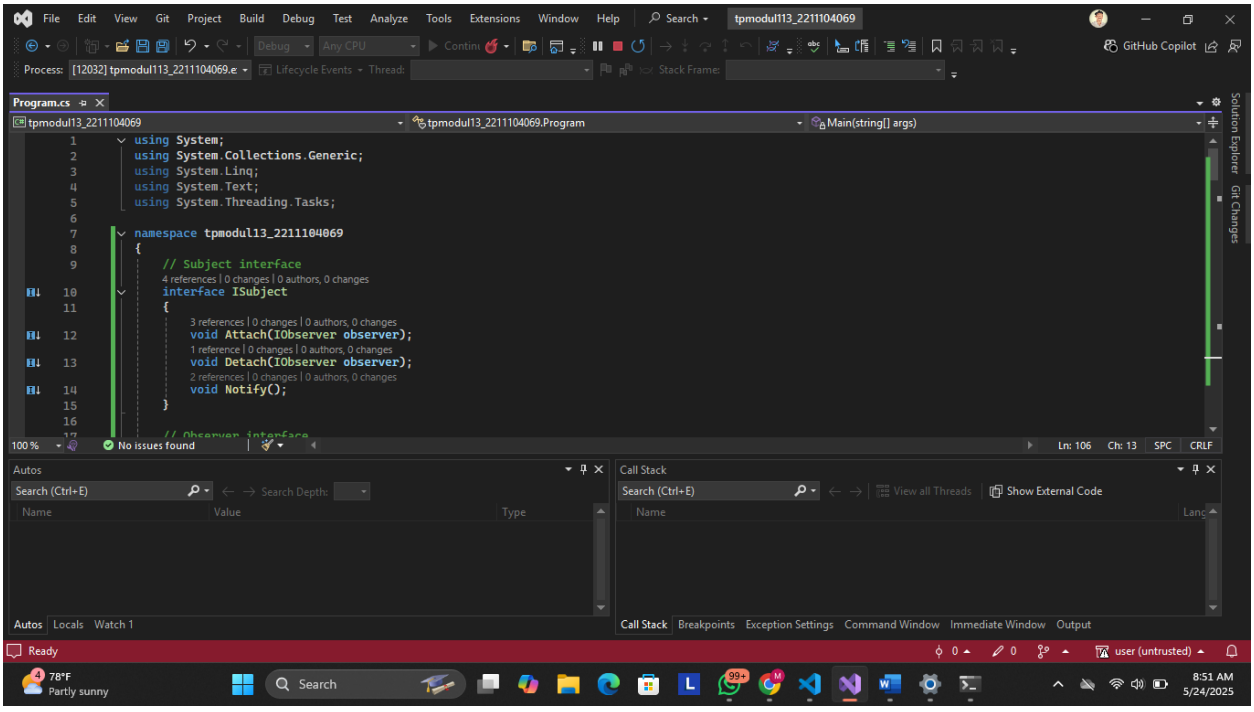
- Memungkinkan hubungan satu-ke-banyak antar objek secara dinamis saat runtime tanpa mengubah kode penerbit.
- Memudahkan penambahan observer baru tanpa memodifikasi subject (prinsip terbuka/tertutup).
- Memudahkan sinkronisasi dan komunikasi antar objek yang saling bergantung.

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

Bukti Pengerjaan



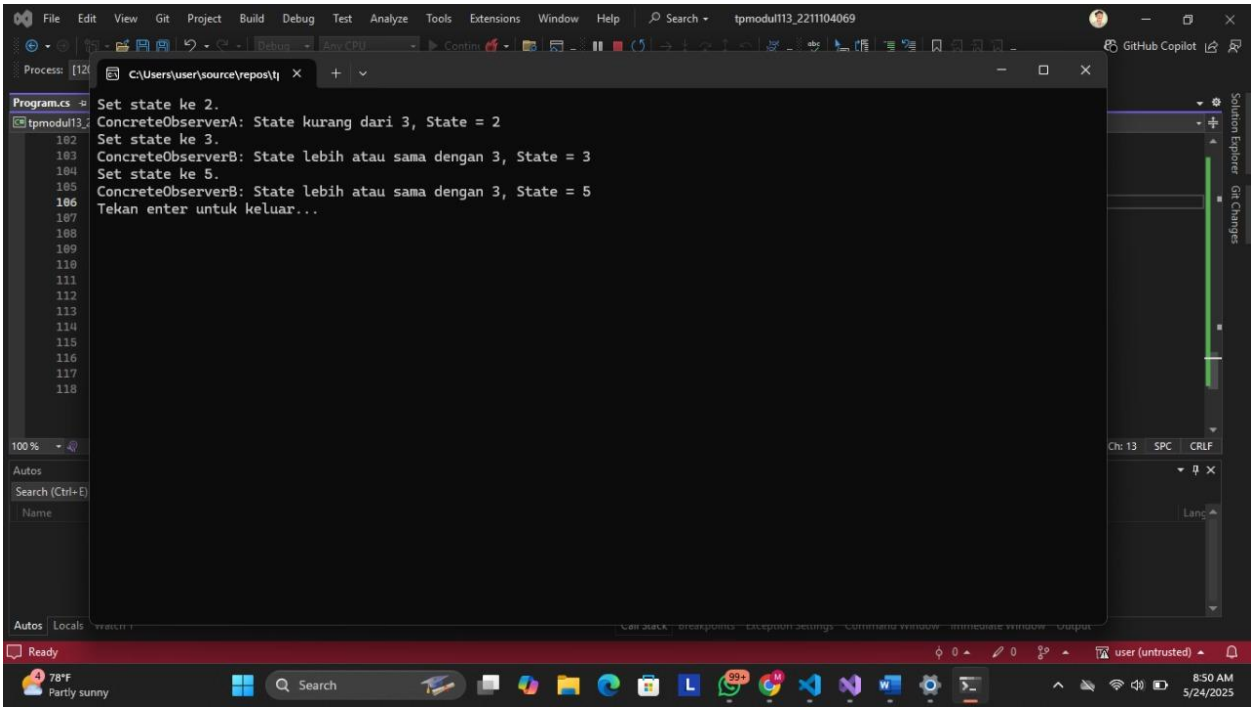
Program.cs

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

namespace tpmodul13_2211104069
{
    // Subject interface
    interface ISubject
    {
        void Attach(IObserver observer);
        void Detach(IObserver observer);
        void Notify();
    }

    // Observer interface
    interface IObserver
    {
        void Update(ISubject subject);
    }

    // Concrete Subject
    class ConcreteSubject : ISubject
    {
        private List<IObserver> observers = new List<IObserver>();

        private int state;
        public int State
        {
            get { return state; }
            set
            {
                state = value;
                Notify(); // notify observers ketika state berubah
            }
        }

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

        public void Detach(IObserver observer)
        {
```

```
            observers.Remove(observer);
        }

        public void Notify()
        {
            foreach (var observer in observers)
            {
                observer.Update(this);
            }
        }
    }

    // Concrete Observer A
    class ConcreteObserverA : IObserver
    {
        public void Update(ISubject subject)
        {
            if (subject is ConcreteSubject concreteSubject)
            {
                if (concreteSubject.State < 3)
                {
                    Console.WriteLine("ConcreteObserverA: State kurang dari 3, State = " +
concreteSubject.State);
                }
            }
        }
    }

    // Concrete Observer B
    class ConcreteObserverB : IObserver
    {
        public void Update(ISubject subject)
        {
            if (subject is ConcreteSubject concreteSubject)
            {
                if (concreteSubject.State >= 3)
                {
                    Console.WriteLine("ConcreteObserverB: State lebih atau sama dengan 3,
State = " + concreteSubject.State);
                }
            }
        }
    }

    internal class Program
    {
```

```
        static void Main(string[] args)
        {
            // Membuat subject
            var subject = new ConcreteSubject();

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

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

            // Mengubah state dan melihat hasil observer
            Console.WriteLine("Set state ke 2.");
            subject.State = 2;

            Console.WriteLine("Set state ke 3.");
            subject.State = 3;

            Console.WriteLine("Set state ke 5.");
            subject.State = 5;

            // Tunggu user tekan enter untuk keluar
            Console.WriteLine("Tekan enter untuk keluar...");
            Console.ReadLine();
        }
    }
}
```

Output



Penjelasan Kode

Penjelasan tiap baris di method Main:

var subject = new ConcreteSubject();

Membuat instance objek Subject yang akan diamati.

var observerA = new ConcreteObserverA();

var observerB = new ConcreteObserverB();

Membuat dua objek observer yang akan menerima notifikasi perubahan dari subject.

subject.Attach(observerA);

subject.Attach(observerB);

Mendaftarkan kedua observer ke subject supaya mereka menerima update.

subject.State = 2;

Mengubah state subject, yang secara otomatis memicu pemanggilan method Notify() dan mengupdate semua observer.

subject.State = 3;

subject.State = 5;

Mengubah state beberapa kali untuk melihat bagaimana observer merespon perubahan tersebut.

Console.ReadLine();

Agar jendela console tetap terbuka sampai user menekan enter.

Penjelasan keseluruhan :

Kode di atas mengimplementasikan design pattern **Observer** dalam bahasa C#. Pola ini terdiri dari dua interface utama, yaitu ISubject yang merepresentasikan objek yang diamati (subject) dan IObserver yang merupakan objek pengamat (observer). Kelas ConcreteSubject mengimplementasikan ISubject dan menyimpan daftar observer yang terdaftar untuk menerima notifikasi. Saat properti State pada ConcreteSubject diubah, metode Notify() dipanggil untuk memberi tahu semua observer dengan memanggil metode Update() masing-masing observer. Dua kelas observer konkret, yaitu ConcreteObserverA dan ConcreteObserverB, mengimplementasikan interface IObserver dan memberikan respons berbeda berdasarkan nilai State. Dalam method Main, objek subject dan dua observer dibuat, kemudian observer didaftarkan (attach) ke subject. Ketika nilai state diubah secara berturut-turut menjadi 2, 3, dan 5, observer akan bereaksi sesuai kondisi yang telah ditentukan, menampilkan pesan yang berbeda pada konsol. Pola ini memungkinkan hubungan dinamis antara objek subject dan observer, di mana observer dapat bereaksi secara otomatis saat terjadi perubahan pada subject.