TUGAS PENDAHULUAN
KONSTRUKSI PERANGKAT LUNAK

MODUL XIII

DESIGN PATTERN IMPLEMENTATION

Disusun Oleh :

Nadia Putri Rahmaniar / 2211104012

S1 SE-06-01

Dosen Pengampu :

Yudha Islami Sulistya, S.Kom., M.Cs.

PROGRAM STUDI S1 SOFTWARE ENGINEERING

TELKOM UNIVERSITY PURWOKERTO

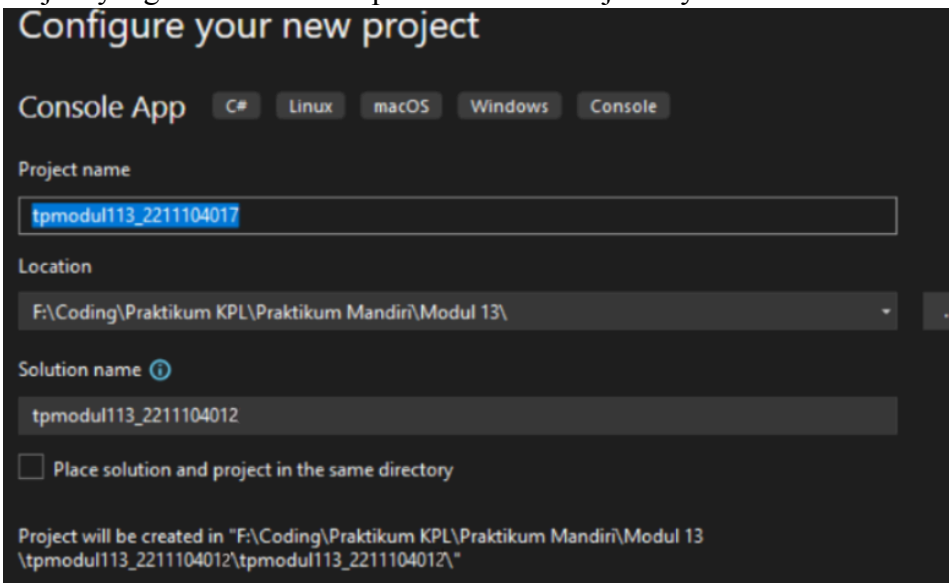
TUGAS PENDAHULUAN 13

1. MEMBUAT PROJECT GUI BARU

Buka IDE misalnya dengan Visual Studio

A. Misalnya menggunakan Visual Studio, buatlah project baru dengan nama tpmodul113_NIM

B. Project yang dibuat bisa berupa console atau sejenisnya



2. MENJELASKAN SALAH SATU DESIGN PATTERN

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:

Observer Pattern dapat digunakan misalnya ketika *Customer* ingin mendapatkan notifikasi dari *Store* tentang ketersediaan produk tertentu, misalnya laptop keluaran terbaru. Daripada pelanggan harus bolak-balik datang ke toko untuk mengecek, atau toko mengirimkan notifikasi ke semua pelanggan (termasuk yang

tidak tertarik), Observer Pattern memungkinkan toko hanya mengirimkan pemberitahuan kepada pelanggan yang secara spesifik telah *subscribe* untuk mendapatkan notifikasi tersebut.

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

Jawab:

Berikut langkah-langkah singkat implementasi Observer Pattern pada studi kasus *subscribe* di toko laptop:

1. Identifikasi objek yang menjadi publisher (subjek) dan subscriber (pengamat).
2. Membuat antarmuka subscriber, minimal mempunyai metode update() yang akan dipanggil saat ada perubahan.
3. Membuat antarmuka publisher yang menyediakan metode attach(), detach(), dan notify() untuk mengatur daftar subscriber.
4. Mengimplementasikan daftar langganan, publisher akan menyimpan daftar subscriber dan memanggil metode update() milik mereka saat terjadi event.
5. Mengimplementasikan subscriber konkret, subscriber akan merespons notifikasi dengan mengambil data dari publisher (jika diperlukan).
6. Di bagian client, buat dan hubungkan objek publisher dengan subscriber secara dinamis sesuai kebutuhan.

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

Jawab:

Kelebihan

- **Loose Coupling (Keterkaitan Longgar)**

  Publisher dan Subscriber tidak saling bergantung secara langsung. Keduanya hanya terhubung melalui antarmuka (interface), sehingga memudahkan pengembangan dan pemeliharaan.
- **Mendukung Open/Closed Principle**

Dapat menambahkan tipe Subscriber baru tanpa mengubah kode Publisher, sehingga sistem lebih fleksibel terhadap perubahan.

- **Dapat Dinamis di Waktu Runtime**

  Objek dapat mendaftar (subscribe) atau keluar (unsubscribe) kapan saja selama program berjalan, menjadikan pola ini sangat fleksibel.

- **Pola Komunikasi yang Efisien untuk Banyak Objek**

  Cocok digunakan ketika satu objek perlu memberi tahu banyak objek lain saat terjadi perubahan status atau event tertentu.

Kekurangan:

- **Sulit Melacak Aliran Notifikasi**

  Karena hubungan antara Publisher dan Subscriber bersifat tidak langsung, sulit untuk menelusuri urutan atau penyebab notifikasi dalam sistem besar.

- **Potensi Masalah Kinerja**

  Jika Publisher memiliki banyak Subscriber, proses notifikasi bisa menjadi lambat, terutama jika masing-masing Subscriber menjalankan operasi berat.

- **Subscriber Bisa Menjadi Bergantung pada Publisher Secara Tidak Langsung**

  Meski antar kelas tidak terikat langsung, implementasi notifikasi bisa menciptakan ketergantungan terselubung melalui data atau metode tertentu.

- **Masalah Urutan Eksekusi**

  Publisher tidak menjamin urutan notifikasi ke setiap Subscriber, yang bisa menjadi masalah jika urutan eksekusi penting.

3. IMPLEMENTASI DAN PEMAHAMAN DESIGN PATTERN OBSERVER

Buka halaman web berikut https://refactoring.guru/design-patterns/observer dan scroll ke bagian “Code Examples”, pilih kode yang akan dilihat misalnya C# dan ikuti langkah-langkah berikut:

A. Pada project yang telah dibuat sebelumnya, tambahkan kode yang mirip atau sama dengan contoh kode yang diberikan di halaman web tersebut
B. Jalankan program tersebut dan pastikan tidak ada error pada saat project dijalankan
C. Jelaskan tiap baris kode yang terdapat di bagian method utama atau “main”

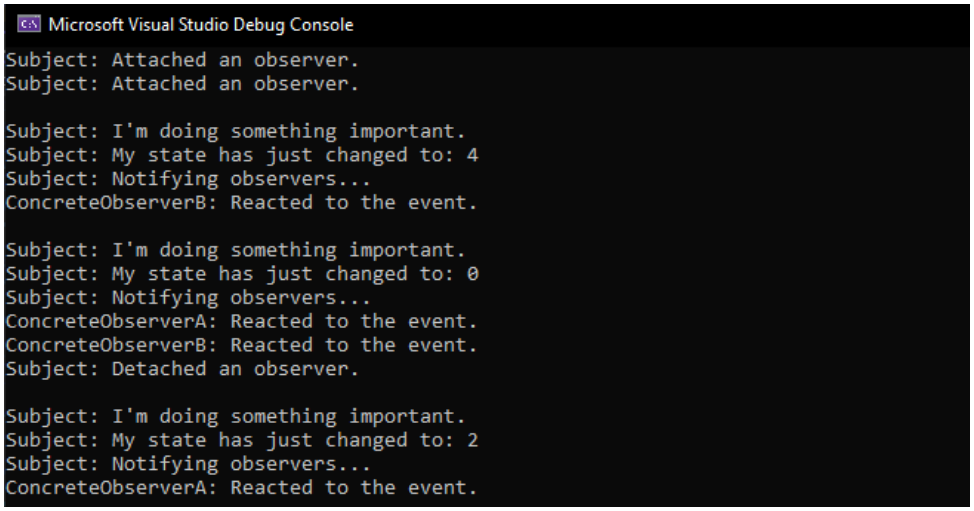
Source Code:

```
using System;
using System.Collections.Generic;
using System.Threading;

namespace ObserverPatternExample
{
    // Interface Observer
    8 references
    public interface IObserver
    {
        3 references
        void Update(ISubject subject);
    }

    // Interface Subject (Publisher)
    4 references
    public interface ISubject
    {
        3 references
        void Attach(IObserver observer);
        2 references
        void Detach(IObserver observer);
        2 references
        void Notify();
    }

    // Concrete Subject
    4 references
    public class Subject : ISubject
    {
        5 references
        public int State { get; set; } = 0;

        private List<IObserver> _observers = new List<IObserver>();

        3 references
        public void Attach(IObserver observer)
        {
            Console.WriteLine("Subject: Attached an observer.");
            _observers.Add(observer);
        }

        2 references
        public void Detach(IObserver observer)
        {
            _observers.Remove(observer);
            Console.WriteLine("Subject: Detached an observer.");
        }
```

```
        2 references
        public void Notify()
        {
            Console.WriteLine("Subject: Notifying observers...");
            foreach (var observer in _observers)
            {
                observer.Update(this);
            }
        }

        3 references
        public void SomeBusinessLogic()
        {
            Console.WriteLine("\nSubject: I'm doing something important.");
            this.State = new Random().Next(0, 10);

            Thread.Sleep(15);

            Console.WriteLine($"Subject: My state has just changed to: {this.State}");
            this.Notify();
        }
    }

    // Concrete Observer A
    1 reference
    public class ConcreteObserverA : IObserver
    {
        2 references
        public void Update(ISubject subject)
        {
            if ((subject as Subject).State < 3)
            {
                Console.WriteLine("ConcreteObserverA: Reacted to the event.");
            }
        }
    }
```

```
    // Concrete Observer B
    1 reference
    public class ConcreteObserverB : IObserver
    {
        2 references
        public void Update(ISubject subject)
        {
            if ((subject as Subject).State == 0 || (subject as Subject).State >= 2)
            {
                Console.WriteLine("ConcreteObserverB: Reacted to the event.");
            }
        }
    }

    // Main Program
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            var subject = new Subject();

            var observerA = new ConcreteObserverA();
            subject.Attach(observerA);

            var observerB = new ConcreteObserverB();
            subject.Attach(observerB);

            subject.SomeBusinessLogic();  // Observer A dan B merespon
            subject.SomeBusinessLogic();  // Observer A dan B merespon

            subject.Detach(observerB);    // Observer B tidak lagi merespon

            subject.SomeBusinessLogic();  // Hanya Observer A yang merespon
        }
    }
}
```

Hasil:



Penjelasan di bagian method utama atau "main":

- var subject = new Subject(); Baris kode ini berfungsi untuk menciptakan subject, yaitu inti dari pola Observer. subject ini akan menyimpan data atau state, dan perubahannya akan dipublikasikan kepada para observer yang mendaftar.
- var observerA = new ConcreteObserverA(); subject.Attach(observerA); Pertama, kita membuat observerA, sebuah objek yang ingin memantau dan bereaksi terhadap perubahan pada subject. Kemudian, melalui metode Attach(observerA), observerA didaftarkan ke subject. Ini berarti observerA kini akan menerima pemberitahuan setiap kali subject mengalami perubahan.
- var observerB = new ConcreteObserverB(); subject.Attach(observerB); Mirip dengan observerA, baris ini membuat observerB, observer kedua yang juga tertarik pada perubahan subject. Setelah itu, observerB didaftarkan ke subject dengan Attach(observerB), memastikan observerB juga akan menerima notifikasi
- subject.SomeBusinessLogic(); Ketika metode ini dijalankan, subject akan melakukan operasi bisnis internalnya. Proses ini mungkin melibatkan perubahan pada state internal subject secara acak. Setelah perubahan terjadi, subject secara otomatis akan memberi tahu semua observer yang telah mendaftar (dalam kasus ini, observerA dan observerB) tentang perubahan tersebut.
- subject.Detach(observerB); Kode ini berfungsi untuk menghapus observerB dari daftar observer yang memantau subject. Setelah operasi ini, observerB tidak akan lagi menerima notifikasi atau pembaruan dari subject, meskipun subject mengalami perubahan state.