RAKTIKUM KONSTRUKSI PERANGKAT LUNAK
TUGAS PENDAHULUAN 13

Design Pattern Implementation

Telkom University

disusun Oleh:
Nita Fitrotul Mar’ah
2211104005
SE0601

Dosen Pengampu :
Yudha Islami Sulistya, S.Kom., M.Cs.

S1 REKAYASA PERANGKAT LUNAK
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

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 cocok diterapkan dalam situasi seperti ketika seorang pelanggan ingin mendapatkan informasi terbaru dari sebuah toko mengenai ketersediaan produk tertentu, misalnya laptop model terbaru. Daripada pelanggan harus terus-menerus mengecek langsung ke toko atau toko mengirim pemberitahuan ke semua pelanggan tanpa kecuali, Observer Pattern memungkinkan toko hanya mengirim notifikasi kepada pelanggan yang sudah mendaftar (subscribe) untuk menerima pembaruan tersebut. Dengan begitu, komunikasi jadi lebih tepat sasaran dan efisien.

   b. Berikan penjelasan singkat mengenai langkah-langkah dalam mengimplementasikan design pattern "Observer".

      Jawab:

      Berikut adalah langkah-langkah dasar dalam menerapkan Observer Pattern berdasarkan studi kasus notifikasi toko laptop:

      - Tentukan objek mana yang berperan sebagai Publisher (Subjek) dan mana yang berperan sebagai Subscriber (Pengamat).
      - Buat interface untuk Subscriber yang minimal memiliki metode update() yang akan dipanggil saat ada perubahan dari Publisher.
      - Buat interface untuk Publisher yang menyediakan metode seperti attach(), detach(), dan notify() guna mengelola daftar subscriber.
      - Implementasikan mekanisme langganan di Publisher, di mana Publisher menyimpan daftar subscriber dan memanggil metode update() saat suatu peristiwa terjadi.
      - Buat implementasi konkrit dari Subscriber yang akan menangani notifikasi dan, jika diperlukan, mengambil data dari Publisher.
      - Pada bagian klien, hubungkan Publisher dan Subscriber secara dinamis sesuai kebutuhan aplikasi.

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

Jawab:

Kelebihan:

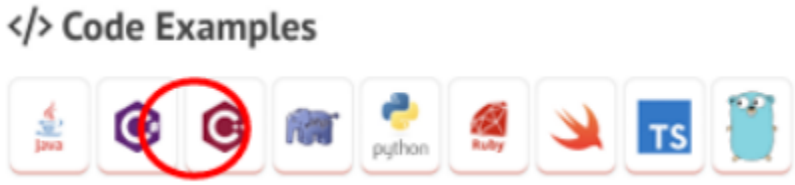
- Keterkaitan Longgar (Loose Coupling): Publisher dan Subscriber tidak saling ketergantungan secara langsung, melainkan hanya melalui interface, yang mempermudah pengembangan dan perawatan sistem.
- Mendukung Prinsip Open/Closed: Kita bisa menambahkan jenis Subscriber baru tanpa perlu mengubah kode dari Publisher, membuat sistem lebih mudah dikembangkan.
- Fleksibilitas Saat Runtime: Objek bisa mendaftar (subscribe) atau berhenti (unsubscribe) kapan saja selama aplikasi berjalan.
- Efisien untuk Komunikasi Satu-ke-Banyak: Cocok dipakai jika satu objek perlu memberi tahu banyak objek lain saat terjadi perubahan atau event.

Kekurangan:

- Sulit Melacak Notifikasi: Karena hubungan antar objek tidak langsung, menjadi sulit untuk menelusuri siapa yang memberi notifikasi dan siapa yang menerimanya, terutama dalam sistem berskala besar.
- Masalah Performa: Jika jumlah Subscriber banyak dan masing-masing melakukan tugas berat, proses pemberitahuan bisa mengganggu kinerja aplikasi.
- Ketergantungan Terselubung: Walaupun terlihat terpisah, implementasi notifikasi bisa menciptakan ketergantungan tidak langsung antara Publisher dan Subscriber.
- Tidak Menjamin Urutan Eksekusi: Urutan notifikasi ke tiap Subscriber tidak selalu terjamin, yang bisa menjadi masalah jika urutan tersebut 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
    {
        2 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;
        3 references
        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();
        }
    }
```

```
    public class ConcreteObserverA : IObserver
    {
        1 reference
        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
    {
        1 reference
        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:

```
Subject: Attached an observer.
Subject: Attached an observer.

Subject: I'm doing something important.
Subject: My state has just changed to: 4
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: 0
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: 2
Subject: Notifying observers...
ConcreteObserverA: Reacted to the event.

D:\KPL\13_Design_Pattern_Implementation\TP\tpmodul13_2211104005\tpmodul13_
13896) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Opt
>Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Penjelasan di bagian method utama atau "main":

- var subject = new Subject();
  Digunakan untuk membuat sebuah instance dari objek Subject, yang bertindak sebagai publisher. Objek ini memiliki status (state) yang dapat berubah, dan akan memberitahu semua observer yang terdaftar ketika terjadi perubahan pada status tersebut.
- var observerA = new ConcreteObserverA();
  subject.Attach(observerA);
  a. Baris pertama membuat objek observerA, yaitu observer yang ingin memantau perubahan yang terjadi pada Subject.
  b. Baris kedua menambahkan observerA ke dalam daftar observer milik Subject, sehingga observerA akan mendapatkan pemberitahuan setiap kali ada perubahan status pada Subject.
- var observerB = new ConcreteObserverB();
  subject.Attach(observerB);
  a. Digunakan untuk membuat objek observerB, yaitu observer kedua yang juga ingin mendapatkan informasi perubahan dari Subject.
  b. Baris kedua menambahkan observerB ke daftar observer pada Subject, sehingga observerB akan menerima notifikasi saat status Subject berubah.
- subject.SomeBusinessLogic();
  a. Menjalankan sebuah metode pada Subject yang berisi proses bisnis utama.
  b. Dalam proses ini, Subject akan secara acak mengubah nilainya (state) dan kemudian memberitahu semua observer yang telah terdaftar, yaitu observerA dan observerB.

- subject.Detach(observerB);
  Digunakan untuk menghapus observerB dari daftar observer yang dipantau oleh Subject. Setelah baris ini dijalankan, observerB tidak akan lagi menerima informasi atau pemberitahuan dari Subject mengenai perubahan status.