

TUGAS PENDAHULUAN
KONSTRUKSI PERANGKAT LUNAK
DESIGN PATTERN IMPLEMENTATION



Dibuat Oleh :

Muhammad Idham Cholid

2211104016

SE0601

PROGRAM STUDI S1 SOFTWARE ENGINEERING

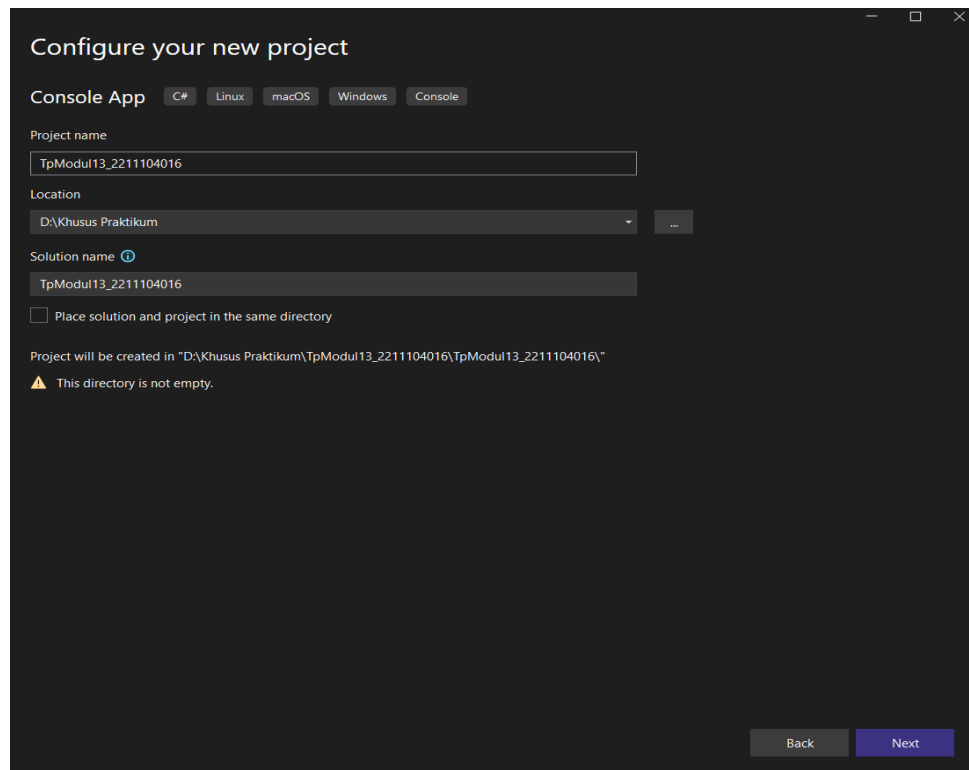
FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025

1. MEMBUAT PROJECT GUI BARU

- Membuat Project baru



2. MENJELASKAN SALAH SATU DESIGN PATTERN

- **Contoh Kondisi Penggunaan Observer**

Observer pattern biasanya digunakan ketika ada satu objek utama yang perubahannya perlu diketahui oleh banyak objek lain secara otomatis. Misalnya, pada aplikasi berita, ketika admin menambahkan artikel baru, semua pengguna yang sudah subscribe akan langsung mendapatkan notifikasi tanpa harus membuka aplikasi secara manual. Ini membuat sistem lebih efisien dan responsif

- **Langkah-Langkah Mengimplementasikan Observer Pattern**

1. Tentukan siapa yang menjadi subject (misalnya, sumber data utama).
2. Buat interface observer, biasanya berisi metode seperti update() yang akan dipanggil saat ada perubahan.
3. Implementasikan class subject yang bertugas menyimpan daftar observer dan memberitahu mereka jika ada update.
4. Tambahkan observer ke dalam subject biasanya menggunakan method attach() atau subscribe().
5. Ketika terjadi perubahan pada subject, semua observer akan dipanggil method update()-nya untuk merespon perubahan tersebut.

- **Kelebihan dan Kekurangan Observer Pattern**

Kelebihan:

- a. Memudahkan sinkronisasi data antara satu objek dengan banyak objek lain.
- b. Memisahkan logika antar komponen, sehingga kode lebih mudah dirawat dan dikembangkan.
- c. Cocok untuk sistem yang membutuhkan notifikasi real-time, seperti sistem chat atau update status.

Kekurangan:

- a. Kalau tidak hati-hati, bisa muncul masalah performa kalau jumlah observer terlalu banyak.
- b. Sulit dilacak ketika ada bug, karena pembaruan terjadi otomatis di banyak tempat.
- c. Perlu perhatian lebih untuk mengelola siklus hidup observer agar tidak menyebabkan memory leak.

3. IMPLEMENTASI DAN PEMAHAMAN DESIGN PATTERN OBSERVER

- Pada project ini menambahkan code yang sana dengan contoh kode yang diberikan di halaman web tersebut

```
Program.cs
TpModul13_2211104016
using System;
using System.Collections.Generic;
using System.Threading;

namespace RefactoringGuru.DesignPatterns.Observer.Conceptual
{
    public interface IObservable
    {
        // Receive update from subject
        void Update(ISubject subject);
    }

    public interface ISubject
    {
        // Attach an observer to the subject.
        void Attach(IObservable observer);

        // Detach an observer from the subject.
        void Detach(IObservable observer);

        // Notify all observers about an event.
        void Notify();
    }

    // The Subject owns some important state and notifies observers when the
    // state changes.
    public class Subject : ISubject
    {
        // For the sake of simplicity, the Subject's state, essential to all
        // subscribers, is stored in this variable.
        public int State { get; set; } = -0;

        // List of subscribers. In real life, the list of subscribers can be
        // stored more comprehensively (categorized by event type, etc.).
        private List<IObservable> _observers = new List<IObservable>();

        // The subscription management methods.
        public void Attach(IObservable observer)
        {
            Console.WriteLine("Subject: Attached an observer.");
            this._observers.Add(observer);
        }
    }
}
```

```
Program.cs
TpModul13_2211104016
public void Detach(IObservable observer)
{
    this._observers.Remove(observer);
    Console.WriteLine("Subject: Detached an observer.");
}

// Trigger an update in each subscriber.
public void Notify()
{
    Console.WriteLine("Subject: Notifying observers...");

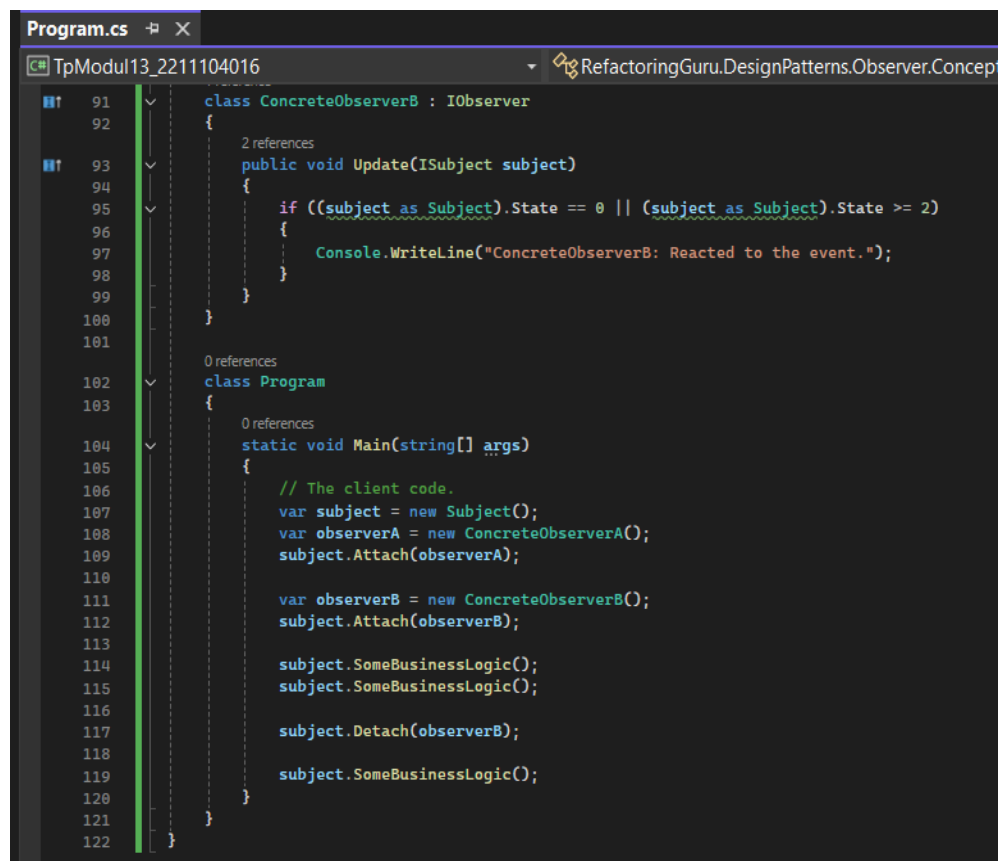
    foreach (var observer in _observers)
    {
        observer.Update(this);
    }
}

// Usually, the subscription logic is only a fraction of what a Subject
// can really do. Subjects commonly hold some important business logic,
// that triggers a notification method whenever something important is
// about to happen (or after it).
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 Observers react to the updates issued by the Subject they had
// been attached to.
class ConcreteObserverA : IObservable
{
    public void Update(ISubject subject)
    {
        if ((subject as Subject).State < 3)
        {
            Console.WriteLine("ConcreteObserverA: Reacted to the event.");
        }
    }
}
```



```
Program.cs
C:\TpModul13_2211104016 RefactoringGuru.DesignPatterns.Observer.Concept

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.");
        }
    }
}

0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        // The client code.
        var subject = new Subject();
        var observerA = new ConcreteObserverA();
        subject.Attach(observerA);

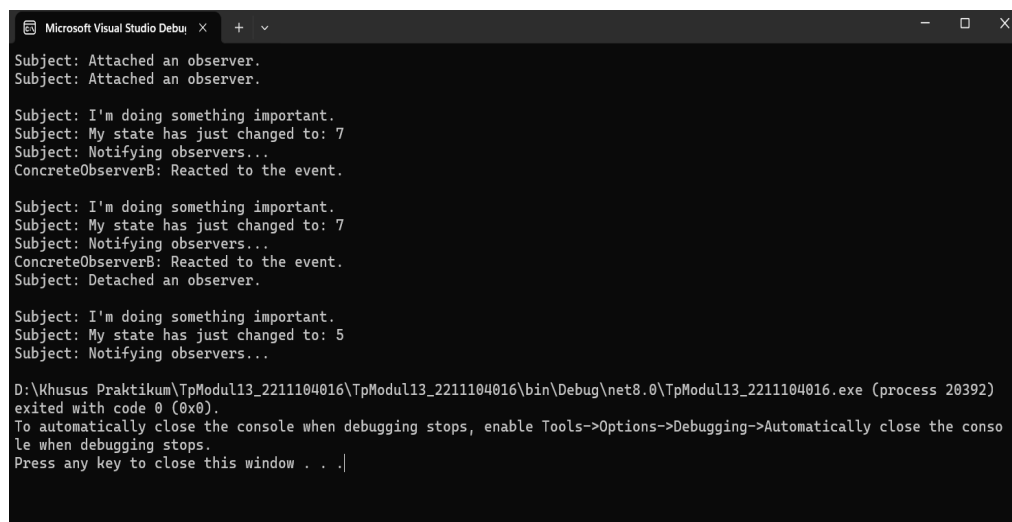
        var observerB = new ConcreteObserverB();
        subject.Attach(observerB);

        subject.SomeBusinessLogic();
        subject.SomeBusinessLogic();

        subject.Detach(observerB);

        subject.SomeBusinessLogic();
    }
}
```

- Hasil Running



```
Microsoft Visual Studio Debug
Subject: Attached an observer.
Subject: Attached an observer.

Subject: I'm doing something important.
Subject: My state has just changed to: 7
Subject: Notifying observers...
ConcreteObserverB: Reacted to the event.

Subject: I'm doing something important.
Subject: My state has just changed to: 7
Subject: Notifying observers...
ConcreteObserverB: Reacted to the event.
Subject: Detached an observer.

Subject: I'm doing something important.
Subject: My state has just changed to: 5
Subject: Notifying observers...

D:\Khusus Praktikum\TpModul13_2211104016\TpModul13_2211104016\bin\Debug\net8.0\TpModul13_2211104016.exe (process 20392)
exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .|
```

- Jelaskan tiap baris kode yang terdapat di bagian method utama atau “main”

```
var subject = new Subject();
```

Membuat Object Subject,yaitu pusat perubahan status yang akan diberi tahu kepada observer

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

Membuat observer A dan mendaftarkan ke subject observerA akan dipanggil saat status subject berubah

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

Membuat observer B dan juga mendaftarkan 2 observer yang akan merespons perubahan status subject.

```
subject.SomeBusinessLogic();  
subject.SomeBusinessLogic();
```

Menjalankan logika bisnis subject yang mengubah status dan memberi tahu observer.

```
subject.Detach(observerB);
```

Menghapus observerB dari daftar observer subject.

```
subject.SomeBusinessLogic();
```

Menjalankan logika bisnis subject untuk terakhir kali, hanya observerA yang diberi tahu untuk perubahan status.