

TP MODUL 13
DESIGN PATTERN IMPLEMENTATION



Nama :

Alya Rabani (2311104076)

Dosen :

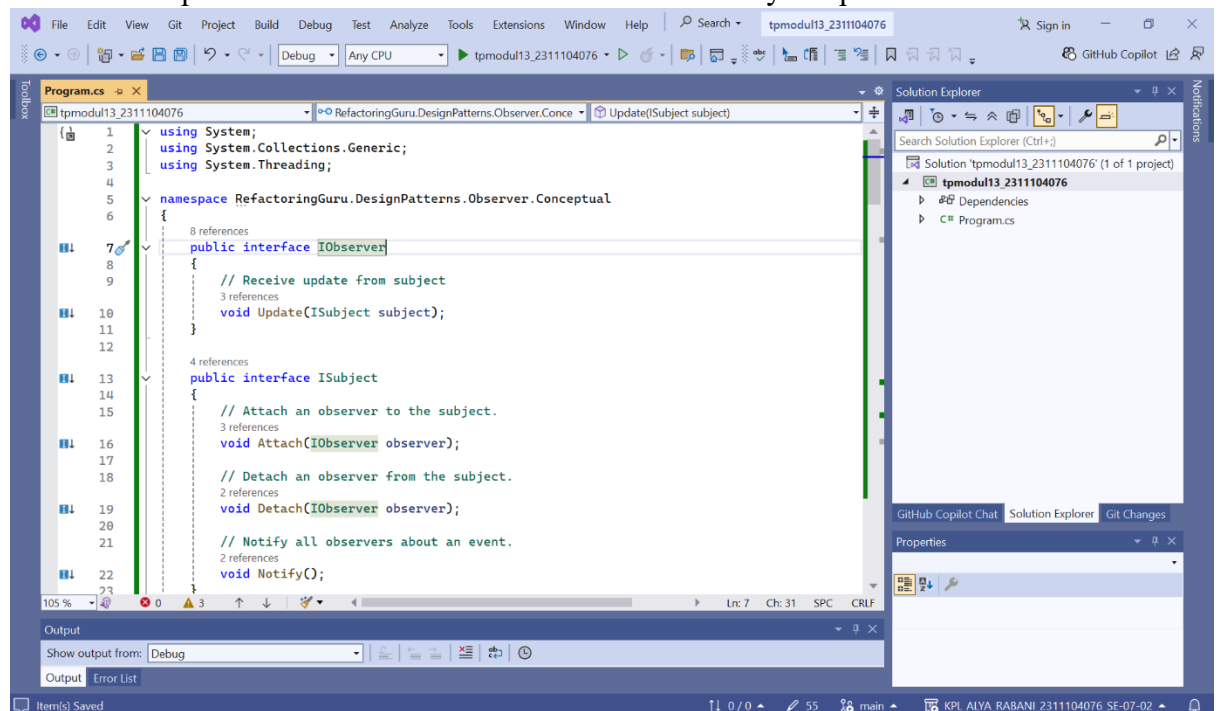
YUDHA ISLAMI SULISTYA

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK
TELKOM UNIVERSITY PURWOKERTO

2025

1. Menjelaskan salah satu design pattern

Pada penerapan Observer Pattern, ini digunakan untuk mengizinkan satu objek (Subject) memberi tahu banyak objek lain (Observers) saat terjadi perubahan status. Kemudian dapat membuat sistem lebih fleksibel dan loosely-coupled.



Namespace ini mendefinisikan ruang nama untuk menyimpan semua class agar tidak berbenturan dengan class lain.

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

namespace RefactoringGuru.DesignPatterns.Observer.Conceptual
```

Interface ini harus diimplementasikan oleh semua *observer*. Update() akan dipanggil ketika Subject berubah.

```
public interface IObserver
{
    // Receive update from subject
    void Update(ISubject subject);
}
```

Interface ini diimplementasikan oleh class Subject. Ia mendefinisikan kontrak tentang bagaimana observer bisa didaftarkan atau dilepas.

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

```

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

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

```

Subject adalah objek utama yang diamati. Saat kondisinya berubah, ia memberi tahu semua observer yang sudah terdaftar.

```

public class Subject : ISubject
{

```

State menyimpan status internal Subject. Perubahan nilai ini akan memicu notifikasi.

```

public int State { get; set; } = -0;

```

Daftar semua observer yang berlangganan.

```

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

```

Menambahkan observer ke daftar, lalu mencetak bahwa observer berhasil ditambahkan.

```

public void Attach(IObserver observer)

```

```

{
    Console.WriteLine("Subject: Attached an observer.");
    this._observers.Add(observer);
}

```

Menghapus observer dari daftar, dan mencetak bahwa observer dihapus.

```

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

```

Memberi tahu semua observer bahwa terjadi perubahan dengan memanggil Update().

```

public void Notify()
{
    Console.WriteLine("Subject: Notifying observers...");

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

```

Ini mensimulasikan logika bisnis. Setelah status diubah, ia memanggil Notify().

```

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();
}

```

Observer A hanya akan merespon jika State < 3.

```

class ConcreteObserverA : IObservable
{
    public void Update(ISubject subject)
    {
        if ((subject as Subject).State < 3)
        {
            Console.WriteLine("ConcreteObserverA: Reacted to the event.");
        }
    }
}

```

Observer B merespon jika State == 0 **atau** State >= 2.

```

class ConcreteObserverB : IObservable
{
    public void Update(ISubject subject)
    {
        if ((subject as Subject).State == 0 || (subject as Subject).State >= 2)
        {
            Console.WriteLine("ConcreteObserverB: Reacted to the event.");
        }
    }
}

```

Pada program utama ini, membuat Subject, menambahkan dua observer, dan menjalankan logika bisnis 3 kali, dengan perubahan observer di tengah-tengah.

```

class Program
{
    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();

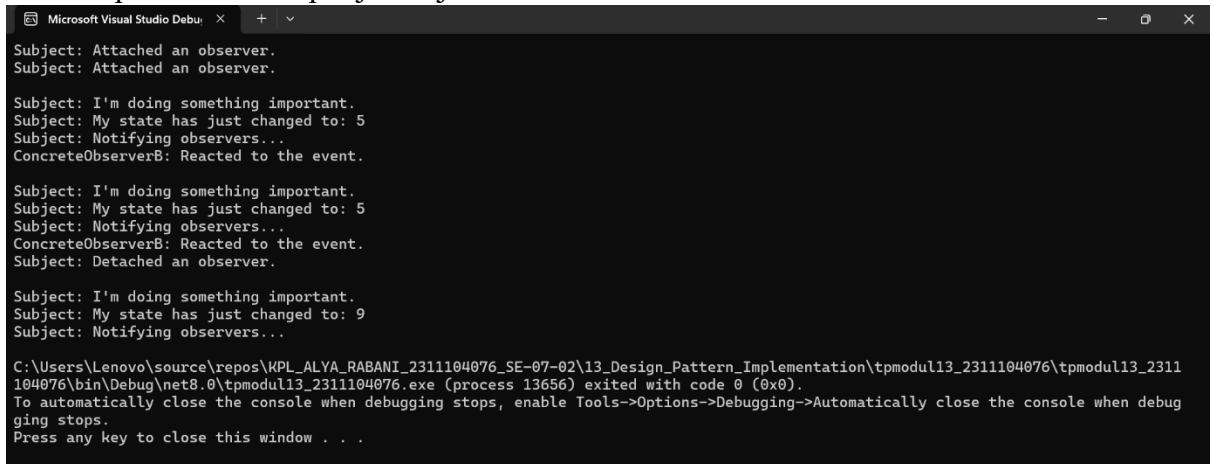
        subject.Detach(observerB);

        subject.SomeBusinessLogic();
    }
}
}

```

2. Implementasi dan pemahaman design pattern observer

Ini merupakan hasil output jika dijalankan



```

Microsoft Visual Studio Debu: x + v
Subject: Attached an observer.
Subject: Attached an observer.

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

Subject: I'm doing something important.
Subject: My state has just changed to: 5
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: 9
Subject: Notifying observers...

C:\Users\Lenovo\source\repos\KPL_ALYA_RABANI_2311104076_SE-07-02\13_Design_Pattern_Implementation\tpmodul13_2311104076\tpmodul13_2311104076\bin\Debug\net8.0\tpmodul13_2311104076.exe (process 13656) 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 . . .

```