

**TUGAS PENDAHULUAN
MODUL 13**



Disusun Oleh :

Izzaty Zahara Br Barus – 23111040452

Kelas :

SE-07-02

Dosen :

Yudha Islami Sulistya

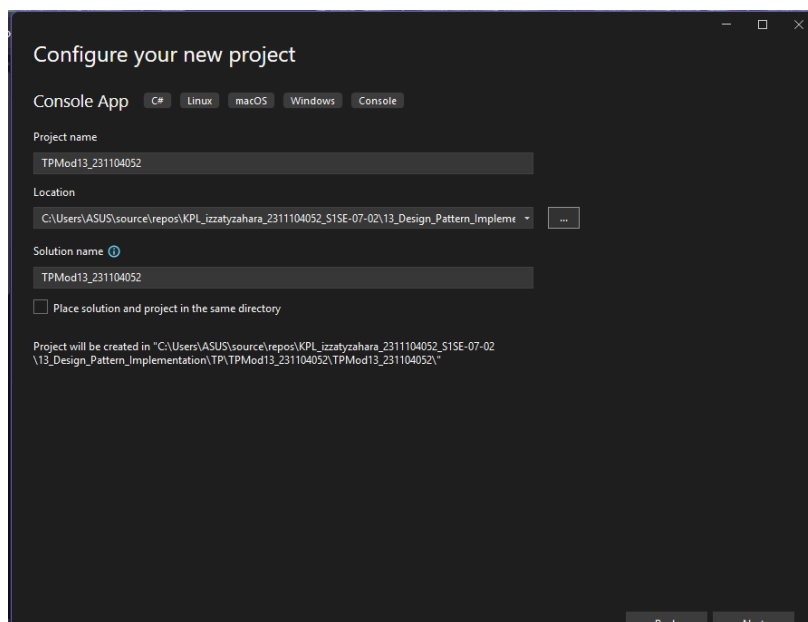
**PROGRAM STUDI SOFTWARE ENGINEERING
DIREKTORAT KAMPUS PURWOKERTO
TELKOM UNIVERSITY PURWOKERTO
2025**

I. Link Github

- https://github.com/Izzaaaaaaaaaa/KPL_izzatyzahara_2311104052_S1SE-07-02.git

II. Alur Pengerjaan

1. Membuat Project Console APP TPModul13_2311104052



2. Memasukkan Code pada Class “WeatherSystem.cs”

```
using System;  
  
using System.Collections.Generic;  
  
namespace TPMod13_2311104052  
{
```

```
// Interface Observer
public interface IObservable
{
    void Update(int temperature);
}

// Subject (Observable)
public class WeatherData
{
    private List<IObservable> observers = new
List<IObservable>();
    private int temperature;

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

    public void Detach(IObservable observer)
    {
        observers.Remove(observer);
    }
}
```

```
public void SetTemperature(int value)
{
    temperature = value;
    Notify();
}

private void Notify()
{
    foreach (var observer in observers)
    {
        observer.Update(temperature);
    }
}

// Concrete Observer
public class TemperatureDisplay : IObservable
{
    public void Update(int temperature)
    {
        Console.WriteLine($"[Display] Suhu sekarang:
```

```
{temperature}°C");  
    }  
}  
}
```

1. Class WeatherData

Class WeatherData merupakan inti dari sistem, yaitu berperan sebagai Subject (Publisher) dalam pola Observer. Di dalamnya terdapat sebuah list dari objek IObserver yang mewakili para subscriber (observer) yang ingin mendapatkan informasi saat suhu berubah. Class ini menyediakan method Attach() untuk menambahkan observer, Detach() untuk menghapus observer, dan SetTemperature() untuk mengubah nilai suhu sekaligus memicu notifikasi ke semua observer yang terdaftar. Saat suhu diubah, method Notify() akan dipanggil untuk memberitahu seluruh observer dengan cara memanggil method Update() milik masing-masing observer. Dengan pendekatan ini, class WeatherData tidak perlu tahu detail dari observer yang berlangganan, sehingga tercipta hubungan loose coupling antara pengirim data dan penerimanya.

2. Interface dan Class IObserver / TemperatureDisplay

Interface `IObserver` mendefinisikan kontrak bagi semua observer, yaitu harus memiliki method `Update(int temperature)` yang akan dipanggil oleh `WeatherData`. Kemudian, class `TemperatureDisplay` merupakan implementasi konkret dari `IObserver`, yang artinya class ini adalah Observer (Subscriber). Dalam method `Update()`, observer menerima data suhu terbaru dan menampilkannya ke layar menggunakan `Console.WriteLine`. Dengan adanya pemisahan peran seperti ini, setiap observer bisa memiliki perilaku berbeda saat menerima update, misalnya selain tampilan suhu, bisa juga ditambahkan notifikasi atau logging ke file.

3. Memasukkan code pada class “Program.cs”

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using TPMMod12_2311104052; // Namespace dari Form1.cs

namespace UnitTestProject1
{
    [TestClass]
    using System;
    using TPMMod13_2311104052;

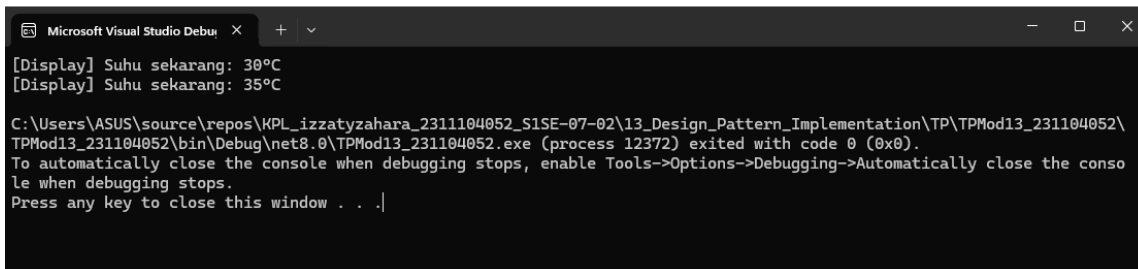
    class Program
```

```
{  
    static void Main(string[] args)  
    {  
        WeatherData weather = new WeatherData();  
        TemperatureDisplay display = new TemperatureDisplay();  
  
        weather.Attach(display);  
  
        weather.SetTemperature(30); // Output: [Display] Suhu  
sekarang: 30°C  
        weather.SetTemperature(35); // Output: [Display] Suhu  
sekarang: 35°C  
    }  
}
```

Class Program merupakan titik masuk utama dari aplikasi dan berfungsi sebagai pengendali alur eksekusi sistem. Di dalamnya terdapat method `Main()`, yang pertama kali dijalankan saat program dimulai. Pada method ini, objek `WeatherData` dibuat sebagai representasi dari subject (penerbit data), dan objek `TemperatureDisplay` dibuat sebagai observer (yang akan menerima notifikasi perubahan data). Melalui pemanggilan method `Attach()`, observer didaftarkan ke subject agar dapat

menerima pembaruan saat terjadi perubahan data suhu. Setelah itu, suhu diatur menggunakan method `SetTemperature()`, yang tidak hanya mengubah nilai internal, tetapi juga secara otomatis memicu method `Notify()` untuk memberitahu semua observer yang terdaftar. Hasilnya, observer `TemperatureDisplay` akan menerima nilai suhu baru melalui method `Update()` dan langsung menampilkannya ke konsol. Dengan cara ini, class Program memperlihatkan proses kerja lengkap dari Observer Pattern mulai dari pembuatan objek, pendaftaran observer, hingga notifikasi dan respons yang terjadi ketika data berubah.

III. Hasil Output



```
Microsoft Visual Studio Debu  X + -
[Display] Suhu sekarang: 30°C
[Display] Suhu sekarang: 35°C

C:\Users\ASUS\source\repos\KPL_izzatyzahara_2311104052_S1SE-07-02\13_Design_Pattern_Implementation\TP\TPMod13_231104052\
TPMod13_231104052\bin\Debug\net8.0\TPMod13_231104052.exe (process 12372) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .|
```

IV. Kesimpulan

Implementasi Observer Pattern dalam project `TPMod13_2311104052` menggunakan dua class utama berhasil menunjukkan konsep komunikasi satu-ke-banyak secara otomatis antara objek subject dan observer. `WeatherData` sebagai subject mampu memberitahu seluruh observer setiap kali data berubah, tanpa harus tahu bagaimana observer bekerja secara spesifik.

Pola ini sangat berguna untuk membangun sistem modular dan fleksibel, terutama dalam aplikasi yang bersifat event-driven atau real-time seperti dashboard cuaca, sistem monitoring, dan lain-lain. Dengan struktur dua class saja, konsep Observer Pattern tetap bisa diterapkan secara sederhana namun tetap mencerminkan prinsip desain perangkat lunak yang baik.