

**LAPORAN PENDAHULUAN
MODUL 13**



Disusun Oleh :

Zaenarif Putra 'Ainurdin – 2311104049

Kelas :

SE-07-02

Dosen :

Yudha Islami Sulistya

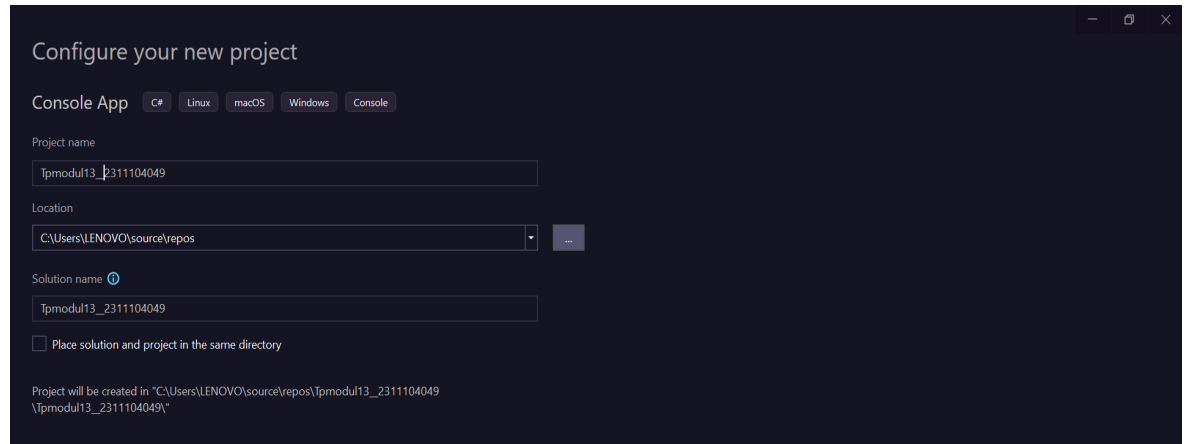
**PROGRAM STUDI SOFTWARE ENGINEERING
DIREKTORAT KAMPUS PURWOKERTO
TELKOM UNIVERSITY
PURWOKERTO
2025**

I. Link Github

https://github.com/zaenarifputra/KPL_Zaenarif-Putra-Ainurdin_2311104049_S1SE-07-02/tree/bed4d4f6329eb688e05f70f26632be0e54982495/10_Library_Construction/TP/Tpmodul10_2311104049

II. Alur Pengerjaan

1. Membuat project baru yang dimana menggunakan Console App yang dimana diberi nama “TpModul13_2311104049”

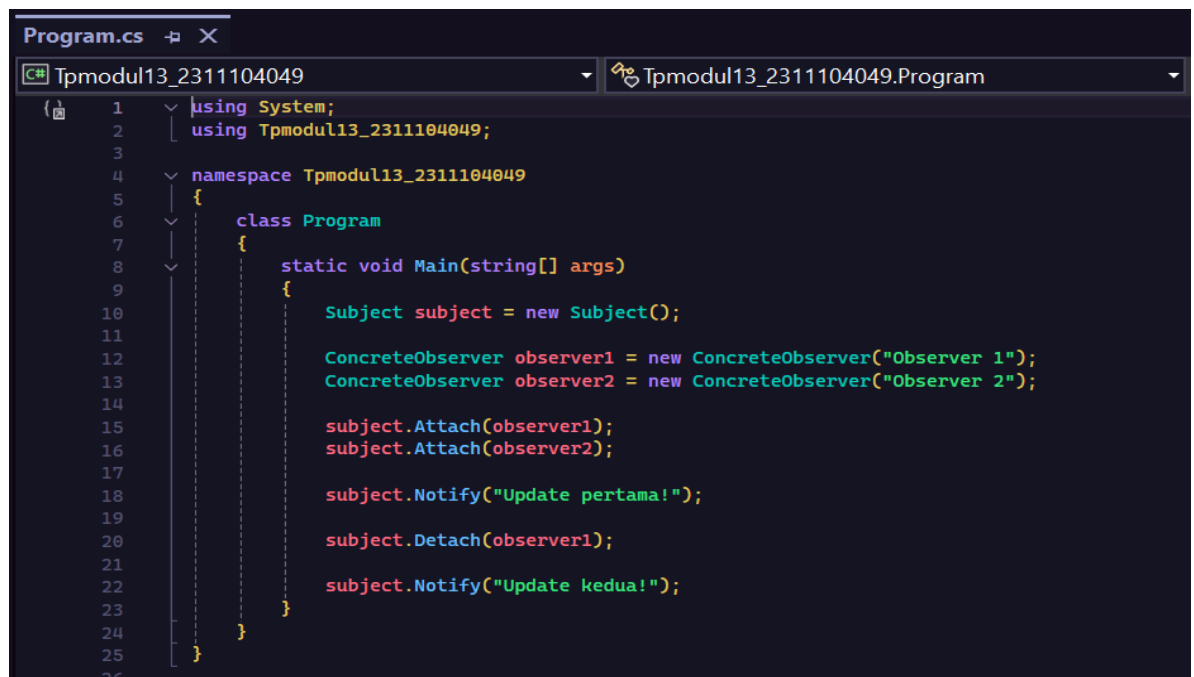


2. setelah membuat project kemudian melakukan pengerjaan terhadap penggunaan Design Pattern “Observer” dalam program yang diberi nama “[Observation.cs](#)”, berikut penjelasan sederhana dari syntaxnya dan juga screenshoot syntaxnya :

```
7 namespace Tpmodul13_2311104049
8 {
9     public interface IObserver
10     {
11         void Update(string message);
12     }
13     public interface ISubject
14     {
15         void Attach(IObserver observer);
16         void Detach(IObserver observer);
17         void Notify(string message);
18     }
19     public class Subject : ISubject
20     {
21         private List<IObserver> _observers = new List<IObserver>();
22
23         public void Attach(IObserver observer)
24         {
25             _observers.Add(observer);
26         }
27
28         public void Detach(IObserver observer)
29         {
30             _observers.Remove(observer);
31         }
32
33         public void Notify(string message)
34         {
35             foreach (var observer in _observers)
36             {
37                 observer.Update(message);
38             }
39         }
40     }
41     public class ConcreteObserver : IObserver
42     {
43         private string _name;
44
45         public ConcreteObserver(string name)
46         {
47             _name = name;
48         }
49
50         public void Update(string message)
51         {
52             Console.WriteLine($"{_name} menerima notifikasi: {message}");
53             Console.ReadLine();
54         }
55     }
56 }
```

Syntax di atas adalah penerapan dari konsep Observer. Pola ini digunakan untuk menciptakan sistem di mana satu objek (Subject) dapat memberi tahu banyak objek lain (Observer) ketika terjadi perubahan. Dalam kode ini, `IObserver` adalah antarmuka yang mendefinisikan metode `Update` yang harus dimiliki oleh setiap observer. `ISubject` adalah antarmuka untuk objek yang dapat diawasi, yang memiliki metode `Attach`, `Detach`, dan `Notify` untuk menambah, menghapus, dan memberi tahu observer. Kelas `Subject` menyimpan daftar observer dan akan memanggil metode `Update` milik observer saat ada pesan baru melalui metode `Notify`. Kelas `ConcreteObserver` mewakili objek yang menerima notifikasi, dan ketika metode `Update` dipanggil, ia akan menampilkan pesan di konsol. Tujuan dari pola ini adalah agar komponen saling terhubung dengan fleksibel tanpa ketergantungan langsung, sehingga memudahkan pemeliharaan dan pengembangan sistem.

3. Setelah menyelesaikan penerapan Design Pattern “Observer” selanjutnya membuat sebuah implementasi yang bisa menjalankan penerapan dari program [Observation.cs](#) yaitu membuat sebuah [program.cs](#) berikut syntax dan juga penjelasan sederhana agar bisa menerapkan konsep Design Pattern :



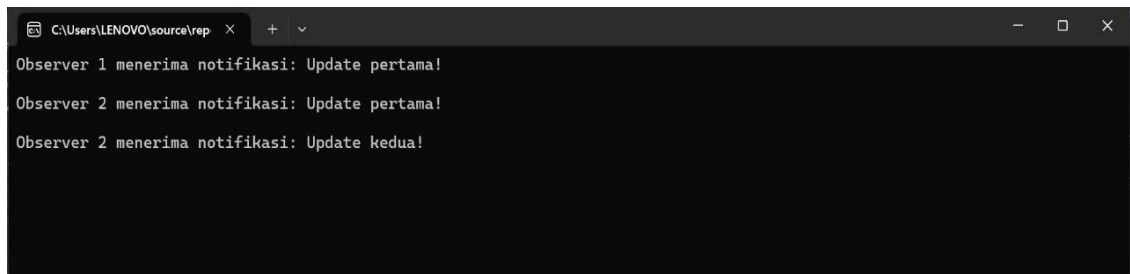
```
Program.cs
1  using System;
2  using Tpmodule13_2311104049;
3
4  namespace Tpmodule13_2311104049
5  {
6      class Program
7      {
8          static void Main(string[] args)
9          {
10             Subject subject = new Subject();
11
12             ConcreteObserver observer1 = new ConcreteObserver("Observer 1");
13             ConcreteObserver observer2 = new ConcreteObserver("Observer 2");
14
15             subject.Attach(observer1);
16             subject.Attach(observer2);
17
18             subject.Notify("Update pertama!");
19
20             subject.Detach(observer1);
21
22             subject.Notify("Update kedua!");
23         }
24     }
25 }
```

Syntax di atas adalah penerapan dari konsep Observer. Pola ini digunakan untuk menciptakan sistem di mana satu objek (Subject) dapat memberi tahu banyak objek lain (Observer) ketika terjadi perubahan. Dalam kode ini, `IObserver` adalah antarmuka yang mendefinisikan metode `Update` yang harus dimiliki oleh setiap observer. `ISubject` adalah antarmuka untuk objek yang dapat diawasi, yang memiliki metode `Attach`, `Detach`, dan `Notify` untuk menambah, menghapus, dan memberi tahu observer. Kelas `Subject` menyimpan daftar observer dan akan memanggil metode `Update` milik observer saat ada pesan baru melalui metode `Notify`. Kelas `ConcreteObserver` mewakili objek yang menerima notifikasi, dan ketika metode `Update` dipanggil, ia akan menampilkan pesan di konsol. Tujuan dari pola ini adalah agar komponen saling terhubung dengan fleksibel tanpa ketergantungan langsung, sehingga memudahkan pemeliharaan dan pengembangan sistem.

4. setelah semua implementasi yang dibutuhkan sudah berhasil di implementasikan semua selanjutnya melakukan push menuju cloud github yang dimana untuk melakukan push bisa melakukannya dengan perintah `git add .`, `git commit -m "mengimplementasikan Design Pattern Observer"` dan kemudian terakhir melakukan `git push -u origin master`. jika berhasil maka akan di tampilkan seperti gambar berikut :



III. Hasil Running & Kesimpulan



Kesimpulannya bahwa penerapan Design Pattern Observer sangat bermanfaat dalam menciptakan sistem yang fleksibel dan terstruktur, terutama ketika dibutuhkan hubungan satu-ke-banyak antar objek. Dengan memisahkan antara komponen utama (Program.cs) dan logika Observer (Observation.cs), kita dapat mengimplementasikan sistem yang mudah diperluas tanpa mengganggu bagian kode lain. Observer pattern memungkinkan objek-objek (observer) untuk selalu mendapatkan informasi terbaru dari satu pusat data (subject) tanpa perlu pengecekan manual, karena pembaruan dikirim secara otomatis. Implementasi ini sangat berguna untuk kasus nyata seperti sistem notifikasi, dashboard informasi, atau aplikasi real-time. Praktikum ini juga menekankan pentingnya penggunaan interface dan pemisahan tanggung jawab kode untuk mendukung prinsip desain perangkat lunak yang baik, seperti low coupling dan high cohesion, sehingga kode menjadi lebih mudah untuk dipelihara dan dikembangkan di masa depan.