

LAPORAN PRAKTIKUM
TUGAS PENDAHULUAN 13



Nama :

Aulia Jasifa Br Ginting 2311104060

S1SE-07-02

Dosen :

Yudha Islami Sulistya, S.Kom., M.Kom

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025

I. LINK GITHUB

https://github.com/auliajsf06/KPL_Aulia-Jasifa-Br-Ginting_2311104060_SE-07-02

II. IMPLEMENTASI DAN PEMAHAMAN DESIGN PATTERN OBSERVER

- a. Pada project yang telah dibuat sebelumnya, tambahkan kode yang mirip atau sama dengan contoh kode yang diberikan di halaman web tersebut
Syntax Class “Program.cs”

```
Program.cs [X]
C# TPModul13_2311104060 RefactoringGuru.DesignPatterns.Observer
1 using System;
2 using System.Collections.Generic;
3 using System.Threading;
4
5 namespace RefactoringGuru.DesignPatterns.Observer.Conceptual
6 {
7     public interface IObservable
8     {
9         // Receive update from subject
10        void Update(ISubject subject);
11    }
12
13    public interface ISubject
14    {
15        // Attach an observer to the subject.
16        void Attach(IObservable observer);
17
18        // Detach an observer from the subject.
19        void Detach(IObservable observer);
20
21        // Notify all observers about an event.
22        void Notify();
23    }
```

```
Program.cs [X]
C# TPModul13_2311104060 RefactoringGuru.DesignPatterns.Observer.Conceptual Notif
24
25 // The Subject owns some important state and notifies observers when the
26 // state changes.
27 public class Subject : ISubject
28 {
29     // For the sake of simplicity, the Subject's state, essential to all
30     // subscribers, is stored in this variable.
31     public int State { get; set; } = -1;
32
33     // List of subscribers. In real life, the list of subscribers can be
34     // stored more comprehensively (categorized by event type, etc.).
35     private List<IObservable> _observers = new List<IObservable>();
36
37     // The subscription management methods.
38     public void Attach(IObservable observer)
39     {
40         Console.WriteLine("Subject: Attached an observer.");
41         this._observers.Add(observer);
42     }
43
44     public void Detach(IObservable observer)
45     {
46         this._observers.Remove(observer);
47         Console.WriteLine("Subject: Detached an observer.");
48     }
```

```
Program.cs X
TPModul13_2311104060 RefactoringGuru.DesignPatterns.Observer.Conce Update(ISubject

49
50 // Trigger an update in each subscriber.
51 public void Notify()
52 {
53     Console.WriteLine("Subject: Notifying observers...");
54
55     foreach (var observer in _observers)
56     {
57         observer.Update(this);
58     }
59
60
61 // Usually, the subscription logic is only a fraction of what a Subject
62 // can really do. Subjects commonly hold some important business logic,
63 // that triggers a notification method whenever something important is
64 // about to happen (or after it).
65 public void SomeBusinessLogic()
66 {
67     Console.WriteLine("\nSubject: I'm doing something important.");
68     this.State = new Random().Next(0, 10);
69
70     Thread.Sleep(15);
71
72     Console.WriteLine("Subject: My state has just changed to: " + this.State);
73     this.Notify();
74 }
75 }
```

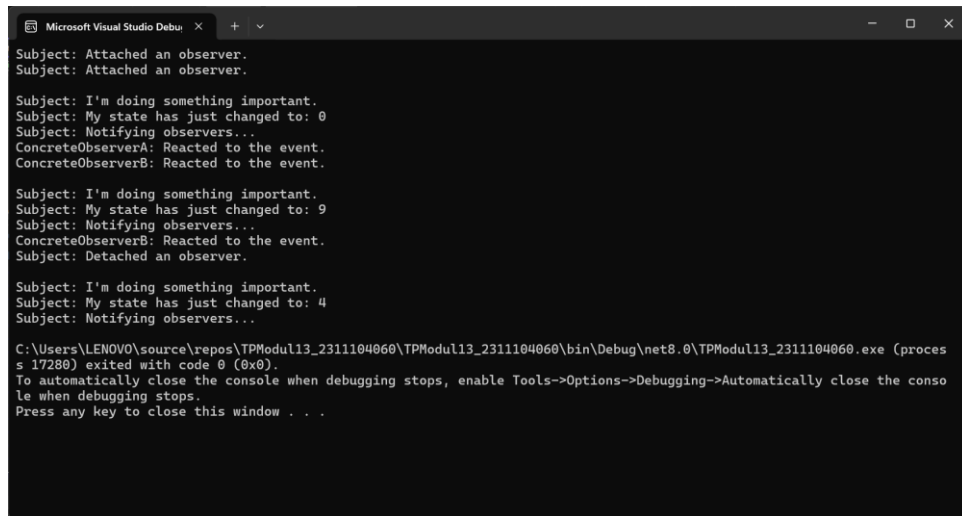
```
Program.cs X
TPModul13_2311104060 RefactoringGuru.DesignPatterns.Observer.Conce Main(strin

76
77 // Concrete Observers react to the updates issued by the Subject they had
78 // been attached to.
79 class ConcreteObserverA : IObserver
80 {
81     public void Update(ISubject subject)
82     {
83         if ((subject as Subject).State < 3)
84         {
85             Console.WriteLine("ConcreteObserverA: Reacted to the event.");
86         }
87     }
88 }
89
90 class ConcreteObserverB : IObserver
91 {
92     public void Update(ISubject subject)
93     {
94         if ((subject as Subject).State == 0 || (subject as Subject).State >= 2)
95         {
96             Console.WriteLine("ConcreteObserverB: Reacted to the event.");
97         }
98     }
99 }
100 }
```

```
Program.cs X
TPModul13_2311104060 RefactoringGuru.DesignPatterns.Observer.Conce

100
101 class Program
102 {
103     static void Main(string[] args)
104     {
105         // The client code.
106         var subject = new Subject();
107         var observerA = new ConcreteObserverA();
108         subject.Attach(observerA);
109
110         var observerB = new ConcreteObserverB();
111         subject.Attach(observerB);
112
113         subject.SomeBusinessLogic();
114         subject.SomeBusinessLogic();
115
116         subject.Detach(observerB);
117
118         subject.SomeBusinessLogic();
119     }
120 }
121 }
```

- b. Jalankan program tersebut dan pastikan tidak ada error pada saat project dijalankan



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

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: I'm doing something important.
Subject: My state has just changed to: 9
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: 4
Subject: Notifying observers...

C:\Users\LENOVO\source\repos\TPModul13_2311104060\TPModul13_2311104060\bin\Debug\net8.0\TPModul13_2311104060.exe (process 17280) 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 . . .
```

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

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

Mengimpor namespace yang diperlukan. System untuk fungsi dasar, System.Collections.Generic untuk menggunakan koleksi generik seperti List, dan System.Threading untuk menggunakan fitur multithreading seperti Thread.Sleep.

```
namespace RefactoringGuru.DesignPatterns.Observer.Conceptual
{
```

Mendefinisikan namespace untuk mengorganisir kode. Ini membantu dalam menghindari konflik nama dan mengelompokkan kode yang terkait.

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

Mendefinisikan antarmuka IObserver yang memiliki metode Update. Metode ini akan dipanggil oleh ISubject untuk memberi tahu pengamat tentang perubahan.

```

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();
}

```

Mendefinisikan antarmuka ISubject yang memiliki metode untuk mengelola pengamat: Attach untuk menambahkan pengamat, Detach untuk menghapus pengamat, dan Notify untuk memberi tahu semua pengamat tentang perubahan.

```

public class Subject : ISubject
{

```

Mendefinisikan kelas Subject yang mengimplementasikan antarmuka ISubject. Kelas ini akan menyimpan status dan mengelola pengamat.

```

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

```

Mendefinisikan properti State yang menyimpan status penting dari Subject. Nilai awalnya diatur ke -0.

```

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

```

Mendeklarasikan daftar _observers untuk menyimpan pengamat yang terdaftar. Ini adalah koleksi dari objek yang mengimplementasikan IObserver.

```

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

```

Metode Attach untuk menambahkan pengamat ke daftar _observers dan mencetak pesan ke konsol.

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

Metode Detach untuk menghapus pengamat dari daftar _observers dan mencetak pesan ke konsol.

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

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

Metode Notify untuk memberi tahu semua pengamat tentang perubahan. Ini mencetak pesan dan memanggil metode Update pada setiap pengamat yang terdaftar.

```
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();
}
```

Metode SomeBusinessLogic yang mensimulasikan logika bisnis. Ini mengubah State ke nilai acak antara 0 dan 9, menunggu selama 15 milidetik, mencetak status baru, dan memanggil Notify untuk memberi tahu pengamat.

```

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

```

Kelas ConcreteObserverA yang mengimplementasikan IObservable. Metode Update akan merespons jika State dari Subject kurang dari 3.

```

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.");
        }
    }
}

```

Kelas ConcreteObserverB yang juga mengimplementasikan IObservable. Metode Update akan merespons jika State dari Subject sama dengan 0 atau lebih

III. KESIMPULAN

Kode di atas mengimplementasikan pola desain Observer dalam C#. Pola ini memungkinkan objek (disebut Subject) untuk memberi tahu pengamat (Observers) tentang perubahan statusnya. Dalam implementasi ini, terdapat dua antarmuka, IObservable dan ISubject, yang mendefinisikan metode untuk mengelola pengamat dan memberi tahu mereka tentang perubahan. Kelas Subject menyimpan status penting dan memiliki metode untuk menambahkan, menghapus, dan memberi tahu pengamat. Dua kelas konkret, ConcreteObserverA dan ConcreteObserverB, mengimplementasikan logika respons terhadap pembaruan dari Subject berdasarkan kondisi tertentu dari status. Dalam metode Main, objek Subject dan pengamatnya

dibuat, dan logika bisnis dijalankan untuk menunjukkan bagaimana pengamat bereaksi terhadap perubahan status. Dengan demikian, kode ini menunjukkan bagaimana pola Observer dapat digunakan untuk memisahkan logika bisnis dari logika tampilan, memungkinkan pengelolaan yang lebih baik terhadap interaksi antara objek dalam aplikasi.