

Nama : Candra Dinata  
Nim : 2311104061  
Kelas : SE0702

## **MENJELASKAN SALAH SATU DESIGN PATTERN**

### **A. Contoh Kondisi Penggunaan Design Pattern "Observer"**

Design pattern *Observer* cocok digunakan saat kita memiliki satu objek (disebut subject) yang perlu memberitahu banyak objek lain (observers) saat terjadi perubahan.

Contoh:

Sistem notifikasi aplikasi cuaca. Ketika data cuaca diperbarui (misalnya suhu berubah), maka beberapa komponen seperti tampilan suhu, grafik, dan notifikasi pengguna semuanya harus diberi tahu secara otomatis dan diperbarui.

### **B. Langkah-Langkah Mengimplementasikan Design Pattern "Observer"**

1. Buat antarmuka Observer  
Mendefinisikan metode yang akan dipanggil saat subject mengalami perubahan (misalnya `update()`).
2. Buat antarmuka Subject  
Mendefinisikan metode untuk menambahkan, menghapus, dan memberi notifikasi ke observer.
3. Implementasikan kelas Subject konkrit  
Kelas ini menyimpan daftar observer dan memanggil `update()` pada setiap observer ketika terjadi perubahan.
4. Implementasikan kelas Observer konkrit  
Kelas ini mengimplementasikan antarmuka Observer dan merespon perubahan dari Subject sesuai kebutuhan.

### **C. Kelebihan dan Kekurangan Design Pattern "Observer"**

Kelebihan:

- Loose coupling (terpisah dengan longgar): Observer tidak perlu mengetahui detail internal subject.
- Fleksibel dan dapat diperluas: Mudah menambah observer baru tanpa mengubah subject.
- Mendukung event-driven programming: Cocok untuk sistem real-time atau dinamis.

Kekurangan:

- Ketergantungan tersembunyi: Sulit untuk melacak hubungan antar objek, terutama saat observer banyak.
- Potensi masalah performa: Jika ada banyak observer, notifikasi dapat menjadi berat.
- Kesulitan debugging: Debugging bisa lebih rumit karena efek samping tersebar ke banyak objek.

## IMPLEMENTASI DAN PEMAHAMAN DESIGN PATTERN OBSERVER

```
1 using System;
2 using System.Collections.Generic;
3 using System.Threading;
4
5 namespace RefactoringGuru.DesignPatterns.Observer.Conceptual
6 {
7     public interface IObserver
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(IObserver observer);
17
18        // Detach an observer from the subject.
19        void Detach(IObserver observer);
20
21        // Notify all observers about an event.
22        void Notify();
23    }
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<IObserver> _observers = new List<IObserver>();
36
37        // The subscription management methods.
38        public void Attach(IObserver observer)
39        {
40            Console.WriteLine("Subject: Attached an observer.");
41            this._observers.Add(observer);
42        }
43
44        public void Detach(IObserver observer)
45        {
46            this._observers.Remove(observer);
47            Console.WriteLine("Subject: Detached an observer.");
48        }
49
50        // Trigger an update in each subscriber.
51        public void Notify()
52        {
53            Console.WriteLine("Subject: Notifying observers...");
54            foreach (var observer in _observers)
55            {
56                observer.Update(this);
57            }
58        }
59
60        // Usually, the subscription logic is only a fraction of what a Subject
61        // can really do. Subjects commonly hold some important business logic,
62        // that triggers a notification method whenever something important is
63        // about to happen (or after it).
64        public void SomeBusinessLogic()
65        {
66            Console.WriteLine("\nSubject: I'm doing something important.");
67            this.State = new Random().Next(0, 10);
68            Thread.Sleep(15);
69            Console.WriteLine("Subject: My state has just changed to: " + this.State);
70            this.Notify();
71        }
72    }
73
74    // Concrete Observers react to the updates issued by the Subject they had
75    // been attached to.
76    class ConcreteObserverA : IObserver
77    {
78        public void Update(ISubject subject)
79        {
80            if ((subject as Subject).State < 3)
81            {
82                Console.WriteLine("ConcreteObserverA: Reacted to the event.");
83            }
84        }
85    }
86
87    class ConcreteObserverB : IObserver
88    {
89        public void Update(ISubject subject)
90        {
91            if ((subject as Subject).State == 0 || (subject as Subject).State >= 2)
92            {
93                Console.WriteLine("ConcreteObserverB: Reacted to the event.");
94            }
95        }
96    }
97
98    class Program
99    {
100        static void Main(string[] args)
101        {
102            // The client code.
103            var subject = new Subject();
104            var observerA = new ConcreteObserverA();
105            subject.Attach(observerA);
106
107            var observerB = new ConcreteObserverB();
108            subject.Attach(observerB);
109
110            subject.SomeBusinessLogic();
111            subject.SomeBusinessLogic();
112            subject.Detach(observerB);
113            subject.SomeBusinessLogic();
114        }
115    }
116 }
```

**static void Main(string[] args)**

Ini adalah entry point dari aplikasi. Ketika program dijalankan, eksekusi dimulai dari method Main.  
string[] args adalah parameter opsional untuk menerima argumen dari command line.

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

Membuat instance baru dari class Subject.

Subject adalah kelas utama yang akan mengelola status dan memberi tahu observer saat status berubah.

**var observerA = new ConcreteObserverA();**

**subject.Attach(observerA);**

Membuat instance observerA dari ConcreteObserverA, yaitu observer pertama.

Kemudian observerA didaftarkan (attached) ke subject agar menerima notifikasi setiap kali subject berubah.

**var observerB = new ConcreteObserverB();**

**subject.Attach(observerB);**

Membuat instance observerB dari ConcreteObserverB, yaitu observer kedua.

observerB juga didaftarkan ke subject, sehingga juga akan menerima notifikasi saat subject berubah.

**subject.SomeBusinessLogic();**

Memanggil method SomeBusinessLogic() pada subject.

Di dalam method ini:

subject akan mengubah nilai State secara acak.

Setelah perubahan, subject akan memanggil Notify() untuk memberi tahu semua observer yang terdaftar.

**subject.SomeBusinessLogic();**

Menjalankan kembali SomeBusinessLogic().

Sama seperti sebelumnya, akan terjadi perubahan state dan semua observer yang masih terdaftar akan diberi tahu.

**subject.Detach(observerB);**

Melepaskan (unsubscribing) observerB dari subject.

Setelah baris ini, observerB tidak akan menerima notifikasi lagi dari subject.

**subject.SomeBusinessLogic();**

Menjalankan lagi SomeBusinessLogic().

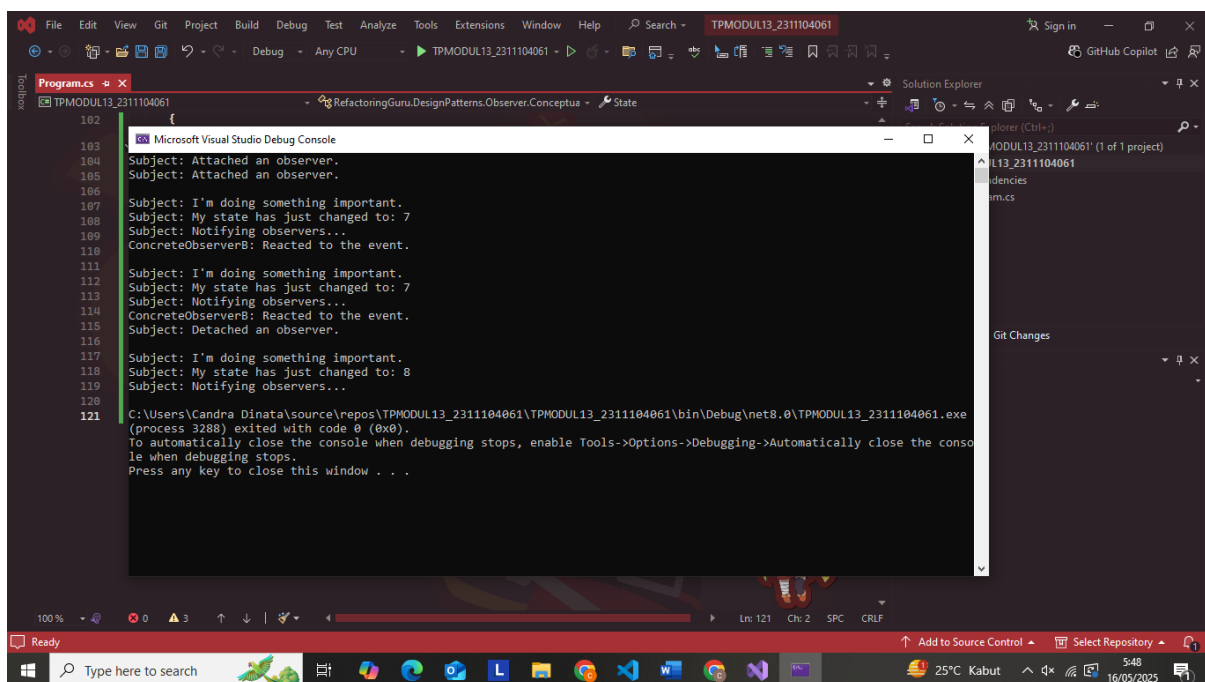
Kali ini hanya observerA yang akan menerima notifikasi, karena observerB sudah dilepas.

## Kesimpulan:

Metode Main ini mendemonstrasikan pola desain Observer:

1. Objek Subject mengelola observer.
2. ConcreteObserverA dan ConcreteObserverB merespons perubahan status Subject.
3. Client (method Main) bisa menambahkan atau menghapus observer kapan saja.

## Output:



```
102 {
103     Microsoft Visual Studio Debug Console
104     Subject: Attached an observer.
105     Subject: Attached an observer.
106
107     Subject: I'm doing something important.
108     Subject: My state has just changed to: 7
109     Subject: Notifying observers...
110     ConcreteObserverB: Reacted to the event.
111
112     Subject: I'm doing something important.
113     Subject: My state has just changed to: 7
114     Subject: Notifying observers...
115     ConcreteObserverB: Reacted to the event.
116     Subject: Detached an observer.
117
118     Subject: I'm doing something important.
119     Subject: My state has just changed to: 8
120     Subject: Notifying observers...
121
122     C:\Users\Candra Dinata\source\repos\TPMODUL13_2311104061\TPMODUL13_2311104061\bin\Debug\net8.0\TPMODUL13_2311104061.exe
    (process 3288) 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 . . .
```