

**TUGAS PENDAHULUAN
KONSTRUKSI PERANGKAT LUNAK**

PERTEMUAN 10



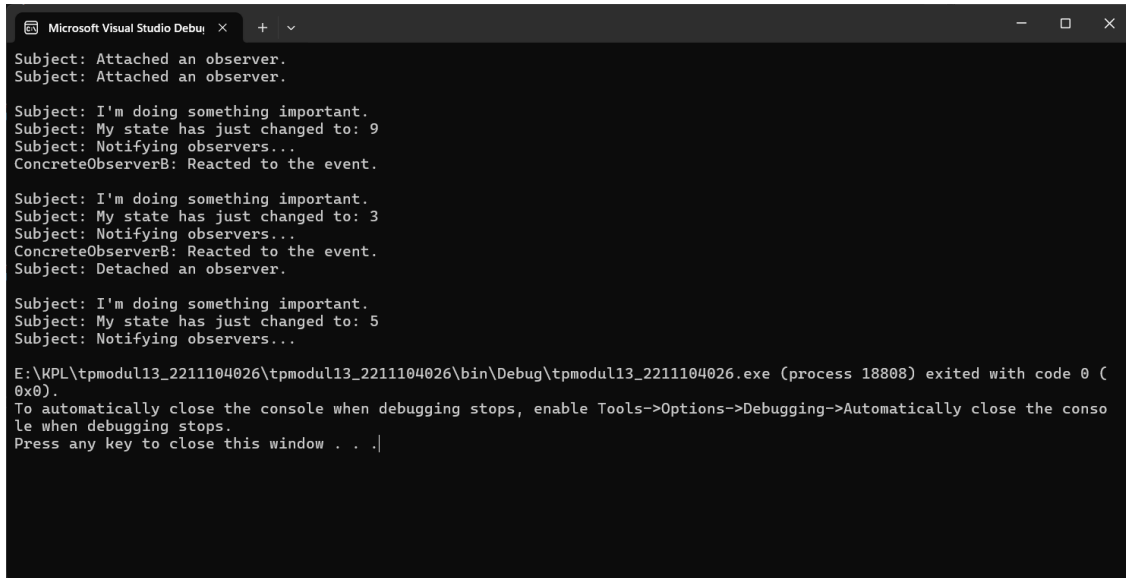
**Disusun Oleh :
Muhammad Abdul Aziz
2211104026
SE0601**

**Asisten Praktikum :
Naufal El Kamil Aditya Pratama Rahman
Imelda**

**Dosen Pengampu :
Yudha Islami Sulistya, S.Kom., M.Cs.**

**PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025**

1. Screenshot hasil run



```
Microsoft Visual Studio Debug Console
Subject: Attached an observer.
Subject: Attached an observer.

Subject: I'm doing something important.
Subject: My state has just changed to: 9
Subject: Notifying observers...
ConcreteObserverB: Reacted to the event.

Subject: I'm doing something important.
Subject: My state has just changed to: 3
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: 5
Subject: Notifying observers...

E:\KPL\tpmodul13_2211104026\tpmodul13_2211104026\bin\Debug\tpmodul13_2211104026.exe (process 18808) 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 . . .|
```

2. Penjelasan singkat dari kode implementasi yang dibuat (beserta screenshot dari potongan source code yang dijelaskan).

Design Pattern

A. Contoh kondisi penggunaan Observer Pattern:

Observer pattern dapat digunakan pada **sistem notifikasi**, misalnya:

- Aplikasi berita yang memberitahu pengguna saat ada berita baru.
- Sistem monitoring cuaca yang mengupdate suhu di berbagai lokasi.

B. Langkah-langkah implementasi Observer Pattern:

1. Buat **Subject** (publisher) yang menyimpan list dari **Observers** (subscriber).
2. Subject menyediakan method untuk:
 - Menambahkan observer.
 - Menghapus observer.
 - Memberi notifikasi ke semua observer saat ada perubahan.
3. Buat interface **IObserver** untuk observer.
4. Implementasikan class-class observer yang menerapkan **IObserver**.
5. Subject akan memanggil **Update()** milik observer saat ada perubahan data.

C. Kelebihan dan Kekurangan:

Kelebihan:

- Mengikuti prinsip **Loose Coupling** (tidak bergantung langsung).
- **Mudah diperluas**, tinggal tambah observer baru.
- Cocok untuk **event-driven systems**.

Kekurangan:

- Bisa sulit dilacak jika terlalu banyak observer.
- Risiko **memory leaks** jika observer tidak dihapus dengan benar.

Source Code :

Program.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Threading;
4
5  namespace RefactoringGuru.DesignPatterns.Observer.Conceptual
6  {
7      8 references
8      public interface IObserver
9      {
10         // Receive update from subject
11         3 references
12         void Update(ISubject subject);
13     }
14
15     4 references
16     public interface ISubject
17     {
18         // Attach an observer to the subject.
19         3 references
20         void Attach(IObserver observer);
21
22         // Detach an observer from the subject.
23         2 references
24         void Detach(IObserver observer);
25
26         // Notify all observers about an event.
27         2 references
28         void Notify();
29     }
30
31     // The Subject owns some important state and notifies observers when the
32     // state changes.
33     4 references
34     public class Subject : ISubject
35     {
36         // For the sake of simplicity, the Subject's state, essential to all
37         // subscribers, is stored in this variable.
38         5 references
39         public int State { get; set; } = -0;
40
41         // List of subscribers. In real life, the list of subscribers can be
42         // stored more comprehensively (categorized by event type, etc.).
43         private List<IObserver> _observers = new List<IObserver>();
44
45         // The subscription management methods.
46         3 references
47         public void Attach(IObserver observer)
48         {
49             Console.WriteLine("Subject: Attached an observer.");
50             this._observers.Add(observer);
51         }
52
53         2 references
54         public void Detach(IObserver observer)
55         {
56             this._observers.Remove(observer);
57             Console.WriteLine("Subject: Detached an observer.");
58         }
59
60         // Trigger an update in each subscriber.
61         2 references
62         public void Notify()
63         {
64             Console.WriteLine("Subject: Notifying observers...");
65
66             foreach (var observer in _observers)
67             {
68                 observer.Update(this);
69             }
70         }
71
72         // Usually, the subscription logic is only a fraction of what a Subject
73         // can really do. Subjects commonly hold some important business logic,
74         // that triggers a notification method whenever something important is
75         // about to happen (or after it).
76         3 references
77         public void SomeBusinessLogic()
78         {
79             Console.WriteLine("\nSubject: I'm doing something important.");
80             this.State = new Random().Next(0, 10);
81
82             Thread.Sleep(15);
83
84             Console.WriteLine("Subject: My state has just changed to: " + this.State);
85             this.Notify();
86         }
87     }
88
89     // Concrete Observers react to the updates issued by the Subject they had
90     // been attached to.
91     1 reference
92     class ConcreteObserverA : IObserver
93     {
94         2 references
95         public void Update(ISubject subject)
96         {
97             if ((subject as Subject).State < 3)
98             {
99                 Console.WriteLine("ConcreteObserverA: Reacted to the event.");
100             }
101         }
102     }
103 }
```

```

86     }
87     }
88 }
89
90 1 reference
91 class ConcreteObserverB : IObserver
92 {
93     2 references
94     public void Update(ISubject subject)
95     {
96         if ((subject as Subject).State == 0 || (subject as Subject).State >= 2)
97         {
98             Console.WriteLine("ConcreteObserverB: Reacted to the event.");
99         }
100     }
101
102 0 references
103 class Program
104 {
105     0 references
106     static void Main(string[] args)
107     {
108         // The client code.
109         var subject = new Subject();
110         var observerA = new ConcreteObserverA();
111         subject.Attach(observerA);
112
113         var observerB = new ConcreteObserverB();
114         subject.Attach(observerB);
115
116         subject.SomeBusinessLogic();
117         subject.SomeBusinessLogic();
118
119         subject.Detach(observerB);
120
121         subject.SomeBusinessLogic();
122     }
123 }

```

Penjelasan

- Isubject (Publisher Interface)

```

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

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

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

```

- Ini adalah antarmuka dari publisher (si pemberi notifikasi).
- Menyediakan method untuk menambahkan, menghapus, dan memberi notifikasi ke observer.

- Subject (Publisher)

```
public class Subject : ISubject
{
    // For the sake of simplicity, the Subject's state, essential to all
    // subscribers, is stored in this variable.
    5 references
    public int State { get; set; } = -0;

    // List of subscribers. In real life, the list of subscribers can be
    // stored more comprehensively (categorized by event type, etc.).
    private List<IObserver> _observers = new List<IObserver>();

    // The subscription management methods.
    3 references
    public void Attach(IObserver observer)
    {
        Console.WriteLine("Subject: Attached an observer.");
        this._observers.Add(observer);
    }

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

    // Trigger an update in each subscriber.
    2 references
    public void Notify()
    {
        Console.WriteLine("Subject: Notifying observers...");

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

    // Usually, the subscription logic is only a fraction of what a Subject
    // can really do. Subjects commonly hold some important business logic,
    // that triggers a notification method whenever something important is
    // about to happen (or after it).
    3 references
    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();
    }
}
```

- Menyimpan data penting (State).
- Menyimpan daftar observer dalam list.
- Saat terjadi perubahan state, akan memanggil Notify() ke semua observer.

- IObserver (Observer Interface)

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

- Setiap observer harus mengimplementasikan method Update().

- ConcreteObserverA dan ConcreteObserverB

```

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

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

```

- Mereka adalah implementasi nyata dari observer.
- Mereka akan melakukan sesuatu berdasarkan kondisi dari subject.State.

- Program.Main() (Client)

```

class Program
{
    0 references
    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();
    }
}

```

- Di sinilah semuanya dijalankan.
- Program membuat objek subject dan menambahkan observer ke dalamnya.
- SomeBusinessLogic() mengubah State lalu memicu Notify() ke observer.