

**TUGAS PENDAHULUAN MODUL 13
KONSTRUKSI PERANGKAT LUNAK**

DESIGN PATTERN IMPLEMENTATION



**DISUSUN OLEH:
KHOLIFAH DINA
2211104004**

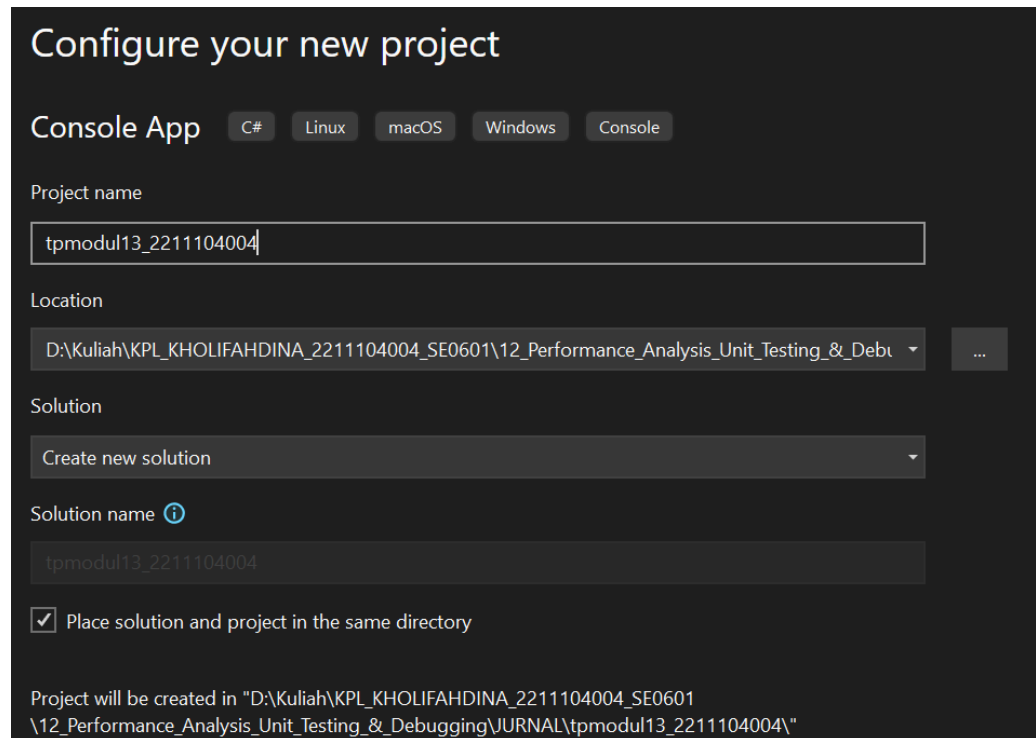
SE 06 01

**S1 REKAYASA PERANGKAT LUNAK
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
2025**

1. MEMBUAT PROJECT GUI

Buka IDE misalnya dengan Visual Studio

- a. Misalnya menggunakan Visual Studio, buatlah solution baru dengan nama tpmodul13_NIM. Project console



2. MENJELASKAN SALAH SATU DESIGN PATTERN

Buka halaman web <https://refactoring.guru/design-patterns/catalog> kemudian baca design pattern dengan nama “Observer”, dan jawab pertanyaan berikut ini (dalam Bahasa Indonesia):

- a. Berikan salah satu contoh kondisi dimana design pattern “Observer” dapat digunakan

Observer pattern cocok digunakan ketika terdapat objek yang statusnya dipantau oleh beberapa objek lainnya. Misalnya, **Aplikasi cuaca**: Jika suhu atau kelembaban berubah, maka semua tampilan yang menampilkan informasi cuaca (misalnya dashboard, widget, notifikasi) harus ikut diperbarui secara otomatis.

- b. Berikan penjelasan singkat mengenai langkah-langkah dalam mengimplementasikan design pattern “Observer”

Langkah-langkah implementasi Observer Pattern

- i. Subject: Buat class yang menyimpan data dan daftar observer.
- ii. Observer Interface: Definisikan interface yang harus diimplementasikan oleh semua observer.
- iii. Concrete Observer: Class-class yang berlangganan (subscribe) ke subject, dan merespons ketika subject berubah.

- iv. Attach/Detach: Subject menyediakan metode untuk menambahkan atau menghapus observer.
 - v. Notify: Ketika ada perubahan pada subject, method Notify() dipanggil untuk memberitahu semua observer.
- c. Berikan kelebihan dan kekurangan dari design pattern “Observer”

Kelebihan Observer Pattern

- Memungkinkan loose coupling (tidak saling ketergantungan kuat antar objek).
- Memudahkan pengembangan sistem yang modular dan scalable.
- Menyederhanakan komunikasi antar objek.

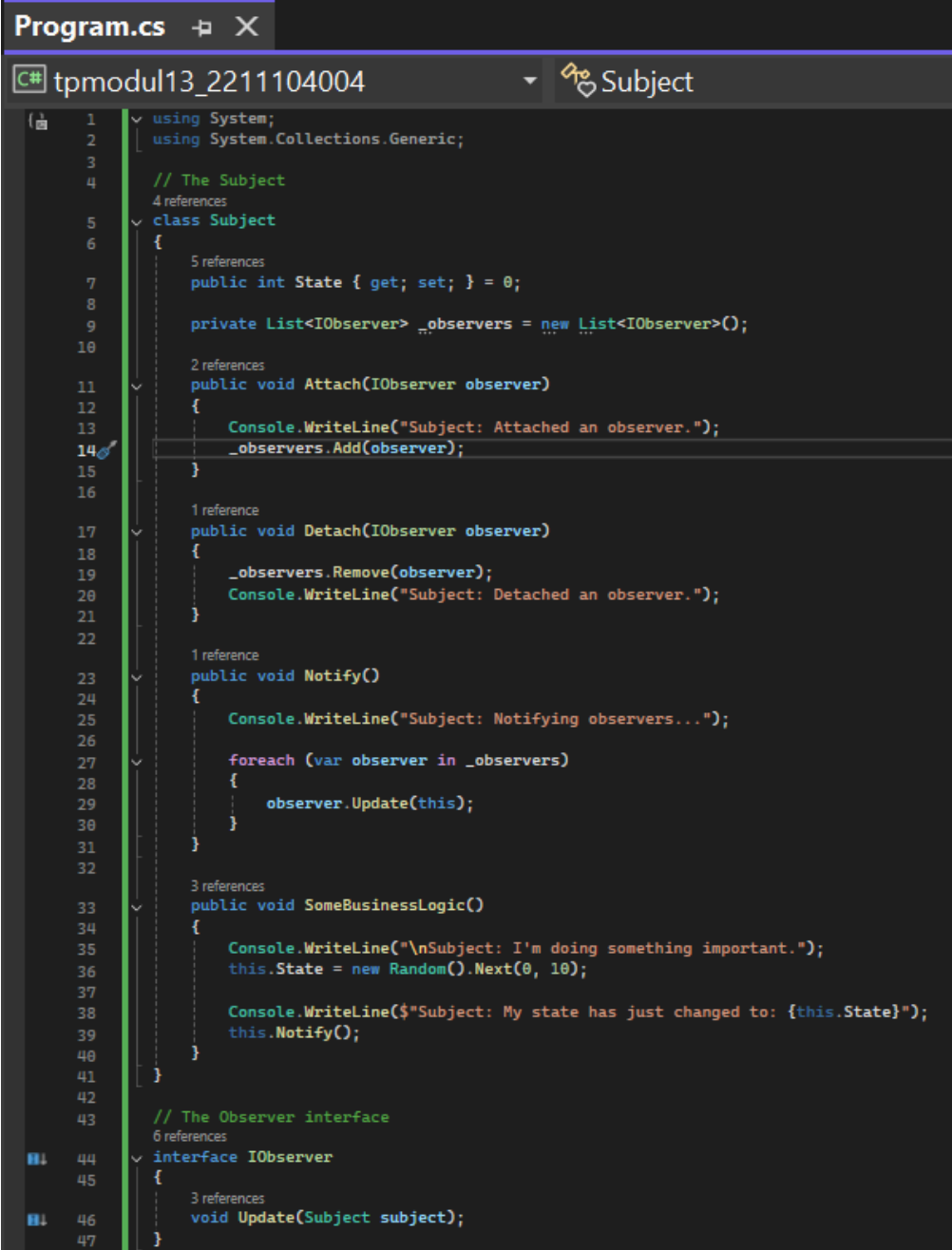
Kekurangan Observer Pattern

- Jika observer terlalu banyak, dapat menimbulkan overhead.
- Debugging dan tracing bisa menjadi sulit karena banyaknya notifikasi yang dikirim.
- Ketika tidak dikelola dengan baik, bisa terjadi memory leaks (karena observer tidak dilepas dari subject).

3. IMPLEMENTASI DAN PEMAHAMAN DESIGN PATTERN OBSERVER

Buka halaman web berikut <https://refactoring.guru/design-patterns/observer> dan scroll ke bagian “Code Examples”, pilih kode yang akan dilihat misalnya C# dan ikuti langkah-langkah berikut:

- a. Pada project yang telah dibuat sebelumnya, tambahkan kode yang mirip atau sama dengan contoh kode yang diberikan di halaman web tersebut



```
Program.cs X
C# tpmodule13_2211104004 Subject
1 using System;
2 using System.Collections.Generic;
3
4 // The Subject
4 references
5 class Subject
6 {
7     5 references
8     public int State { get; set; } = 0;
9
10     private List<IObservable> _observers = new List<IObservable>();
11
12     2 references
13     public void Attach(IObservable observer)
14     {
15         Console.WriteLine("Subject: Attached an observer.");
16         _observers.Add(observer);
17     }
18
19     1 reference
20     public void Detach(IObservable observer)
21     {
22         _observers.Remove(observer);
23         Console.WriteLine("Subject: Detached an observer.");
24     }
25
26     1 reference
27     public void Notify()
28     {
29         Console.WriteLine("Subject: Notifying observers...");
30
31         foreach (var observer in _observers)
32         {
33             observer.Update(this);
34         }
35     }
36
37     3 references
38     public void SomeBusinessLogic()
39     {
40         Console.WriteLine("\nSubject: I'm doing something important.");
41         this.State = new Random().Next(0, 10);
42
43         Console.WriteLine($"Subject: My state has just changed to: {this.State}");
44         this.Notify();
45     }
46 }
47
48 // The Observer interface
49 6 references
50 interface IObservable
51 {
52     3 references
53     void Update(Subject subject);
54 }
```

```

49 // Concrete Observer A
50 class ConcreteObserverA : IObserver
51 {
52     public void Update(Subject subject)
53     {
54         if (subject.State < 3)
55         {
56             Console.WriteLine("ConcreteObserverA: Reacted to the event.");
57         }
58     }
59 }
60
61 // Concrete Observer B
62 class ConcreteObserverB : IObserver
63 {
64     public void Update(Subject subject)
65     {
66         if (subject.State == 0 || subject.State >= 2)
67         {
68             Console.WriteLine("ConcreteObserverB: Reacted to the event.");
69         }
70     }
71 }
72
73 class Program
74 {
75     static void Main(string[] args)
76     {
77         var subject = new Subject();
78
79         var observerA = new ConcreteObserverA();
80         subject.Attach(observerA);
81
82         var observerB = new ConcreteObserverB();
83         subject.Attach(observerB);
84
85         subject.SomeBusinessLogic();
86         subject.SomeBusinessLogic();
87
88         subject.Detach(observerB);
89
90         subject.SomeBusinessLogic();
91     }
92 }
93

```

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

```

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: 5
Subject: Notifying observers...
ConcreteObserverB: Reacted to the event.

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

D:\Kuliah\KPL_KHOLIFAHADINA_2211104004_SE0601\12_Performance_Analysis_Unit_Testing_&_Debugging\JURNAL\tpmodul13_2211104004\bin\Debug\net8.0\tpmodul13_2211104004.exe (process 2404) 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”

Pada method Main, pertama-tama dibuat sebuah objek dari class Subject yang akan menjadi pusat pengamatan. Kemudian dibuat dua observer, yaitu ConcreteObserverA dan ConcreteObserverB, yang masing-masing di-*attach* ke subject menggunakan method Attach(). Ini berarti kedua observer akan menerima notifikasi jika terjadi perubahan pada state dari subject. Setelah itu, method SomeBusinessLogic() dipanggil dua kali, yang akan mengubah state dari subject secara acak dan memicu pemanggilan method Notify() untuk memberitahu semua observer yang terdaftar.

Selanjutnya, observer B di-*detach* dari subject sehingga ia tidak akan menerima notifikasi lagi. Terakhir, method SomeBusinessLogic() dipanggil sekali lagi, namun hanya observer A yang akan menerima notifikasi karena observer B sudah dilepas. Melalui alur ini, diperlihatkan bagaimana observer pattern bekerja untuk memisahkan logika pengamatan dan reaksi antar objek secara fleksibel dan dinamis.