

**TUGAS PENDAHULUAN
KONSTRUKSI PERANGKAT LUNAK**

**MODUL XIII
DESIGN PATTERN IMPLEMENTATION**



Disusun Oleh :

Rizky Hanifa Afania

2211104017

SE-06-01

Dosen Pengampu :

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

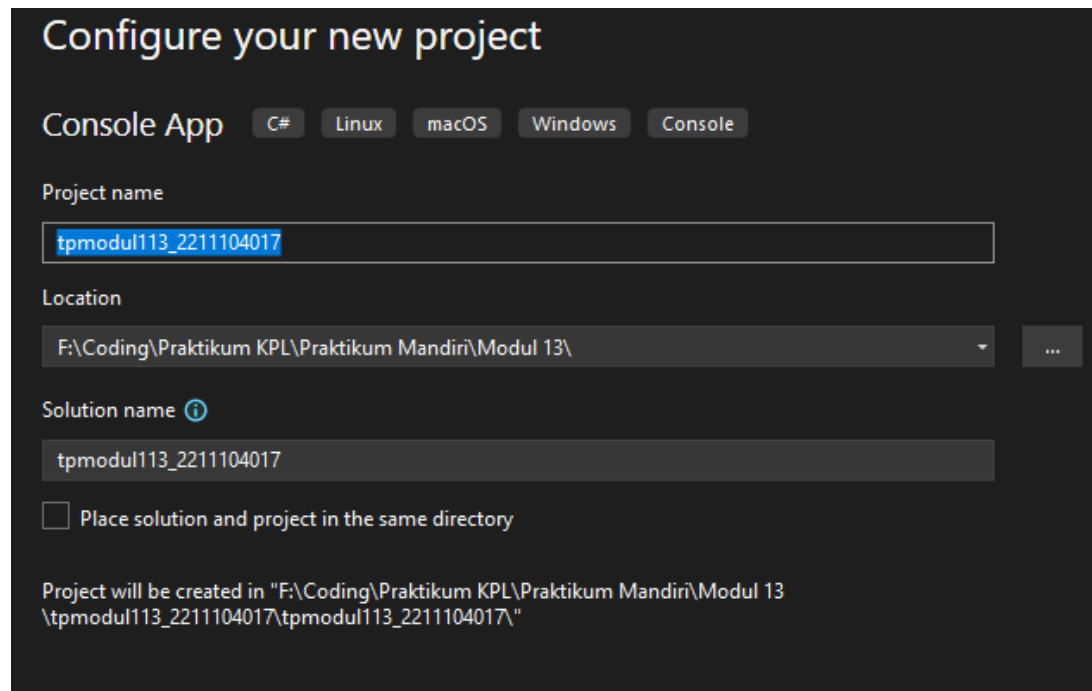
**PROGRAM STUDI S1 SOFTWARE ENGINEERING
TELKOM UNIVERSITY PURWOKERTO**

TUGAS PENDAHULUAN 13

1. MEMBUAT PROJECT GUI BARU

Buka IDE misalnya dengan Visual Studio

- A. Misalnya menggunakan Visual Studio, buatlah project baru dengan nama tpmodul113_NIM
- B. Project yang dibuat bisa berupa console atau sejenisnya



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?

Jawab:

Observer Pattern dapat digunakan misalnya ketika *Customer* ingin mendapatkan notifikasi dari *Store* tentang ketersediaan produk tertentu, misalnya laptop keluaran terbaru. Daripada pelanggan harus bolak-balik datang ke toko untuk mengecek, atau toko mengirimkan notifikasi ke semua pelanggan (termasuk yang

tidak tertarik), Observer Pattern memungkinkan toko hanya mengirimkan pemberitahuan kepada pelanggan yang secara spesifik telah *subscribe* untuk mendapatkan notifikasi tersebut.

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

Jawab:

Berikut langkah-langkah singkat implementasi Observer Pattern pada studi kasus *subscribe* di toko laptop:

1. Identifikasi objek yang menjadi publisher (subjek) dan subscriber (pengamat).
2. Membuat antarmuka subscriber, minimal mempunyai metode `update()` yang akan dipanggil saat ada perubahan.
3. Membuat antarmuka publisher yang menyediakan metode `attach()`, `detach()`, dan `notify()` untuk mengatur daftar subscriber.
4. Mengimplementasikan daftar langganan, publisher akan menyimpan daftar subscriber dan memanggil metode `update()` milik mereka saat terjadi event.
5. Mengimplementasikan subscriber konkret, subscriber akan merespons notifikasi dengan mengambil data dari publisher (jika diperlukan).
6. Di bagian client, buat dan hubungkan objek publisher dengan subscriber secara dinamis sesuai kebutuhan.

- C. Berikan kelebihan dan kekurangan dari design pattern “Observer”

Jawab:

Kelebihan

- **Loose Coupling (Keterkaitan Longgar)**

Publisher dan Subscriber tidak saling bergantung secara langsung. Keduanya hanya terhubung melalui antarmuka (interface), sehingga memudahkan pengembangan dan pemeliharaan.

- **Mendukung Open/Closed Principle**

Dapat menambahkan tipe Subscriber baru tanpa mengubah kode Publisher, sehingga sistem lebih fleksibel terhadap perubahan.

- **Dapat Dinamis di Waktu Runtime**

Objek dapat mendaftar (subscribe) atau keluar (unsubscribe) kapan saja selama program berjalan, menjadikan pola ini sangat fleksibel.

- **Pola Komunikasi yang Efisien untuk Banyak Objek**

Cocok digunakan ketika satu objek perlu memberi tahu banyak objek lain saat terjadi perubahan status atau event tertentu.

Kekurangan:

- **Sulit Melacak Aliran Notifikasi**

Karena hubungan antara Publisher dan Subscriber bersifat tidak langsung, sulit untuk menelusuri urutan atau penyebab notifikasi dalam sistem besar.

- **Potensi Masalah Kinerja**

Jika Publisher memiliki banyak Subscriber, proses notifikasi bisa menjadi lambat, terutama jika masing-masing Subscriber menjalankan operasi berat.

- **Subscriber Bisa Menjadi Bergantung pada Publisher Secara Tidak Langsung**

Meski antar kelas tidak terikat langsung, implementasi notifikasi bisa menciptakan ketergantungan terselubung melalui data atau metode tertentu.

- **Masalah Urutan Eksekusi**

Publisher tidak menjamin urutan notifikasi ke setiap Subscriber, yang bisa menjadi masalah jika urutan eksekusi penting.

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:

</> Code Examples



- Pada project yang telah dibuat sebelumnya, tambahkan kode yang mirip atau sama dengan contoh kode yang diberikan di halaman web tersebut
- Jalankan program tersebut dan pastikan tidak ada error pada saat project dijalankan
- Jelaskan tiap baris kode yang terdapat di bagian method utama atau “main”

Source Code:

```
modul113_2211104017
Observer

1  using System;
2  using System.Collections.Generic;
3  using System.Threading;
4
5  namespace ObserverPatternExample
6  {
7      // Interface Observer
8      public interface IObserver
9      {
10         void Update(ISubject subject);
11     }
12
13     // Interface Subject (Publisher)
14     public interface ISubject
15     {
16         void Attach(IObserver observer);
17         void Detach(IObserver observer);
18         void Notify();
19     }
20
21     // Concrete Subject
22     public class Subject : ISubject
23     {
24         public int State { get; set; } = 0;
25
26         private List<IObserver> _observers = new List<IObserver>();
27
28         public void Attach(IObserver observer)
29         {
30             Console.WriteLine("Subject: Attached an observer.");
31             _observers.Add(observer);
32         }
33
34         public void Detach(IObserver observer)
35         {
36             _observers.Remove(observer);
37             Console.WriteLine("Subject: Detached an observer.");
38         }
39     }
40 }
```

```

40 public void Notify()
41 {
42     Console.WriteLine("Subject: Notifying observers...");
43     foreach (var observer in _observers)
44     {
45         observer.Update(this);
46     }
47 }
48
49 // 3 references
50 public void SomeBusinessLogic()
51 {
52     Console.WriteLine("\nSubject: I'm doing something important.");
53     this.State = new Random().Next(0, 10);
54
55     Thread.Sleep(15);
56
57     Console.WriteLine($"Subject: My state has just changed to: {this.State}");
58     this.Notify();
59 }
60
61 // Concrete Observer A
62 // 1 reference
63 public class ConcreteObserverA : IObserver
64 {
65     // 2 references
66     public void Update(ISubject subject)
67     {
68         if ((subject as Subject).State < 3)
69         {
70             Console.WriteLine("ConcreteObserverA: Reacted to the event.");
71         }
72     }
73 }

```

```

73 // Concrete Observer B
74 // 1 reference
75 public class ConcreteObserverB : IObserver
76 {
77     // 2 references
78     public void Update(ISubject subject)
79     {
80         if ((subject as Subject).State == 0 || (subject as Subject).State >= 2)
81         {
82             Console.WriteLine("ConcreteObserverB: Reacted to the event.");
83         }
84     }
85 }
86
87 // Main Program
88 // 0 references
89 class Program
90 {
91     // 0 references
92     static void Main(string[] args)
93     {
94         var subject = new Subject();
95
96         var observerA = new ConcreteObserverA();
97         subject.Attach(observerA);
98
99         var observerB = new ConcreteObserverB();
100         subject.Attach(observerB);
101
102         subject.SomeBusinessLogic(); // Observer A dan B merespon
103         subject.SomeBusinessLogic(); // Observer A dan B merespon
104
105         subject.Detach(observerB); // Observer B tidak lagi merespon
106         subject.SomeBusinessLogic(); // Hanya Observer A yang merespon
107     }
108 }

```

Hasil:

```
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: 4
Subject: Notifying observers...
ConcreteObserverB: Reacted to the event.

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: Detached an observer.

Subject: I'm doing something important.
Subject: My state has just changed to: 2
Subject: Notifying observers...
ConcreteObserverA: Reacted to the event.

F:\Coding\Praktikum KPL\Praktikum Mandiri\Modul 13\tpmodul113_2211104017\tpmodul113_2211104017.exe (process 13896) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->De
le when debugging stops.
Press any key to close this window . . .
```

Penjelasan di bagian method utama atau “main”:

- `var subject = new Subject();`
Digunakan untuk membuat objek subject, yaitu objek utama (publisher) yang memiliki state dan akan menginformasikan perubahan tersebut ke para observer.
- `var observerA = new ConcreteObserverA();`
`subject.Attach(observerA);`
 - a. Digunakan untuk membuat observerA, sebuah objek yang ingin mengetahui setiap perubahan pada subject.
 - b. Baris kedua mendaftarkan observerA ke subject, sehingga sekarang observerA akan mendapat notifikasi jika terjadi perubahan.
- `var observerB = new ConcreteObserverB();`
`subject.Attach(observerB);`
 - a. Digunakan untuk membuat observerB, observer kedua yang juga ingin mengetahui perubahan dari subject.

- b. Baris kedua mendaftarkan observerB ke subject, sehingga observerB juga akan mendapatkan notifikasi.
- subject.SomeBusinessLogic();
 - a. Menjalankan fungsi yang melakukan proses bisnis pada subject.
 - b. Subject akan mengubah nilainya (state) secara acak, dan kemudian memberitahu semua observer yang terdaftar (yaitu observerA dan observerB).
- subject.Detach(observerB);

Berfungsi untuk menghapus observerB dari daftar observer. Mulai dari titik ini, observerB tidak akan lagi menerima notifikasi dari subject.