

TP MODUL 13
DESIGN PATTERN IMPLEMENTATION



Nama :

Dhiemas Tulus Ikhsan (2311104046)

Dosen :

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

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK
TELKOM UNIVERSITY PURWOKERTO
2025

1. 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:

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

Salah satu contoh kondisi di mana design pattern "Observer" dapat digunakan adalah dalam aplikasi cuaca. Misalkan kita memiliki sebuah aplikasi yang menampilkan informasi cuaca terkini. Pengguna dapat mendaftar untuk menerima pembaruan cuaca secara real-time. Ketika data cuaca baru tersedia (misalnya, suhu, kelembaban, atau prakiraan hujan), aplikasi cuaca akan memberi tahu semua pengguna yang terdaftar (subscribers) tentang perubahan tersebut. Dengan cara ini, pengguna tidak perlu terus-menerus memeriksa aplikasi untuk mendapatkan informasi terbaru, dan mereka hanya akan menerima notifikasi ketika ada perubahan yang relevan.

- b. Berikan penjelasan singkat mengenai langkah-langkah dalam mengimplementasikan design pattern “Observer”
 1. Identifikasi Publisher dan Subscriber: Pisahkan logika bisnis menjadi dua bagian, di mana satu bagian akan bertindak sebagai publisher (objek yang memiliki status yang menarik) dan bagian lainnya sebagai subscriber (objek yang ingin mendapatkan notifikasi tentang perubahan status).
 2. Deklarasikan Interface Subscriber: Buat interface untuk subscriber yang setidaknya memiliki satu metode update yang akan dipanggil oleh publisher saat ada perubahan.
 3. Deklarasikan Interface Publisher: Buat interface untuk publisher yang mendeskripsikan metode untuk menambahkan dan menghapus subscriber dari daftar.
 4. Implementasikan Daftar Subscription: Tentukan dimana daftar subscription akan disimpan dan implementasikan metode untuk menambah dan menghapus subscriber. Biasanya, ini dapat dilakukan dalam kelas abstrak yang diturunkan dari interface publisher.
 5. Buat Kelas Publisher Konkret: Implementasikan kelas konkret yang mewakili publisher dan pastikan untuk memanggil metode notifikasi kepada semua subscriber saat terjadi perubahan penting.
 6. Implementasikan Metode Notifikasi di Kelas Subscriber Konkret: Kelas subscriber harus mengimplementasikan metode update untuk menangani notifikasi yang diterima dari publisher.

7. Buat dan Daftarkan Subscriber: Di bagian klien, buat objek publisher dan subscriber, lalu daftarkan subscriber ke publisher untuk menerima notifikasi.

c. Berikan kelebihan dan kekurangan dari design pattern “Observer”

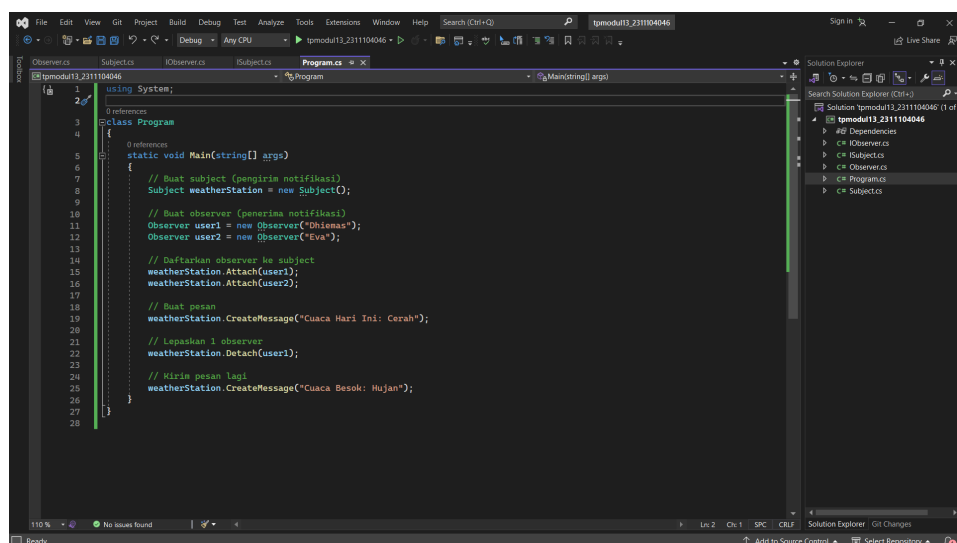
Kelebihan:

1. Prinsip Open/Closed: Anda dapat menambahkan kelas subscriber baru tanpa harus mengubah kode publisher, dan sebaliknya.
2. Hubungan Dinamis: Anda dapat mengatur hubungan antara objek pada waktu runtime, memungkinkan fleksibilitas dalam pengelolaan notifikasi.
3. Pengurangan Ketergantungan: Publisher tidak perlu mengetahui detail tentang subscriber, hanya berinteraksi melalui interface yang telah ditentukan.

Kekurangan:

1. Urutan Notifikasi Acak: Subscriber mungkin menerima notifikasi dalam urutan yang tidak terduga, yang dapat menyebabkan masalah jika urutan penting.
2. Pengelolaan Memori: Jika tidak dikelola dengan baik, daftar subscriber dapat menyebabkan kebocoran memori jika subscriber tidak dihapus dengan benar.
3. Kompleksitas: Implementasi pattern ini dapat menambah kompleksitas pada kode, terutama jika ada banyak publisher dan subscriber yang saling berinteraksi.

2. Implementasi dan Pemahaman Design Pattern Observer



```
1 using System;
2
3 class Program
4 {
5     static void Main(string[] args)
6     {
7         // Buat subject (pengirim notifikasi)
8         Subject weatherStation = new Subject();
9
10        // Buat observer (penerima notifikasi)
11        Observer user1 = new Observer("Dhienas");
12        Observer user2 = new Observer("Eva");
13
14        // Daftarkan observer ke subject
15        weatherStation.Attach(user1);
16        weatherStation.Attach(user2);
17
18        // Buat pesan
19        weatherStation.CreateMessage("Cuaca Hari Ini: Cerah");
20
21        // Lepaskan 1 observer
22        weatherStation.Detach(user1);
23
24        // Kirim pesan Lagi
25        weatherStation.CreateMessage("Cuaca Besok: Hujan");
26
27    }
28 }
```

Penjelasan tiap baris kode yang terdapat di bagian method utama atau “main”

a. Membuat objek Subject

```
Subject weatherStation = new Subject();
```

Baris ini membuat instance dari Subject, yang akan bertindak sebagai pengirim notifikasi. Dalam konteks observer pattern, ini adalah "yang diawasi" — misalnya: stasiun cuaca yang akan memberitahu perubahan cuaca ke para pengguna.

b. Membuat beberapa Observer

```
Observer user1 = new Observer("Dhiemas");  
Observer user2 = new Observer("Eva");
```

Di sini dibuat dua objek observer bernama user1 dan user2. Observer adalah pihak yang akan menerima notifikasi saat ada perubahan pada Subject.

c. Mendaftarkan observer ke subject

```
weatherStation.Attach(user1);  
weatherStation.Attach(user2);
```

Baris ini memanggil method Attach() dari Subject untuk menambahkan observer user1 dan user2 ke dalam list. Jadi, setiap kali Subject memberi update, keduanya akan diberitahu.

d. Mengirim pesan pertama

```
weatherStation.CreateMessage("Cuaca Hari Ini: Cerah");
```

Ketika CreateMessage() dipanggil, internal method Notify() secara otomatis akan memanggil method Update() pada semua observer yang sudah terdaftar, yaitu user1 dan user2.

Output-nya:

```
[Dhiemas] menerima update: Cuaca Hari Ini: Cerah  
[Eva] menerima update: Cuaca Hari Ini: Cerah
```

Ini menandakan bahwa observer mendapat update secara otomatis setelah ada perubahan data.

e. Menghapus salah satu observer

```
weatherStation.Detach(user1);
```

Baris ini memanggil method Detach() untuk menghapus user1 dari daftar observer. Setelah ini, hanya user2 yang akan menerima update berikutnya.

f. Mengirim pesan kedua

```
weatherStation.CreateMessage("Cuaca Besok: Hujan");
```

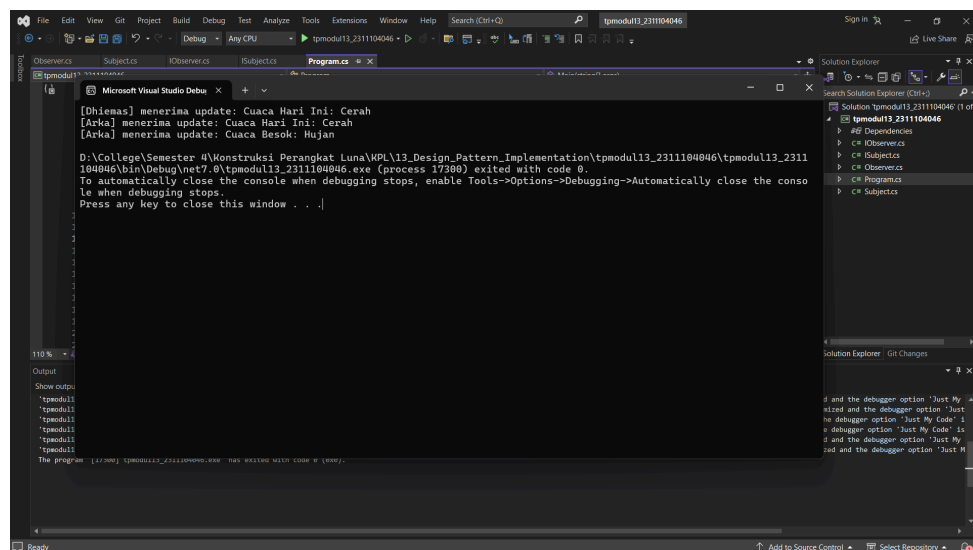
Sama seperti sebelumnya, method ini akan memicu Notify() untuk memanggil Update() pada observer yang masih aktif, dalam hal ini hanya user2.

Output-nya:

```
[Eva] menerima update: Cuaca Besok: Hujan
```

user1 tidak lagi menerima update karena sudah di-detach sebelumnya.

3. Hasil Program



4. Kesimpulan

Melalui implementasi design pattern Observer dalam project ini, dapat disimpulkan bahwa pola ini sangat efektif digunakan ketika sebuah objek perlu memberi tahu banyak objek lain tentang perubahan statusnya tanpa harus mengetahui detail dari objek-objek tersebut. Pada program ini, objek Subject berperan sebagai pusat notifikasi, sedangkan objek Observer bertindak sebagai penerima informasi yang akan melakukan respons terhadap perubahan.

Dengan pendekatan ini, kita dapat melihat bahwa saat observer (user1 dan user2) mendaftar ke Subject menggunakan Attach(), mereka akan langsung menerima notifikasi saat Subject mengalami perubahan status melalui CreateMessage(). Proses ini terjadi secara otomatis dan menunjukkan prinsip dasar dari Observer Pattern, yaitu pemantauan terhadap perubahan state secara terpisah namun terhubung.

Kemudian saat salah satu observer dihapus dari daftar menggunakan Detach(), observer tersebut tidak akan menerima update lagi, sementara observer lain tetap menerima informasi. Hal ini menunjukkan fleksibilitas dan efisiensi dari Observer Pattern dalam mengelola dependensi satu-ke-banyak yang longgar (loose coupling).

Pola desain ini sangat cocok digunakan dalam berbagai aplikasi, seperti sistem event-driven, antarmuka pengguna (UI), aplikasi cuaca, notifikasi berita, dan banyak lagi, di mana banyak objek perlu sinkron terhadap perubahan pada satu sumber data utama.

Secara keseluruhan, penerapan Observer Pattern tidak hanya memperkuat pemahaman konsep design pattern, tetapi juga memberikan gambaran nyata bagaimana arsitektur perangkat lunak yang rapi dan scalable bisa dibangun dengan prinsip pemrograman berorientasi objek.