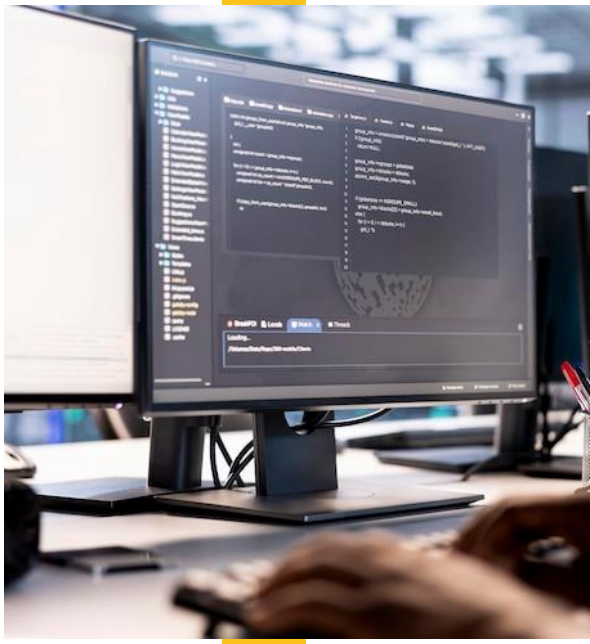
POLBAN

Pertemuan 13
Design Pattern



Yadhi Aditya Permana

D4 Teknik Informatika
Politeknik Negeri Bandung

Pertemuan Sebelumnya

- Recording Software Design
- Model-Based Design (MBD)
  - Structural Design Descriptions
  - Behavioral Design Descriptions
- Design Patterns
- Domain-Specific Languages (DSL)

POLBAN

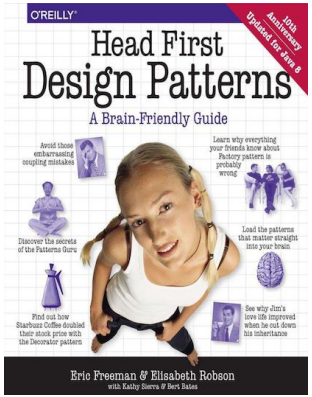
Good programmers write code.
Great programmers reuse it.

—David Thomas

co-author of The Pragmatic Programmer

Materi Pertemuan

- Definisi design pattern
- Implementasi Factory Pattern
- Implementasi Decorator Pattern
- Implementasi Observer Pattern



Referensi:
Gamma et al. (1994), *"Design Patterns: Elements of Reusable Object-Oriented Software"*.
Freeman et al. (2004). "*Head First Design Patterns*." O'Reilly Media.

Tujuan Pembelajaran

Setelah mengikuti perkuliahan ini, mahasiswa mampu:

- Memahami konsep dan prinsip dasar design pattern sebagai solusi desain berulang dalam pengembangan perangkat lunak berorientasi objek.
- Mengidentifikasi permasalahan desain perangkat lunak yang dapat diselesaikan dengan menggunakan pattern Observer, Decorator, dan Factory.
- Mengimplementasikan ketiga pattern dalam konteks studi kasus nyata menggunakan bahasa pemrograman

POLBAN

01

Design Pattern

Definisi Design Pattern

Design Pattern adalah **solusi generik yang dapat digunakan kembali** untuk mengatasi **masalah desain perangkat lunak yang sering terjadi** dalam konteks tertentu.

Design Pattern bukan kode siap pakai, tapi **pola solusi**.
Terbukti efektif karena sudah digunakan di banyak sistem.
Membantu **membentuk struktur dan perilaku sistem** dengan cara yang reusable dan maintainable.

Kelebihan design patterns (1)

Solusi Tervalidasi

Design Patterns mendokumentasikan solusi standar untuk masalah desain umum → menjadi bagian dari knowledge base desain.

Bagian dari Artefak Desain

Saat tim menggunakan design pattern (misal Observer, Factory), pola tersebut direpresentasikan dalam Class Diagram, Sequence Diagram, dsb. → menjadi recorded design artifacts.

Meningkatkan Konsistensi Desain

Dokumentasi pattern memastikan implementasi tetap konsisten, meski oleh tim berbeda.

Kelebihan design patterns (2)

- Mempermudah Komunikasi Tim
  - Menyebut pattern (misal: “Gunakan Decorator Pattern”) lebih ringkas & dipahami lintas tim daripada mendeskripsikan solusi dari awal.
- Sebagai Bagian dari Model-Based Design
  - Patterns biasanya diadopsi dalam Model-Based Design (MBD) sebagai bagian dari struktur model.
- Mendukung Reusability & Maintainability
  - Karena pattern terdokumentasi, mempermudah pemeliharaan & pengembangan di masa depan.

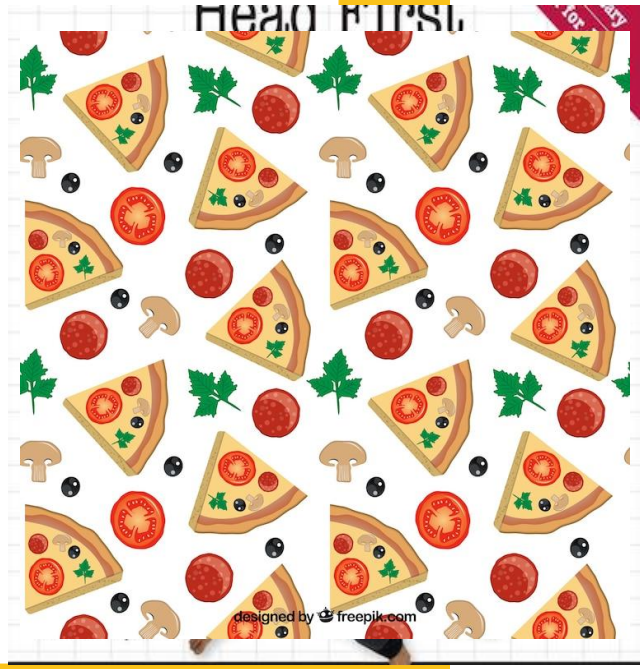
Kategori dalam design patterns

Kategori	Fungsi Utama	Patterns
Creational	Mengatur cara pembuatan objek agar lebih fleksibel dan reusable tanpa tergantung pada implementasi konkret.	Singleton, **Factory Method**, Abstract Factory, Builder, Prototype
Structural	Menjelaskan cara menyusun kelas dan objek untuk membentuk struktur sistem yang besar, efisien, dan mudah dikelola.	Adapter, Bridge, Composite, **Decorator**, Facade, Flyweight, Proxy
Behavioral	Mengelola interaksi antar objek, pola komunikasi, dan alur proses dalam sistem perangkat lunak.	**Observer**, Strategy, Command, State, Template Method, Chain of Responsibility, Mediator, Memento, Visitor, Iterator, Interpreter

POLBAN

02

Factory Pattern (Creational)



Definisi Factory Pattern

Factory Pattern adalah pola desain (design pattern) dalam pemrograman berorientasi objek yang **mengatur cara pembuatan objek**, sehingga proses tersebut:

- terpisah dari kode utama (klien),
- tidak bergantung langsung pada kelas konkret, dan
- lebih fleksibel untuk pengembangan di masa depan.

Tujuan Factory Pattern

Tujuan Utama:

- Menghindari penggunaan new langsung pada kelas konkret.
- Mengurangi ketergantungan antara kode utama (klien) dan class yang dibuat.
- Mendukung prinsip Open-Closed: kelas terbuka untuk ekstensi, tapi tertutup untuk modifikasi.
- Mempermudah ekspansi sistem saat perlu menambah jenis objek baru.

Cara Kerja Factory Pattern

Tanpa Factory (Cara Lama)

```
Pizza pizza;
if (type.equals("cheese")) {
    pizza = new CheesePizza();
}
```

Kekurangan: Harus ubah kode jika ingin jenis pizza baru.

Dengan Factory (Cara yang Benar)

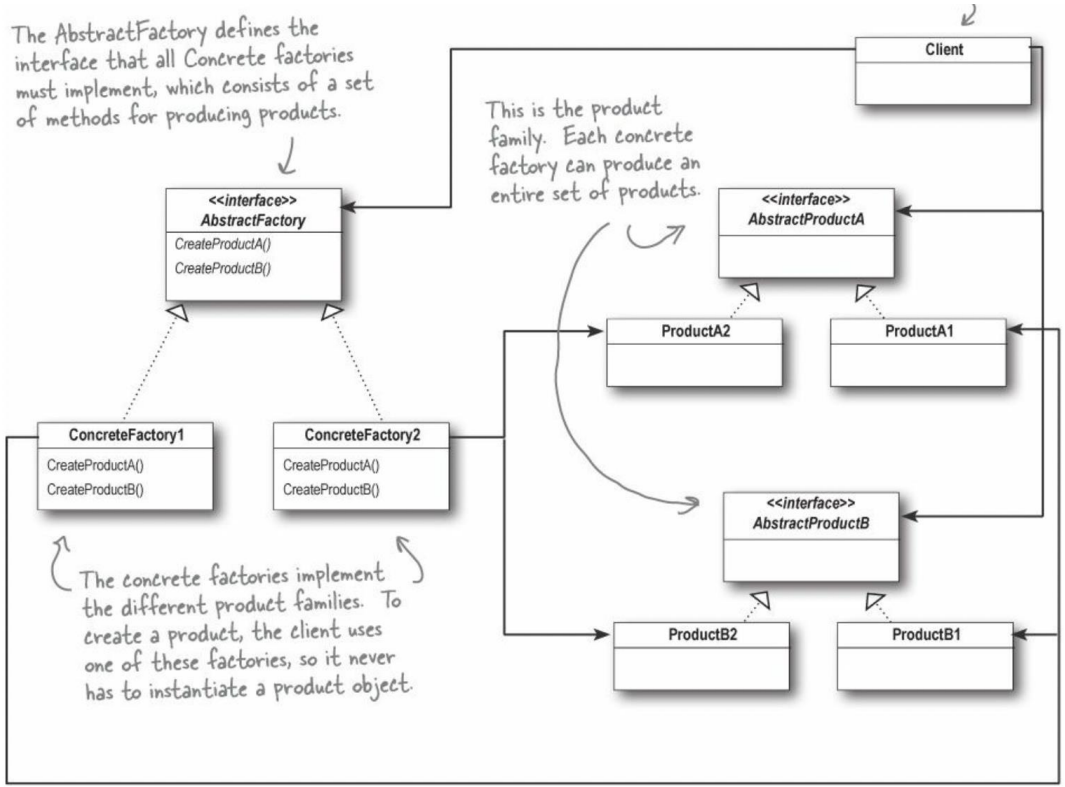
```
Pizza pizza = pizzaFactory.createPizza("cheese");
```

Keuntungan: Proses pembuatan objek diserahkan ke Factory, klien tidak tahu (dan tidak peduli) tentang new.

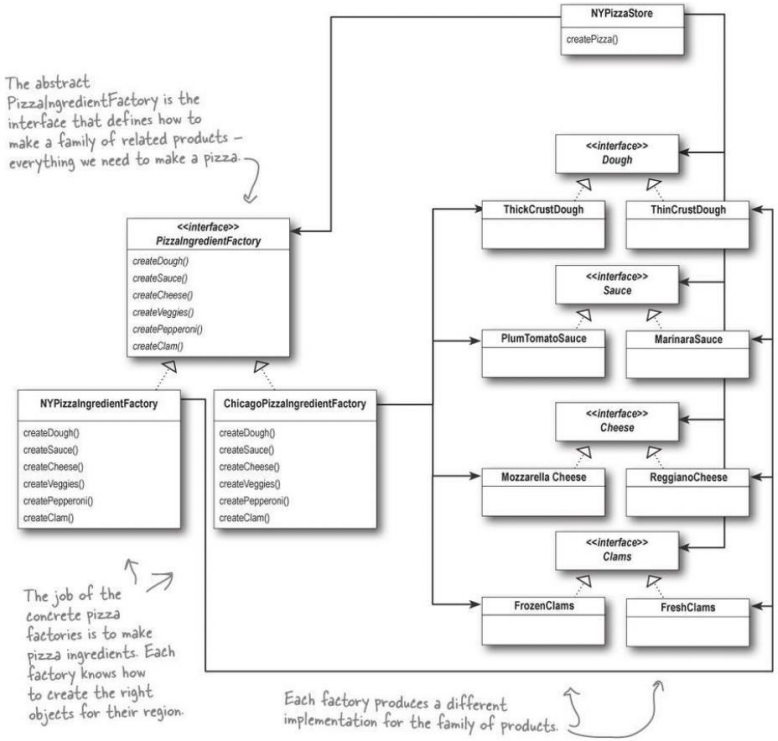
Manfaat Factory Pattern

Manfaat	Penjelasan
Loose Coupling	Kode utama tidak bergantung pada implementasi
Open for Extension	Mudah menambahkan jenis baru
Testability	Mudah diuji karena pembuatan objek bisa disimulasikan (mock)
Reusability	Factory bisa digunakan oleh banyak bagian lain

Konsep class diagram di factory pattern



Konsep class diagram di factory pattern



Running factory pattern

```
public class PizzaTestDrive {

   public static void main(String[] args) {
      PizzaStore nyStore = new NYPizzaStore();
      PizzaStore chicagoStore = new ChicagoPizzaStore();

      Pizza pizza = nyStore.orderPizza("cheese");
      System.out.println("Ethan ordered a " + pizza.getName() + "\n");

      pizza = chicagoStore.orderPizza("cheese");
      System.out.println("Joel ordered a " + pizza.getName() + "\n");
   }
}
```

Running factory pattern

```
public class PizzaTestDrive {

   public static void main(String[] args) {
      PizzaStore nyStore = new NYPizzaStore();
      PizzaStore chicagoStore = new ChicagoPizzaStore();

      Pizza pizza = nyStore.orderPizza("cheese");
      System.out.println("Ethan ordered a " + pizza.getName() + "\n");

      pizza = chicagoStore.orderPizza("cheese");
      System.out.println("Joel ordered a " + pizza.getName() + "\n");
   }
}
```

Running factory pattern

```
yadhiaditya@MBP-YA src % java headfirst/designpatterns/factory/pizzafm/PizzaTestDrive
--- Making a NY Style Sauce and Cheese Pizza ---
Prepare NY Style Sauce and Cheese Pizza
Tossing dough...
Adding sauce...
Adding toppings:
   Grated Reggiano Cheese
Bake for 25 minutes at 350
Cut the pizza into diagonal slices
Place pizza in official PizzaStore box
Ethan ordered a NY Style Sauce and Cheese Pizza

--- Making a Chicago Style Deep Dish Cheese Pizza ---
Prepare Chicago Style Deep Dish Cheese Pizza
Tossing dough...
Adding sauce...
Adding toppings:
   Shredded Mozzarella Cheese
Bake for 25 minutes at 350
Cutting the pizza into square slices
Place pizza in official PizzaStore box
Joel ordered a Chicago Style Deep Dish Cheese Pizza
```

POLBAN

03

Decorator Pattern (Structural)

Definisi Decorator Pattern

Decorator Pattern adalah **pola desain struktural** yang digunakan untuk **menambahkan tanggung jawab (perilaku atau fitur) ke objek secara dinamis** *tanpa mengubah kode kelas aslinya*.

Analogi sederhana: Seperti membungkus kado dengan berbagai lapisan—kado tetap sama, namun masih bisa menambahkan hiasan seperti pita, stiker, atau label secara fleksibel.

Tujuan Decorator Pattern

Tujuan:

- Menambah **perilaku baru ke objek secara fleksibel** tanpa harus membuat banyak subclass.
- Menghindari **class explosion** akibat kombinasi fitur (misal: CoffeeWithMilkAndSoyAndWhip).
- Mendukung prinsip **Open-Closed**: kelas boleh diperluas tapi tidak boleh diubah.

Cara Kerja Decorator Pattern

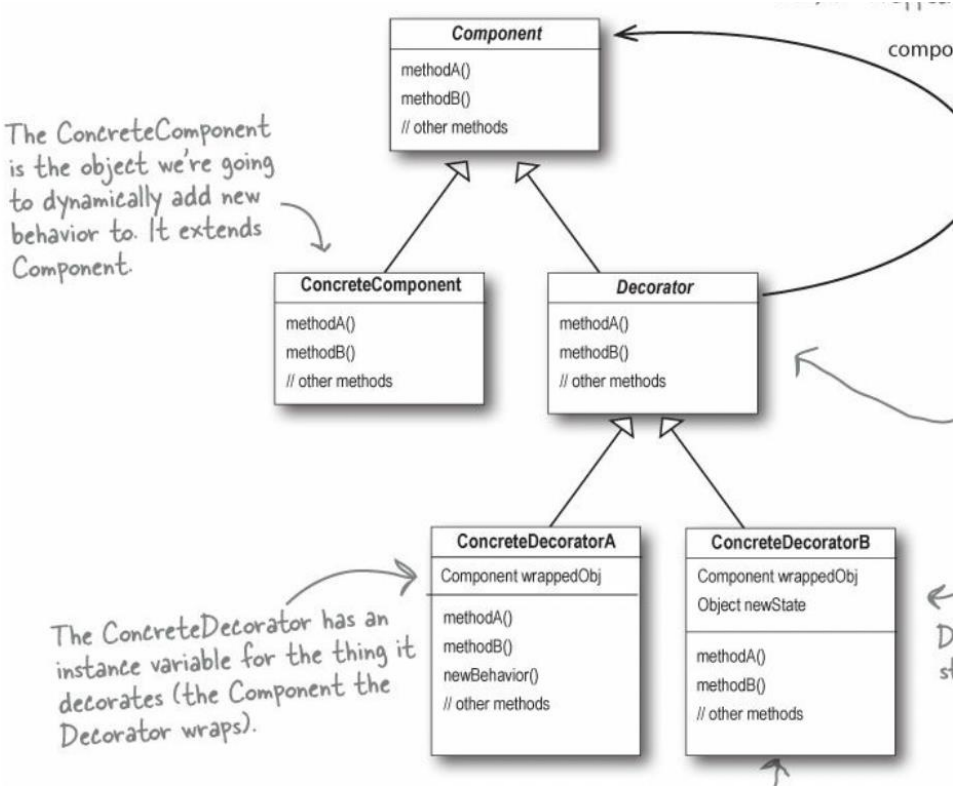
Cara Kerjanya?

- Buat **komponen inti** (interface atau abstract class).
- Implementasikan **komponen konkret** (misal Espresso).
- Buat **abstract decorator** (juga mewarisi komponen).
- Buat **concrete decorator** (seperti Mocha, Whip) yang membungkus komponen dan menambahkan fungsionalitas.

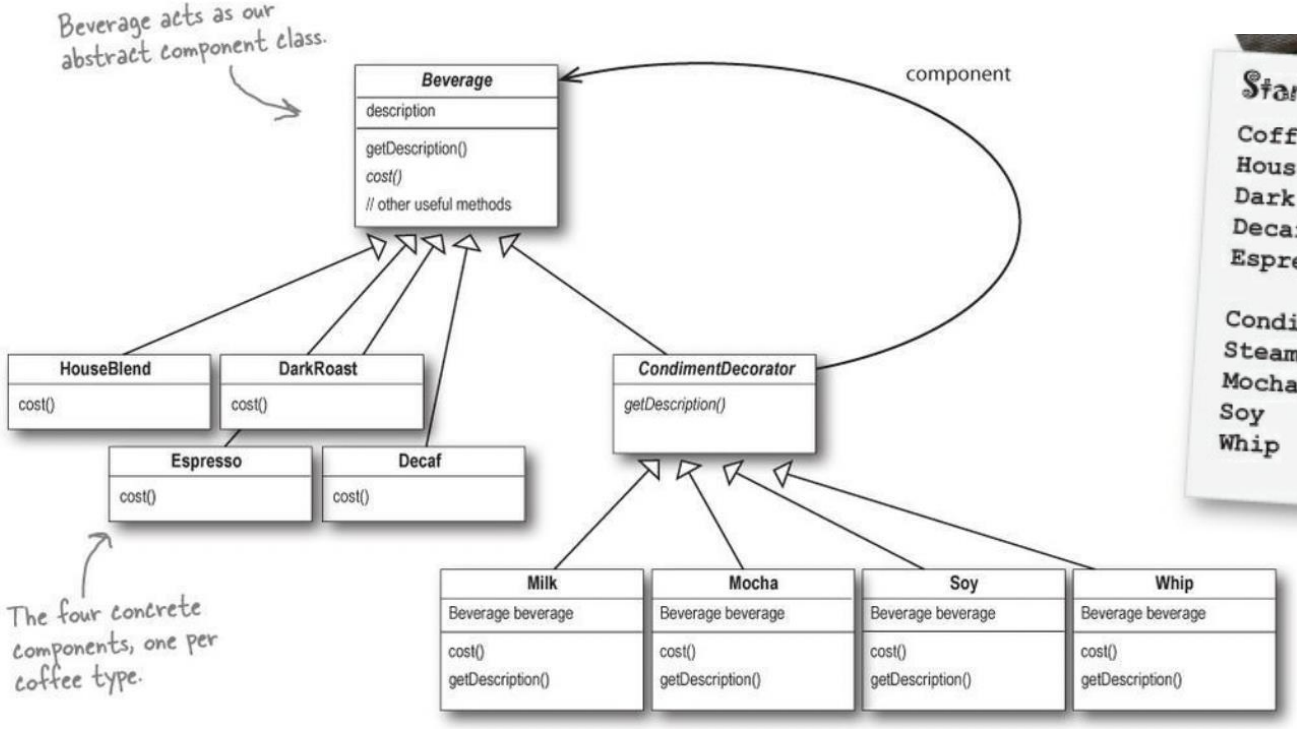
Manfaat Decorator Pattern

Manfaat	Penjelasan
Fleksibel & reusable	Bisa tambah fitur baru tanpa mengubah kode asli
Hindari subclass berlebihan	Tidak perlu bikin semua kombinasi subclass yang mungkin
Runtime behavior	Bisa tambah/mengubah perilaku objek **saat runtime**
Mendukung prinsip SOLID	Khususnya **Open-Closed Principle**

Konsep class diagram di Decorator pattern



Konsep class diagram di Decorator pattern



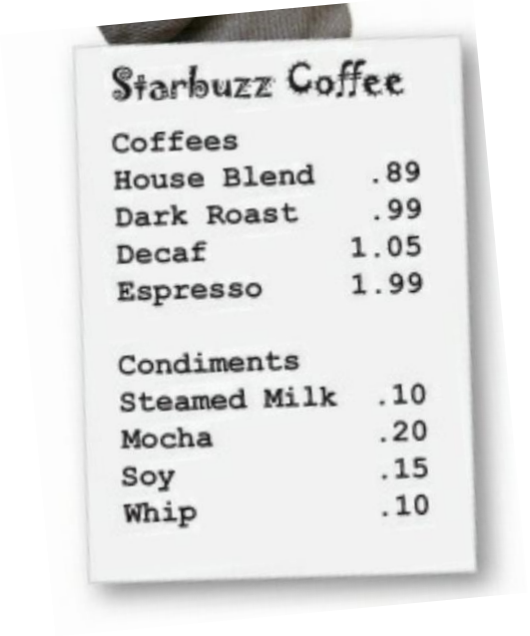
Running Decorator pattern

```
public class StarbuzzCoffee {

   public static void main(String args[]) {
      Beverage beverage = new Espresso();
      System.out.println(beverage.getDescription()
            + " $" + beverage.cost());

      Beverage beverage2 = new DarkRoast();
      beverage2 = new Mocha(beverage2);
      beverage2 = new Mocha(beverage2);
      beverage2 = new Whip(beverage2);
      System.out.println(beverage2.getDescription()
            + " $" + beverage2.cost());

      Beverage beverage3 = new HouseBlend();
      beverage3 = new Soy(beverage3);
      beverage3 = new Mocha(beverage3);
      beverage3 = new Whip(beverage3);
      System.out.println(beverage3.getDescription()
            + " $" + beverage3.cost());
   }
}
```



Running Decorator pattern

```
yadhiaditya@MBP-YA src % java headfirst/designpatterns/decorator/starbuzz/StarbuzzCoffee
Espresso $1.99
Dark Roast Coffee, Mocha, Mocha, Whip $1.49
House Blend Coffee, Soy, Mocha, Whip $1.34
```

POLBAN

04

Observer Pattern (Behavioral)

Definisi Observer Pattern

Observer Pattern adalah pola desain perilaku (behavioral pattern) yang digunakan untuk **membangun hubungan satu-ke-banyak (one-to-many)** antar objek, di mana ketika satu objek (**Subject**) berubah, semua objek yang **mengamati (Observers)** akan diberi tahu dan diperbarui secara otomatis.

Analogi sederhana: Seperti langganan berita. Ketika berita baru terbit (perubahan state), semua pelanggan (observer) akan menerima notifikasi.

Tujuan Observer Pattern

Tujuan:

Menghubungkan objek tanpa saling tergantung langsung (loose coupling).

Ketika satu objek (subject) berubah, otomatis memberi tahu semua yang tergantung padanya (observers).

Digunakan pada situasi **real-time updates** seperti:

- Sistem cuaca (Weather Station)
- UI yang menampilkan data yang terus berubah
- Sistem event listener pada GUI atau aplikasi game

Cara Kerja Observer Pattern

1. Ada dua peran utama:

- **Subject**: Objek yang diamati (memiliki data/state).
- **Observers**: Objek-objek yang ingin tahu saat Subject berubah.

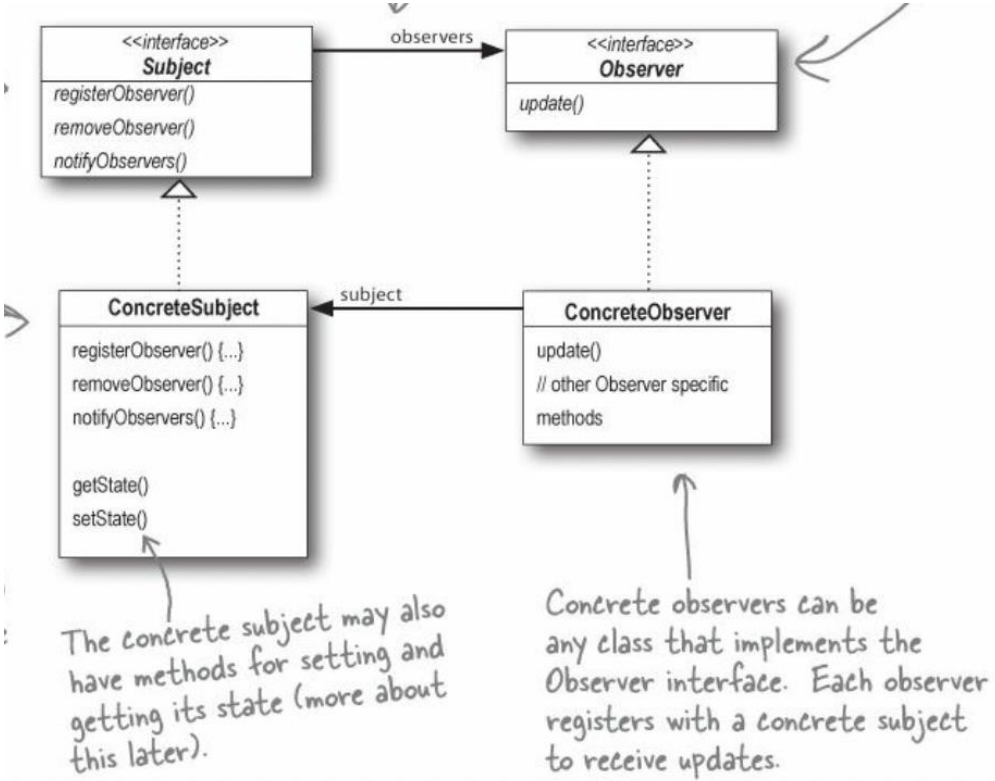
2. Proses alur:

- Observer **mendaftar ke Subject** (registerObserver()).
- Ketika Subject berubah, ia akan **memanggil update() semua Observer**.
- Observer kemudian bisa **mengambil data** dari Subject (pull) atau **menerima data** dari Subject langsung (push).

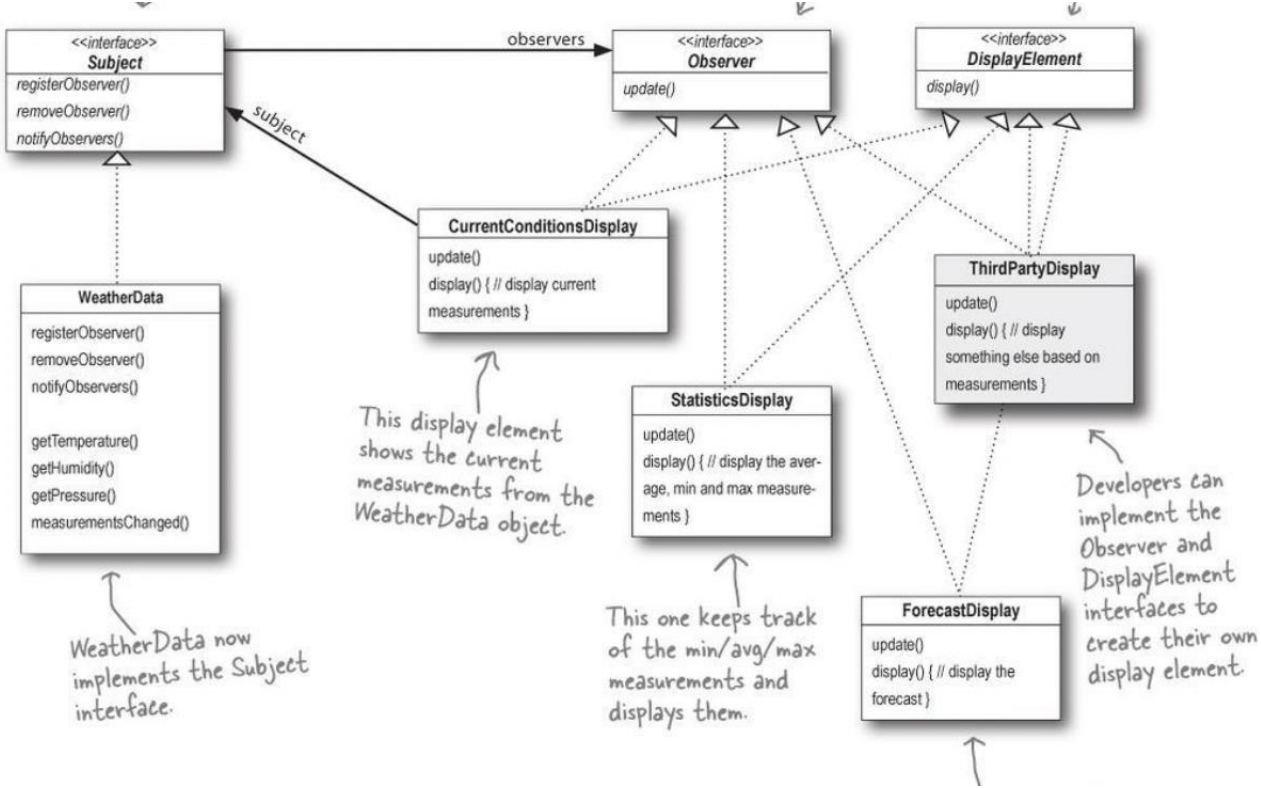
Manfaat Observer Pattern

Manfaat	Penjelasan
Otomatisasi notifikasi	Tidak perlu manual memberi tahu setiap objek
Fleksibel dan mudah diperluas	Tambah/menghapus observer tanpa ubah subject
Reusable dan testable	Subject dan Observer dapat diuji terpisah
Dipakai luas di framework	Contoh: **Swing**, **JavaBeans**, **RxJS**, **EventEmitter** di Node.js

Konsep class diagram di Observer pattern



Konsep class diagram di Observer pattern



Running Observer pattern

```
public class WeatherStation {

   public static void main(String[] args) {
      WeatherData weatherData = new WeatherData();

      CurrentConditionsDisplay currentDisplay =
         new CurrentConditionsDisplay(weatherData);
      StatisticsDisplay statisticsDisplay = new StatisticsDisplay(weatherData);
      ForecastDisplay forecastDisplay = new ForecastDisplay(weatherData);

      weatherData.setMeasurements(80, 65, 30.4f);
      weatherData.setMeasurements(82, 70, 29.2f);
      weatherData.setMeasurements(78, 90, 29.2f);

      weatherData.removeObserver(forecastDisplay);
      weatherData.setMeasurements(62, 90, 28.1f);
   }
}
```

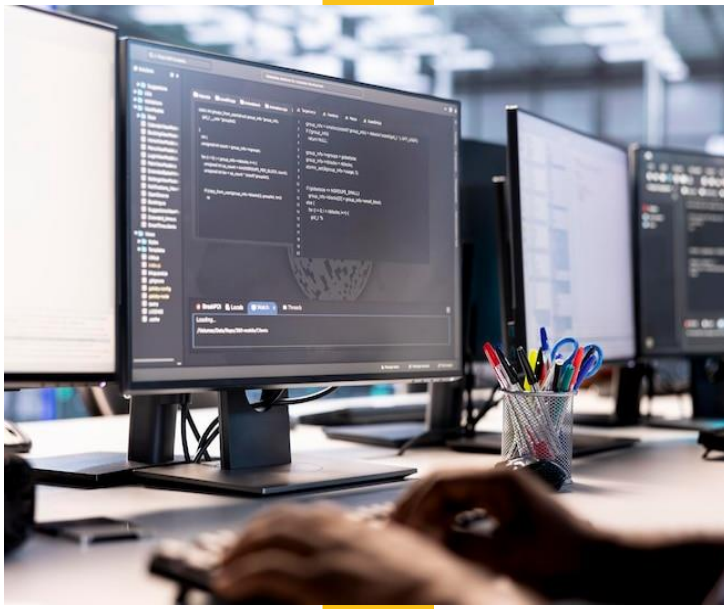
Running Observer pattern

```
yadhiaditya@MBP-YA src % java headfirst/designpatterns/observer/weather/WeatherStation
Current conditions: 80.0F degrees and 65.0% humidity
Avg/Max/Min temperature = 80.0/80.0/80.0
Forecast: Improving weather on the way!
Current conditions: 82.0F degrees and 70.0% humidity
Avg/Max/Min temperature = 81.0/82.0/80.0
Forecast: Watch out for cooler, rainy weather
Current conditions: 78.0F degrees and 90.0% humidity
Avg/Max/Min temperature = 80.0/82.0/78.0
Forecast: More of the same
Current conditions: 62.0F degrees and 90.0% humidity
Avg/Max/Min temperature = 75.5/82.0/62.0
```

POLBAN

05

Latihan Individual



Tugas design pattern (individu)

- Kerjakan secara individual
- Jalankan aplikasi yang diberikan,
  - pizzafm
  - starbuzz
  - Weather
- Ganti mainclassnya, berikan perintah lain, Misalkan untuk factory pattern
  - pizza = chicagoStore.orderPizza("clam");
  - System.out.println("Joel ordered a " + pizza.getName() + "\n");
  - pizza = nyStore.orderPizza("pepperoni");
  - System.out.println("Ethan ordered a " + pizza.getName() + "\n");
- Buat laporan yang sudah dilakukan, kumpulkan dalam link yang sudah disediakan:

Bahan Materi dan pengumpulan

- https://drive.google.com/drive/folders/13woM4H0uPXW-Ni2uo_mb29f1s70xFSwb?usp=sharing

- Penamaan TugasDP_NIM_NAMA.pdf
- Misalkan TugasDP_231524043_FANZA.pdf

Minggu Depan

Strategi & Metode Desain

Terima kasih

Do you have any questions?