Design Pattern Command

Command Pattern adalah salah satu pola desain perilaku (behavioral design pattern) yang memisahkan antara objek yang memberi perintah dan objek yang menjalankan perintah, dengan cara membungkus setiap perintah menjadi sebuah objek terpisah. Pola ini memudahkan kita untuk mengatur dan menjalankan perintah-perintah tersebut secara fleksibel—misalnya disimpan, dijalankan nanti, diatur urutannya, atau bahkan dibatalkan (undo). Dengan pendekatan ini, kita bisa membuat sistem yang lebih modular, mudah diatur, dan tidak saling ketergantungan antara pemicu dan pelaksana aksi.

Ciri-ciri Behavioral Pattern:

- Fokus pada alur proses atau aksi
- Memisahkan siapa yang meminta dan siapa yang melaksanakan
- Membuat sistem fleksibel dan bisa dimodifikasi tanpa ngacak-ngacak semua
- Lacak aksi yang pernah dilakukan
- Atur aksi kompleks secara modular

Behavioral Pattern: fokus ke aliran aksi antar objek

Struktur Kerja

1. Client
    - Membuat objek ConcreteCommand, dan menghubungkannya dengan Receiver.
    - Mengatur command tersebut ke Invoker.
2. Invoker
    - Objek yang menerima command dari client.
    - Menyimpan dan men-trigger command lewat metode seperti press_button() atau invoke().
3. Command (Interface)
    - Mendeklarasikan method execute() (kadang juga undo() jika diperlukan).
    - Jadi jembatan antara Invoker dan aksi sebenarnya.
4. ConcreteCommand
    - Implementasi dari Command.
    - Menyimpan referensi ke Receiver dan menentukan aksi spesifik.
    - Saat execute() dipanggil, dia akan memanggil metode di Receiver.
5. Receiver
    - Objek yang menjalankan aksi aktual.
    - Tidak tahu siapa yang meminta perintah—fokus hanya ke “bagaimana caranya melakukan itu

Komponen Utama

1. Command: Interface/abstraksi yang mendeklarasikan metode execute()
2. ConcreteCommand: Implementasi spesifik dari Command, mengikat Receiver dengan action

3. Receiver: Objek yang mengetahui cara melakukan operasi sebenarnya
4. Invoker: Meminta Command untuk menjalankan request
5. Client: Membuat ConcreteCommand dan mengatur penerimanya

KEGUNAAN (USE CASE)

Use Case	Kegunaan
Undo / Redo	Simpan riwayat command agar bisa dibatalkan/dikembalikan
Button Binding	Setiap tombol UI bisa menjalankan perintah berbeda tanpa tahu isi aksi
Task Queue / Scheduler	Simpan perintah lalu jalankan kemudian
Macro (Batch)	Gabungkan beberapa perintah dalam satu aksi
Remote Command	Kirim command ke server atau perangkat lain (IoT, robot, dll)

UML Diagram (Remote Command)

Contoh use case

```python
# Command interface
class Command:
    def execute(self):
        pass

# Receiver
class TV:
    def __init__(self):
        self.is_on = False

    def turn_on(self):
```

```
        if not self.is_on:
            self.is_on = True
            print("TV Dinyalakan 📺")
        else:
            print("TV sudah menyala")

    def turn_off(self):
        if self.is_on:
            self.is_on = False
            print("TV Dimatikan ❌")
        else:
            print("TV sudah mati")

# ConcreteCommand
class PowerToggleCommand(Command):
    def __init__(self, tv):
        self.tv = tv

    def execute(self):
        if self.tv.is_on:
            self.tv.turn_off()
        else:
            self.tv.turn_on()

# Invoker
class RemoteControl:
    def __init__(self):
        self.command = None

    def set_command(self, command):
        self.command = command

    def press_button(self):
        self.command.execute()

# Client Code
tv = TV()
remote = RemoteControl()

power_command = PowerToggleCommand(tv)
remote.set_command(power_command)

# Tekan tombol power beberapa kali
remote.press_button()  # TV Dinyalakan 📺
remote.press_button()  # TV Dimatikan ❌
remote.press_button()  # TV Dinyalakan 📺
```

output

```
TV Dinyalakan 📺
TV Dimatikan ❌
TV Dinyalakan 📺
```

URUTAN KERJA (Flow teknisnya)

1. Client menyiapkan:
    - Tombol Power (ConcreteCommand)
    - Remote (Invoker)
    - TV (Receiver)
    - Lalu tombol Power ini dipasang ke remote
2. User (client) menekan tombol di Remote (invoker)
3. Remote menjalankan execute() pada objek Command
4. ConcreteCommand memanggil turn_on() ke objek TV (receiver)
5. TV merespons dan menyalakan layar.

KELEBIHAN & KEKURANGAN

Kelebihan:

- Loose coupling, **Artinya:** Komponen dalam program tidak saling tergantung secara langsung
- Mendukung undo/redo, **Artinya:** Bisa membatalkan atau mengulang perintah sebelumnya.
- Extensibility, **Artinya:** Mudah menambahkan aksi baru tanpa mengubah kode client atau receiver
- Reusability, **Artinya:** Perintah bisa disimpan dan digunakan ulang kapan pun (misal untuk redo, batch processing, scripting)
- Bisa simpan dan antri perintah, **Artinya:** Perintah bisa ditunda dulu, lalu dijalankan nanti.
- Mempermudah pembuatan UI yang dinamis (tombol, shortcut, dll), **Artinya:** Tombol, shortcut, atau menu bisa dikaitkan dengan perintah apa pun.

Kekurangan:

- Tambahan kelas lebih banyak (1 command = 1 class), **Artinya:** Setiap aksi harus punya class sendiri.
- Overhead untuk aplikasi sederhana, **Artinya:** Terlalu rumit dan berat untuk aplikasi kecil.
- Kompleksitas tinggi jika command terlalu kecil dan banyak, **Artinya:** Kalau kamu punya banyak aksi kecil (misalnya 50 tombol kecil), jumlah `Command` bisa jadi sangat banyak dan susah dikelola.

Alasan Memilih Bahasa Python untuk penerapan Command Pattern

- Karena Sintaks yang cukup sederhana dan mudah di pahami
- Mendukung OOP