BINAR

Design Pattern

Gold - Chapter 5 - Topic 3

BINAR

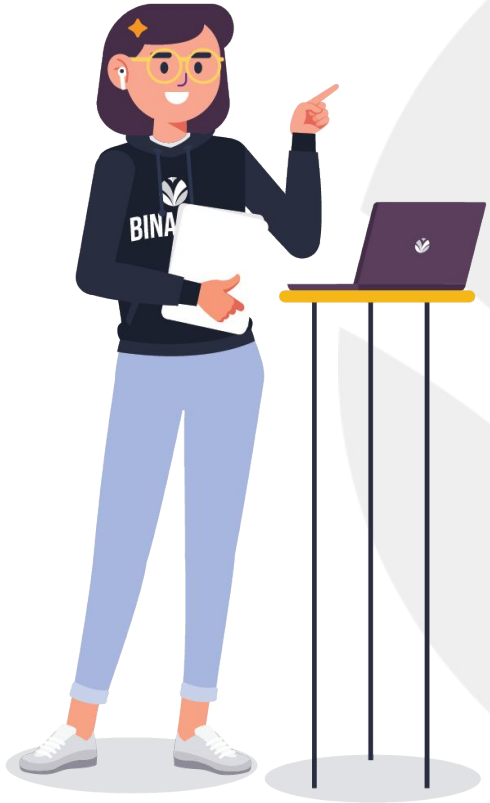
Selamat datang di Chapter 5 Topic 3 online course Back End Java dari Binar Academy!

Selamat datang, Binarian!

Pada topic kedua, kamu udah belajar tentang Spring Web lengkap bingits secara mendalam.

Pada topic ketiga ini, kita bakal mengelaborasi tentang **Design Pattern.** Mulai dari konsep design pattern, singleton, factory dan pool pattern, sampai contoh dari Structural Pattern dan Behavioral Pattern.

Yuk, langsung aja kita kepoin~



Dari sesi ini, kita bakal bahas hal-hal berikut:

- Pengantar Design Pattern
- Konsep Singleton, Factory dan Pool Pattern
- Contoh Structural Pattern dan Behavioral Pattern dalam best practice

Kita udah spoiler dikit nih di topik kemarin, kalau konsep Design Pattern ini mirip kayak resep masakan.

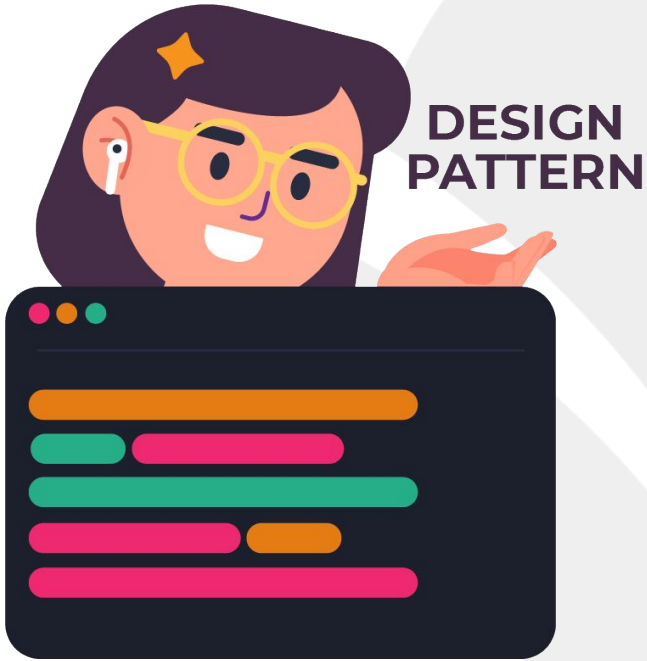
Tapi, sebelum menuju detail ke penjelasan itu, kita bakal flashback dulu ke konsep Class di chapter sebelumnya.

Yepp, karena ada kaitannya juga dengan class.

Kita flashback dulu ya, bestie~

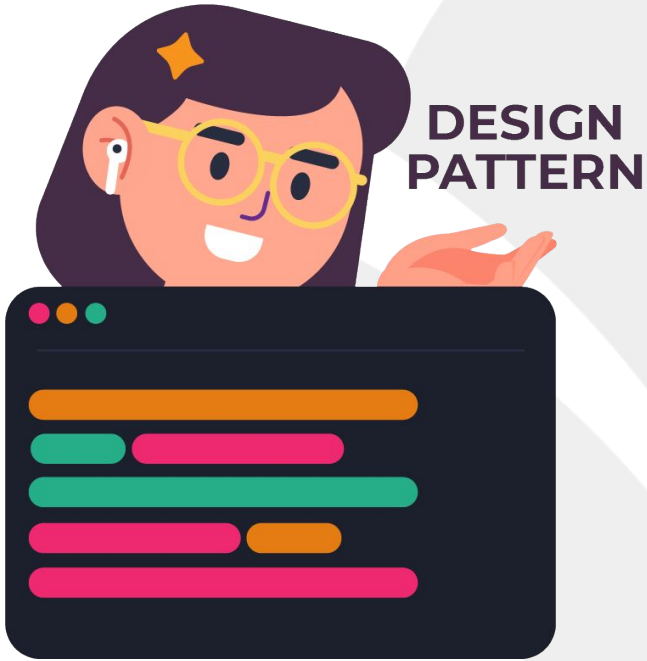
Pada chapter 2 kita udah belajar tentang konsep OOP, dimana disitu ada konsep class. Kamu masih ingat kan apa itu class?

Materi tersebut sebenarnya ada kaitannya sama materi yang mau kita bahas sekarang.



Design pattern ini mirip kayak class, yaitu suatu blueprint.

Bedanya, kalau class ibarat blueprint atau acuan buat bikin object, kalau design pattern ibaratnya **blueprint atau template buat bikin program**.



Sekarang kita masuk ke konsep dari Design Pattern-nya, ya!

Pas kita bikin suatu program, biasanya kita bakal dihadapkan dengan masalah berulang kali. Kalau masalahnya beda, solusi beda juga, dong?

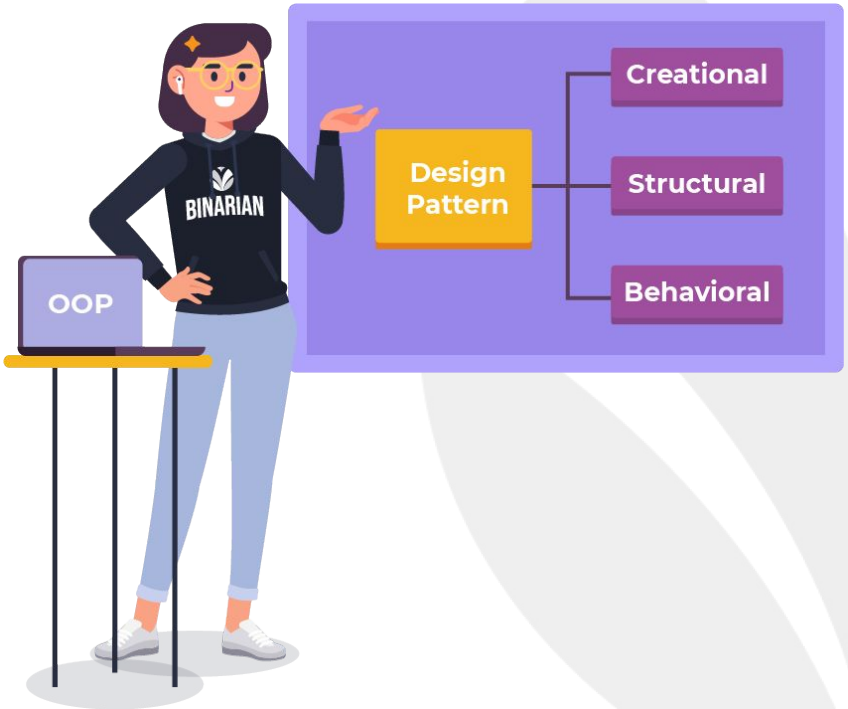
Seluruh masalah akhirnya bisa diselesaikan dengan teknik yang beda-beda karena punya situasi yang beda-beda juga. Iya, kan?

Sayangnya, hal itu bakal mengakibatkan pengembangan program yang lagi dibikin malah jadi lama.

Di sinilah design pattern mengubah dunia jadi lebih indah. Doi merupakan **kumpulan solusi yang dicoba dan diuji untuk masalah umum yang biasa terjadi dalam software design**.

Dengan mengetahui design pattern, ternyata nantinya bakal bermanfaat banget buat kita yang menyelesaikan masalah pakai prinsip OOP.

Keren, kan?!



Kalau kita baru belajar bikin program, biasanya kita bakal sering bingung dan bertanya-tanya nih~

Iya, berhubung seorang developer punya tugas bikin program, pasti ada concern berupa pertanyaan tentang hal-hal berikut:

- Kapan sih kita harus bikin file baru untuk menulis kode kita?
- Ada nggak sih struktur folder yang bikin kita nyaman saat ngoding?
- Bisa nggak sih kita mengelompokkan beberapa kode yang punya kemiripan dalam fungsi-fungsinya?

Hayo siapa yang sering overthinking dengan pertanyaan sebelumnya?

Semua pertanyaan itu bisa dijawab sama design pattern!

Dengan menerapkan design pattern, **kode yang kita bikin bakal lebih terstruktur dan lebih gampang buat dipahami sama orang lain**.

Selain itu, kita juga jadi lebih gampang buat mengelompokkan kode-kode yang kita buat.

Coba kamu ingat pengalaman di sebuah restoran yang menyajikan makanan yang super enak~

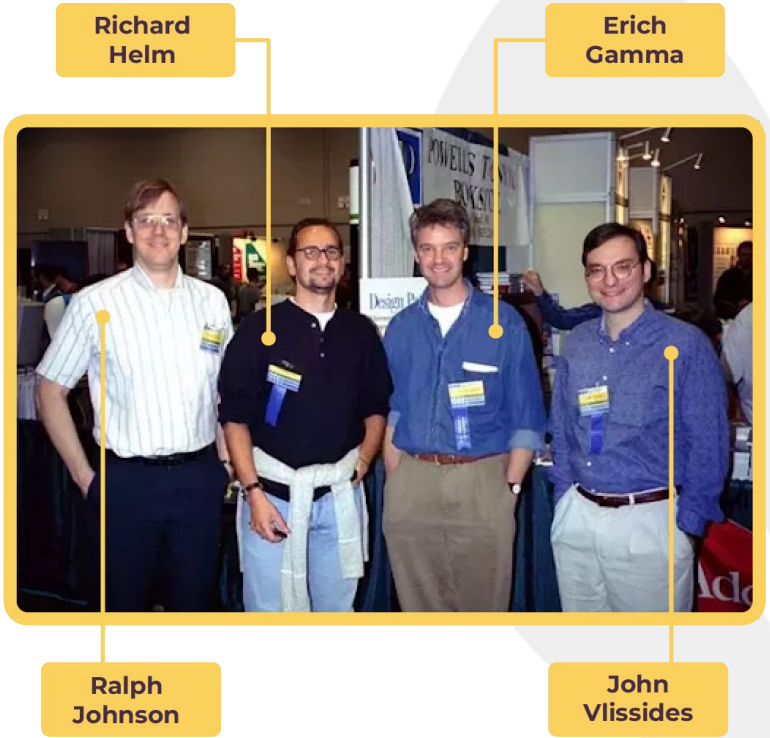
Dibalik makanan enak, tentu ada koki yang punya resep dan teknik masak khusus yang diciptakan melalui berbagai trial and error supaya bisa menghasilkan makanan yang super enak.

Kira-kira kayak gitu kalau design pattern ini diterapkan di kehidupan nyata. Resepnya tadi bisa diibaratkan sebagai design pattern.

“Kalau design pattern itu kayak resep, terus kokinya diibaratkan jadi siapa dong?”

Kalau dari sejarah asli ditemukannya design pattern, koki yang dimaksud adalah beberapa software developer berpengalaman yang menulis buku **Design Patterns: Elements of Reusable Object-Oriented Software**.

Mereka dikenal dengan sebutan **Gang of Four (GoF)**.



Pokoknya inget resep makanan favorit jadi inget tentang Design Pattern

Selanjutnya kita bakal intip-intip materi dari Creational Pattern: Singleton, Factory, dan Pool.

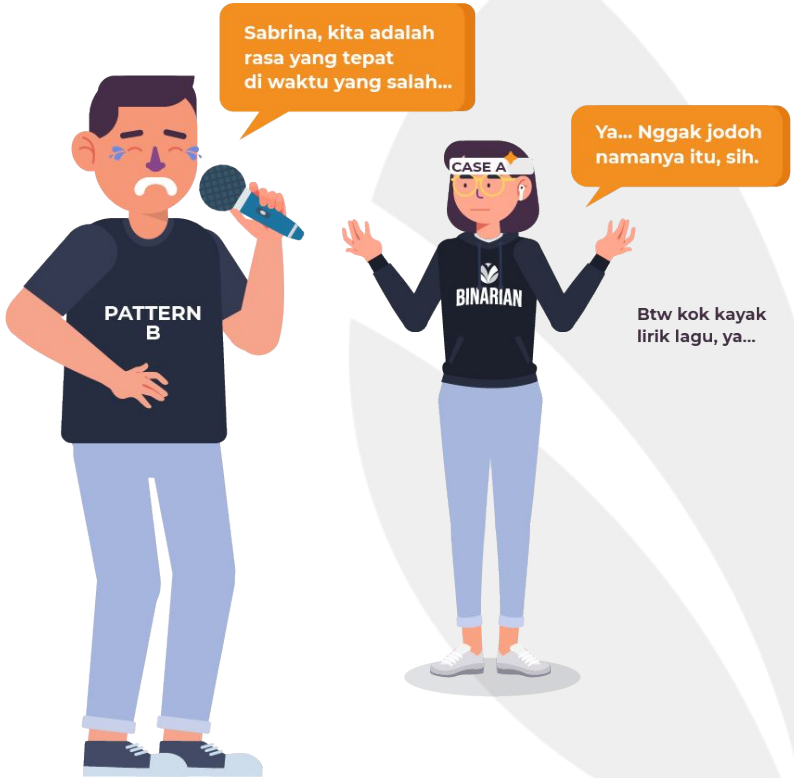
Pasti udah nggak sabar, kan? Yuk kita berangkatt~

Ibarat jodoh, sebuah pattern bakal sangat efektif kalau dipakai di kasus yang tepat, gengs~

Yap, bener banget.

Sebuah pattern **harus dipakai di kasus yang tepat**. Kalau nggak tepat, meskipun implementasi dari pattern-nya udah sesuai, nantinya malah menimbulkan masalah yang baru, lho.

So penting banget nih buat kita untuk tahu kasus apa aja yang bisa diselesaikan oleh pattern. Biar nggak bikin pusing nantinya~



“Tapi kalau tetap dipakai di kasus yang nggak tepat, gimana?”

Kondisi di mana sebuah pattern dipakai secara barbar tanpa menganalisa kasus yang mau diselesaikan bisa jadi **anti pattern**, gengs.

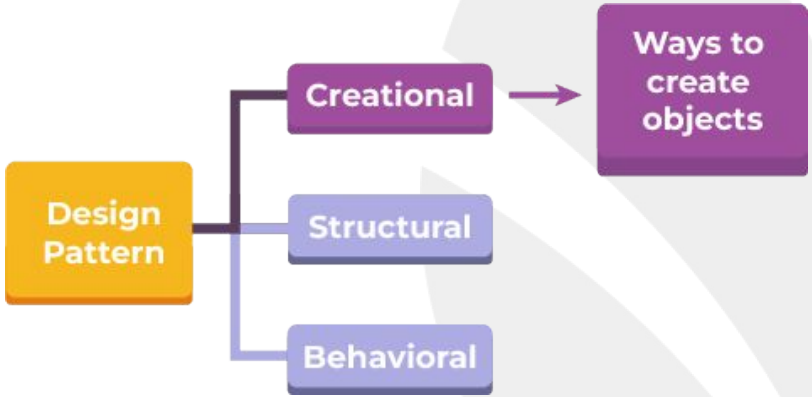
Maksudnya, anti pattern adalah **coding practice yang buruk**.

Istilah ini muncul pada tahun 1995 oleh Andrew Koenig.

“Kalau gitu, arti dari Creational Pattern itu apa, dong?”

Hayo, masih inget nggak kalau Creational Pattern tadi disebutin di awal sub topik?

Creational Pattern adalah solusi yang berkaitan dengan pembuatan object. Dengan creational pattern, kita bisa **me-manage penggunaan object.**

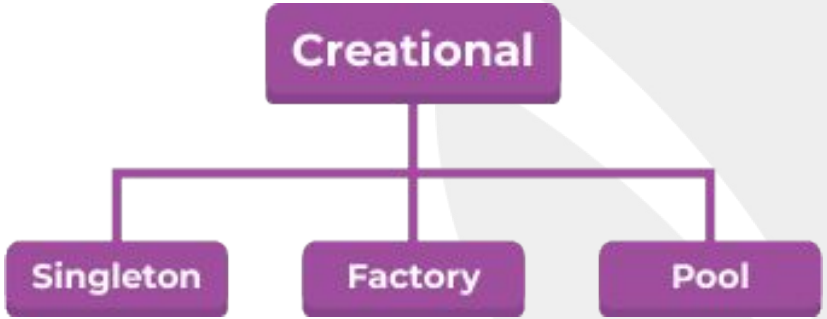


Iya. Kamu tahu nggak kalau sebuah object bisa jadi sesuatu yang mahal buat dibikin?

Hal ini karena sebuah object bisa memakan banyak sekali resource, sehingga sebuah object nggak bisa di-instance sebanyak-banyaknya.

Maka dari itu kita perlu belajar tentang managing object melalui Creational Pattern, yang diturunkan lagi jadi tiga pattern, yaitu **singleton, factory, dan pool pattern.**

Kita coba bahas satu-satu, yuk!

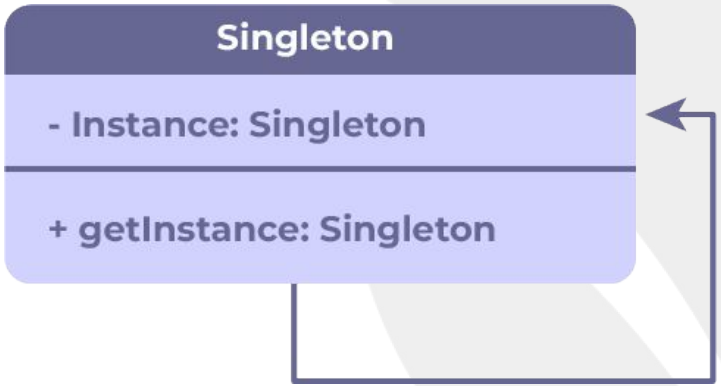


Singleton Pattern

Yaitu pattern yang bisa **memanggil satu object di dalam suatu aplikasi.** Pokoknya inget aja kata single.

Kenapa cuma satu instance saja? Karena object ini mahal banget, gengs.

Pada penggunaan Spring, pattern ini bisa dilakukan pakai annotation bean. Di mana instance dari annotation ini secara default pakai singleton (bisa diatur pakai scope dan bisa pakai pattern lain yaitu prototype).



Contoh object yang pakai singleton adalah database connection.

Sebuah aplikasi menggunakan satu object buat melakukan database connection terhadap satu database.

Kalau terhubung dengan dua database, berarti cuma dua connection.

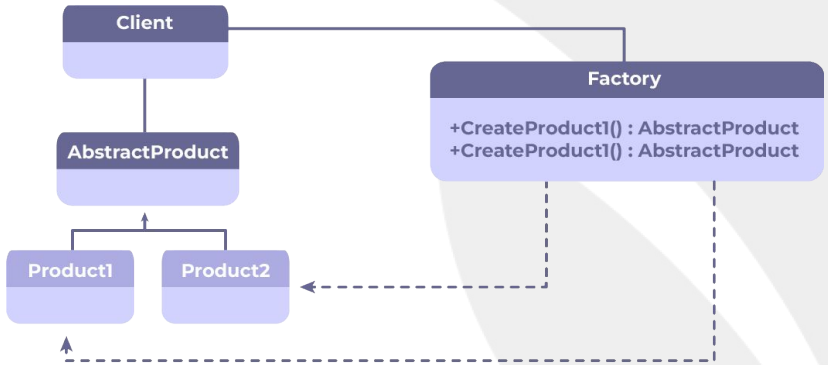
```
public final class ClassSingleton {

    private static ClassSingleton INSTANCE;
    private String info = "Initial info class";

    private ClassSingleton() {
    }

    public static ClassSingleton getInstance() {
        if(INSTANCE == null) {
            INSTANCE = new ClassSingleton();
        }

        return INSTANCE;
    }

    // getters and setters
}
```

Factory Pattern

Yaitu pattern yang cukup sering dipakai.

Factory pattern bisa **bikin berbagai macam object dari satu sumber**.

Object tersebut tentunya harus punya kesamaan (meng-implement interface yang sama) ya, gengs.



Misalnya ada suatu class service yang dipakai buat melakukan generate document xlsx, pdf dan csv.

Pas mau memproses file tersebut, semuanya udah mengimplement interface yang sama dengan method generate(). Tapi, implementasi atau cara buat men-generate masing-masing tipe document tersebut ternyata beda-beda. Jadi masing-masing tipe document bakal punya class-nya masing-masing.

Oleh karena itu, pada class service tersebut harus menentukan object apa yang harus dikembalikan.

Berikut contoh dari factory pattern!

Design Pattern - Factory Pattern

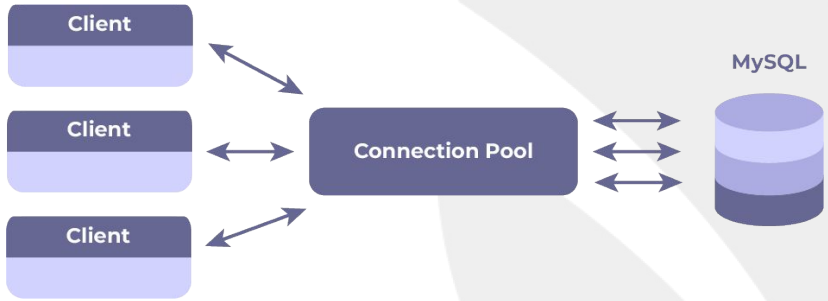
Kalau dihubungkan sama contoh sebelumnya, class service merupakan **ShapeFactory**, sedangkan class dari tipe-tipe document merupakan class yang meng-implement interface shape.

Pool Pattern

Atau disebut juga object pool pattern merupakan **versi singleton pattern yang lebih kompleks.**

Dengan object ini bukan cuma satu aja yang di-instance sob, tapi object yang dibuat sesuai dengan kebutuhan.

Contoh penggunaannya adalah pada SpringDataJPA yang pakai Hikari Connection Pool. Dengan menggunakan pool ini, kita bisa menyesuaikan berapa thread yang aktif pada kondisi idle, dan thread maksimum yang bisa diproses oleh pool.



Berikut contoh implementasi dari object pool ya, sob!

Design Pattern - Object Pool

Sekarang kamu jadi tahu beberapa pattern dan contohnya, kan?

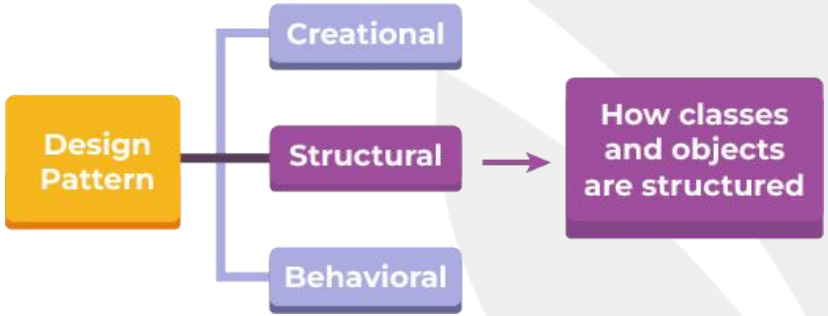
Selanjutnya, kita bakal bahas tipis-tipis tentang **Facade Pattern**.

Langsung aja kita cari tahu, yuk!

Sebelum bahas Facade Pattern, kita harus tahu tentang Structural Pattern dulu nih, sob!

Structural pattern merupakan **pattern yang mengatur gimana class-class berinteraksi**.

Kalau pakai pattern yang udah ada, kita bisa gampang banget buat melihat gimana sebuah class dikumpulkan dan membentuk sebuah struktur yang besar.



Ada tujuh pattern di structural pattern. Apa aja, ya?

Berikut adalah tujuh structural pattern!

- **Adapter pattern**
- **Bridge pattern**
- **Composite Pattern**
- **Decorator pattern**
- **Flyweight pattern**
- **Proxy pattern**
- **Facade pattern**

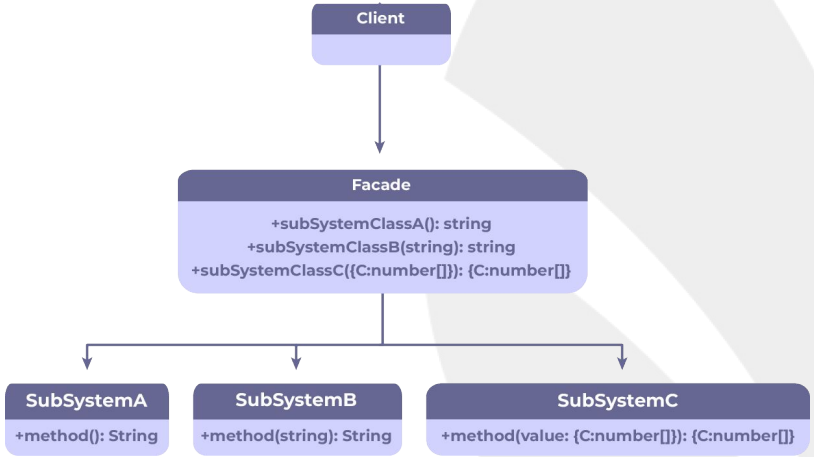
Tapi, di subtopic ini kita bakal bahas **Facade Pattern** aja, ya!

Jadi, apa itu Facade Pattern?

Facade pattern merupakan solusi untuk menampilkan **code yang complex** dan membaginya jadi beberapa bagian.

Facade ini adalah tempat bertemunya beberapa class yang berisi business logic. Sehingga, class yang terpisah tersebut bakal bertemu di satu class.

Mantep, khann?!

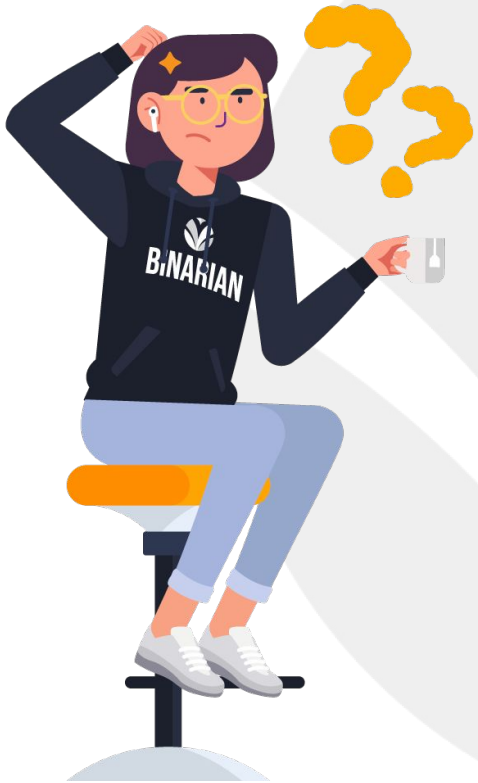


“Kenapa sih seluruh business logic nggak dijadikan satu class aja?!”

Sabar, sabar. Semua ini ada alasannya, kok.

Soalnya gini, kalau dijadikan satu class aja, maka **class yang isinya seluruh business logic bakal jadi kompleks banget**. Bahkan bisa jadi sulit buat dibaca dan dipahami.

Oleh karena itu class-class tersebut harus dikelompokkan jadi class-class yang lebih kecil supaya mudah dipahami. Dan sebaiknya class-class kecil tersebut punya interface yang berisi method utama.



Berikut adalah contoh dari facade pattern!

Facade Pattern

Simak baik-baik ya, bestie~

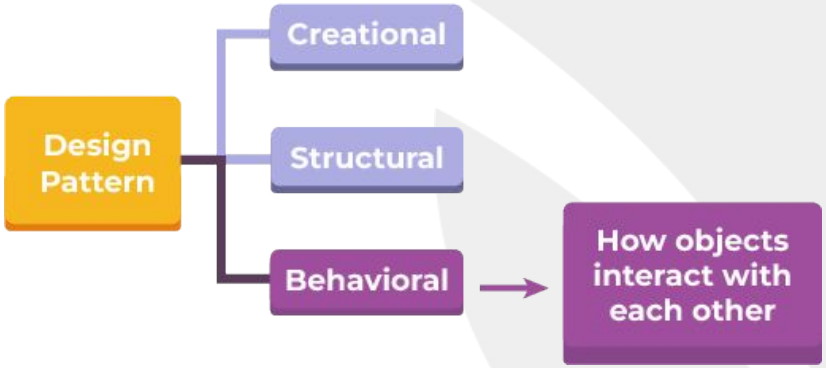
Menuju ke materi paling terakhir!

Udah banyak banget pattern yang kita bahas, tapi masih ada satu pattern lagi yang belum kita obrolin, yaitu Behavioral Pattern: Strategy pattern.

Hmm, kira-kira konsepnya gimana, yaaaa?

Behavioral Pattern itu sebenarnya adalah…

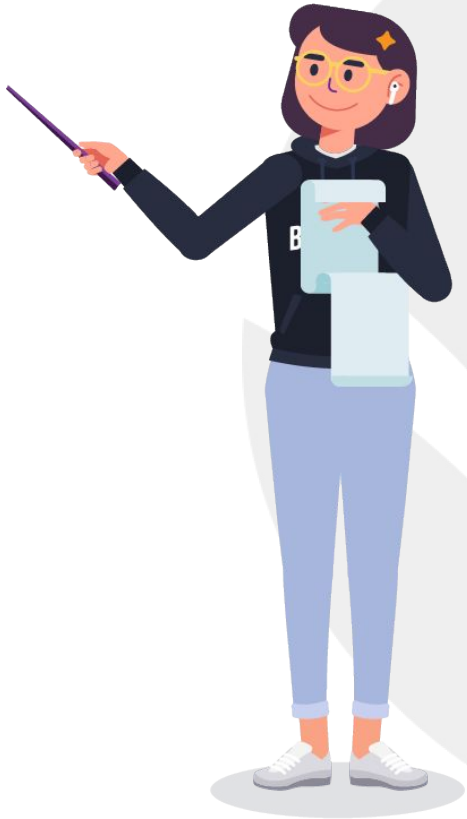
Behavioral Pattern merupakan pattern yang isinya **gimana sebuah object berinteraksi dan berperilaku di dalam sebuah system**.



Behavioral Pattern yang cukup sering dipakai itu yang di bawah ini nih, gengs:

- **Command pattern**
- **Interpreter pattern**
- **Chain of Responsibility pattern**
- **Strategy pattern**
- **Memento pattern**
- **Observer pattern**

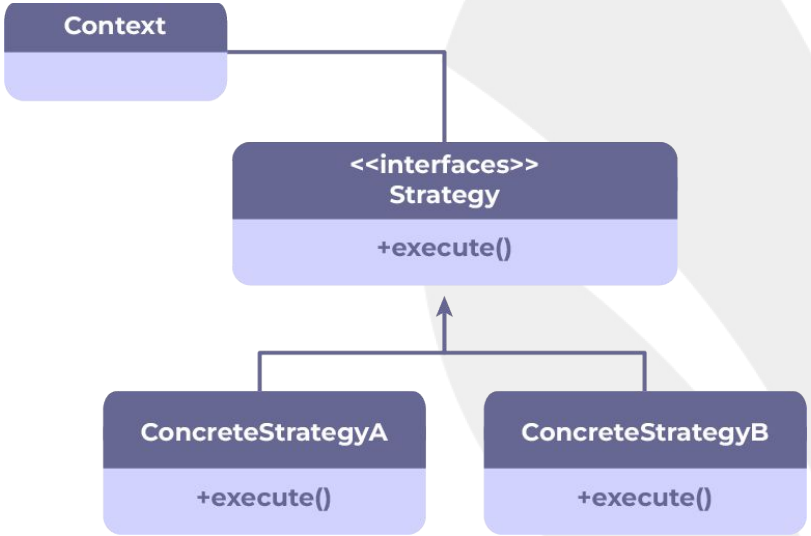
Di subtopic ini, kita bakal bahas **Strategy Pattern**.



“Strategy Pattern itu cara buat bikin pattern, ya?”

Hmm. Gini, gini. Strategy Pattern adalah **pattern untuk mengelompokkan algoritma**.

Strategy pattern ini agak mirip sama pattern yang kita bahas di awal. Tapi yang ditekankan di strategy pattern bukan tentang pembuatan objectnya, melainkan pengelompokkan algoritma yang jadi penekanannya.



Ada yang lebih keren lagi, sebuah If else atau switch case sederhana bisa membentuk pattern ini.

Berikut adalah contoh penggunaan strategy pattern!

Strategy Pattern

Coba deh kamu intip contoh berikut. Di situ bisa jadi ada lebih dari satu pattern yang dipakai.

Misalnya penggabungan structural pattern dan behavioral pattern.

Selama pattern-pattern tersebut **secara tepat digunakan**, maka nantinya bakal membentuk suatu coding practice yang baik.

By the way, tadi kan sempet mention tuh, kalo dalam implementasi pattern kamu harus pakai di kasus yang tepat.

Nah, untuk bisa menganalisa pattern di kasus yang tepat itu menurut kamu harus gimana? Apakah perlu strategi khusus?

Hayoo.. coba kamu diskusikan ya jawabannya.

Nah, selesai sudah pembahasan kita di Chapter 5 Topic 3 ini.

Selanjutnya, kita bakal bahas tentang API Documentation with Swagger.

Penasaran kayak gimana? Yuk, langsung ke topik selanjutnya~