**Spring Boot Design Patterns - Creational**

## **Pendahuluan**

**Sebelumnya harus sudah paham materi terkait** :

* Java
* Spring Boot

**Agenda**

* Pengenalan Design Pattern
* Pengenalan Creational Design Pattern
* Singleton
* Prototype
* Abstract Factory
* Builder
* Factory Method

## **Pengenalan Design Patterns**

**Pendahuluan**

* Membuat aplikasi berbasis object oriented bukanlah hal yang mudah
* Dan membuat aplikasi berbasis object oriented yang mudah dikembangkan, itu lebih sulit lagi
* Bahkan tak jarang, kadang desain object oriented awal ketika membuat aplikasi, perlu diubah ketika ukuran aplikasi kita semakin kompleks dan besar
* Pengembang aplikasi, biasanya akan mengambil pengalaman dari pengembangan aplikasi sebelumnya, entah dari pengalaman sendiri, atau pengalaman orang lain
* Tujuan course ini adalah belajar membuat desain aplikasi object oriented sebagai Design Patterns

**Apa itu Design Patterns?**

* Design patterns adalah solusi umum yang biasa digunakan ketika menghadapi masalah saat melakukan desain aplikasi
* Design patterns bukanlah solusi yang bisa di transform langsung menjadi kode program
* Design patterns adalah template atau panduan untuk menyelesaikan masalah, dan bisa diimplementasikan berbeda-beda, namun dengan tujuan yang sama

**Buku Design Patterns**

* Design Pattern : Elements of Reusable Object-Oriented Software
* Para penulisnya dikenal dengan panggilan GOF (Gang of Four)

**Kegunaan Design Patterns**

* Design patterns bisa mempercepat proses development dengan menyediakan solusi yang sudah terbukti.
* Menggunakan Design Patterns bisa menolong kita dari kesalahan dimasa depan jika mengimplementasikan solusi sendiri yang belum terbukti baik.
* Design Patterns menyediakan solusi umum, sehingga bisa digunakan di berbagai kasus
* Design Patterns juga sudah umum diadopsi oleh para pengembang perangkat lunak, sehingga menggunakan Design Patterns akan memudahkan perangkat lunak kita mudah dikembangkan

**Jenis-Jenis Design Patterns**

Secara garis besar, Design Patterns dibagi menjadi tiga jenis

* Creational Design Patterns, yaitu Design Patterns yang berhubungan dengan mekanisme pembuatan object
* Structural Design Patterns, yaitu Design Patterns yang berhubungan dengan interaksi antar object dan class ketika membentuk struktur yang lebih besar.
* dan Behavioral Design Patterns, yaitu Design Patterns yang berhubungan dengan penugasan, tanggung jawab, dan hubungan antar object

Pada pembelajaran ini, kita akan fokus membahas Creational Design Patterns

## **Pengenalan Creational Design Patterns**

* Creational Design Patterns adalah Design Patterns untuk membuat object.
* Creational Design Patterns membuat system menjadi lebih independen ketika membuat dan melakukan komposisi object
* Creational Design Patterns kadang menggunakan perawisan untuk membuat object
* Penggunaan Design Patterns tidak diharuskan hanya satu Pattern saja, kita bisa menggabungkan beberapa Design Patterns sekaligus jika memang dibutuhkan

**Jenis-Jenis Creational Design Patterns**

Terdapat banyak Design Pattern yang berjenis Creational, yaitu :

* Singleton,
* Prototype,
* Builder,
* Factory Method,
* dan Abstract Factory

## **Membuat Project**

<https://start.spring.io/>

Dependencies :

* Lombok

## **Singleton Pattern**

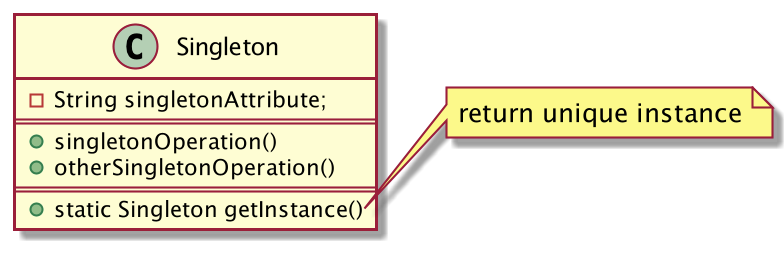
**Singleton**

* GOF : Ensure a class only has one instance, and provide a global point of access to it.
* Memastikan bahwa sebuah class hanya memiliki satu instance object, dan menyediakan cara mengaksesnya secara global (artinya bisa diakses dari mana saja)

**Motivasi**

* Pada beberapa kasus, memastikan sebuah class hanya memiliki satu instant object sangat penting
* Pertanyaannya, bagaimana memastikan sebuah class hanya memiliki satu instance object?
* Solusinya adalah membiarkan class itu sendiri yang bertanggung jawab untuk membuat instance object nya. Class tersebut bisa memastikan tidak ada instance lain yang dibuat, dan akan menyediakan cara mengakses singleton object tersebut
* Ketika terdapat perubahan pada instance object, client yang menggunakan singleton object tersebut tidak perlu melakukan perubahan

**Struktur**



**Keuntungan**

* Class singleton bisa mengatur bagaimana dan kapan client bisa mengakses object singleton tersebut
* Singleton lebih aman dibandingkan global variable. Ini mencegah client secara tidak sengaja mengubah global variable
* Class singleton bisa mengubah implementasi dari instance object nya, tanpa harus melakukan perubahan di client yang mengaksesnya

**Singleton Pattern di Spring Boot**

* Implementasi singleton pattern di Spring sangatlah mudah, kita bahkan tidak perlu membuat class khusus singleton
* Secara default object bean yang terdapat di Spring sudah singleton, jadi secara tidak sadar sebenarnya kita sudah mengimplementasikan singleton pattern

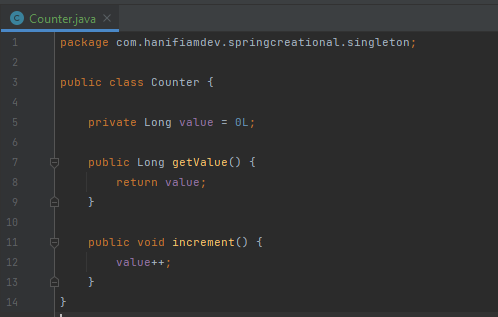
**Kapan Menggunakan Singleton Pattern**

* Saat menggunakan Spring, sebisa mungkin selalu menggunakan singleton pattern, kecuali memang ada kebutuhan khusus sehingga kita tidak ingin menggunakan singleton
* Secara default, Spring selalu membuat object secara singleton, dan secara default object akan dibuat secara eager (langsung di awal ketika aplikasi dijalankan), namun jika kita ingin membuat object singleton ketika object akan diakses, kita bisa mengubahnya menjadi lazy (artinya hanya dibuat ketika pertama kali diakses)
* Namun jika objectnya berat, seperti koneksi ke database, membuat http client, dan lain-lain, sangat disarankan membuat object singleton yang eager

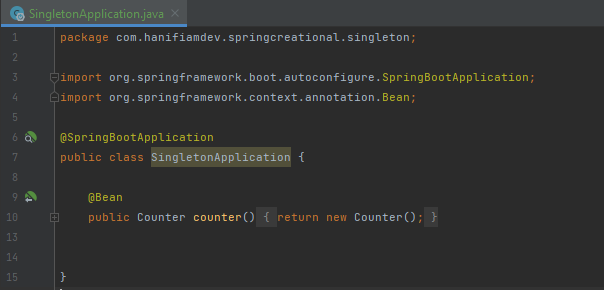
**Contoh Kasus**

* Kita akan membuat sebuah counter sebagai statistic berapa banyak request yang sudah ditangani oleh aplikasi kita
* Kita akan membuat class Counter, dimana class counter ini akan menyimpan data jumlah counter, dan memiliki method untuk menaikkan data counter nya
* Kita akan membuat instance class Counter sebagai singleton object, karena data counter harus selalu sama, sehingga tidak memerlukan lebih dari satu instance object

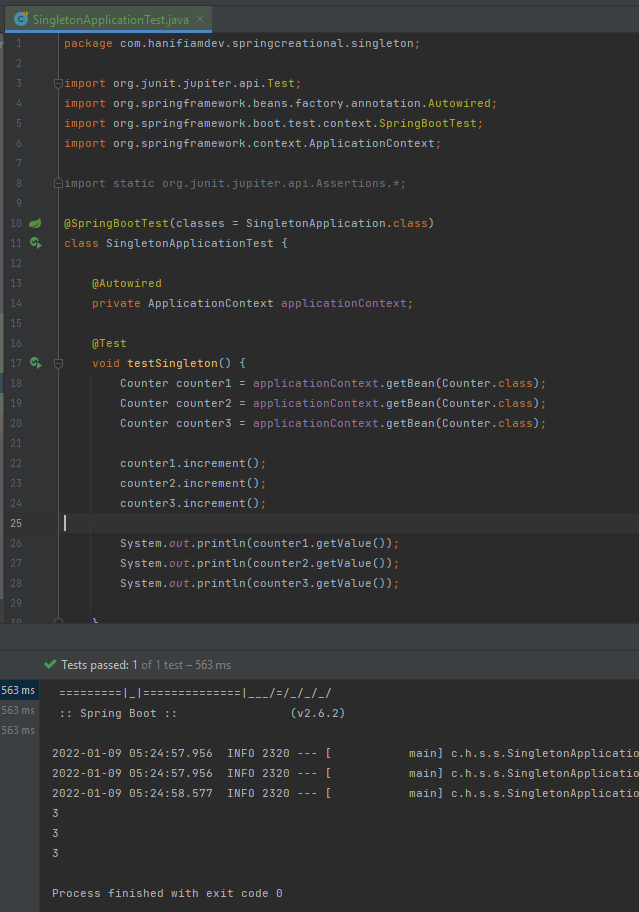
Kode : Class Counter



Kode : Membuat Singleton



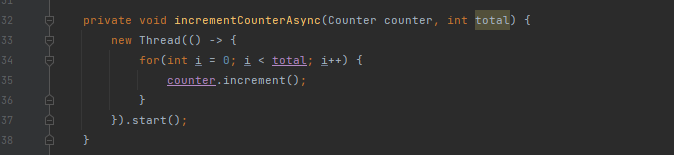
Kode : Test Singleton



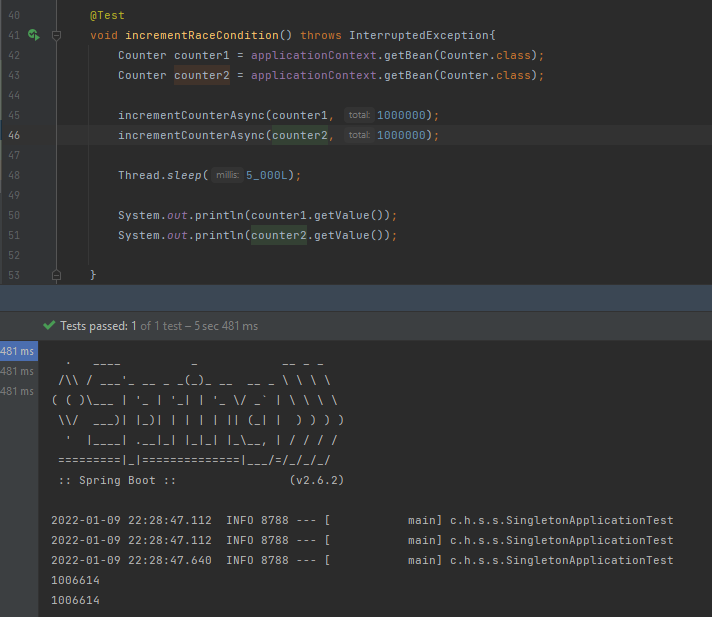
Yang Perlu Diperhatikan

* Singleton object secara otomatis akan di sharing ke semua client
* Oleh karena itu pastikan singleton object itu immutable (tidak bisa diubah)
* Jika singleton object nya sifatnya mutable (bisa diubah), pastikan tidak terjadi race condition ketika diubah secara bersamaan oleh beberapa client secara sekaligus

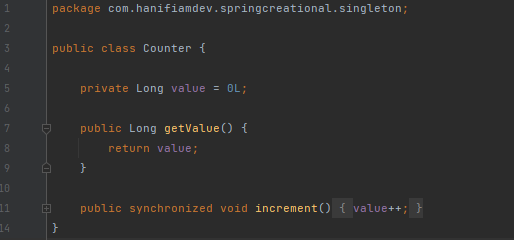
Kode : Race Condition



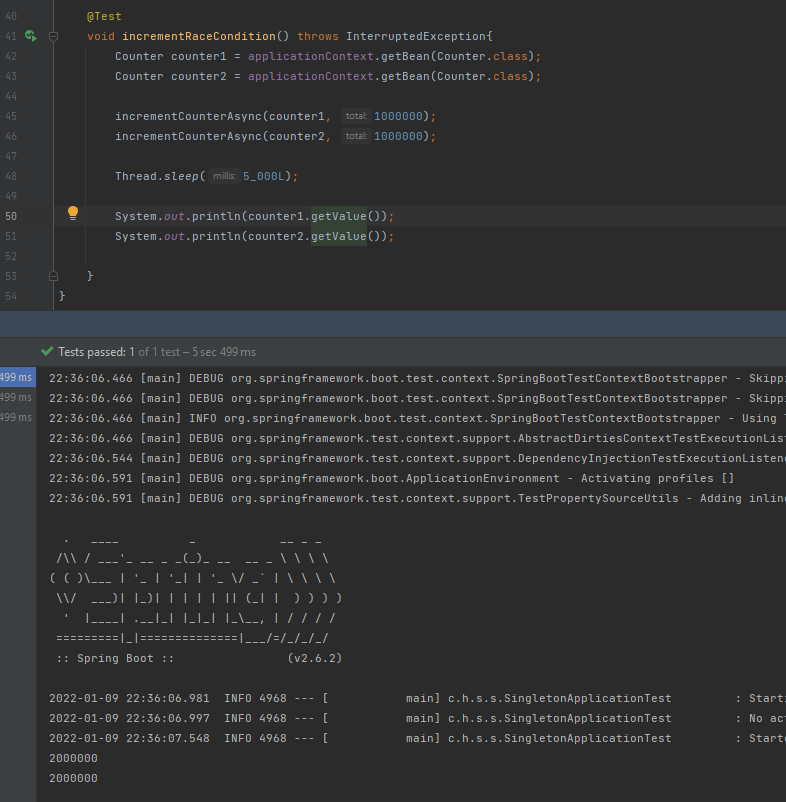
Berikut test ketika terjadi race condition dimana harusnya hasilnya 2000000, ini karena mengakses objek bersamaan dalam ms yang bersamaan



Solusinya buat synchronized agar tidak terkena race condition



Done



**Prototype Pattern**

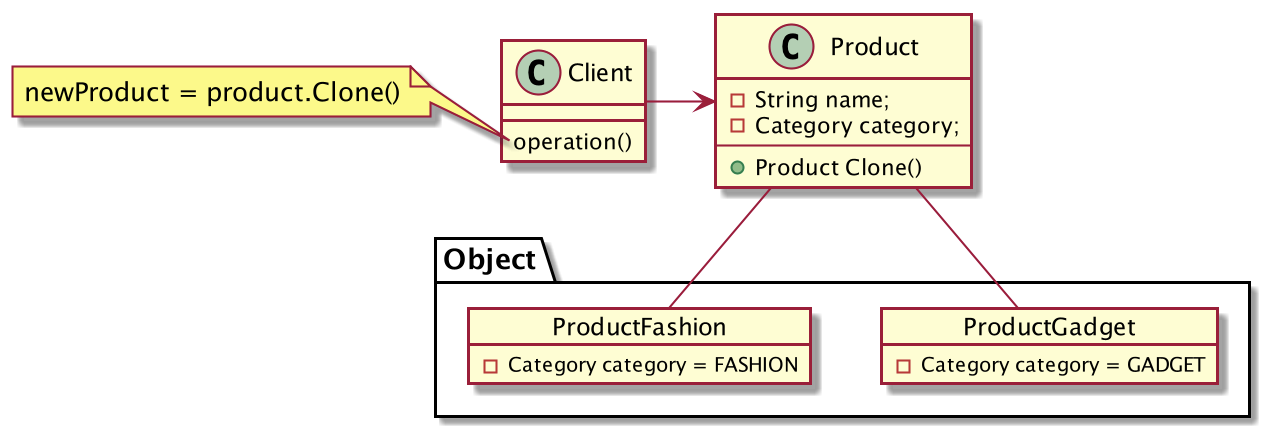
**Prototype**

* GOF : Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.
* Tentukan jenis objek yang akan dibuat menggunakan contoh prototipe, dan buat objek baru dengan menyalin objek prototipe-nya.

**Motivasi**

* Pada beberapa kasus, kita sering sekali menemui pembuatan object dengan beberapa data yang sama
* Jika kita membuat object tersebut secara manual satu persatu, maka akan rentang terhadap kesalahan, terutama ketika object dengan jenis tertentu harus berubah datanya
* Prototype pattern merupakan pattern dimana kita akan membuat object prototype, lalu saat kita membuat object baru, kita tidak perlu membuat object dari awal lagi, kita cukup menyalin data dari object prototype
* Hal inin membuat pembuatan object lebih mudah, dan jika data prototype ingin diubah, kita bisa dengan mudah mengubah prototype, dan secara otomatis semua object baru dari prototype tersebut akan berubah datanya

**Struktur**



**Keuntungan**

* Menambah dan menghapus prototype object sangat mudah, karena bukan sub-class, sehingga menambah dan menghapus prototype object tidak sesulit jika menggunakan class inheritance
* Membuat jenis object baru sangat mudah, hanya dengan menambah atau mengubah attribute object prototype, tidak perlu ada penambahan class baru
* Menurunkan jumlah subclass, pada pattern Factory Method, kadang menghasilkan class hirarki yang banyak, prototype bisa mencegah terjadinya class hirarki yang terlalu banyak
* Prototype object bahkan bisa dibuat pada saat runtime (aplikasi berjalan), karena tidak membutuhkan deklarasi class baru

**Prototype Pattern di Spring Boot**

* Kita sudah tahu secara default bean di Spring adalah singleton, artinya ketika kita akses bean nya, sebenarnya dia adalah object yang sama
* Di Spring, kita bisa ubah scope bean dari singleton menjadi prototype. Saat kita ubah menjadi prototype, secara otomatis object akan selalu dibuat baru sesuai dengan deklarasi yang ada di bean
* Dengan demikian, implementasi prototype di Spring sangat mudah

**Kapan Menggunakan Prototype Pattern**

* Saat kita memiliki sebuah class yang jenis objectnya banyak kesamaan, maka sangat cocok menggunakan prototype pattern
* Dengan prototype pattern, kita bisa membuat object dari prototype bean yang sudah kita deklarasikan
* Hati-hati ketika membuat bean dari prototype, karena secara otomatis object akan selalu dibuat baru oleh Spring

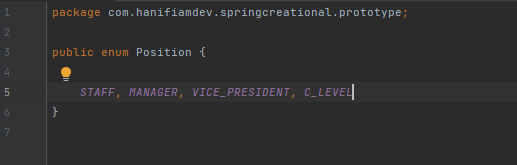
**Contoh Kasus**

* Misal kita akan membuat aplikasi kepegawaian, dimana terdapat data Employee
* Setiap Employee memiliki title / jabatan masing-masing, dan salary-nya selalu sama antar title
* Pada kasus ini, kita bisa menggunakan prototype pattern untuk membuat object Employee, dimana kita bisa buat beberapa prototype sesuai dengan title nya

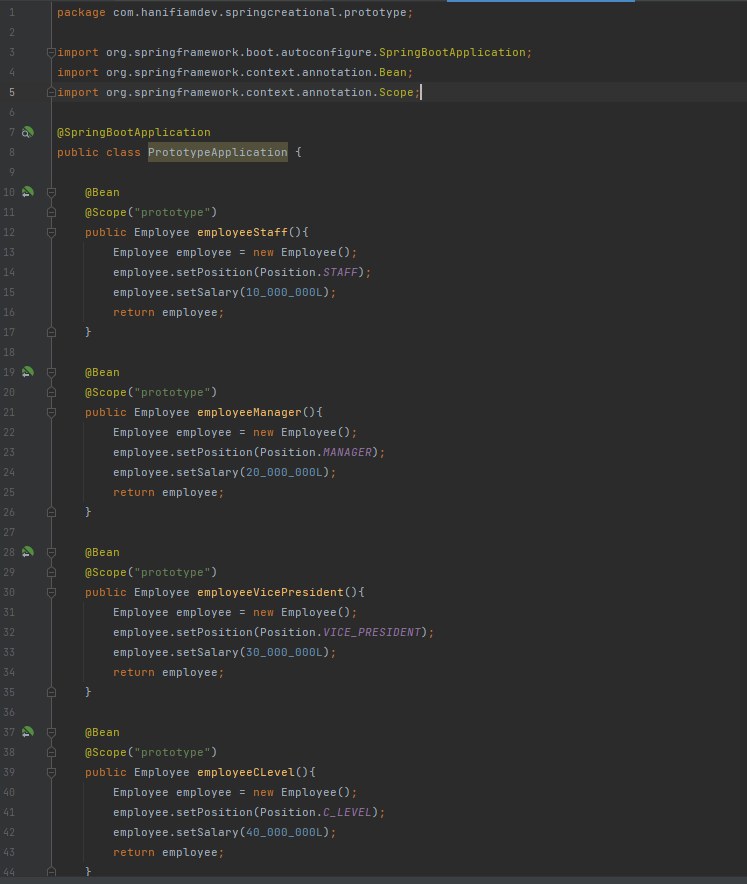
Kode : Class Employee( Getter and Setter, toString())



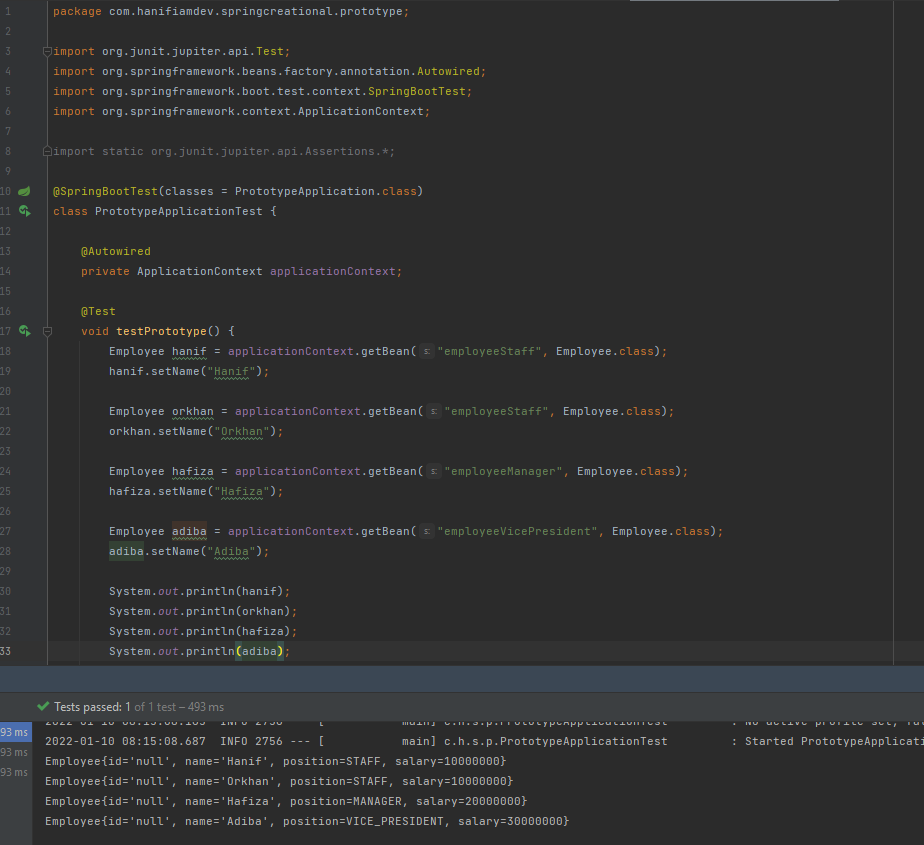
Kode : Enum position



Kode : Membuat Prototype



Kode : Membuat Object dari Prototype ( Unit Test )



**Builder Pattern**

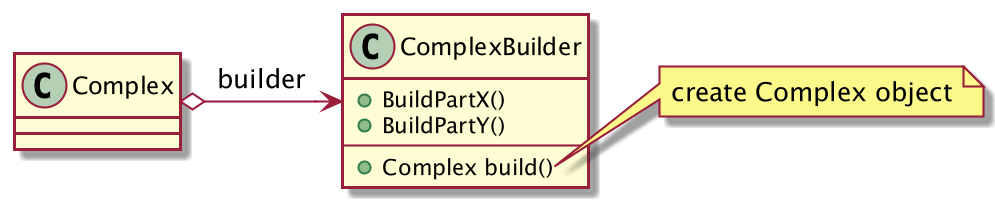
**Builder**

* GOF : Separate the construction of a complex object from its representation so that the same construction process can create different representations.
* Memisahkan pembuatan objek yang kompleks dari representasi sehingga proses pembuatan yang sama dapat membuat representasi yang berbeda.

**Motivasi**

* Pada kenyataannya, kita sering sekali menemui class yang sangat kompleks dan butuh banyak hal yang perlu disiapkan sebelum membuat object dari class tersebut
* Ada baiknya pada kasus ini, kita implementasikan builder pattern
* Dalam builder pattern, kita akan membuat builder class, yang digunakan sebagai class untuk melakukan konstruksi pembuatan class yang telah ditentukan

**Struktur**



**Keuntungan**

* Memisahkan kompleksitas konstruksi object dari class nya
* Menghilangkan kompleksitas class dari client, sehingga client hanya perlu menggunakan builder ketika membuat instance object
* Pembuatan object dengan builder dilakukan bertahap step by step, ini membuat client lebih mudah ketika pembuatan object dibandingkan langsung menggunakan constructor atau function/method