

Java OOP Part 1

Silver Chapter 2 - Topic 3

**Selamat datang di Chapter 2 Topic 3
online course Back End Java dari
Binar Academy!**



Welcome Back! 😊 ✨

Gimana nih latihan kamu pada topik sebelumnya? Jadi makin kebayang nggak sama Algoritma?

Pokoknya, kamu kerenn banget udah bisa menyelesaikan semua latihan sekaligus sampai di topik baru ini. Kasih jempol dulu buat kamu! 👍

Pada slides pembukaan, kamu udah baca sekilas tentang Konsep OOP di Java. Iya, sama kok. Sama-sama mau belajar konsep baru ini. Apa sih **konsep OOP di Java?**



Detailnya, kita bakal bahas hal-hal berikut:

- Konsep OOP
- 4 Pilar OOP
- Klasifikasi Class, Object, Field dan Method
- Konsep Access Modifiers
- Dua pilar OOP - Encapsulations dan Polymorphism
- Mengenal Lombok sebagai Library



OOP merupakan singkatan dari Object Oriented Programming.

Yes, bener. Masih ada kaitannya sama bahasa pemrograman.

Supaya pemahaman kita makin detail, gimana kalau kita jelaskan konsep dasar OOP lewat Karakteristik OOP?



OOP itu apa sih?

OOP atau kependekan dari **Object Oriented Programming** adalah bahasa pemrograman yang berorientasi pada object.

Menurut si OOP ini, **segala kegiatan, aktor dan detail yang terlibat di dalam Java** dianggap sebuah object.



Terus karakteristiknya OOP itu gimana?

Biar makin paham sama si OOP, kamu perlu tahu dulu karakteristik dari OOP itu ada apa aja.

Di dalam OOP itu sendiri kamu bakal ketemu sama keyword **class** dan **object**.

Class ibarat sebuah resep masakan. Sedangkan object adalah hidangan yang udah matang dari resep tersebut.



Lebih jelasnya tuh gini.

Pada sebuah kerajaan ada seorang koki yang menciptakan suatu resep masakan kesukaan raja, yaitu nasi goreng teri.

Berhubung raja suka banget sama nasi goreng buatan koki, dia minta si koki buat masak terus nasi goreng teri kesukaannya tiap pagi, siang, sore, atau sampai 1000 tahun lagi dengan resep yang sama tanpa ada perubahan sedikit pun.



Nah, saking seringnya si koki masak nasi goreng teri dengan resep yang sama, jadi dia tuh udah nggak perlu lagi buat bikin resep yang baru.



Kalau di konsep Java, cerita koki dan raja tersebut bisa dibilang sebagai pengaplikasian dari OOP.

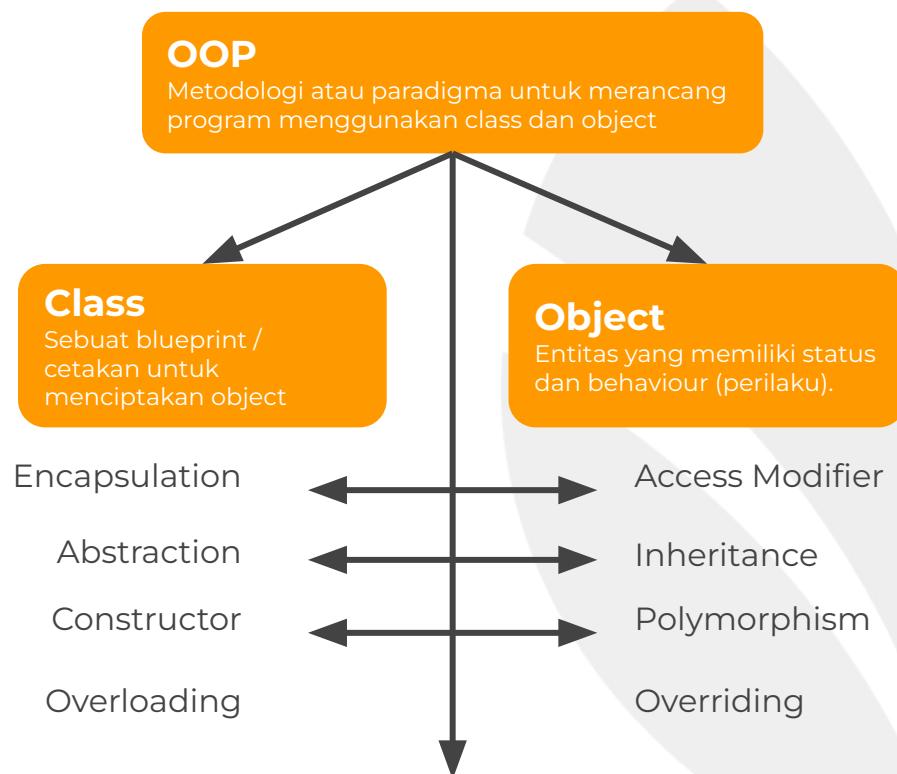
Yaitu, kamu bisa membuat berbagai object dengan cara **me-reuse atau memakai kembali sebuah class**.

Atau gampangnya, kamu udah nggak perlu lagi nulis kode berulang-ulang dalam satu script.



Jadi, Konsep OOP itu meliputi apa aja?

Kurang lebih, bagan konsep **OOP** bisa dijelaskan seperti gambar di sebelah ini:



Ibarat membangun rumah yang
membutuhkan pilar untuk menyangga
kerangka diatas tanah, si OOP juga
memanfaatkan hal yang sama.

Bedanya, konsep pilar yang ada di OOP
berkaitan sama bahasa pemrograman.



4 pilar dari OOP~

Konsep pilar dari OOP ini bisa dibayangkan kayak gambar di samping 

Kalau pilar yang di rumah fungsinya buat menyangga rumah biar kokoh, kalau pilar OOP ini adalah aspek-aspek yang membuat bahasa pemrograman menjadi sebuah OOP.



Sebagaimana gambar disamping udah kasih kamu spoiler tentang pilar-pilar OOP, kita bisa tahu bahwa empat pilar OOP terdiri dari:

1. **Encapsulation**
2. **Polymorphism**
3. **Inheritance**
4. **Abstraction**

Dari empat pilar diatas, khusus pada topik ini kita bakal jelasin dua pilar pertama dulu, yaitu Encapsulation dan Polymorphism.



Sebelum jelasin dua pilar, kita balik dulu nih ke karakteristik OOP.

Pada awal materi sempet di-mention tuh ada istilah *Class*.

Ada yang masih inget *Class* itu ibarat apa?



Jadi, Class itu apa, ya?

Class adalah **blueprint** yang didefinisikan sebagai variabel-variabel **untuk menyimpan data dan fungsi/method** dalam melakukan operasi di suatu data.





Dalam suatu class, ada beberapa komponen, seperti:

1. **Field/atribute**

yaitu variable untuk menyimpan suatu value.

1. **Method**

yaitu fungsi untuk mengoperasikan data yang ada.

3. Constructor

yaitu method khusus yang memiliki nama sama, dengan nama class.

Code block yang ada di constructor bakal dieksekusi pas class dibentuk jadi object atau **diinstance**. Dari situ, kalau nggak ada constructor, maka class nggak bakal bisa diinstance buat bikin object.



Nama class: Nasabah

Field: Nama dengan tipe data String

Method

Method

Constructor

```
public class Nasabah {  
    private String nama;  
    public void setNama(String nama){  
        this.nama = nama;  
    }  
    public String getNama(){  
        return this.nama;  
    }  
    public Nasabah(){};  
    public Nasabah (int nama){  
        this.nama = nama;  
    }  
}
```

Sampai di topic ini, kamu udah kenal nih sama keyword public dan private.

Lebih lengkapnya, nanti bakal kita bahas di subtopic Access Modifier ya!

Masih ada hubungannya sama Class, Object juga masih termasuk dalam karakteristik utamanya OOP.

Apa sih peran Object dalam OOP?

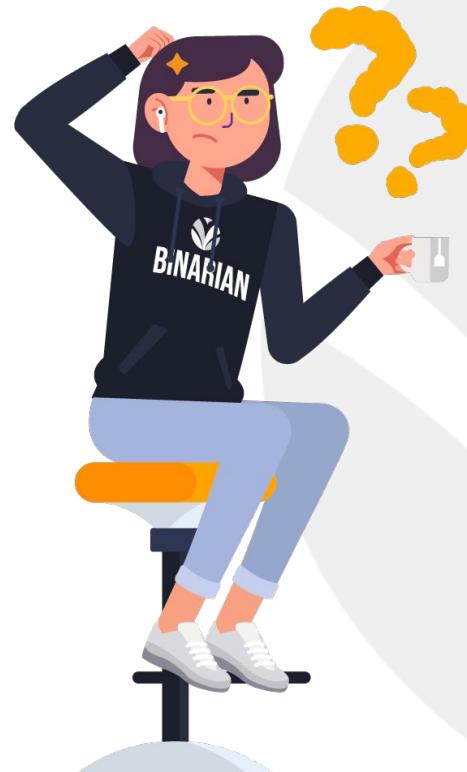


Sebenarnya object itu apa, sih?

Sebelum paham sama konsep Object, kita harus tahu juga asal muasal dari object itu sendiri.

Object adalah **instance** dari suatu **class**.

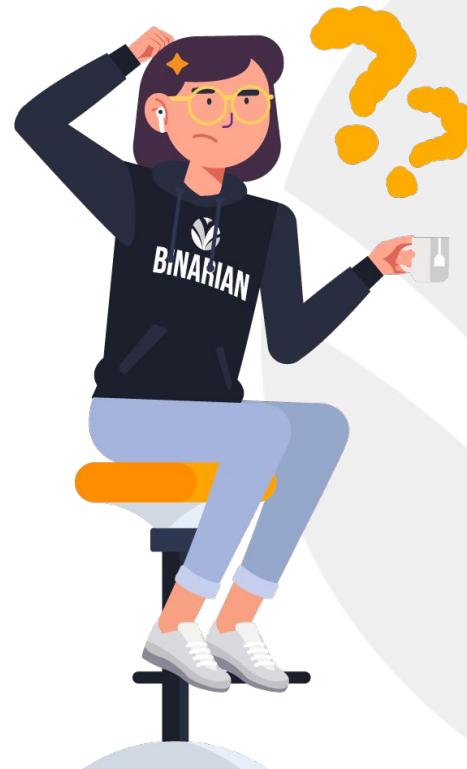
Instantiate sendiri bisa bilang sebagai proses membuat suatu object. Gampangnya, meskipun cuma pake satu class, object bisa dibikin jadi lebih dari satu alias banyak banget.



Dibawah ini merupakan contoh notasi pembuatan object:



```
NamaClass NamaObj = new constructorClass
```



Disamping ini ada contoh instance dari class Nasabah di subtopic sebelumnya:

Ada 2 instance dari class Nasabah, yaitu nasabahBaru dan nasabahLama.

```
Nasabah nasabahBaru = new Nasabah();
```

```
Nasabah nasabahLama = new Nasabah("Bob");
```

Instance dari class Nasabah constructor kosong

Instance dari class Nasabah constructor dengan satu parameter String

```
public class Nasabah {  
    private String nama;  
    public void setNama(String nama){  
        this.nama = nama;  
    }  
    public String getNama(){  
        return this.nama;  
    }  
  
    public Nasabah(){};  
  
    public Nasabah (int nama){  
        this.nama = nama;  
    }  
}
```

Constructor kosong

Constructor dengan satu parameter

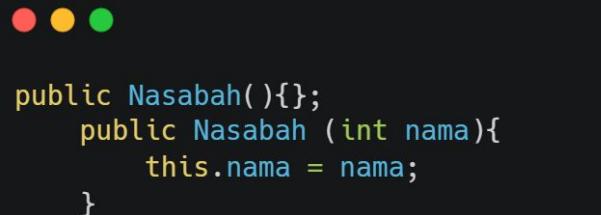


```
public Nasabah(){};
```

- NasabahBaru di-instantiate pakai constructor kosong.

Terus, coba kita lihat struktur dari constructor kosong.

Dengan constructor ini kita nggak ngasih parameter apa-apa dan nggak melakukan eksekusi code pada block constructor karena kosong. Kalau kita akses field nama di object NasabahBaru, maka datanya masih kosong (null).

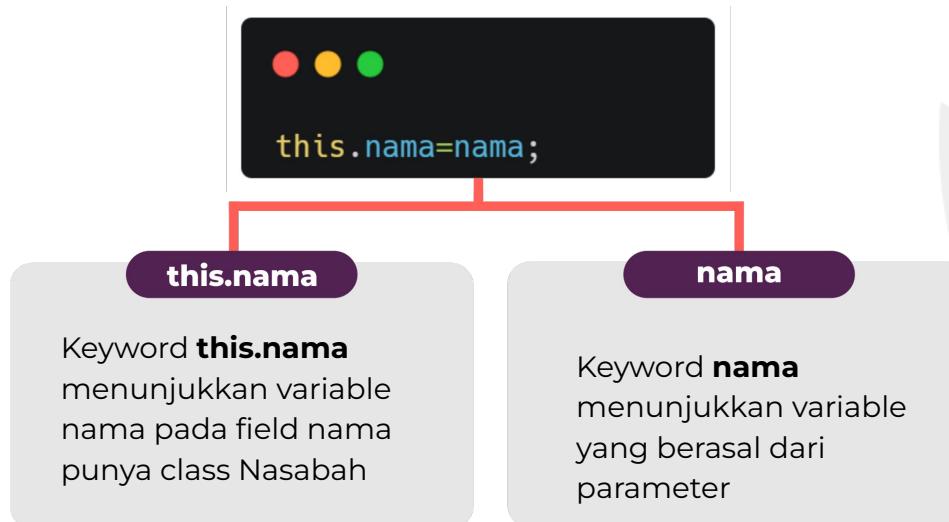


A screenshot of a Java code editor showing a constructor definition. The code is as follows:

```
public Nasabah(){};
public Nasabah (int nama){
    this.nama = nama;
}
```

- NasabahLama di-instantiate pake constructor berparameter.
Coba kita lihat lagi constructor yang punya parameter pada gambar di samping~

Untuk memakai constructor ini kita butuh 1 parameter. Sedangkan di block tersebut udah ada statement.





```
Nasabah nasabahBaru = new  
nasabah( );  
Nasabah nasabahLama = new  
nasabah("Bob");
```

Arti dari statement tersebut adalah kita meng-assign field nama pada class Nasabah pakai argument ketika constructor dipanggil.

Pada object NasabahLama, field nama udah nggak kosong, karena udah di-assign sama argument "Bob".

Setelah udah kebayang sama Class dan Object, kita lanut nih ke **Field**.

Field ini bagian dari Class, dan erat kaitannya dengan Value.

Kayak gimana sih visualisasi field?



Bukan cuma foto aja yang bisa menyimpan sebuah value, Field juga sama lho~

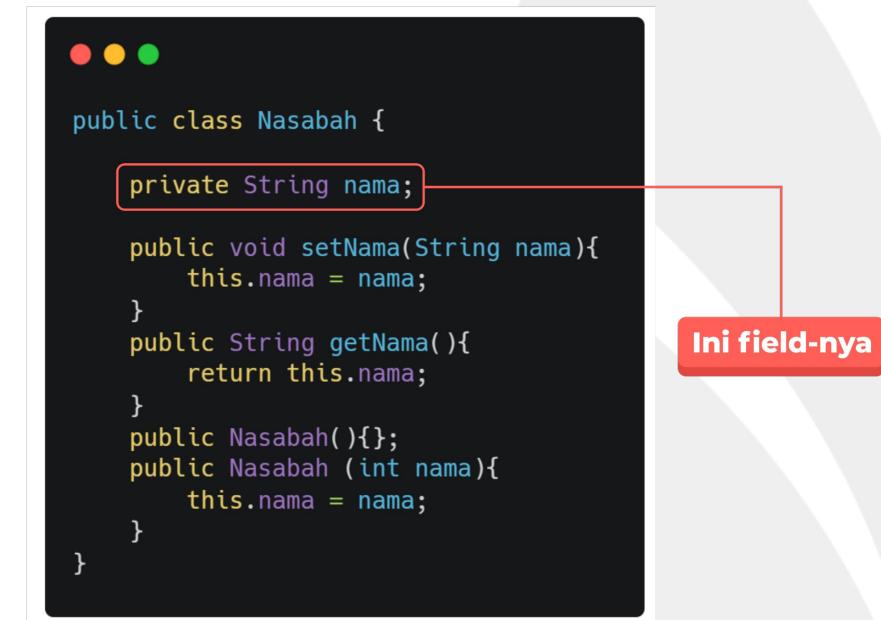
Iya, soalnya **field adalah variable yang fungsinya buat menyimpan suatu value**, seperti numerical value atau text.

```
public class Nasabah {  
    private String nama;  
  
    public void setNama(String nama){  
        this.nama = nama;  
    }  
    public String getNama(){  
        return this.nama;  
    }  
    public Nasabah(){};  
    public Nasabah (int nama){  
        this.nama = nama;  
    }  
}
```

Ini field-nya

Selain itu, field juga punya karakteristik kayak gini, nih:

- Class bisa punya lebih dari satu field
- Field bisa pake berbagai tipe data
- Best practice di OOP, field dilakukan encapsulation.



```
public class Nasabah {  
    private String nama;  
  
    public void setNama(String nama){  
        this.nama = nama;  
    }  
    public String getNama(){  
        return this.nama;  
    }  
    public Nasabah(){};  
    public Nasabah (int nama){  
        this.nama = nama;  
    }  
}
```

Ini field-nya

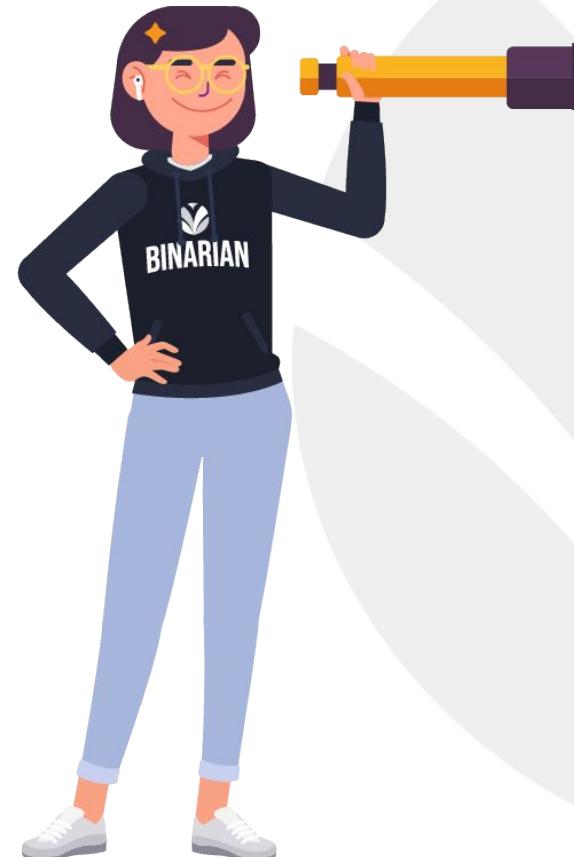
Karena udah ada gambaran tentang OOP dan Field, sekarang saatnya kita bahas tipis-tipis tentang **Method**.

Apa itu Method?



Method merupakan sebuah fungsi untuk mengoperasikan data yang ada.

Misalnya, kalau kita punya **class human**, berarti **method itu ibarat aksi yang bisa dilakukan manusia** seperti berjalan, berbicara, makan, dsb.



Berikut adalah karakteristik method pada Java, yaitu:

- Sebuah class bisa punya lebih dari satu method.
- Method bisa berupa **void** atau method yang memiliki return.
- Method juga bisa berupa method dengan berbagai jumlah parameter.

```
public class Nasabah {  
    private String nama;  
    public void setNama(String nama){  
        this.nama = nama;  
    }  
    public String getNama(){  
        return this.nama;  
    }  
    public Nasabah(){};  
    public Nasabah (int nama){  
        this.nama = nama;  
    }  
}
```

LANJUTTT

Kita udah mulai masuk ke pertengahan materi, nih.

Setelah Method, kita bakal mempelajari yang namanya **Access Modifiers**.

Berangkatt~



Access modifier adalah keyword untuk menandai hak akses dan modifikasi suatu data.

Kalau pake access modifier, kita bisa membatasi resource-resource mana aja yang bisa diakses.

Untuk melihat access modifier, ada yang berupa public dan private.



Ada 4 Access Modifiers dari Java, Iho~

Kita kupas tuntas satu-satu ya!

1



Public, bisa diakses sama semua class

2



Protected, cuma bisa diakses dari dalam class dan class-class turunannya atau dalam package yang sama, method-method dan konstruktor.

Lanjutannya lagi nih~



Private, cuma bisa diakses dari dalam class aja



Default, cuma bisa diakses sama class-class dalam package yang sama.

Access modifier ini bisa dipakai tanpa harus mendefinisikan access modifier lainnya. Kalau kita nggak mendefinisikan, berarti access modifier yang dipakai adalah default secara otomatis.

Eitss sebentar, ada catatan nih~

Jadi, Local variable di suatu method itu nggak punya access modifier karena variable tersebut cuma bisa diakses sama method content saja.



Oke deh, sekarang kita bakal masuk ke pembahasan inti tentang dua Pilar OOP.

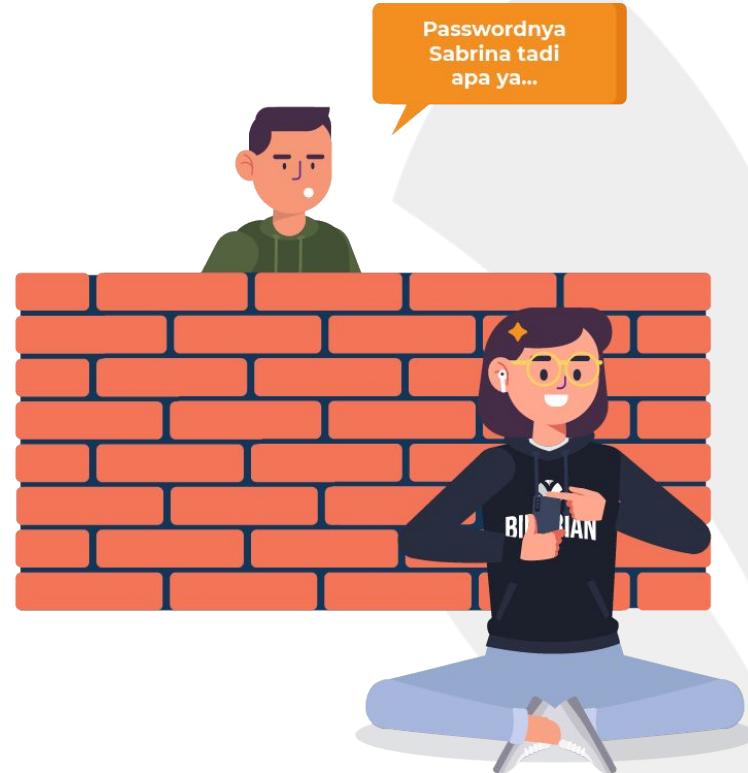
Mulai dari yang pertama dulu, yaitu **Encapsulations**.



Jadi...

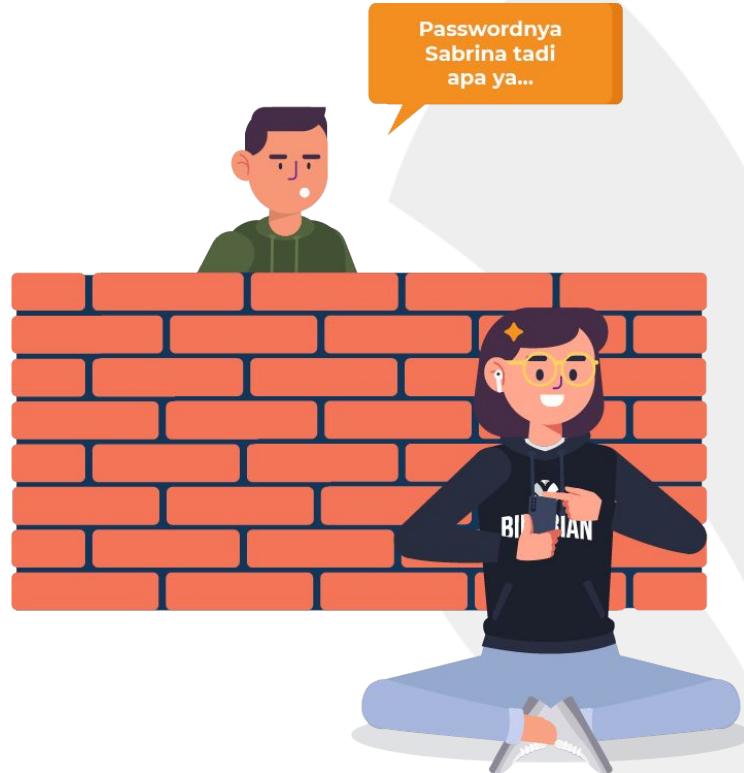
Encapsulation adalah prinsip pada OOP untuk menyembunyikan field dari class lainnya.

Tujuannya, supaya field nggak bisa diakses sembarangan.



Iya. Mirip kayak pas kita bikin kata sandi smartphone buat ngehindarin temen-temen yang suka kepo~

Cara kerja encapsulation ini bisa dengan memanfaatkan access modifier, yaitu **bikin access modifier dari semua field jadi private**.



Karena encapsulation membuat field nggak bisa diakses secara sembarangan, terus gimana caranya untuk bisa mengakses dan modifikasi field tersebut?

Caranya, butuh dua method bantuan yaitu method getter dan setter.

- **Method getter**, dipakai untuk mengambil value field dari suatu object.
- **Method setter**, dipakai untuk memodifikasi value field dari suatu object.



Coba kita lihat contoh class Nasabah.

Field pada class Nasabah, yaitu “nama” punya access modifier private yang artinya nggak bisa langsung di-set. Misalnya, “nama” punya access modifier public, berarti bisa diassign dan diakses seperti berikut:

```
Nasabah nasabahBaru = new  
nasabah();  
nasabahBaru.nama =  
“Patrick”;  
String namaAkun =  
nasabahBaru.nama;
```

Coba kita bandingkan.

field "nama" bisa langsung diassign
tanpa pake method setter dan bisa
diakses tanpa pake method getter



```
Nasabah nasabahBaru = new
nasabah();
NasabahBaru.nama =
"Patrick";
String namaAkun =
NasabahBaru.nama;
```

Sedangkan kalau private bakal jadi:



```
Nasabah nasabahBaru = new
nasabah();
NasabahBaru.setNama("Patrick")
;
String namaAkun =
NasabahBaru.getNama();
```

Berikut bentuk umum dari method getter dan setter!

- Penamaan method selalu get dan set. Selain itu, nama field yang dibuat harus dalam bentuk camel case.
- Method setter nggak punya return, sedangkan method getter punya return. Return dari method getter sesuai dengan tipe data dari field tersebut.
- Access modifier getter dan setter harus public. Method ini bisa diauto-generate dengan IDE

```
● ● ●  
public void setNama(String nama){  
    this.nama = nama;  
}
```

• **method setter**

```
● ● ●  
public String getNama(){  
    return this.nama;  
}
```

• **method getter**

Pilar pertama udah ada bayangan, selanjutnya kita bakal bahas pilar kedua, yaitu **Polymorphism**.

Polymorphism itu apa, sih?



Polymorphism adalah **prinsip yang memungkinkan sebuah method memiliki berbagai bentuk**.

Maksud dari “bentuk” di sini adalah isi, tipe data, dan parameternya berbeda walau namanya sama.

Kalau di kehidupan, kita punya contoh yang bakal dijelasin lewat cerita Mas Gun. Yuk geser dulu slide-nya!



Jadi gini...

Mas Gun adalah seorang object pria yang bisa punya karakteristik/method yang beda-beda.

Mas Gun bisa jadi seorang ayah, suami, karyawan, ataupun seseorang dalam keluarga.

Dari situ bisa kita simpulin nih, kalau Mas Gun adalah seorang pria yang sama, tapi punya karakteristik yang berbeda dalam situasi yang berbeda.





Method Overloading Vs Method Overriding

Pada subtopic sebelumnya, kita udah kenalan sama contoh dari polymorphism yaitu penggunaan constructor kosong dan constructor berparameter.

Polymorphism yang dimungkinkan pada Java adalah Method Overloading dan Method Overriding.

Terus, bedanya apa?



Perbedaan Method Overloading dengan Method Overriding

- **Method Overloading** adalah penggunaan method dengan **nama yang sama**, tapi pakai **parameter yang berbeda** atau jumlah parameter yang berbeda.
- **Method Overriding** adalah penggunaan method dengan **nama yang sama**, tapi pake **implementasi method yang berbeda**.

Di topic ini, kita nggak bakal bahas detail tentang Method Overriding karena menyinggung pilar lain, yaitu inheritance yang bakal kita pelajari di topic 3. Sabar, ya~



Berikut adalah contoh dari method overloading!

Pada method overloading, diperbolehkan kalau jumlah parameternya sama.

Walaupun begitu, tapi tipe parameternya harus berbeda ya!

```
public AuthenticationException(Throwable t) {  
    super(t);  
}  
public AuthenticationException(String message) {  
    super(message);  
}  
public AuthenticationException(String message, Throwable t) {  
    super(message, t);  
}
```

Pada topik ini kita jelasin dua pilar OOP dulu, nih. Ditunggu ya penjelasan dua pilar lainnya~

Selanjutnya, kita bakal move on ke **Lombok Library Java**.

Nama yang unik, ya. Nggak cuma namanya, penggunaan Lombok Library juga unik banget!



Lombok si Library Java~

Lombok adalah library Java yang memungkinkan pembuatan beberapa method seperti getter, setter dan constructor.

Dengan memanfaatkan Lombok, ketiganya jadi lebih mudah dibuat karena pakai annotation. Terus, annotation itu apa?



Annotation adalah keyword yang dipakai dengan prefix @ yang bisa diletakkan sebelum class, field, method ataupun parameter untuk pakai librarynya.

Kalau pake Lombok, coding bakal jauh lebih rapi.



Berikut adalah Annotation yang sering dipakai~

1. **@Getter**, buat bikin method getter
2. **@Setter**, buat bikin method setter
3. **@RequiredArgsConstructor**, buat bikin Constructor dengan parameter seluruh field yang ada
4. **@NoArgsConstructor**, buat bikin Constructor kosong



Berikut contoh penggunaan Lombok sehingga code menjadi jauh lebih ringkas~

Sebelum menggunakan library Lombok

```
● ● ●  
public class Nasabah {  
    private String nama;  
    public void setNama(String nama){  
        this.nama = nama;  
    }  
    public String getNama(){  
        return this.nama;  
    }  
    public Nasabah(){};  
    public Nasabah (int nama){  
        this.nama = nama;  
    }  
}
```

Setelah menggunakan library Lombok

```
● ● ●  
@Getter  
@Setter  
@RequiredArgsConstructor  
@NoArgsConstructor  
public class Nasabah {  
    private String nama;  
}
```



The screenshot shows a Java IDE interface with two code files and an outline view:

- Runner.java**:

```
1 public class Runner {  
2     public static void main(String[] args) {  
3         Alien a1 = new Alien();  
4     }  
5 }  
6  
7 }
```
- Alien.java**:

```
1 import lombok.Data;  
2 import lombok.Getter;  
3 import lombok.Setter;  
4  
5 @Getter  
6 @Setter  
7 public class Alien {  
8     private int age;  
9     private String name;  
10    private String tech;  
11  
12 }  
13  
14  
15  
16 }  
17
```
- Outline** view:
 - Alien class
 - getAge(): int
 - getName(): String
 - getTech(): String
 - setAge(int)
 - setName(String)
 - setTech(String)
 - age: int
 - name: String
 - tech: String



Walau code bakal terlihat lebih rapi, Lombok tidak akan menghilangkan fungsi utama dari code yang kamu tulis.
Kalo kamu mau eksplor Lombok lebih lanjut, kamu bisa tonton video di bawah ini 

[Project Lombok | GoodBye Boilerplate Code](#)

Bip bip~

Ada misi buat kamu nih biar makin mantap belajar OOPnya.

Buatlah implementasi konsep OOP berupa encapsulations dan polymorphism dengan konsep Java.

Nanti diskusikan jawabannya bersama teman sekelas dan facilitator ya!



Gimana nih sob tanggapan kamu setelah kenalan sama si OOP ini?

Menurutmu, seberapa penting sih OOP ini dipelajari oleh seorang Back End Java engineer? Dan apa yang bakalan terjadi kalo seorang BE Java Engineer nggak menguasai OOP ketika bekerja?



Nah, selesai sudah pembahasan kita di Chapter 2 Topic 3 ini.

Selanjutnya, kita bakal bahas tentang Java OOP part 2.

Penasaran kayak gimana? Yuk, langsung ke topik selanjutnya~

