|  |
| --- |
| **Java** |
| **Catatan Belajar Java** |

# Java Dasar

## Sejarah Java

Java merupakan Bahasa pemrograman yang banyak digunakan oleh developer dalam mengembangkan aplikasi desktop, mobile dan juga website. Java dikenal dengan motto *“Write Once, Run Anywhere”* artinya java merupakan Bahasa pemrograman yang mampu diciptakan di berbagai platform tanpa harus penyeusaian ulang di platformnya.

Alasan Java dapat dijalankan di berbagai platform karena Bahasa ini merupakan Bahasa Tingkat tinggi. Di java terdapat **JVM** (*Java Virtual Machine*) Dimana akan merubah syntax menjadi kode numeric (*bytecode*) sehingga dapat dijalankan diberbagai perangkat.

Java dibuat oleh *James Gostling* saat bekerja di *Sun Microsystem* dan java mulai dirilis public saat tahun 1995.

Komponen – Komponen JAVA :

* JVM (*Java Virtual Machine*)

= Jantung dari java yang merubah code menjadi kode byte.

* JDK (*Java Development Kit*)

= Komponen utama dari java di dalamnya mencakup JRE dan juga JVM.

* JRE (*Java Runtime Environment*)

= Menjalankan aplikasi yang sudah dikompilasi kedalam bytecode & JRE tidak termasuk compiler karena hanya menjalankan aplikasi java yang sudah jadi.

### Menjalankan Kode Java

File Java itu berekstensi file .java Ketika kita menjalankan java terdapat 2 cara, seperti berikut :

* Compile ke dalam ekstensi .class

Anggap kita mempunyai sebuah file java “HelloWorld.java”

***javac HelloWorld.java***

= akan menghasilkan hasil kompilasi yaitu file .class

***java Hello World***

= akan menjalankan file .class

* Langsung dijalankan

***java HelloWorld.java***

= langsung dikompilasi dan dijalankan

Berikut adalah contoh Gambaran Proses Development Java.

A diagram of a computer program

Description automatically generated

## Instalasi Java (JDK)

Pada url tersebut <https://openjdk.org/> merupakan salah satu implementasi JDK yang open-source dan gratis. Terdapat alternatif lain selain OpenJDK yaitu Oracle JDK, Amazon Corrreto, dan Zulu.

Download JDK java pada url berikut <https://jdk.java.net/>

Installasi :

* Setelah download install dan simpan di dalam folder C:
* Kemudian copy path urlnya seperti berikut “*C:\openjdk-22.0.1\jdk-22.0.1*”
* Buka setting sistem environtment & pergi untuk menambahkan *System Variables* baru
* Klik New dengan nama **JAVA\_HOME** dan value nya diisi path tersebut.
* Kemudian Pilih di *System Variables* Path klik edit, klik new dan tambahkan “*%JAVA\_HOME%\bin*”
* Buka cmd lakukan java –version untuk mengecek JRE dan javac –version untuk mengecek compiler atau JVM.

### Integreted Development Environment (IDE)

IDE adalah smart editor yang digunakan untuk mengedit kode program. IDE juga digunakan untuk melakukan otomatisasi proses kompilasi kode program dan otomatisasi proses menjalankan program.

IDE yang biasanya diguanakn untuk membuat code program JAVA :

* Intelli JDEA Ultimate / Community
* Eclipse
* NetBeans
* JDeveloper
* Visual Studio Code

## Program Awal - Hello World

A screen shot of a computer program

Description automatically generated

Sebelumnya buat folder dengan nama **java\_dasar** didalamnya akan berisi kode kode program java dasar. Karena disimpan di dalam sebuah folder **java\_dasar** kita harus memanggil foldernya menggunakan *package*.

Di dalam java kita harus membat sebuah Class utama yang didalam nya berisi sebuah ***main*** function, agar Ketika dijalankan maka main function tersebut yang akan dijalankan. Pembuatan main function :

***public static void main (String[] args) { … }***

System.out.println() = digunakan untuk mencetak data dan memberikan new line.

System.out.print() = sama dengan println namun tidak memberikan new line.

## Tipe Data Number

### Integer Number

Integer Number, atau kita anggap adalah bilangan bulat di dalam java terdapat banyak kita bisa menggunakan *byte, short, int, dan long*. Semuanya punya min max pada nilainya yang berbeda beda, semakin besar maka semakin banyak juga memakan memory.

A computer screen shot of numbers and lines

Description automatically generatedA table with numbers and a number

Description automatically generated

Kita dapat memberikan sebuah Keyword (***L***) pada long sebagai menandakan bahwa variable tersebut menggunakan tipe data *long.*

### Floating Point Number

Sedangkan floating point merupakan sebuah nilai (number) yang memiliki nilai di belakang koma, seperti 3,14 maka kita gunakan floating number baik itu *float* atau *double*.

**A close-up of a number

Description automatically generated**

**A number and lines on a black background

Description automatically generated**

Float juga mirip seperti Long dapat diberikan penanda dengan (***F***) bahwa nilai floating point tersebut adalah float.

### Literals Number

Literals Number merupakan sebuah nilai (number) yang biasanya digunakan saat kondisi tertentu seperti membuat nilai hexadecimal yang diawali dengan (***0x***), sedangkan untuk bilangan binary dapat diawali dengan (***0b***).

A computer screen shot of a number

Description automatically generated

### Underscore Number

Terkadang jika nilai kita berupa angka Panjang kita memisahkannya menggunakan tanda . (titik) namun di java tidak seperti itu di java kita menggunakan tanda \_ (underscore) untuk nilai yang panjang karena . (titik) digunakan untuk floating point number.

**A number and numbers on a black background

Description automatically generated**

### Konversi Tipe Data Number

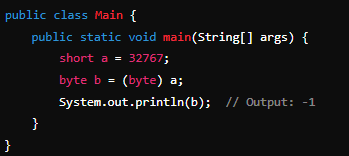
Di dalam Konversi Tipe Data Number terdapat dua cara dalam melakuan konversi, ada *Widening Casting* (Konversi secara otomatis) dan ada juga *Narrowing Casting* (Konversi secara manual). Lantas apa perbedaan diantara keduanya?

***Widening Casting*** *kita dapat merubah tipe data yang lebih kecil ke tipe data yang lebih besar.*

***Narrowing Casting*** merupakan konversi tipe data yang lebih besar ke tipe data yang lebih kecil, ini harus dilakukan secara eksplisit karena ada kemungkinan kehilangan data atau lebih dikenal dengan *Downcasting*.

*\*perlu diketahui Narrowing Casting nilai dari sebuah tipe data akan dibuat ke dalam byte code hal ini akan menyebabkan kehilangan data / perubahan nilai aslinya.*

Contoh Narrowing Casting :



Nilai awal *a* adalah 32767. *A* merupakan tipe data short yang sizenya merupakan 2 byte. ***1 byte adalah 8 bit.***

Ketika *a* diubah kedalam bit code maka menghasilkan 0111 1111 1111 1111 (16 bit).

Sedangkan tipe data byte hanya bernilai 1 byte atau 8 bit maka, Ketika dikonversi dari short ke bit akan terjadi perubahan data karena nilai yang diambil hanya pada 1111 1111 (8 bit).

Lalu bagaimana dengan hasilnya? Sebelum kesana kita harus mengetahui matematika ***kompenen***. Dimana kompenen ini sangat dipakai dalam menghitung byte.

#### Menghitung Nilai Kompenen

Nilai komplemen pada byte ada 2, ada one’s complement dan two’s complement. Komplemen ini akan ***mereverse*** dari nilai awalnya. Kita beri contoh perhitungannya seperti berikut :

Tadi nilai yang didapat adalah 1111 1111

*One’s Complements (reverse to binary)*

1111 1111 -> 0000 0000 = 0

*Two’s Complements (reverse to binary)*

1111 1111 -> 0000 0000 + 1

0000 0001 = -1

*Jika diawali dengn nilai 0 maka bilangan tersebut merupakan bilangan negative sedangkan jika diawali dengan 1 maka bilangan tersebut merupakan bilangan positif.*

Kesimpulannya, Java itu menerapkan two’s complement dalam melakukan narrowing casting, ini dilakukan karena untuk menghitung nilai wal yang memiliki nilai yang besar Ketika dikonveri ke nilai yang kecil.

*\*jika konversi nilainya tidak sesuai dari max / min tipe data konversinya maka akan terjadi ‘number overflow’ artinya nilainya dapat direpresentasikan oleh tipe dengan bilangan tertentu.*

A screenshot of a computer program

Description automatically generated

## Tipe Data Character

Di java tipe data ***char*** merupakan tipe data untuk 1 buah karakter (huruf) dengan ditandai menggunakan ‘ ’ (petik satu).

A computer screen shot of a computer code

Description automatically generated

## Tipe Data Boolean

Boolean di java direpresentasikan dengan nama tipe data ***boolean***. Default nilai dari Boolean di java adalah false.

A computer screen shot of a number of text

Description automatically generated with medium confidence

## Tipe Data String

Tipe data String adalah tipe data yang berisikan karakter atau sederhananya adalah teks. Di java ini direpresentasikan dengan *String*. Default nilai String adalah null dan untuk membuatnya pastikan menggunakan petik 2.

A computer code with colorful text

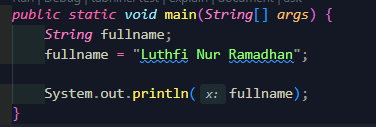
Description automatically generated

*Untuk menggabungkan lebih dari 1 string dapat menggunakan* ***+***

## Variabel

Variabel adalah tempat untuk menyimpan data. Java adalah Bahasa yang static type (*pendeklarasian tipe data saat kompilasi*), sehingga sebuah variable hanya bisa digunakan untuk menyimpan tipe data yang sama, tidak bisa berubah ubah.

Untuk membuat variable di Java kita bisa menggunakan nama tipe data lalu diikuti dengan nama variabelnya.

**

### Keyword Var

Java sekarang sudah mendukung kata kunci var sehingga kita bisa tidak perlu menyebutkan secara eksplisit tipe datanya. Namun perlu diingat, saat kita membuat kata kunci var untuk membuat variable, kita harus menginisialisasi value secara langsung.

A screen shot of a computer code

Description automatically generated

### Keyword Final

Secara default variable datanya dapat diubah ubah nilainya. Jika kita ingin membuat sebuah variable yang datanya tidak boleh diubah ubah setelah pertama kali dibuat kita bisa menggunakan kata kunci ***final*** *(constant).*

A computer screen with colorful text

Description automatically generated

## Tipe Data Bukan Primitif

Tipe data primitive adalah tipe data bawaan Bahasa pemorgraman. Tipe data ini tidak bisa diubah lagi. Tipe data *number, char, boolean* adalah tipe data primitive. **Tipe data primitive harus selalu memiliki default value**. Sedangkan String bukan termasuk tipe data primitive, Tipe data yang bukan primitive tidak memiliki default value dan bisa bernilai Null.

*\*Jika data bukan primitive bisa memiliki sebuah method / function. Di Java, semua tipe data primitive memiliki representasi tipe data bukan primitive-nya.*

Sebelah kiri merupakan tipe data primitive sedangkan disebelah kanan merupakan tipe data bukan primitive berupa sebuah ***Object***.

A screenshot of a computer

Description automatically generated

A graph with red line

Description automatically generated with medium confidence

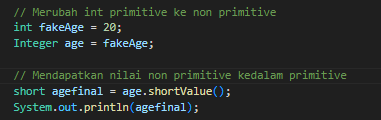
### Konversi ke Bukan Primitif

Jika tipe data awal kita primitive dan ingin menggunakan Object dari tipe data yang bukan primitive, kita bisa lakukan seperti berikut.

A computer screen shot of a math equation

Description automatically generated

Jika ingin melakukan konversi dari Object / Bukan Primitive ke dalam Primitive dapat dengan menggunakan method namaTipeDataValue(), seperti berikut.



## Tipe Data Array

Array adalah tipe data yang berisikan Kumpulan data dengan *tipe data yang sama*, jumlah data di array tidak bisa diubah sejak pertama kali dibuat. Diawali dengan nama tipe data nya, kemudian inisialisasi datanya dan berikan jumlah data yang bisa ditampung.

A computer screen shot of a program code

Description automatically generated

### Array Initializer

Cara membuat array tidak hanya seperti itu kita bisa membuat array dengan beberapa cara :

A computer screen shot of a code

Description automatically generated

*Cara biasa ini kita harus menambahkan keyword new dan nama tipe data beserta jumlah array yang bisa ditampungnya.*

A computer screen shot of a code

Description automatically generated

*Cara kedua ini hamper sama dengan yang pertama namun kita tidak menyisipkan jumlah arraynya oleh karena itu array ini tidak memiliki batasan namun kita harus membuat isi datanya secara langsung karena jika tidak maka akan error.*

A screenshot of a computer code

Description automatically generated

*Cara ketiga ini adalah cara yang paling simple karena kita cukup gunakan { } untuk mengisikan arraynya, namun array wajib diisi saat pertama kali dibuat.*

Di java tidak ada istilah menghapus data array, jika kita menggunakan Object maka kita bisa secara langsung memberikan nilai null agar mengkosongkan nilainya. Jika menggunakan tipe data bukan primitive maka kita bisa mengisikan nilainya menjadi **null**.

### Array in Array

Ketika kita ingin membuat array di dalam array, caranya masih mirip dengan cara sebelumnya hanya saja diberikan 2 array di dalamnya contohnya seperti berikut.

A screen shot of a computer code

Description automatically generated

## Operasi Matematika

Operasi matematika adalah operasi digunakan untuk perhitungan di Java Berikut adalah symbol operasi Matematika di java

A screenshot of a computer

Description automatically generated

### Augmented Assignment

Augemented assignment merupkana cara / gaya penulisan agar lebih singkat lagi di dalam operasi matematika di Java.

A table with text and numbers

Description automatically generated

A screenshot of a computer program

Description automatically generated

### Unary Operator

Unary operator juga merupakan cara / style penulisan untuk mempersingkat operasi matematika di dalam blok kode.

A screenshot of a computer

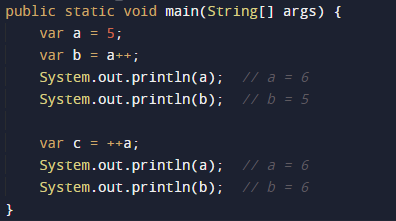
Description automatically generated

A screen shot of a computer program

Description automatically generated

Perlu diingat antara perbedaan a++ dengan ++a.

Jika *a++* biasanya dilakukan saat perulangan karena akan menambahkan value setiap datanya dilooping. Namun jika *++a* maka nilainya akan ditambahkan terlebihi dahulu +1 saat pertama kali. Sebagai contoh berikut.



## Operasi Perbandingan

Operasi perbandingan adalah operasi untuk membandingkan 2 buah data, dan menghasilkan nilai Boolean (true / false).

A screenshot of a computer

Description automatically generated

A screen shot of a computer code

Description automatically generated

## Operasi Boolean

Sebelumnya operasi perbandingan itu menghasilkan sebuah hasil true or false, jika operasi boolean itu melibatkan logika boolean seperti berikut.

A screenshot of a computer

Description automatically generated

**&&** = jika true keduanya maka akan true, sedangkan jika ada false maka akan false.

**||** = jika keduanya false maka akan false, sedangkan jika salah satunya true atau keduanya true maka akan true.

**!** = jika true maka akan false, dan false akan jadi true.

A screen shot of a computer program

Description automatically generated

## Expression, Statement & Block

Ekspression adalah konstruksi dari variabel, operator dan pemanggilan method yang mengevaluasi menjadi sebuah single value.

*Expression adalah core component dari statement.*

A black background with white text and green lines

Description automatically generatedYang hijau merupakan sebuah expression dan juga statement.

Dan yang warna kuning merupakan sebuah statement.

**Statement berisikan execution component**, biasanya diakhiri dengan tanda ; (titik koma).

Ada beberapa jenis statement.

* Assignment expression
* Penggunaan ++ dan –
* Method Invocation
* Object Create Expression

A computer screen shot of a code

Description automatically generated

Block adalah kumpulan statement yang terdiri dari 0 atau lebih statement.

Block diawali dan diakhiri dengan kurung kurawal **{ }**

A screen shot of a computer code

Description automatically generated

## If Statement

Dalam Java **If** statement itu merupakan kata kunci untuk percabangan. Percabangan artinya kita bisa mengeksekusi kode program tertentu ketika suatu kondisi terpenuhi.

**Else if** Statement

Kadang di dalam If kita membutuhkan beberapa kondisi, kita bisa gunakan else if untuk melakukan kondisi pengecekan sama seperti if.

**Else** Statement

Blok if akan dieksekusi ketika kondisi if bernilai true, kadang kita ingin melakukan eksekusi tertentu jika if bernilai false.

A screen shot of a computer program

Description automatically generated

## Switch Statement

Kadang kita hanya butuh menggunakan kondisi sederhana di if statement, seperti hanya menggunakan perbandingan ==.

Switch adalah statement percabangan yang sama dengan if, namun lebih sederhana pembuatannya. Kondisi Switch statement hanya untuk perbandingan == .

A screen shot of a computer program

Description automatically generated

*Keuntungan menggunakan switch itu alur nya lebih mudah untuk dipahami dibandingkan terjebak dengan if else hell.*

*Namun switch case, tidak bisa seperti if adanya operator perbandingan di dalamnya. Sehingga jika kita ingin melakukkan perbandingan cukup lakukan dengan if statement.*

### Switch Lambda

Saat java v.14 diperkenalkan lah switch expression dengan lambda. Ini lebih mempermudah saat penyusunan switch expression kita tidak perlu lagi menggunakan kata kunci break.

A screen shot of a computer code

Description automatically generated

### Kata kunci Yeild

Di Java v.14 ada kata kunci baru juga yaitu yield, dimana kita menggunakan kata kunci yeild untuk mengembalikan nilai pada switch statement. Ini akan sangat mempermudah kita ketika butuh membuat data berdasarkan kondisi switch statement.

*Contoh Switch dengan yield :*

*A screen shot of a computer program

Description automatically generated*

Yield itu seperti melakukan return value. Dimana sebelumnya kita simpan logika switch kedalam sebuah tipe data, dan mengembalikan value dari tipe data tersebut yang sudah diolah menggunakan switch.

## Ternary Operator

Ternary Operator adalah operator sederhana dari if statement. Ternary Operator terdiri dari kondisi yang dievaluasi, jika menghasilkan true maka mengembalikan nilai pertama, jika false maka akan mengembalikan nilai kedua.

Cara menggunakan ternary operator ini lebih cepat, jika if else hanya memiliki 1 kondisi.

A blue and green text

Description automatically generated

## For Loop

For adalah salah satu kata kunci yang bisa digunakan untuk melakukan perulangan. Blok kode yang ada di dalam for akan diulang hingga kondisi for terpenuhi.

A computer code with colorful text

Description automatically generated

Perulangan itu akan dilakukan hingga i bernilai sama dengan 10 dan itu akan berhenti mengulangi block yang ada di dalam for.

*Jika kita melakukan perulangan tanpa kondisi dan initstatement atau (;;) maka nanti akan menghadapi for tanpa henti.*

## While Loop

While loop adalah versi perulangan yang lebih sederhana dibandingkan forLoop. Di dalam while loop hanya terdapat kondisi perulangan tanpa adanya *init statement* dan *post statement*. Namun while loop itu mirip seperti if else lalu loop, karena while melakukan pengecekan terlebih dahulu jika true maka loop akan dijalankan.

Dan untuk memberhentikan loop seperti biasa kita gunakan keyword ***break;***

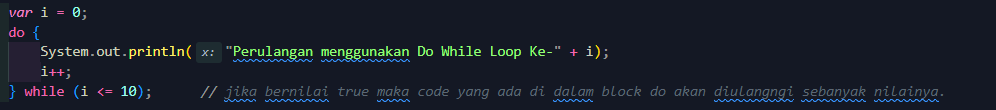
A screen shot of a computer

Description automatically generated

## Do While Loop

Do While loop adalah perulangan yang mirip dengan while. Perbedaannya hanya saat melakukan pengecekan kondisi, Pengecekan kondisi pada while loop dilakukan diawal, sedangkan do while dilakukan setelah perulangan dilakukan.

Oleh karena itu meskipun nilainya tidak sesuai akan tetap diulangi 1x lalu melakukan looping selanjutnya.



## Keyword Break & Continue

Pada switch statement kita sudah mengenal kata kunci break, yang digunakan untuk menghentikan case. Sama juga pada perulangan break juga digunakan untuk menghentikan seluruh perulangan.

Namun berbeda dengan continue, continue akan digunakan untuk menghentikan perulangan saat ini dan akan melanjutkan perulangan selanjutnya.

Contoh break :

A screenshot of a computer screen

Description automatically generated

Contoh continue :

A screen shot of a computer program

Description automatically generated

Dapat dilihat kita membuat sebuah kondisi ketika nilai i modulus 2 maka continue, maksudnya berhentikan, lalu lanjutkan di loop berikutnya.

## ForEach

Kadang kita bisa mengakses data array menggunakan perulangan. Mengakses data array menggunakan for loop itu cukup bertele tele, kita harus membuat counter lalu mengakses array menggunakan counter yang kita buat. Namun terdapat forEach yang bisa digunakan untuk mengakses seluruh data array secara otomatis.

Tanpa menggunakan forEach

A screen shot of a computer code

Description automatically generated

Menggunakan forEach

A screenshot of a computer code

Description automatically generated

*forEach di java cukup berbeda dengan beberapa bahasa pemrograman lainnya karena biasanya (datas as data) sedangkan di java (data : datas). Keduanya mirip hanya saja terbalik di java dan juga java membutuhkan tipe data sebelum pemanggilan data di dalam foreachnya.*

## Method & Method parameter

Method adalah block kode yang berjalan ketika dipanggil. *System.out.println()* ini juga merupakan termasuk method yang betujuan untuk menampilkan data ke console.

Untuk membuat method di java kita juga bisa menggunakan *void* lalu diikuti *nama\_method­() dan diakhiri dengan block. Method* dapat kita sebut juga sebagai *function*.

Karena biasanya function itu identik berada di luar class, jika di java semuanya berada di dalam class oleh karena itu *function-nya* disebut dengan *method*.

A screenshot of a computer program

Description automatically generated

*Karena main itu merupakan static maka hanya boleh memanggil method static juga.*

Di dalam main kita memanggil sebuah method yang berada di class yang sama dan itu static, dengan mengirimkan argument dan ditangkap oleh parameter.

*Argument* adalah data yang dikirimkan saat pemanggilan method, sedangkan *parameter* adalah deklarasi / nilai yang ditangkap oleh Method.

## Method Return Value

Secara default, method itu tidak menghasilkan value apapun, namun kita juga bisa melakukan dimana method mengembalikan return value.

Agar method bisa mengembalikan return value kita harus mengubah kata kunci *void* dengan *tipe\_data* yang akan dihasilkannya.

A screen shot of a computer program

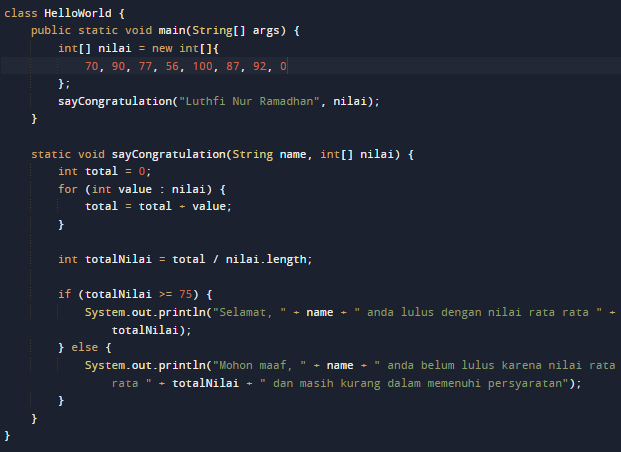
Description automatically generated

## Method Variabel Argument

Kadang kita butuh mengirimkan data yang tidak pasti ke dalam sebuah method. Biasanya dalam kasus seperti ini, kita akan gunakan array sebagai parameter di method tersebut.

Namun di java, kita bisa gunakan variabel *argument*, untuk mengirimkan data yang berisi jumlah tak tentu, bisa nol atau lebih.

Parameter dengan tipe variabel *argument*, hanya bisa ditempatkan di posisi akhir parameter.



## Method Overloading (Same name method)

Method overloading adalah kemampuan untuk membuat method dengan nama yang sama lebih dari 1x. Ketentuannya yaitu data parameter di method tersebut harus berbeda beda, entah jumlah atau tipe parameternya. Jika sama maka akan terjadi error.

A computer screen shot of text

Description automatically generated

## Recursive Method (Factorial)

Recursive Method adalah kemampuan method untuk memanggil method dirinya sendiri. Kadang memang ada banyak problem, yang lebih mudah diselesaikan menggunakan recursive method, contoh kasusnya adalah factorial.

Factorial itu misal saya membuat angka 5 maka akan dikalikan 4, 3, 2, 1.

Berikut Contoh Factorial tanpa menggunakan recursive method

A screenshot of a computer program

Description automatically generated

Penjelasan nya sederhana, ketika valuenya adalah 1 maka kembalikan nilai 1, tapi jika tidak maka jalankan kembali functionnya dan valuenya dikali dan value di dalam method dikurang 1.

A screen shot of a computer code

Description automatically generated

### Problem dengan recursive

Walaupun menggunakan recursive itu menarik, namun kita perlu berhati hati. Jika recursive terlalu dalam, maka akan terjadi error *StackOverflow*, yaitu errornya dimana stack method terlalu banyak dijava.

Kenapa problem ini terjadi?

Karena ketika kita memanggil method, Java akan menyimpannya di dalam stack, jika method itu memanggil method lain, maka stack akan menumpuk terus menerus, dan jika terlalu dalam maka stack nya akan terlalu besar dan menyebabkan error *StackOverflow*.

Contoh kode untuk mengecek kendala rekursive

A screen shot of a computer code

Description automatically generated

## Scope

Di Java, variabel hanya bisa diakses di dalam area dimana mereka dibuat (Scope). Contoh jika kita membuat variabel di dalam sebuah method maka variabel tersebut hanya bisa diakses di dalam block tersebut.

*A screen shot of a computer program

Description automatically generated*

Contoh codenya hi error jika dipanggil diluar dari scope if karena itu variabel hanya bisa diakses di dalam if saja.

## Comment

Kadang kita membutuhkan sebuah komentar seperti memberikan sebuah dokumentasi, komentar ini akan dihiraukan saat melakukan kompilasi / dihapus.

Komentar multi line & single line. Komentar Multi line lebih digunakan untuk dokumentasi, sedangkan single line hanya untuk memberikan informasi pada baris Code.

A blue background with white text

Description automatically generated

# Studi Kasus Java Dasar Aplikasi To Do List

## Prototype Aplikasi ToDoList

Prototype adalah bentuk dasar dari sebuah aplikasi. Prototype bisa digunakan agar memahami flow aplikasi yang akan kita buat, sehingga tidak salah ketika membuat aplikasi. Berikut adalah gambaran prototype aplikasi TodoList yang akan dibuat :

A screenshot of a cell phone

Description automatically generated

## Create Project Java

Untuk membuat project java, di vs code kita bisa dengan cara *CTRL + SHIFT + P*, kemudian cari java: create project. Pastikan sebelumnya sudah menginstall ekstension yang berhubungan dengan Java.

## Membuat Main Class

Main class merupakan class yang akan pertama kali / akan dijalankan di dalam program Java, Biasanya kita membuat 1 main class.

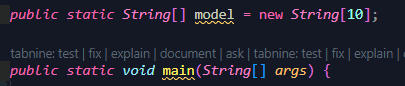
A computer screen shot of text

Description automatically generated

## Membuat Model

Model merupakan Representasi dari data. Nanti setiap data yang masuk akan dimasukkan ke dalam array untuk disimpan. Apabila kita ingin membuat model dengan nilai yang tidak tentu kita bisa ubah *new String[10]* menjadi *{ }*. Namun jika kita menggunakan *{ }* kita tidak bisa menyisipkan data array ke dalamnya.

Model dibuat secara static dan public diluar method main, kenapa diluar? Karena model ini akan diakses oleh method method yang lain.



## Menentukan Bussiness Logic

Sebelumnya kita tentukan di dalam aplikasi akan ada logika seperti apa di dalamnya, misal menampilkan data, create data, dan hapus data. Berikut adalah method method lainnya yang akan dibuat selain method main.



## Menentukan View

Sebelumnya merupakan logic / method untuk menjalankan logika, sedangkan method selanjutnya ini adalah untuk menampilkan view dari aplikasi meskipun nanti view nya hanya berupa command line.

A screenshot of a computer program

Description automatically generated

## Menampilkan Todo List

Ketika kita ingin menampilkan todo list yang berupa array String, pastinya kita melakukan perulangan pada datanya, baik menggunakan forLoop biasa ataupun forEach. Berikut adalah contoh dalam menampilkan todoList

A screenshot of a computer program

Description automatically generated

Pada contoh menampilkan todo list, kita buat sebuah method tambahan untuk melakukan testing, kenapa harus melakukan testing? Karena agar pengujian yang dilakukan tidak pada code utama nya.

## Menambah Todo list

Berikutnya kita akan coba menambahkan todo list baru ke dalam model. Untuk menambahkan data ke dalam model pastikan kita buat sebuah pengecekan terlebih dahulu, jadi jika datanya belum penuh maka data tersebut akan disimpan ke dalam model, sedangkan jika sudah penuh ukuran model / total array-nya akan dikali 2 dengan ukuran array sekarang.

Lalu untuk menambahkan datanya kita lakukan perulangan Kembali, jika datanya masih null maka datanya null tersebut akan diisi oleh data yang sekarang diterima.

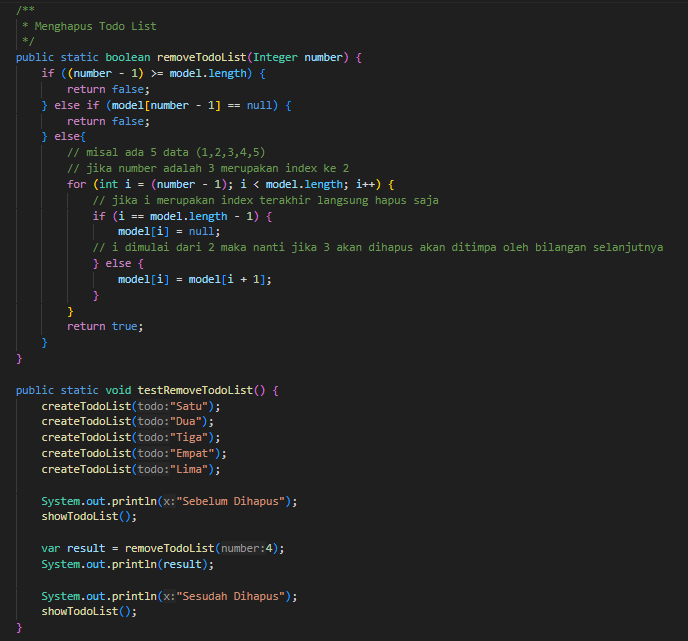
A screen shot of a computer program

Description automatically generated

Sama seperti menampilkan todo List disini kita juga harus membuat method untuk melakukan testing agar menghindari perubahan code pada method utamanya.

## Menghapus Todo List

Selanjutnya adalah menghapus Todo List berikut adalah contoh kode dalam menghapus todolist

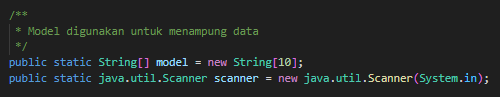


Di dalam method remove kita cek terlebih dahulu jika inputannya tidak sesuai / jika inputannya kosong, dan jika ada datanya dan mau dihapus maka buat data tersebut jadi null kemudian naikkan posisinya ke posisi data yang sudah dihapus, apabila menghapus data ditengah agar data selanjutnya naik. Pastikan membuat method testing agar melakukan percobaan di dalam method tersebut.

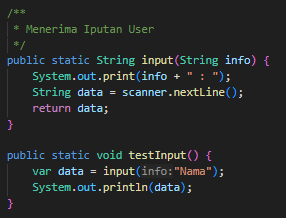
## Input Data

Terkadang kita tidak mungkin membuat inputan melalui codingan langsung, kadang kita sendiri yang ingin mengisikan nilai datanya. Oleh karena itu ikuti cara berikut.

Tambahkan code berikut untuk bisa menerima inputan



Kemudian buat method untuk menerima dan mengembalikan inputan

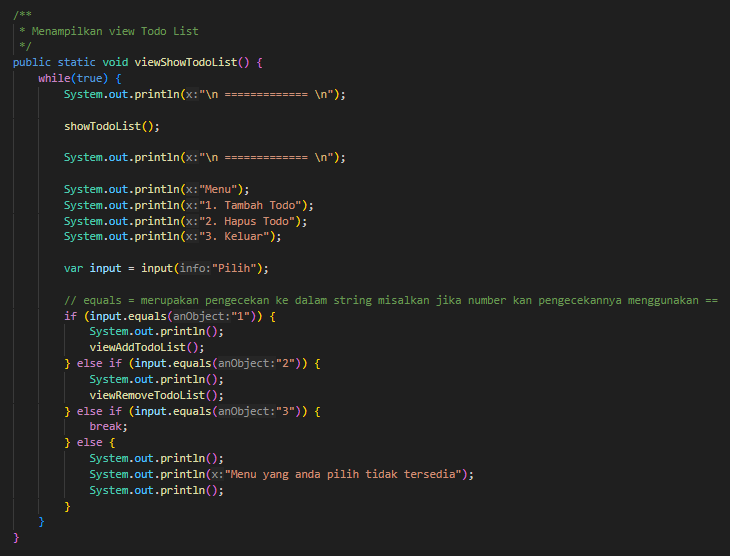


*scanner.nextLine() digunakan untuk mendapatkan inputan dari scanner.*

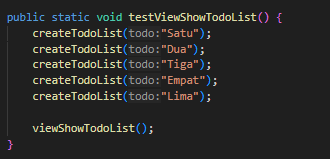
Pastikan juga kita tetap membuat method testing untuk menguji coba method input tersebut.

## View Menampilkan Todo List

Menampilkan todo list ini hanya berupa command line yang akan ditampilkan, berikut adalah method untuk menampilkan todo list.



Terdapat sebuah kondisi dengan menggunakan function equals(). Fungsi tersebut merupakan sebuah pengecekan pada inputan apabila terdapat / menginputkan berupa nilai angka 1, 2, atau 3 maka akan menjalankan method-nya.



Pastikan juga membuat sebuah method testing agar membuat code lebih clean dan menghindari bug.

## View Menambahkan Todo List

Dalam menambahkan todo list juga kita lakuan pengecekan pada inputannya, dan Ketika inputan sesuai maka data akan ditambahkan ke dalam model.



Pastikan kita membuat sebuah method testing juga, agar Ketika ingin melakukan uji coba tidak pada code utama.

## View Menghapus Todo List

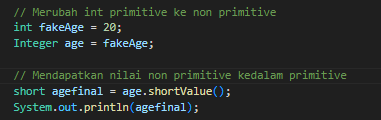
Untuk menghapus todo list kita membutuhkan id / nomor dari todo list nya, karena dengan nomor tersebut maka datanya akan dihapus. Berikut adalah contoh codenya.



Sebelumnya kita bisa melakukan pengecekan pada model dan memberikan message apabila modelnya masih kosong, dan tampilkan jika modelnya emang ada data di dalamnya.

Dikarenakan Ketika ingin melakukan remove adalah Tipe data non Primitive maka kita harus mendapatkan valuenya.

Sebelumnya kita pernah melakukan code berikut



Perlu diingat bahwa menggunakan cara tersebut untuk mendapatkan value dari tipe data non primitive ke primitive oleh karena itu karena *Integer* merupakan sebuah object maka akan memiliki fungsi seperti *shortValue().*

Sedangkan ***valueOf*** itu digunakan untuk mendapatkan value dari tipe data primitive ke non primitive. Oleh karena itu cara diatas menggunakan *valueOf* dikarenakan number disimpan ke dalam tipe data var, dan inputan mengembalikan String.

Pastikan juga untuk membuat method testingnya…



# Object Oriented Programming (OOP) Java

## Pengenalan OOP

Object Oriented Programming adalah sudut pandang Bahasa pemrograman yang berkonsep “Object”. Terdapat banyak sudut pandang dalam Bahasa pemrograman namun yang sangat popular adalah OOP. Ada beberapa istilah di dalam OOP yaitu *Object* dan *Class*.

**Object**, adalah data yang berisi field / properties / attributes dan method / function / behavior. Semua data non primitive juga adalah sebuah Object.

**Class**, adalah sebuah *blueprint, prototype, atau cetakan* untuk membuat Object. Class berisikan deklarasi semua properties dan function yang dimiliki oleh Object. Setiap object selalu dibuat dari Class, dan sebuah class bisa membuat Object tanpa batas.

Sebagai contoh kita bisa analogikan seperti berikut :

Jika kita ingin membuat rumah pasti memiliki rancangan / bahan yang sama, hal ini bisa kita anggap sebagai Class. Dari rancangan atau bahan yang sama ini kita bisa membuat beberapa rumah karena Class bisa dianggap sebagai template. Setiap rumah yang dibuat nantinya itu adalah Object yang telah dibuat menggunakan class, Setiap Object di dalamnya pasti berbeda beda juga yang dibedakan oleh property dan methodnya.

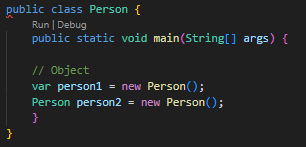
## Class

Untuk membuat class, kita bisa menggunakan kata kunci class, penamaan class bisa menggunakan format CamelCase. Untuk membuat class cukup sederhana hanya seperti berikut.



## Object

Object adalah hasil insatansiasi dari sebuah class, untuk membuat object kita bisa menggunakan kata kunci *new* dan diikuti dengan nama Class dan kurung(). Berikut adalah contoh sederhana dari Object.



## Field / Properties / Attribute

Adalah data yang bisa kita sisipkan ke dalam Object, sebelum kita bisa memasukkan data di dalam fields, kita harus mendeklarasikan data apa saja yang dimiliki object tersebut di dalam deklarasi class-nya, membuat field ini sama seperti membuat variable namun disimpan di dalam class.

Field atau properties ini bisa kita manipulasi, untuk memanipulasinya sama seperti cara pada variable, dan untuk bisa dapat mengaksesnya kita butuh kata kunci . (titik) setelah nama object dan diikuti nama fieldnya.



Berikut adalah contohnya, Ketika kita ingin memanipulasi properties pada object person maka kita bisa dengan cara menambahkan . (titik) dan nama propertiesnya.

Field jika tidak dideklarasikan maka nilainya secara otomatis akan null.

## Method

Selain menambahkan field kita juga bisa menambahkan method ke object, cara mendeklarasikan method tersebut di dalam block class. Sama seperti method biasanya, kita bisa menambahkan return value, parameter dan method overloading di method yang ada di dalam block class.

Untuk dapat bisa mengakses method tersebut, kita bisa menggunakan tanda . (titik) dan diikuti dengan nama method-nya, Sama seperti mengakses field.



## Constructor Method

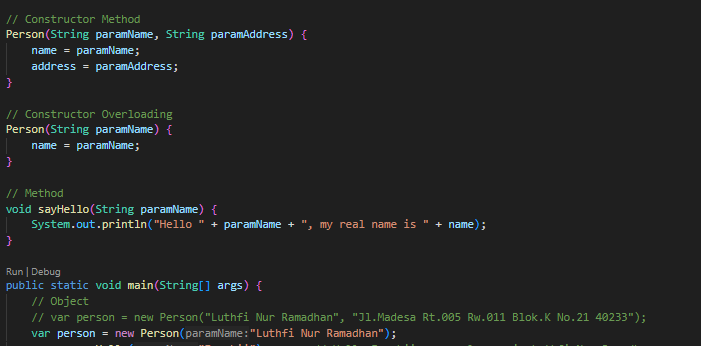
**Constructor** adalah sebuah method yang dipanggil saat object pertama kali dijalankan. Apabila constructor menerima sebuah parameter maka object tersebut Ketika ingin dijalankan harus aga argument yang dikirimkan.

Nama Constructor harus sama dengan nama class, dan tidak membutuhkan kata kunci void atau return value karena constructor tidak mengembalikan data apapun. Sehingga kita dapat merubah kode kita sebelumnya menjadi seperti berikut : 

Parameter wajib di isi, dan mengisi nilai Constructor Ketika kita membuat object baru. Lalu apabika kita mendeklarasikan Kembali name Ketika sesudah constructor maka nilai name hasil constructor akan ditimpa.

## Constructor Overloading

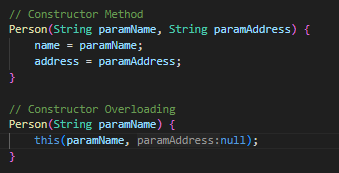
Constructor di java memiliki kemampuan overloading seperti method, constructor dapat dibuat lebih dari 1 syaratnya adalah tipe data parameter harus berbeda atau jumlah parameternya harus berbeda. Contohnya seperti berikut



Karena ada 2 buah Constructor kita jadinya bisa memilih ingin menggunakan constructor mana, apabila ada data yang tidak diisi kan ke dalam constructor maka nilainya akan null.

### Memanggil Constructor Lain

Constructor bisa memanggil constructor lain, hal ini akan memudahkan kita saat butuh menginisialisasi data dengan berbagai kemungkinan, untuk memanggil constructor lain pemanggilannya sama seperti method namun di tambahkan kata kunci *this*. Contohnya sebagai berikut



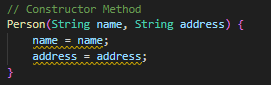
Dengan begini kita mengirimkan data ke constructor ke 2, data tersebut akan ditangkap dan dikirimkan Kembali ke constructor pertama. Jadi Ketika kita memanggil constructor ke 2 ini akan sama hasilnya dengan gambar sebelumnya namun perbedaanya yaitu bahwa constructor 2 ini memanggil constructor lain.

Sebenarnya *this* itu akan merujuk ke Object itu sendiri, karena di dalam Object itu terdapat sebuah Constructor yang pertama kali dijalankan maka akan membutuhkan argument yang dikirimkan untuk constructor pertama.

## Variabel Shadowing

Variabel shadowing adalah kejadian Ketika membuat nama variable dengan nama yang sama di scope yang menutupi variable dengan nama yang sama di scope diatasnya. Ini biasa terjadi seperti kita membuat nama parameter di method sama dengan nama field di class. Ketika terjadi variable shadowing maka variable diatasnya tidak akan bisa diakses.

Berikut adalah contohnya :



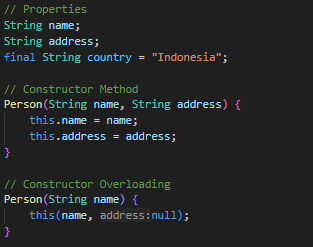
Apabila kode kita buat seperti ini maka property name tidak akan berubah, karena dianggapnya property name diisikan oleh property name itu sendiri bukan berasal dari parameter.

## This Keyword

Saat kita membuat kode di dalam block constructor atau method di dalam class, kita bisa menggunakan kata kunci *this* **untuk mengakses object saat ini**.

Misal kadang kita butuh mengakses sebuah field yang Namanya sama dengan parameter method, hal ini tidak bisa dilakukan jika langsung menyebutkan nama field, kita bisa mengakses nama field tersebut dengan kata kunci *this*. This juga tidak hanya digunakan untuk mengakses field milik object saat ini, namun juga bisa digunakan untuk mengakses method.

*This* bisa digunakan untuk mengatasi masalah variable shadowing.



Perhatikan pada contoh berikut, karena *this* mengacu pada object tersebut maka this.name akan mengacu kepada properties di dalam object tersebut. Dengan begini kita bisa mengatasi masalah variable shadowing.

Kita juga nantinya bisa memanggil this.name atau name untuk mengakses properties di dalam method object tersebut.

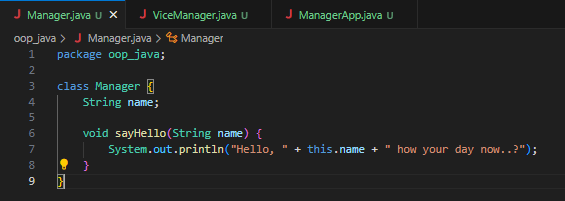
## Inheritance

Inheritance atau Pewarisan adalah kemampuan untuk menurunkan sebuah class ke class lain, dalam artian kitab isa membuat class parent dan class child.

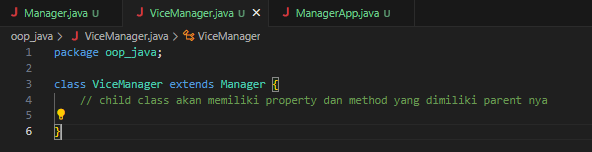
Class child, hanya bisa punya satu class parent, namun satu class parent bisa memiliki banyak class child. Ketika sebuah class diturunkan maka semua field dan method yang ada di class Parent akan dimiliki juga oleh class child nya. Untuk melakukan pewarisan di class child kita harus menggunakan kata kunci extends lalu diikuti dengan nama class parent nya.

Berikut adalah contoh dan penggunaannya

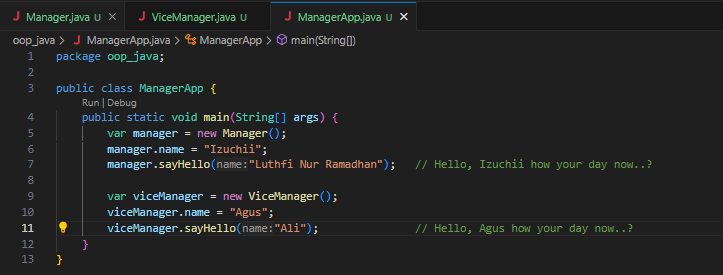
Class parent



Class child, child terhubung dengan menggunakan extends ke parent class



Dan berikut adalah class yang memanggil class parent dan child

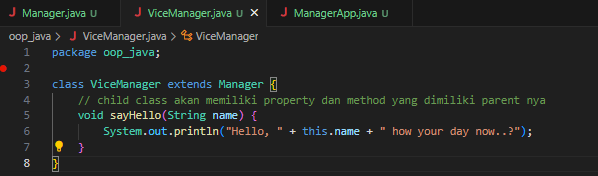


## Method Overiding

Method overriding adalah kemampuan untuk mendeklarasikan ulang method di child class, yang sudah ada di parent class. Saat melakukan proses overriding tersebut kita membuat object dari class child, method yang di class parent tidak bisa diakses lagi.

Contohnya seperti berikut

Method di class child sama persis dengan method di class parent maka method di class parent akan ditimpa dengan method di childnya.

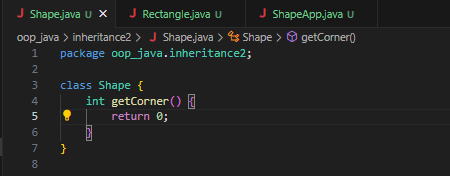


## Super Keyword

Kadang kita ingin mengakses method yang terdapat di class parent yang sudah terlanjut kita override di class child. Untuk dapat mengakses method milik class parent, kita bisa gunakan kata kunci *super*.

Sederhana nya *super* ini digunakan untuk mengakses class parent. Tidak hanya method bahkan property milik parent class pun bisa diakses.

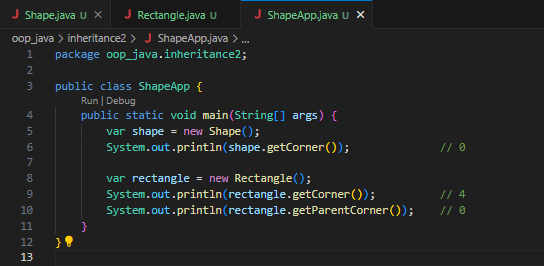
Parent Class



Child Class



Dengan menggunakan super maka akan memanggil method getCorner milik parent class.



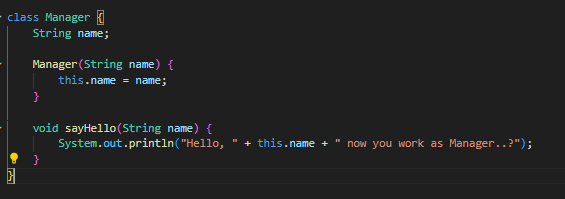
Dan berikut adalah contoh hasilnya Ketika dijalankan.

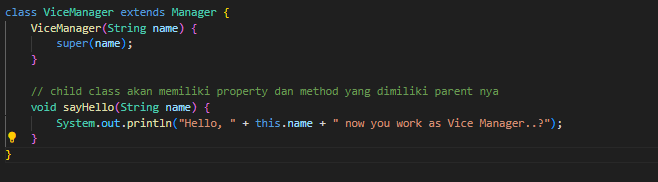
## Super Constructor

Tidak hanya untuk mengakses method atau field yang ada di parent class, kata kunci super juga bisa digunakan untuk mengakses constructor. Namun syaratnya untuk mengakses parent class constructor kita harus mengaksesnya ***di dalam class child constructor.***

*Jika sebuah class parent tidak memiliki constructor yang tidak ada parameternya maka class child wajib mengakes constructor parent tersebut.*

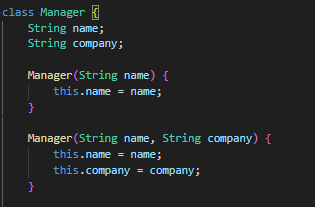
Berikut adalah contoh codenya





*Namun apabila di dalam Parent class terdapat constructor yang tidak menerima apapun itu merupakan default constructor sehingga child class bisa tidak membuat constructor juga.*

Apabila ada 2 constructor di parent class maka kita bisa hanya menggunakan salah satunya saja, anggap kita hanya menggunakan constructor yang hanya menerima Name.

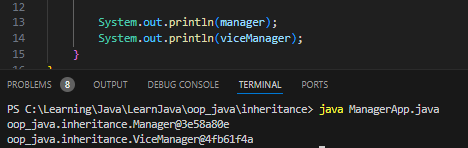


## Object Class

Di Java, setiap class yang dibuat secara otomatis adalah turunan dari class object, walaupun tidak secara langsung kita eksplisit menyebutkan extends object, tapi secara otomatis java akan membuat class kita extends object.

**Bisa dikatakan class Object adalah superclass untuk semua class yang ada di Java.**

Ketika kita melakukan print pada sebuah objek di Java, metode toString() dari objek tersebut dipanggil untuk menghasilkan representasi string dari objek tersebut. Secara default, kelas Object di Java menyediakan implementasi toString() yang mengembalikan nama kelas dari objek yang diikuti oleh tanda "@" dan representasi heksadesimal dari hash code objek.



Method toString() ini merupakan method bawaan dari Object jadi Ketika kita menjalankan code diatas sama saja memanggil dengan toString dan diberikan hash code pada objectnya.

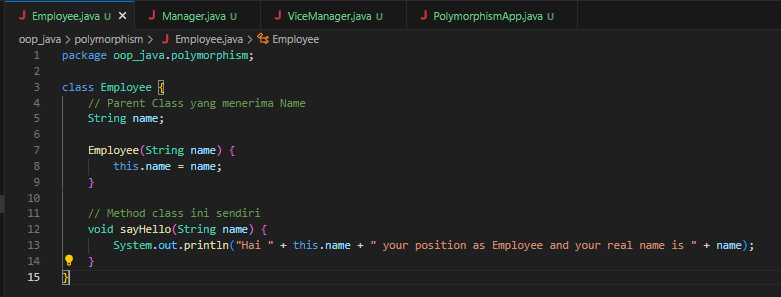
## Polymorphism

Polymorphism berasal dari Bahasa Yunani yang berarti banyak bentuk. Dalam OOP, Polymorphism adalah kemampuan sebuah object berubah bentuk menjadi bentuk lain, Polymorphism erat hubungannya dengan Inheritance.

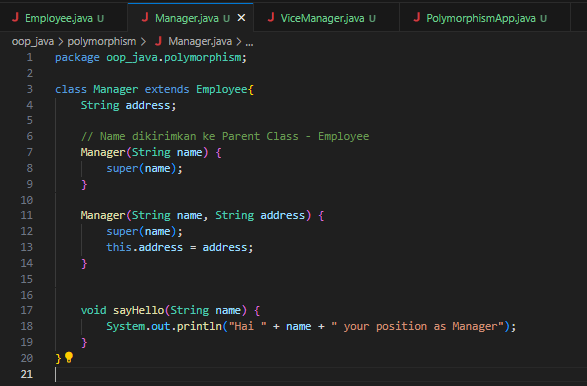
*Employee >> Manager >> Vice Manager*

Object ke Object yang lain itu lah polymorphism. Perhatikan baik baik code berikut.

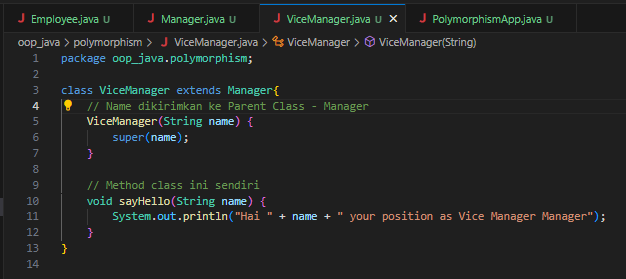
Parent Class Employee



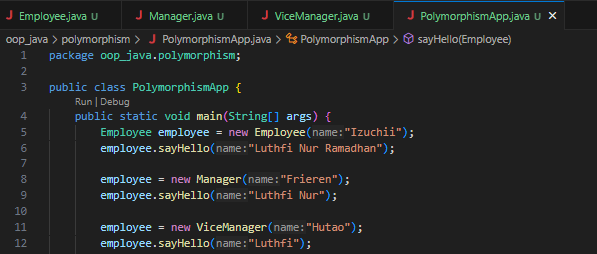
Class Manager



Class Vice Manager



Penjelasan, jika dilihat dari inheritance berikut semua class baik child dan parent memiliki sebuah constructor dan juga method. Hanya Class utama yang menyimpan data name. Untuk dapat menjalankannya adalah seperti berikut

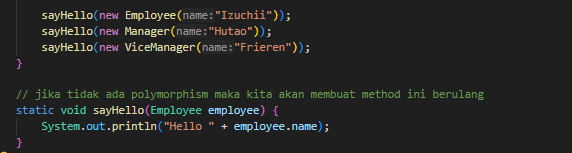


*Employee employee = new Employee("Izuchii");*

Perhatikan contoh code berikut Employee yang dipanggil adalah sebuah tipe data yang berupa class yang mendefinisikan class employee, kemudian untuk mendefinsikan objectnya kita bisa berikan Object class yang dituju.

*Tipe data Polymorphism pasti merujuk ke class utama sebagai template / blue print. Dapat dianalogikan seperti berikut saya memiliki kartu nama yang kosong, diberikan ‘keyword new sama seperti membuat untuk object baru’ misalkan kartu tersebut untuk mahasiswa, maka datanya akan berisi data mahasiswa sedangkan jika dosen maka datanya untuk dosen.*

*Inheritance saja cukup namun masih ada kendala dalam blueprint atau template.*

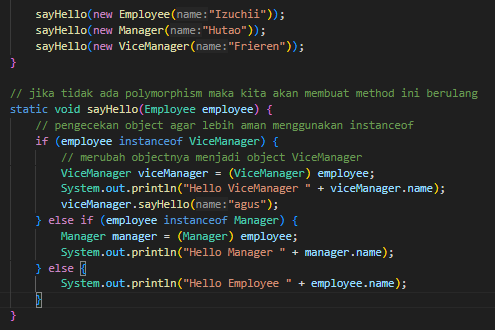
**

Berikut adalah sebuah method yang dijalankan padahal method tersebut sama saja, jika kita menggunakan tipe data biasa / tidak menerapkan polymorphism maka method tersebut akan diulang sebanyak 3x.

## Type Check & Casts

Sebelumnya kita sudah tahu melakukan konversi tipe data (casts) di tipe data primitive. *Casts* juga bisa digunakan untuk tipe data bukan primitive (Object). Namun agar aman, sebelum melakukan casts, pastika kita melakukan type check (pengecekan tipe data) dengan menggunakan kata kunci ***instanceof.***

Hasil operator ***instanceOf*** adalah Boolean, true jika tipe data sesuai dan false jika tipe data tidak sesuai.



*Instanceof = digunakan untuk pengecekan Object*

Perhatikan code berikut, Ketika kita mengirimkan Object ke dalam method sayHello, itu sama seperti disimpan kedalam sebuah tipe data Employee, kita bisa merubahnya ke dalam sebuah tipe data lain / ke object lain.

Misal dari tipe data Employee, menjadi ViceManager.

## Variabel Hiding

Variabel Hiding merupakan masalah yang terjadi Ketika kita membuat nama field sama di class child dengan nama di class parent.

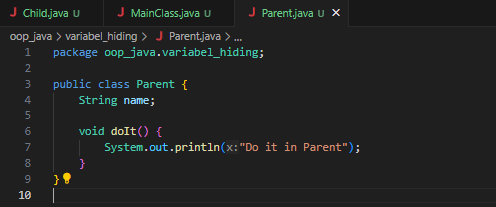
Tidak ada yang Namanya field / property overriding, Ketika kita buat ulang nama field di class, class itu berarti variable hiding, untuk mengatasinya kita bisa gunakan super keyword.

Yang membedakan variable hiding dengan method overriding adalah Ketika sebuah object di casts, saat di casts maka method akan tetap bisa mengakses method overriding, namun variable akan mengakses variable yang ada di dalam classnya.

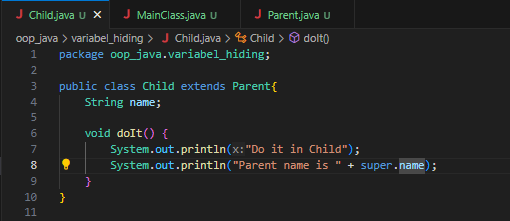
*Simpelnya variable hiding itu merupakan variable yang disembunyikan / berbeda, saat kita melakukan casting object pada polymorphism maka object yang di casting tidak akan lagi memiliki variable sebelumnya namun masih memiliki method sebelumnya.*

Perhatikan pada code berikut.

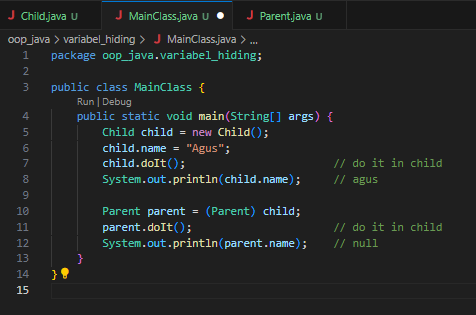
Parent Class



Child Class



Main Class



Lihat, *Ketika kita melakukan cast, maka method doIt akan terjadi override sedangkan property tidak ada istilah override hanya saja property seperti di set ulang Ketika dikakukannya casting. Oleh karena itu propertynya null.*

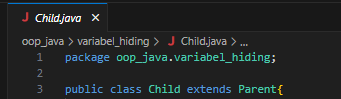
*Artinya jika override method parent jadi tidak bisa diakses.*

## Package

Saat kita membuat aplikasi, bisa dipastikan akan banyak sekali membuat class, Jika class terlalu banyak, kadang akan menyulitkan kita untuk mencari atau mengklasifikasikan jenis jenis class.

Java memiliki fitur package, yaitu fitur mirip folder / direktori, Dimana kita bisa menyimpan class class kita di dalam package. Sama seperti folder / direktori kita juga dapat melakukan nested dengan menggunakan tanda . (titik)

Berikut adalah contoh code nya



## Access Modifier (Public, Private, Protected & no modifier)

Adalah kemampuan membuat class, field, method dan constructor dapat diakases dari mana saja. Sebelumnya kit akita sudah membuat 2 access modifier yaitu *public* dan *default (no-modifier)*.

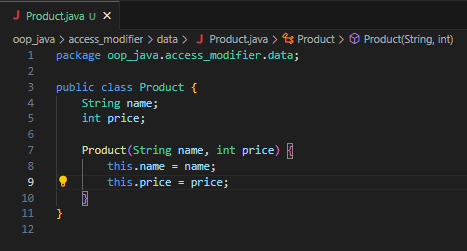
Access Level

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Modifier | Class | Package | Subclass | World |
| Public | Y | Y | Y | Y |
| Protected | Y | Y | Y | N |
| *No modifier* | Y | Y | N | N |
| Private | Y | N | N | N |

*Public,* dapat diakses di class yang sama, dia bisa diakses dari package yang sama, turunannya juga bisa diakses dan diluar package juga masih bisa diakses.

Saat membuat 1 public class hanya bisa 1 di 1 file java selain itu nama public class harus sama dengan nama file.

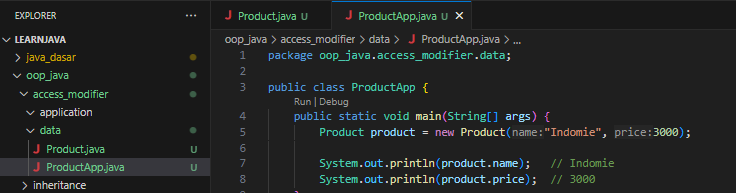
Contoh :



*Protected*, maka Ketika di luar package yang berbeda maka tidak akan bisa diakses.

Contoh :

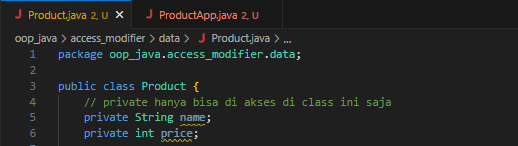




*No modifier*, subclass diluar package tidak akan bisa diakses atau diluar yang lain juga tidak bisa diakses.

*Private*, hanya bisa diakses di class yang sama tidak bisa diakses jika di package lain, subclass atau diluar dari itu.

Contoh:

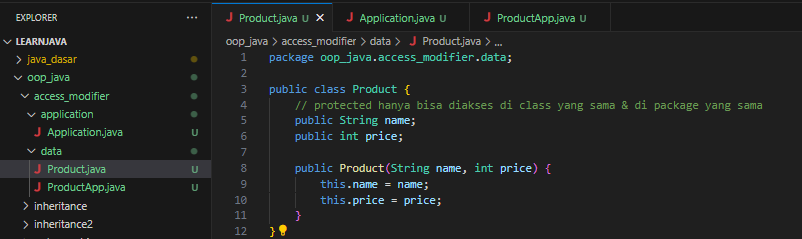


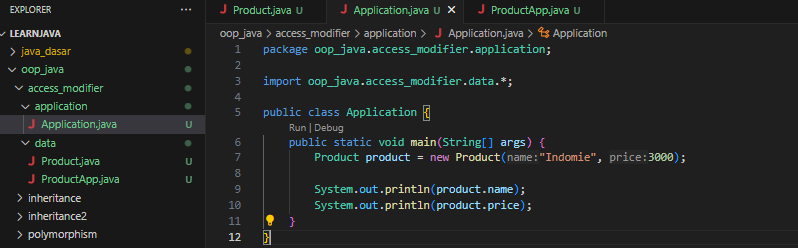
## Import

Import adalah kemampuan untuk menggunakan class yang berada di package yang berbeda, syarat class yang bisa digunakan jika packagenya berbeda adalah class yang harus digunakan adalah *public.*

Apabila tidak public pada class di package lain maka nanti akan error. Pastikan jika ingin bisa mengaksesnya harus public. Untuk import semuanya bisa tambahkan tanda \*.

Berikut adalah contohnya :





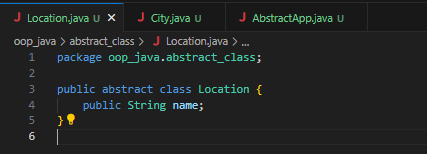
Sebenarnya ada import java.lang.String; namun karena sudah auto import maka kita tidak perlu lagi melakukan import seperti berikut.

## Abstract Class

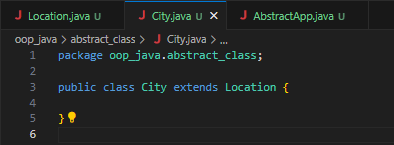
Saat kita membuat class, kita bisa menjadikan sebuah class sebagai abstract class. Abstract class artinya, *class tersebut tidak bisa dibuat sebagai object secara langsung, hanya bisa diturunkan*.

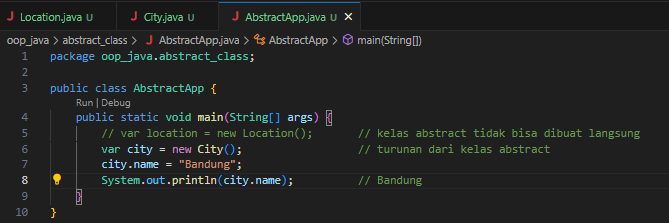
Untuk membuat sebuah class menjadi abstract, kitab isa menggunakan kata kunci abstract sebelum kata kunci class. Dengan demikian abstract class bisa kita gunakan sebagai kontrak untuk class child.

Parent class diberikan abstract sehingga tidak bisa langsung dibuat objectnya



Child class, memiliki property parent nya





## Abstract Method

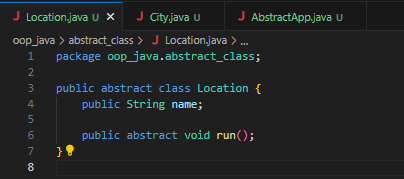
Saat kita membuat class yang abstract, kita bisa membuat abstract method juga di dalam class abstract tersebut. Saat kita membuat sebuah abstract method, kita tidak boleh membuat block method untuk method tersebut.

Artinya, abstract method **wajib di override di class child**. Abstract method juga tidak boleh memiliki access modifier private.

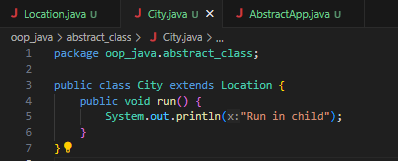
*Abstract itu tidak boleh dideklarasikan itu hanya mendefiniskan mehodnya saja.*

*Kesimpulannya : Abstract itu menuntut child class harus mempunyai method di dalamnya apabila didefinisikannya abstract, dan abstract class tidak bisa dideklarasikan dan hanya bisa di childnya saja.*

Abstract Class

**

Child Class





## Getter & Setter (Enkapsulasi)

Enkapsulasi artinya memastikan data sensitive sebuah object tersembunyi dari akses luar. Hal ini bertujuan agar kita bisa menjaga agar data sebuah object tetap baik dan valid.

Untuk mencapai ini, biasanya kita akan membuat semua property dengan private, sehingga tidak bisa diubah dari luar. Agar bisa diubah kita menyediakan method untuk mengubah dan mendapatkan field tersebut.

Di java proses enkapsulasi sudah dibuat standarisasinya, Dimana kita bisa menggunakan getter dan setter method.

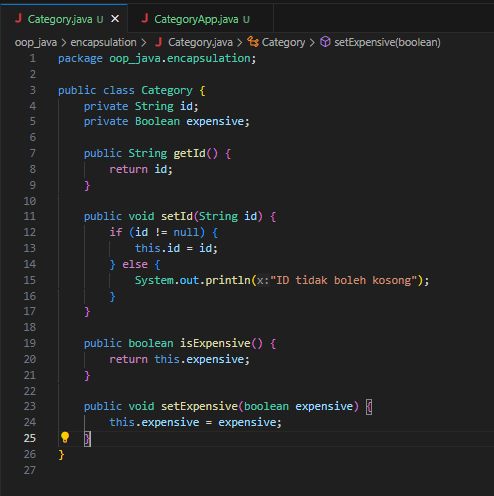
**Getter**, adalah fungsi yang dibuat untuk mengambil data field

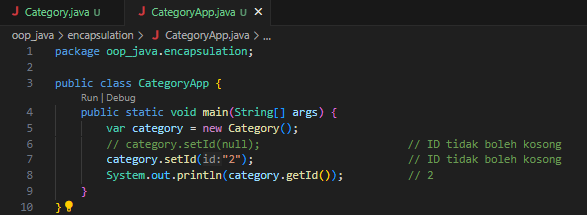
**Setter**, function untuk mengubah data field.

Pembuatan Getter dan Setter Method

|  |  |  |
| --- | --- | --- |
| Tipe data | Getter Method | Setter Method |
| Boolean | isXxx() | setXxx(Boolean value) |
| Primitive | getXxx() | setXxx(Primitive value) |
| Object | getXxx() | setXxx(Object value) |

Kegunaannya adalah agar tidak ada orang yang langsung merubah / mengakses property utamanya. Karena jika dibuat Setter kita bisa juga memberikan validasi di dalamnya. Misal validasi jika null / datanya sama dan sudah ada.





## Interface (implements)

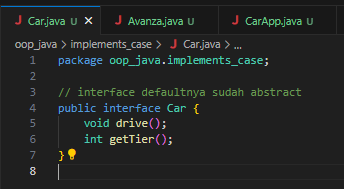
Sebelumnya kita sudah tahu bahwa abstract class bisa kita gunakan sebagai kontrak untuk class childnya. Namun sebenarnya yang lebih tepat untuk kontrak adalah interface.

Interface mirip seperti abstract class, yang membedakan adalah di interface semua method otomatis abstract, tidak memiliki block.

Di interface kita tidak boleh memiliki field, kita hanya boleh memiliki constant (field yang tidak bisa diubah) untuk mewariskan interface kita tidak menggunakan kata kunci extends, melainkan ***implements***.

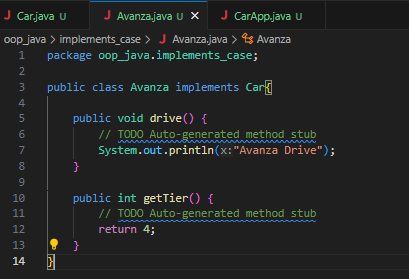
Sehingga sebenarnya kita seharusnya menggunakan Interface karena didalamnya sudah termasuk Abstract Class dan Enkapsulasi berikut adalah contohnya.

Car.java



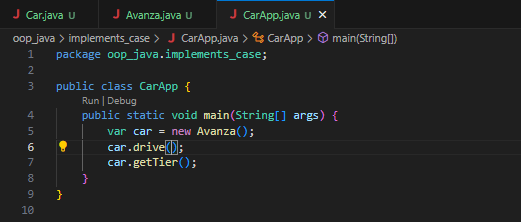
Avanza.java

Disini cukup gunakan kata kunci *implements*, yang membuktikan bahwa Avanza itu adalah child class yang harus mempunyai method milik class Car.



CarApp.java

Pada code ini kita bisa menjalankan seperti berikut.



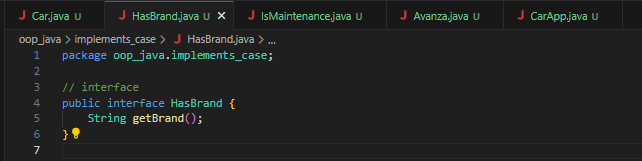
Apabila ada construct maka kita gunakan interface bukan abstract class.

## Interface Inheritance (extends interface)

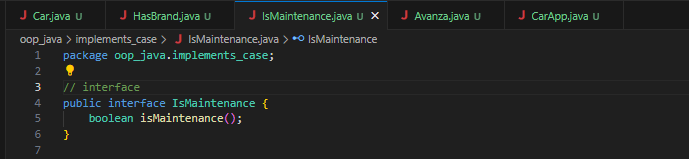
Sebelumnya kita sudah tahu bahwa child class itu memiliki 1 parent class, namun berbeda dengan interface, sebuah ***child bisa implement lebih dari 1 interface***. Bahkan interface pun bisa mengimplement interface lain, bisa lebih dari 1. Namun jika interface ingin mewarisi interface lain, kita gunakan kata kunci ***extends*** bukan ***implements***.

Pada contoh code seblumnya kita bisa tambahkan class baru berikut adalah contohnya

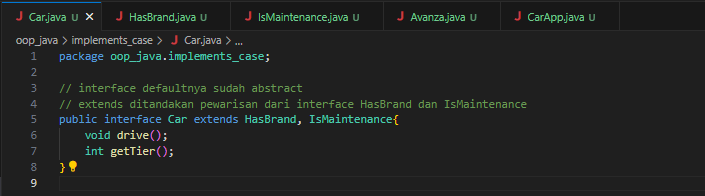
HasBrand.java



IsMaintenance.java

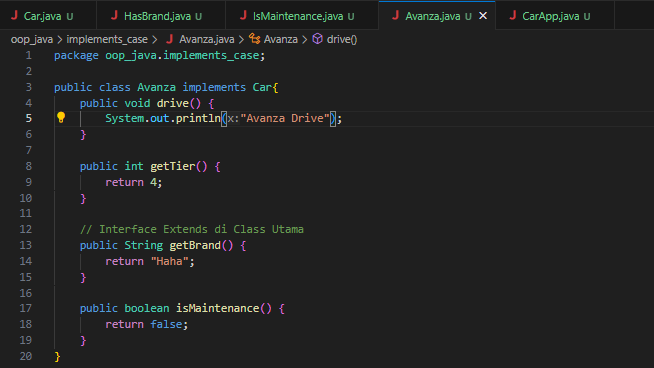


Car.java

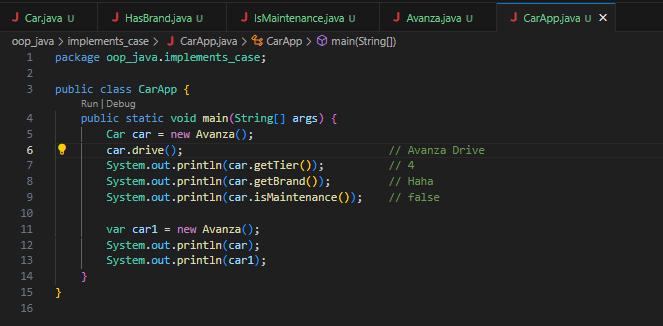


Sekarang Car melakukan inheritance dari interface lain, oleh karena itu Ketika jika class child melakukan implements pada car maka kita harus mengisikan nilai untuk method dari semua interface nya.

Avanza.java



CarApp.java



## Kesimpulan Abstract, Interface, dan Class Biasa

Kasus dapat kita anggap dalam melakukan CRUD sederhana.

1. Class Biasa

Ketika melakukan CRUD sederhana sebenarnya menggunakan class biasa sudah cukup tanpa perlu hirarki yang kompleks.

1. Abstract

Karena class Abstract merupakan sebuah blue print yang harus diterapkan oleh child class dan di override karena class abstract tidak dapat langsung dipanggil jadi Ketika dijalankan kita cukup panggil class childnya.

Abstract class cocok digunakan ketika Anda memiliki beberapa entitas yang membutuhkan operasi CRUD yang mirip atau bahkan identik dalam beberapa aspek.

*Terkadang dengan kedua desain arsitektur penulisan code tersebut tidak apa apa dan sudah terbilang cukup.*

1. Interface

Namun dalam penerapan desain arsitektur Interface mungkin cukup membuat aplikasi kompleks, namun karena interface merupakan sebuah kontrak dari class maka child class nya harus mengikuti class utama / mengikuti aturan dari class utama.

Interface memungkinkan Anda untuk mencapai polimorfisme yang lebih luas, karena kelas-kelas yang berbeda bisa mengimplementasikan satu interface tanpa harus berada dalam hirarki yang sama.

Abstract membuat blue print yang harus diimplementasikan untuk turunannya (terikat hirarki), sedangkan Interface dapat dianggap sebagai kontrak yang harus dipenuhi oleh sebuah class dan tidak terikat hirarki.

## Default Method

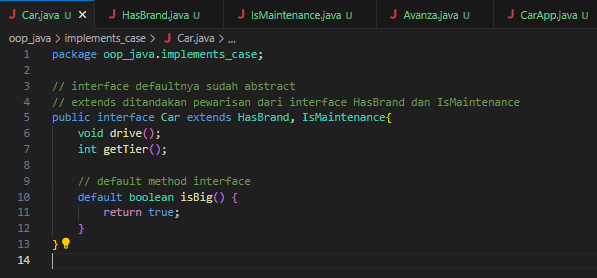
Sebelumnya kan kita tahu bahwa di interface kita tidak bisa membuat method konkrit yang memiliki blok method.

Namun di Java 8 ada fitur default method di interface, fitur ini terjadi karena sulit untuk maintance kontarak interface jika sudah terlalu banyak class yang implement interface tersebut.

*Karena jika kita menambah satu method di interface maka secara otomatis semua class yang implement akan rusak dan harus mengoverride method tersebut.*

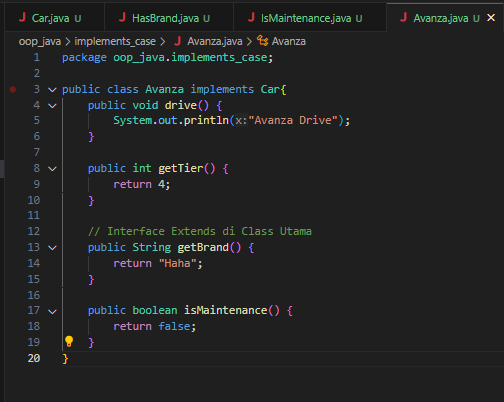
Dengan menggunakan default method, kita bisa menambahkan konkrit method di interface.

Car.java

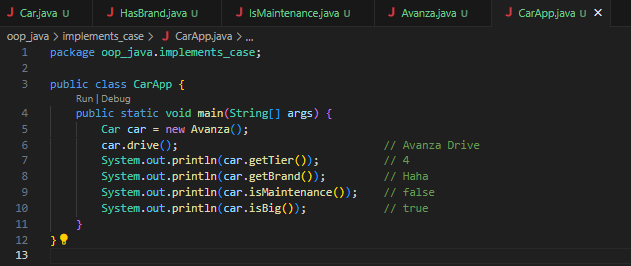


Dengan menerapkan ini seakan akan code ini jika tidak dioverride akan seperti di copas ke class yang implements interface, jika butuh method tersebut cukup lakukan override.

Avanza.java



CarApp.java



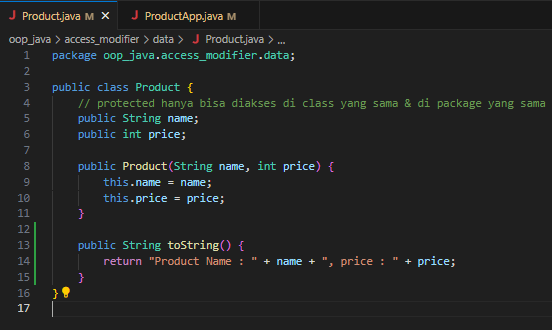
## toString() method

toString() adalah method yang terdapat di class Object.

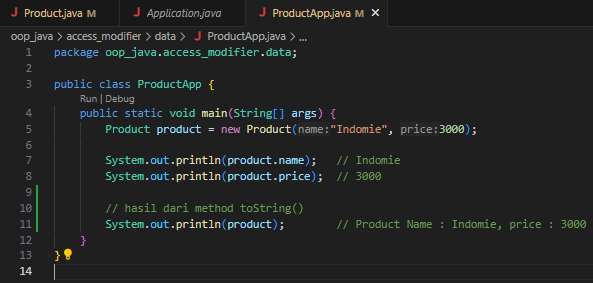
Method ini biasanya digunakan untuk merepresentasikan object ke dalam bentuk String. Secara default .toString() akan menghasilkan :

* namaClass + @ + hashCode

Namun kita tetap bisa merubahnya jika kita mau agar object yang dibuat lebih mudah dibaca.

Product.java 

Product App.java



Keuntungannya kita tidak perlu melakukan sout satu satu untuk mendapatkan propertynya cukup dengan mengubah / overridemethod toString().

## equals() method (Penting)

Hal yang agak membungungkan di Java adalah, cara membandingkan Object.

Di Bahasa pemrograman lain, untuk mengecek apakah sebuah method sama, biasanya menggunakan operator ==, di java operator == hanya untuk mengecek data primitive.

Untuk non Primitive pengecekannya menggunakan method equals().

Secara default equals itu akan membandingkan 2 buah object secara kesamaan diposisi object di memory, artinya jika kita membuat 2 object yang isi fieldnya sama, tetap dianggap beda oleh method equals.

Oleh karena itu ada baiknya kita mengoverride method equals milik class Object tersebut.

Sebelumnya perhatikan code berikut.

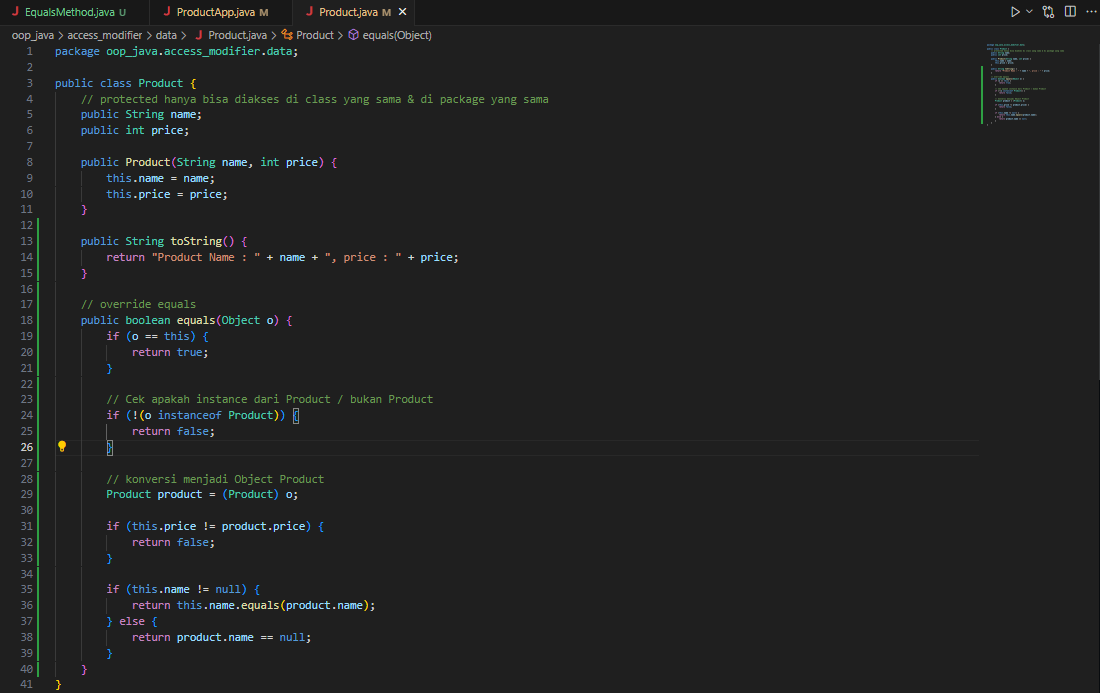


Ketika first dan second dibandingkan hasilnya false karena meskipun isinya sama namun objectnya yang berbeda, sedangkan itu bandingkan second dengan third meskipun sama tapi hasilnya true.

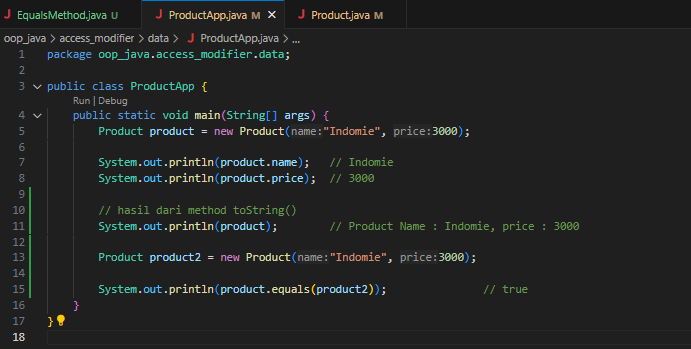
Sebenarnya karena data second sudah tersimpan di memory maka Ketika membandingkan dengan third hasilnya akan true.

Pentingnya melakukan Override pada Equals, dengan melakukan override pada equals kita mempunyai kendali penuh mengenai blok kode kita,

Perhatikan contoh pada Product.java



Pengecekan equals dilakukan secara manual, namun jika kita menggunakan IDE IntelliJ IDEA maka aka nada helper yang membantu melakukan override pada equals



## HashCode() method

Method hashCode() adalah method representasi integer object kita, mirip toString yang merupakan representasi String, ***hashCode adalah representasi integer***.

HashCode sangat bermanfaat ***untuk membuat struktur data unique*** seperti HashMap, Set, dan lain lain, karena cukup menggunakan hashCode method untuk mendapatkan identitas unique object kita.

Secara default hashCode akan mengembalikan nilai integer sesuai data di memory, namun kita bisa mengoverride nya jika kita ingin.

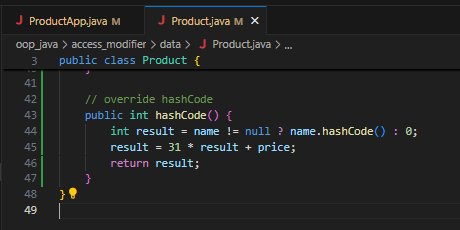
Kontrak HashCode Method

Tidak mudah meng-override method hashCode, karena ada kontraknya :

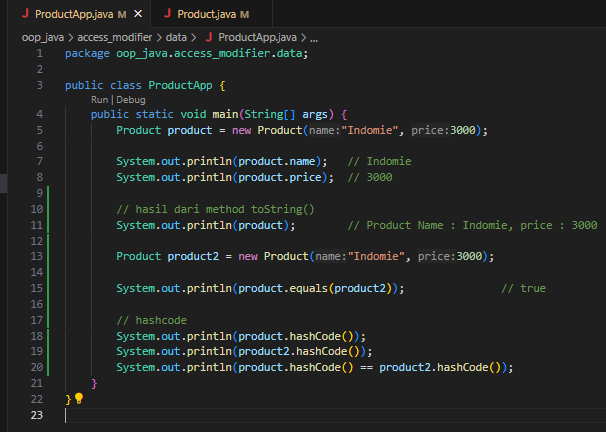
* Sebanyak apapun hashCode dipanggil, untuk object yang sama, harus menghasilkan data integer yang sama.
* Jika ada 2 Object yang sama jika dibandingkan menggunakan method equals, maka nilai hashCodenya juga harus sama.
* Tidak wajib hashCode berbeda jika method equals menghasilkan false, karena memang keterbatasan jumlah integer sekitar 2 miliyar.

Overide hashCode lebih baik menggunakan ide dibandingkan manual.

Product.java



ProductApp.java



Jadi jika dua object dibandingkan menggunakan method equals dianggap sama maka kedua object tersebut harus memiliki hashCode yang sama.

## Kesimpulan Equals() & HashCode()

Mungkin apabila masih bingung dengan equals dan hashCode akan dibahas pada materi kali ini.

**Equals method**

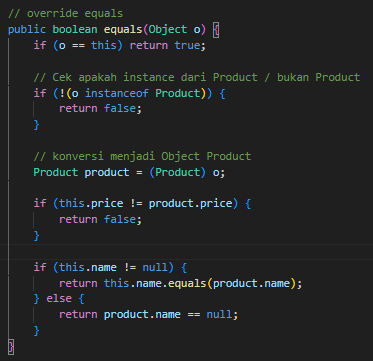
Terkadang didalam Bahasa lain jika kita membuat beberapa object dengan value / nilai yang sama kita dapat membandingkannya menggunakan tanda ==

Namun, di java apabila kita menggunakan == itu sama saja membandingkan Object bukan value di dalamnya. Oleh karena itu di java terdapat sebuah method equals() untuk dapat membandingkan nilai di dalamnya.

Misal ada object 1 berisikan sebuah name “Luthfi Nur Ramadhan”, dan object ke 2 juga sama name berisikan “Luthfi Nur Ramadhan”.

Secara default jika kita membandingkan keduanya menggunakan equals itu akan false, kita harus melakukan override terlebih dahulu pada equals method agar Ketika membandingkan kedua Object tersebut harusnya bernilai true.

*Kenapa bisa false, karena keduanya adalah Object setiap object memiliki method equalsnya masing masing oleh karena itu kita harus melakukan override.*

**

**HashCode Method**

Terkadang kita membandingkan kedua object dengan nilai yang sama seperti pada kasus equals diatas, Karena nilainya / value di dalam kedua Objectnya sama sebaiknya kita override juga HashCode yang dimiliki pada object tersebut.

Keuntungannya, dapat mengurangi duplikasi memory dan kita bisa membandingkan kedua Objectnya. Sehingga nanti Object tersebut memiliki HashCode yang sama dan dapat dibandingkan menggunakan ==

*Namun sama hashCode juga harus di override pada Class utamanya, agar kita memiliki kendali penuh di dalam aplikasi dan dapat memberikan data integer unik.*

Kesimpulan :

* Method equals(): Digunakan untuk membandingkan nilai di dalam objek. Harus dioverride untuk memastikan objek dengan nilai yang sama dianggap equal.
* Method hashCode(): Digunakan untuk menghasilkan nilai hash dari objek. Harus dioverride bersama dengan equals() untuk memastikan konsistensi antara keduanya dan mendukung struktur data berbasis hash.

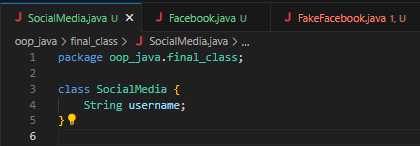
## Final Class

Sebelumnya kita pernah menggunakan kata kunci final di Java. Jika digunakan di variable, maka variable tersebut tidak bisa berubah lagi datanya.

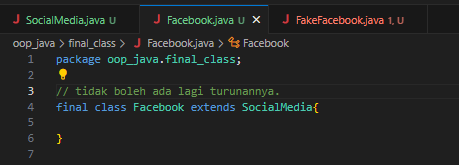
Final juga dapat digunakan di class, Dimana jika kita menggunakan kata kunci final sebelum class, maka kita menandakan bahwa class tersebut **tidak bisa diwariskan lagi**.

Secara otomatis semua class childnya akan error.

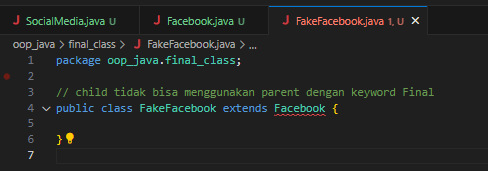
SocialMedia.java



Facebook.java - class final tidak akan bisa diturunkan.



FakeFacebook.java

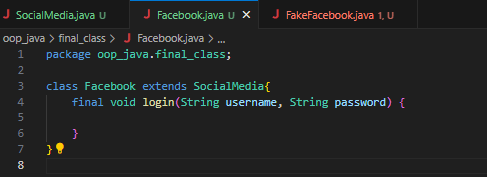


## Final Method

Kata kunci final juga bisa digunakan di Method. Jika sebuah method kita tambahkan kata kunci final, maka artinya method tersebut **tidak bisa di override lagi di class child nya**.

Ini sangat cocok jika kita ingin mengunci implementasi dari sebuah method agar tidak bisa diubah lagi oleh childnya.

Facebook.java - buat sebuah final method



FakeFacebook.java – tidak bisa override method

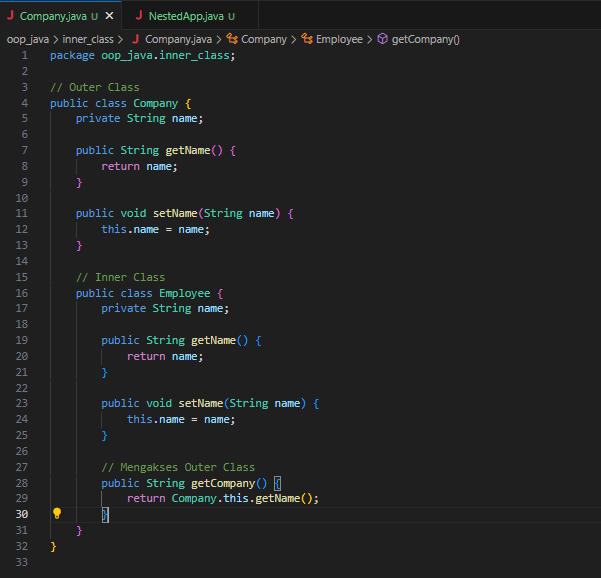


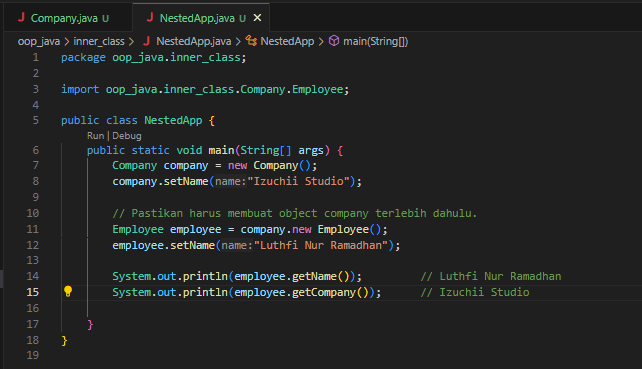
## Inner Class

Di Java, kitab isa membuat Class di dalam Class, atau disebut dengan *inner class*. Salah satu kasus kita membuat inner class biasanya Ketika kita butuh membuat beberapa class yang saling berhubungan, Dimana sebuah class tidak bisa dibuat tanpa class lain.

Misal kita perlu membuat class Employee, Dimana membutuhkan class Company, maka kitab isa membuat class Employee sebagai inner class dari class Company.

Cara membuat inner class, cukup membuatnya di dalam blok class outer class-nya.





Keuntungannya saat kita membuat inner class adalah, kemampuan untuk mengakses outer class nya. Inner class bisa membaca semua private member yang ada di outer class nya.

Untuk mengakses object outer class nya, kitab isa menggunakan nama class outer nya diikuti dengankaya kunci *this*. Misal: *Company.this*

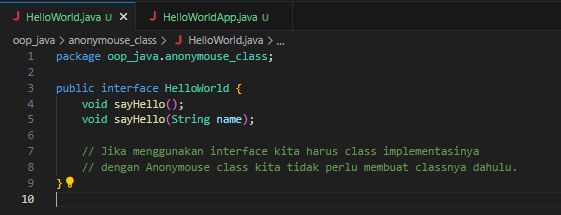
Dan untuk mengaskes super class Outer classnya, kita bisa menggunakan nama class outernya diikuti dengan kata kunci *super*. Misal: *Company.super*.

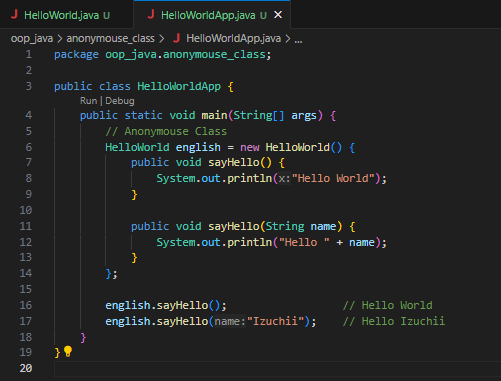
## Anonymouse Class

Anonymouse class atau class tanpa nama, adalah kamampuan mendeklarasikan class, sekaligus meng-inisialisasi object-nya secara langsung.

Anonymouse class sebenarnya termasuk inner class, Dimana **outernya adalah tempat Dimana kita membuat anonymouse class tersebut.**

Anonymouse class sangat cocok Ketika kita berhadapan dengan kasus membuat implementasi interface atau abstract class sederhana, tanpa harus membuat implementasi class nya.





Kekurangan dari anonymouse class tidak bisa di reuse dan harus bikin ulang. Biasanya kasusnya diterapkan pada kasus yang sederhana.

## Static Keyword

Sebelumnya kita sudah sering melihat kata kunci static, kata kunci ini bisa **membuat field, method atau class bisa diakses langsung tanpa melalui objectnya.**

Perlu diingat, static hanya bisa mengakses static lainnya.

**STATIC DAPAT DIGUNAKAN DI …**

Field, atau disebut class variable, artinya field tersebut bisa diakses langsung tanpa membuat object terlebih dahulu.

Method, atau disebut class method, artinya method tersebut bisa diakses langsung tanpa membuat object terlebih dahulu.

Block, static block akan otomatis dieksekusi Ketika sebuah class di load.

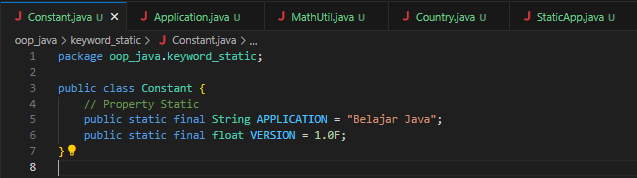
Inner Class, artinya inner class tersebut bisa diakses secara langsung tanpa harus membuat object outer classnya terlebih dahulu. Static pada inner class menyebabkan kita tidak bisa mengakses lagi object outer classnya.

*Biasanya static property itu ditandai dengan Huruf besar jika berada di dalam field dan final namun ini opsional.*

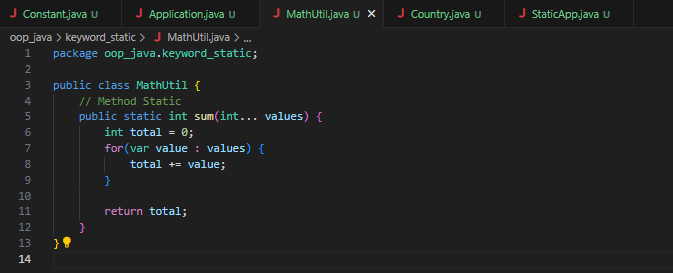
*Static Block, ini akan dieksekusi sekali sebelum classnya di load & hanya bisa mengakses yang static juga.*

Berikut adalah contoh Code

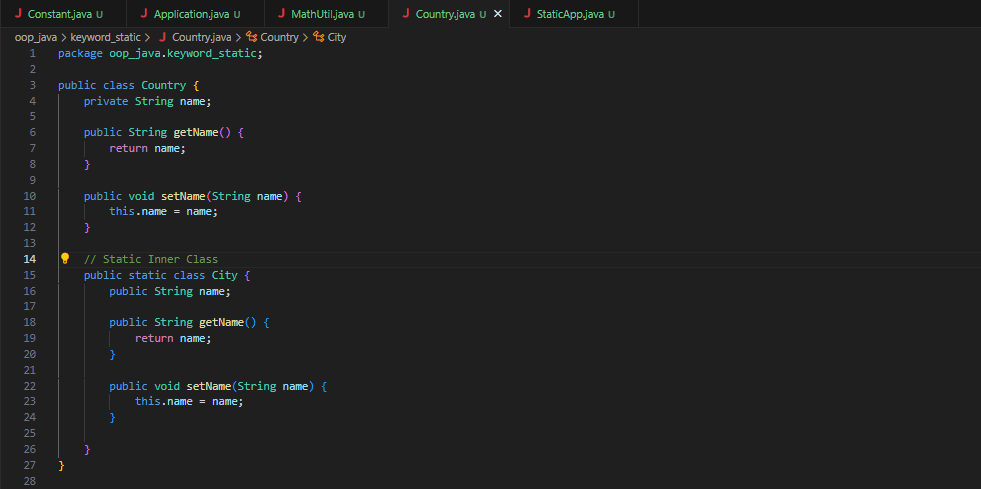
Constant.java - Property Static



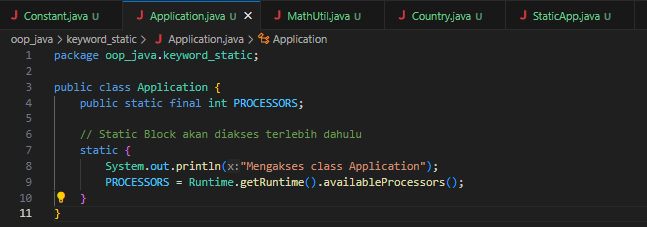
MathUtil.java – Method Static



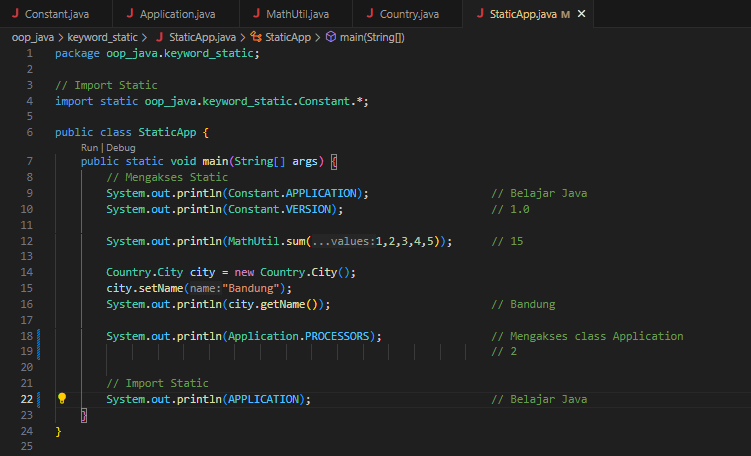
Country.java – Inner Class Static



Application.java – Block Static (Otomatis dijalankan)



StaticApp.Java – Menjalankan Static Keyword



Kesimpulan :

Penggunaan static sangat berguna dalam konteks di mana Anda ingin memiliki anggota kelas yang bersifat global untuk semua instance atau tidak bergantung pada instance. Ini membantu dalam mengurangi penggunaan memori yang tidak perlu dan memungkinkan akses mudah ke variabel atau metode yang sering digunakan tanpa perlu membuat objek.

## Record Class

Fitur ini masih versi preview dan belum stabil di versi Java 14, namun kita sudah bisa mencobanya. Perlu diingat, bahwa karena fitur ini masih experimental, artinya tidak ada jaminan di java versi mendatang, fitur ini bisa aja tetap ada / dihilangkan.

RECORD CLASS

Kadang kita sering membuat class, hanya untuk class yang berisikan data. Hanya berisi getter, equals, hashCode dan toString method.

Record class digunakan untuk mempermudah pembuatan jenis class tersebut. Saat kita membuat record class, secara otomatis Java akan membuatkan getter, equals, hashCode dan toString method secara otomatis, dan juga constructor secara otomatis.

Saat membuat record class, secara otomatis kita akan meng-extends class ***java.lang.Record*** yang artinya **kita tidak bisa extends class lain**. Namun kita tetap bisa mengimplement interface.

*Tidak membuat setter di record class, karena record class itu bersifat immutable (tidak bisa dirubah)*

Berikut adalah Contoh penulisan Record Class

LoginRequest.java



RecordApp.java



Java record itu akan secara otomatis membuat constructornya sebagai parameter, jika tidak menambahkan data di dalam tempat parameter maka tidak akan berguna java Recordnya.

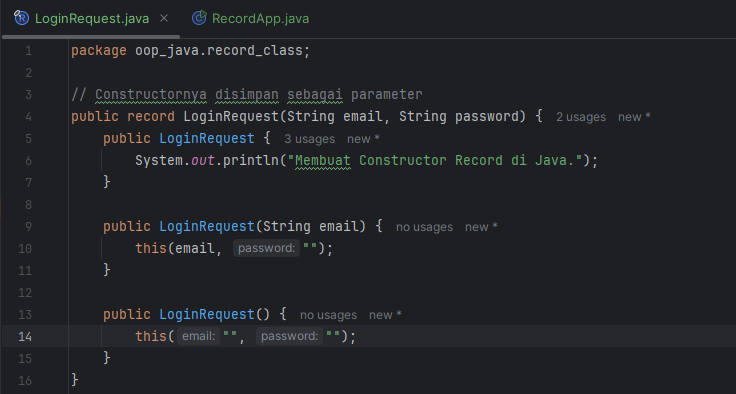
Secara otomatis juga record java method toString secara otomatis degenerate oleh Java.

Untuk getternya kita bisa akses nilainya seperti contoh diatas.

### Record Class Constructor

Secara default, constructor di record class akan dibuat secara otomatis, sesuai dengan definisi record class parameter. Namun jika kita ingin melakukan sesuatu di constructor tersebut, kitab isa membuat *compact constructor*, yaitu constructor tanpa tanda ().

Selain itu, kita juga bisa melakukan constructor overloading, namun ada syaratnya, yaitu constructor overloadingnya harus tetap memanggil constructor utama yang secara otomatis dibuatkan di record class.

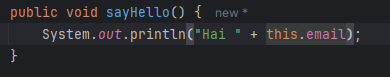


RecordApp.java

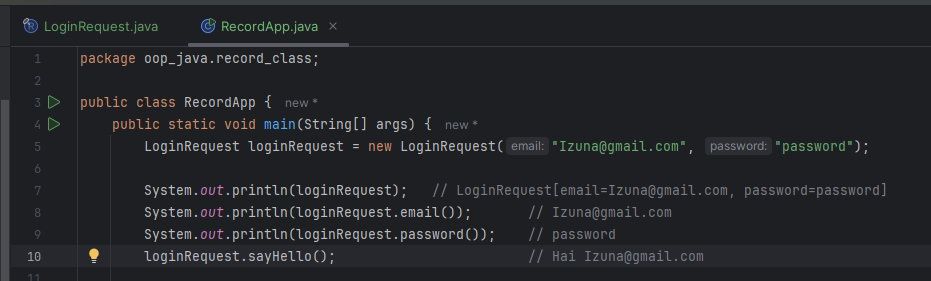


### Record Class Method

Tambahkan method berikut di LoginRequest.java



RecordApp.java



*Tambahan kita tidak bisa menimpa ulang nilai propertynya karena secara default property ini dibuatkan (final) sehingga kita tidak bisa menimpa / menggantinya.*

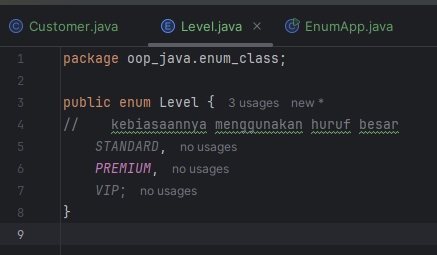
## Enum Class

Saat kita membuat aplikasi, kadang kita akan bertemu dengan jenis jenis data yang nilainya terbatas. Misal, gender ada male dan female

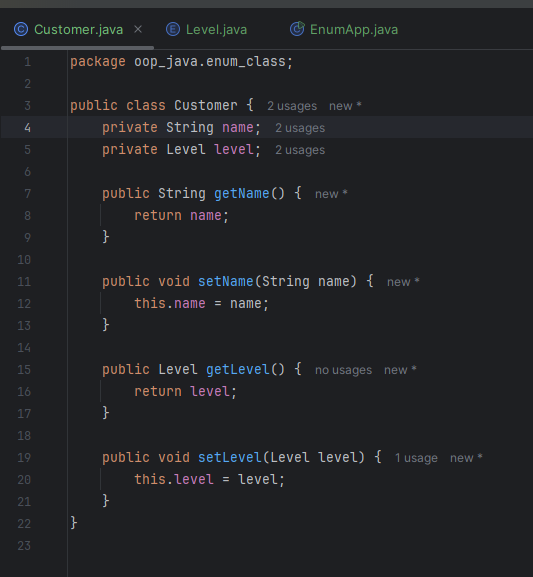
Dalam kasus ini kita bisa menggunakan enum Class, yaitu class yang berisikan nilai terbatas yang sudah ditentukan.

Saat membuat enum class, secara otomatis dia akan mengextends class **java.lang.Enum** oleh karena itu class enum tidak bisa extends class lain, namun masih bisa implements interface.

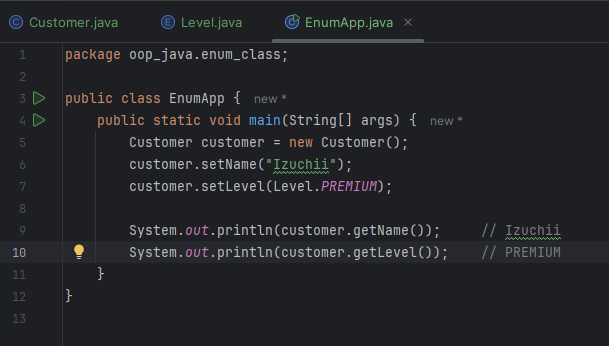
Class Enum Level.java



Customer.java



EnumApp.Java

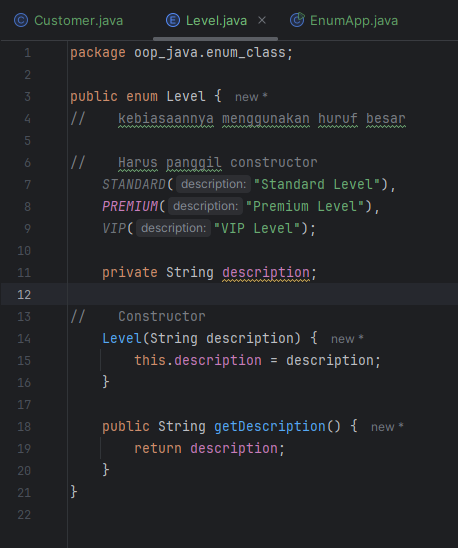


### Enum Members

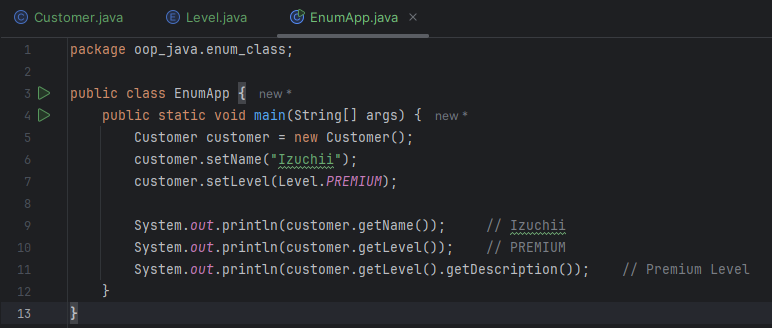
Sama seperti class biasanya, di class enum pun kitab isa menambahkan members (field, method dan constructor)

Khusu Constructor, kita tidak bisa membuat public constructor karena memang tujuan enum bukan untuk diinstansiasi secara bebas.

Level.java

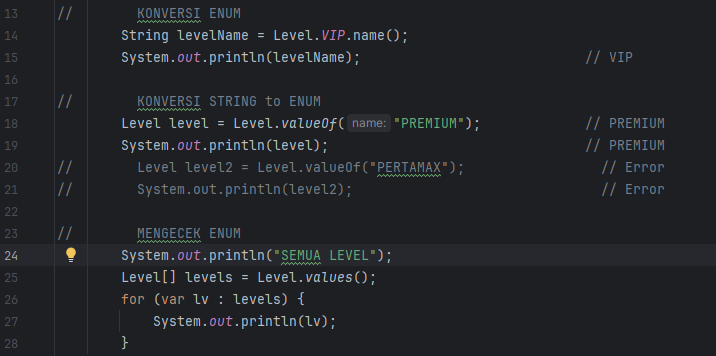


EnumApp.java



### Konversi Enum ke String

Kadang kadang kita butuh konversi dari String menjadi Enum / dari Enum menjadi String.



valueOf = diunakan untuk konversi tipe data primitive ke non primitive

valus untuk mengambil semua data di dalam class Enum.

## Exception (Error)

Saat kita membuat aplikasi, kita tidak akan terhindar dengan yang Namanya error.

Di Java, error direpresentasikan dengan istilah exception dan semua direpresentasikan dalam bentuk class Exception. Kita bisa menggunakan class Exception sendiri, atau menggunakan yang sudah disediakan oleh Java.

Jika kita ingin membuat exception, maka kita harus membuat class yang extends class Throwable atau turunan-turunannya.

**MEMBUAT EXCEPTION**

Exception biasanya terjadi di method, Ketika kita membuat exception di sebuah method, maka method tersebut harus ditandai dengan kata kunci **throws** diikuti denan class exceptionnya.

Jika method tersebut bisa menimbukan lebih dari satu jenis exception, kitab isa menambah lebih dari satu class Exception.

Di dalam kode program kita, untuk membuat exception kita cukup gunakan kata kunci ***throw*** diikuti dengan object exceptionnya.

### Try Catch

Saat kita memanggil sebuah function yang bisa menyebabkan exception, maka kita wajib menggunakan try-catch expression di Java.

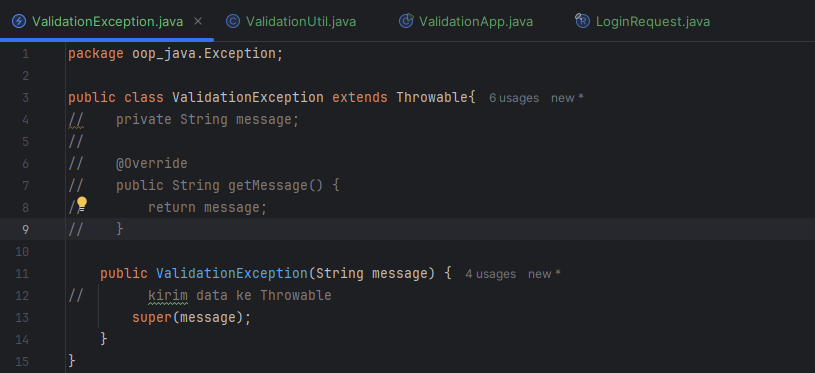
*Ini gunanya agar kita bisa menangkap exception yang terjadi, karena jika tidak ditangkap, lalu terjadi exception maka secara otomatis program kita berhenti.*

Cara menggunakan try-catch expression di Java sangat mudah di block Try kita tinggal panggil method yang bisa menyebabkan exception, dan di block catch kita bisa melakukan sesuatu jika terjadi exception.

Contoh Penerapan :

ValidationException.java

Kita buat Validation Exception yang extends ke Throwbale, karena ValidationException di Throwable sudah ada maka kita cukup kirimkan data ke parent nya.



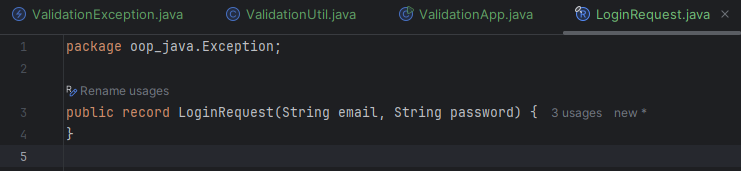
ValidationUtil.java

Logika untuk melakuan pengecekan di dalamnya.

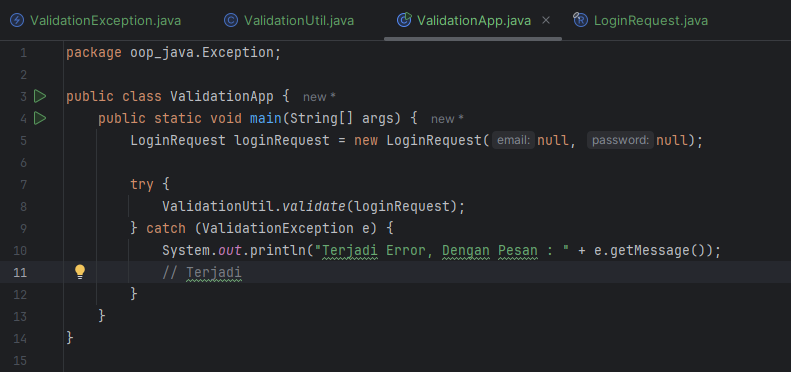


LoginRequest.java

Contoh Record Class yang sudah kita terapkan sebelumnya



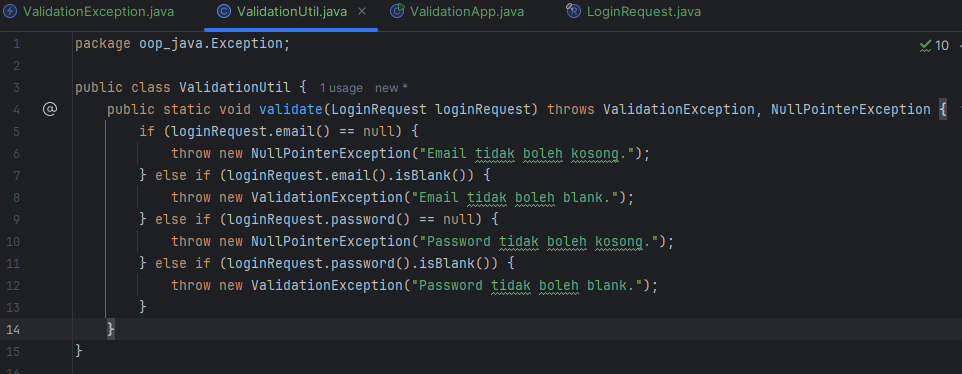
ValidationApp.java



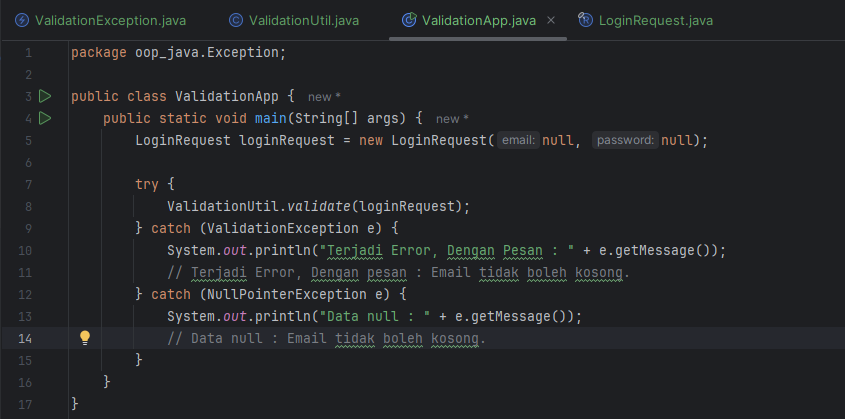
### Multiple Try Catch 1

Misal kita ingin menggunakan NullPointerException yang sudah menjadi bawaan dari Throwable.

Jadi di dalam ValidationApp.java block try menjalankan code di dalam class ValidationUtil.validate, jika ada error maka throw akan menjalankan class ValidationException di dalamnya kan ada secara default method ValidatioException dan NullPointerException, kemudian kita tangkap datanya yang akan masuk ke dalam catch.

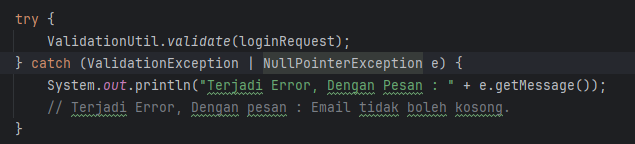


ValidationApp.java



### Multiple Try Catch 2

Kita dapat menangkap exception yang sama secara langsung dengan code seperti berikut

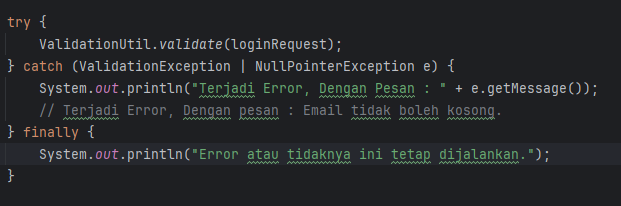


### Finally Keyword

Dalam try-catch, kita bisa menambahkan block finally, Block ini adalah block Dimana akan selalu dieksekusi baik terjadi exception ataupun tidak.

Contoh kasus

Ini sangat cocok Ketika kita ingin melakukan sesuatu, tidak peduli sukses / gagal, missal di block try kita ingin membaca file di block catch kita akan tangkap jika terjadi error, dan di block finally error ataupun sukses membaca file kita wajib menutup koneksi ke file tersebut, biar tidak menggantung di memory.



## Runtime Exception

**JENIS EXCEPTION**

Secara garis besar, di Java exception dibagi menjadi 3 jenis :

* Checked Exception, yaitu exception yang wajib di try catch, seperti yang sudah kita bahas.
* Runtime Exception
* Error

Runtime Exception adalah jenis exception yang tidak wajib ditangkap menggunakan try catch.

Compiler java tidak akan protes walaupun kita tidak menggunakan try catch Ketika kita memanggil method yang bisa menyebabkan runtime exception.

Untuk membuat class runtime exxception, kita wajib mengextends class RuntimeException. Ada banyak di Java yang merupakan runtime Exception, seperti NullPointerException, IllegalArgumentException, dll.

Tetep saja RuntimeException itu extends ke Throwable.

Jika checkedException nullPointerException itu harus mengarah ke exception yang melakukan extends ke throwable karena jika tidak akan error.

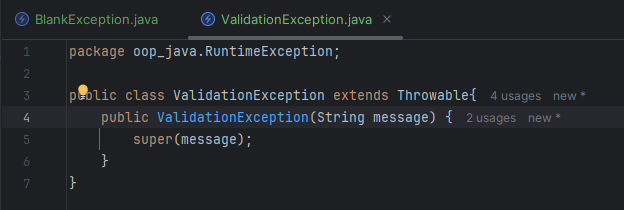
Sedangkan runtime Exception adalah seperti exception yang tidak wajib dan tetap mengarah ke Throwable.

Contoh Code :

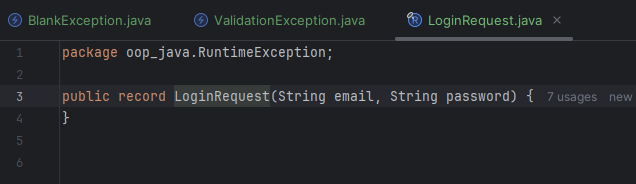
BlankException.java



ValidationException.java

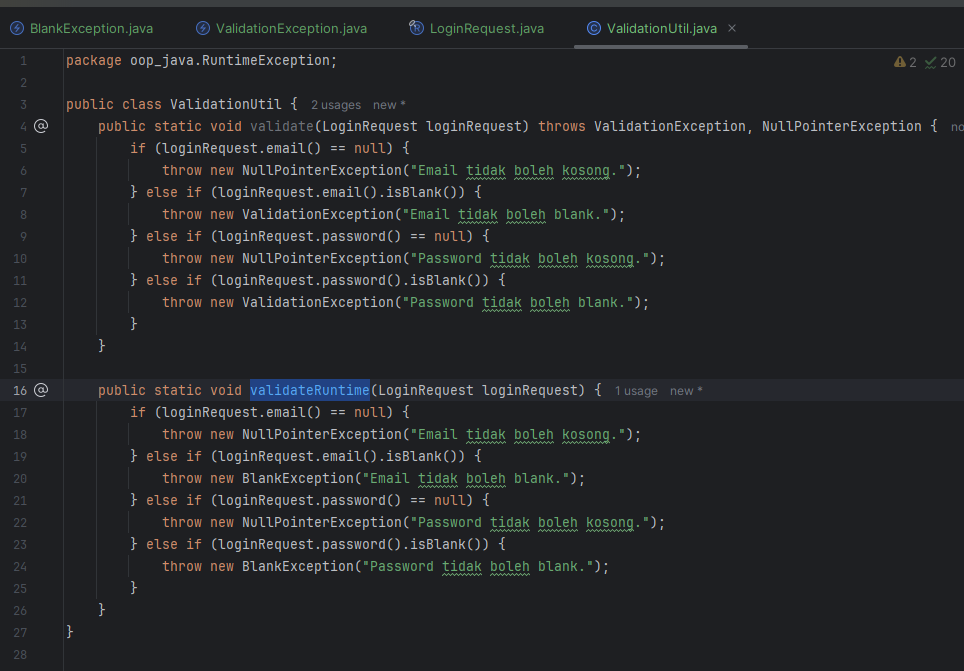


LoginRequest.java

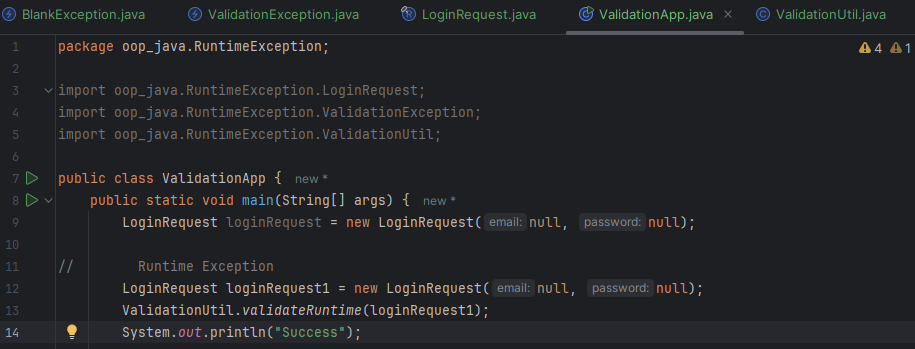


ValidationUtil.java

Berikut adalah contoh penerapan pada RuntimeException



ValidationApp.java



Ketika di jalankan akan error dan tidak muncul pesan success, dengan menggunakan runtime exception dapat digunakan untuk menyimpan error handler di suatu tempat dan Ketika terjadi error cukup error handlernya di satu tempat agar tidak terlalu banyak try catch, Biasanya digunakan di aplikasi yang besar.

## Error

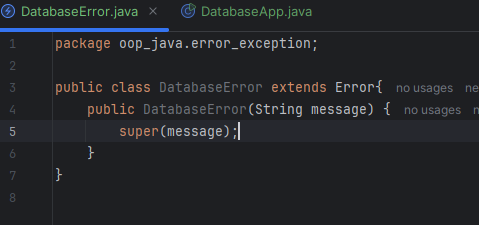
Error adalah jenis exception yang terakhir, error adalah sebuah class di Java, yang tidak direkomendasikan untuk di try catch.

Biasanya error terjadi Ketika ada masalah serius, dan sangat tidak direkomendasikan untuk di try catch.

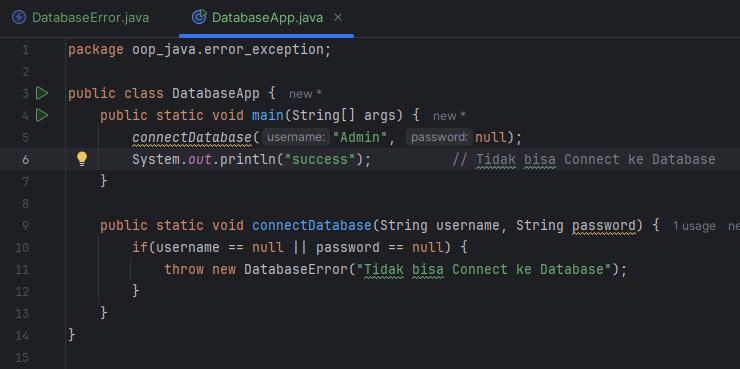
Artinya direkomendasikan **untuk mematikan aplikasi**.

Contoh, missal jika di awal aplikasi kita tidak bisa terkoneksi ke database, direkomendasikan untuk membuat exception jenis Error, dan menghentikan aplikasi.

DatabaseError.java



DatabaseApp.java



Sebenarnya ini lebih cocok untuk jika kendala yang fatal, seperti salah konfigurasi, tidak bisa konek ke database & yang menyebabkan aplikasi tdk berjalan dengan baik.