

Java 8 (Optional Class, Lambda) Part 1

Silver - Chapter 3 - Topic 4

Selamat datang di **Chapter 3 Topic 4**
online course **Back End Java** dari
Binar Academy!



Java 8 for infinity!

Siapa yang udah nggak sabar buat ngobrolin Java 8? Denger denger sih Java 8 adalah versi upgrade dengan fitur yang makin ajib.

Nah, berhubungan fiturnya bakalan bervariasi, kita bakal memecah pembagian Java 8 menjadi dua part. Pertama dulu nih, kita bakal cari tahu tentang **Java 8 yang isinya adalah optional class dan lambda**.

Yuk, langsung aja kita kepoin part 1 ini~



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

- Fitur di Java 8 - Optional Class
- Fitur di Java 8 - Lambda
- Cara membuat Lambda
- Fitur di Java 8 - Method Reference
- Lambda in Collection



Mari kita awali pembahasan kali ini dengan materi Java 8 tentang **Optional Class**.

Eits.. fitur ini bisa mengatasi satu masalah tanpa membuat kode jadi jelek, lho~

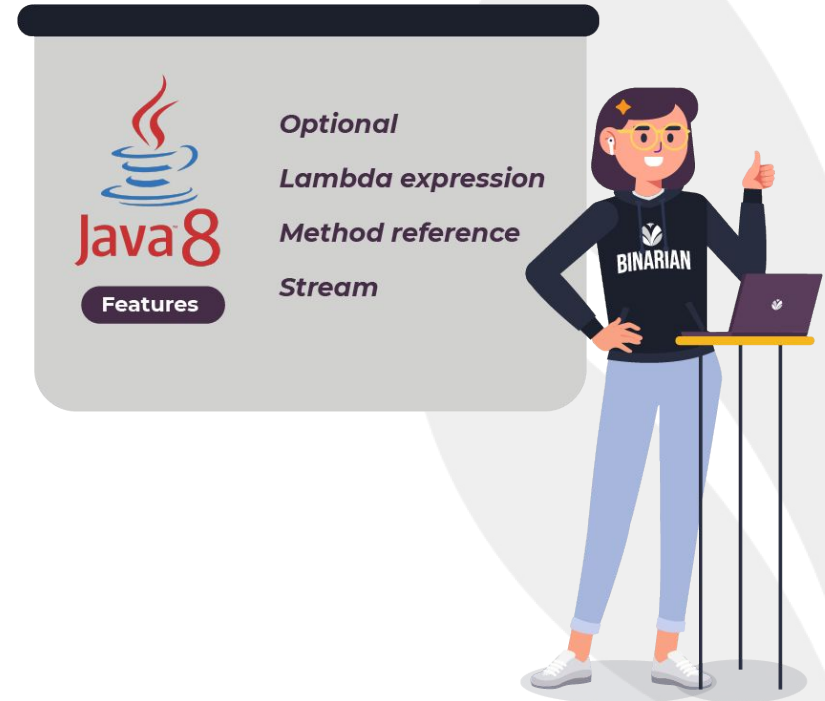


Java 8 yang canggih banget fiturnya~

Java 8 memiliki upgrade yang signifikan dibandingkan dengan versi sebelumnya, lho.

Di antara fitur-fitur yang di upgrade ini, ada empat fitur yang cukup penting nih, yaitu:

- **Optional**
- **Lambda expression**
- **Method reference**
- **Stream**



Emang fiturnya buat apa sih sampai sebegitu pentingnya?

Penting banget dong!

Karena fitur tersebut udah dipakai di teknologi Java yang terkini, artinya dengan menguasai fitur tersebut, kita bisa gampang banget buat menguasai teknologi baru.



Kita bahas fitur yang pertama dulu ya! Yang pertama ada fitur **Optional Class**~

Optional merupakan fitur baru dari java untuk **mencegah terjadinya NullPointerException**.

NullPointerException bakal terjadi kalau suatu variable object bernilai null, sedangkan pada saat yang sama si object tersebut harus mengeksekusi sebuah method.




“Kalau udah terjadi `NullPointerException`, gimana dong?”

Tenang~ sebenarnya bisa aja kita melakukan validasi dulu. Tapi bisa aja si object tersebut adalah object yang nested dan bakal terjadi validasi yang berulang-ulang.

Sayangnya, hal kayak gitu yang bakal bikin code jadi jelek. Nah, **buat mengatasi `NullPointerException` tapi nggak bikin code jadi jelek, ada si optional class** ini.



**Biar kebayang sama case yang bisa diselesaikan sama Optional Class,
kita pake contoh ya~**



```
String version =  
computer.getSoundcard().getUSB().getVersion();
```

Computer punya variable object Soundcard, terus Soundcard punya variable object USB dan USB punya variable Version. Disini, menurut kamu, apa yang bisa bikin NullPointerException?

Terus gimana caranya supaya nggak terjadi `NullPointerException`?

Di samping adalah cara buat yang nggak pakai Java Optional.

Meskipun bisa, tapi cara kayak gitu bakal bikin code jadi jelek karena ada **nested validation** atau **validasi di dalam validasi**. Coba deh kamu lihat, if-nya jadi banyak kan?

“Kalau mau yang bagus, gimana?”

```
String version = "UNKNOWN";
if(computer != null){
    Soundcard soundcard = computer.getSoundcard();
    if(soundcard != null){
        USB usb = soundcard.getUSB();
        if(usb != null){
            version = usb.getVersion();
        }
    }
}
```

Di samping adalah cara buat bikin code yang bagus, bestie~

Simak langkah-langkahnya dengan baik, ya!

Dalam mendeklarasikan variable-variable-nya, kita harus menggunakan container optional terlebih dahulu.

Dan jangan lupa kalau seluruh getter juga punya return berupa Optional, ya.

```
public class Computer {  
    private Optional<Soundcard> soundcard;  
    public Optional<Soundcard> getSoundcard() { ...  
    }  
    ...  
}  
  
public class Soundcard {  
    private Optional<USB> usb;  
    public Optional<USB> getUSB() { ... }  
}  
  
public class USB{  
    public String getVersion(){ ... }  
}
```

Dengan penggunaan Optional Class, hasil akhirnya bakal jadi kayak gini!

Bimsalabim~

Kamu bisa tahu sendiri, kalau code yang dibuat jadi lebih singkat dibandingkan contoh sebelumnya?

Pembahasan secara lengkap tentang penggunaan optional bisa kamu cek [di sini](#). Tapi buat penggunaan dari flatmap dan methodnya bakal dibahas di topic selanjutnya, yaaa~

```
String version =  
computer.flatMap(Computer::getSoundcard)  
  
    .flatMap(Soundcard::getUSB)  
        .map(USB::getVersion)  
        .orElse("UNKNOWN");
```

Biar makin paham, berikut adalah beberapa method yang ada di Optional Class~

- **ifPresent()**, method yang mengembalikan nilai true kalau object nggak null.
- **get()**, method yang mengembalikan value dari object yang ada di dalam container.



- **orElse(T other)**, method yang bakal mengembalikan default value kalau object bernilai null dan bakal mengembalikan value dari object yang ada di dalam container kalau nggak null.

Method yang mirip sama orElse() ini ada banyak, contohnya kayak orElseGet dan orElseThrow.



Oke, sekarang kamu udah ada gambaran tentang Optional Class?

Selanjutnya, kita bakal cari tahu lebih dalam lagi tentang fitur Java 8 yang kedua, yaitu **Lambda**.



Fitur yang kedua namanya Lambda~

Lambda merupakan fitur dari Java 8 yang bisa **menyederhanakan implementasi dari suatu method di interface.**

Kalau pakai lambda, kita nggak perlu mendefinisikan ulang implementasi suatu method di dalam suatu class.

Buat pembahasan lengkap tentang lambda, bisa kamu cek [di sini](#) dan [di sini juga](#) ya, bestie~



Sekarang, coba cek contoh berikut, ya!

Dari contoh di samping, runnable merupakan sebuah interface yang punya method run.

Tapi, method run() ini belum punya implementasi. Nah, penulisan implementasi bisa kita lakukan dengan memanfaatkan lambda.

```
Runnable r = new Runnable() {  
    public void run() {  
        System.out.println("Howdy, world!");  
    }  
};  
r.run();  
}
```

Berikut adalah contoh penggunaan lambda pada case Runnable~

```
Runnable r2 = () -> System.out.println("Howdy, world!");  
r2.run();
```

Penulisan implementasinya bisa secara langsung pakai lambda. Di mana hasilnya juga jadi jauh lebih singkat dibandingkan sama yang sebelumnya.

Buat penggunaan lambda sendiri cuma bisa dipakai di interface yang punya satu method aja, ya. Walaupun begitu, si lambda tetep bisa dipakai buat method yang bersifat void atau punya return, kok.

“Terus bentuk syntax dari lambda itu gimana?”

Bentuknya kayak gini, nih:

(parameter) -> expression

Kalau contoh buat cara pakainya, begini ya~

$(x, y, z) \rightarrow x*y*z;$

- $x*y*z$ adalah sebuah implementasi.
- sedangkan x, y, z adalah parameter variable.



Kalau bentuk dari lambda-nya sendiri ada beberapa macam, gengs!

Berikut adalah beberapa bentuk dari lambda:

- **Consumer**
- **Biconsumer**
- **Supplier**

Bahas satu persatu, yuk!



- **Consumer** merupakan functional interface yang cuma menerima satu input aja. Contohnya kayak gini nih~



```
Consumer<String> printConsumer = t → System.out.println(t);go(f, seed, [])  
}
```

- **Biconsumer** merupakan functional interface yang menerima dua input. Contohnya begini yaaa~



```
Consumer<String, String> printConsumer = (x,y) → System.out.println(x + " : " + y);
```

- **Supplier** merupakan functional interface tanpa input. Contohnya bisa kamu cek pada gambar di bawah ya!



```
Consumer<String, String> printConsumer = (x,y) → System.out.println(x + " : " + y);
```

LANJUTTT

Fitur Optional Class dan Lambda ini super ngebantu banget!

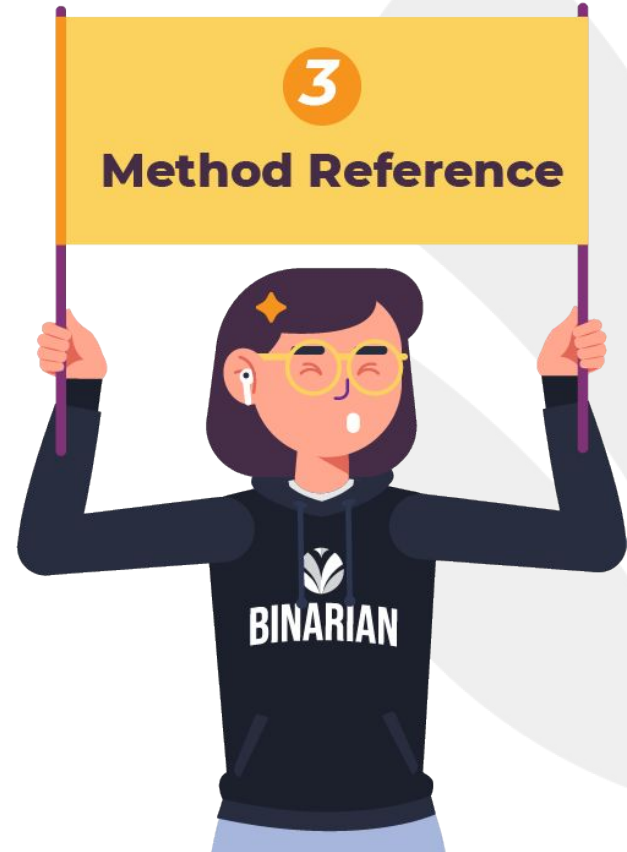
Selanjutnya kita bakal bahas tentang fitur yang nggak kalah keren, yaitu **Method Reference**.



Fitur yang ketiga adalah Method Reference~

Method references merupakan fitur yang dikembangkan di Java 8 untuk melakukan **reference terhadap suatu method**.

Method references menggunakan symbol `::` atau yang sering disebut **double semicolon**.



Ada tiga macam method references pada Java lho, gengs!

Method reference ini di antaranya:

1. **Reference terhadap static method**
2. **Reference terhadap instance method**
3. **Reference terhadap constructor**

Penjelasan lengkap tentang method references bisa ditemukan [di sini](#), yaaa~


Selanjutnya, kita bakal coba bahas contoh dari masing-masing reference method diatas.




Reference terhadap Static Method

Berikut adalah notasi umum dalam memakai reference di method static.

- **Containing Class** merupakan class yang punya method static.
- **staticMethodName** merupakan nama static method tersebut.




```
ContainingClass::staticMethodName
```



```
(args) -> Class.staticMethod(args)
```

MENJADI




```
Class::staticMethod
```


Reference terhadap Instance Method

Berikut adalah notasi umum dalam menggunakan reference di instance method.

- **ObjectType** merupakan tipe object atau nama class yang punya instance method.
- **instanceMethod** merupakan nama method tersebut.




```
ObjectType::instanceMethod
```



```
(obj, args) ->  
obj.instanceMethod(args)
```

MENJADI




```
ObjectType::instanceMethod
```


Reference pada constructor

Berikut adalah notasi umum untuk reference terhadap constructor.

- **ClassName** merupakan nama class yang akan di-instance.
- **new** keyword menandakan bahwa constructor dipanggil pakai method reference.




```
ClassName::new
```



```
(args) -> new ClassName(args)
```

MENJADI



```
ClassName::new
```

Setelah mempelajari konsep Lambda, supaya makin komplit kita bakal bahas tentang contoh dari **penggunaan Lambda di Collection**.

Yuk kita intip materinya!

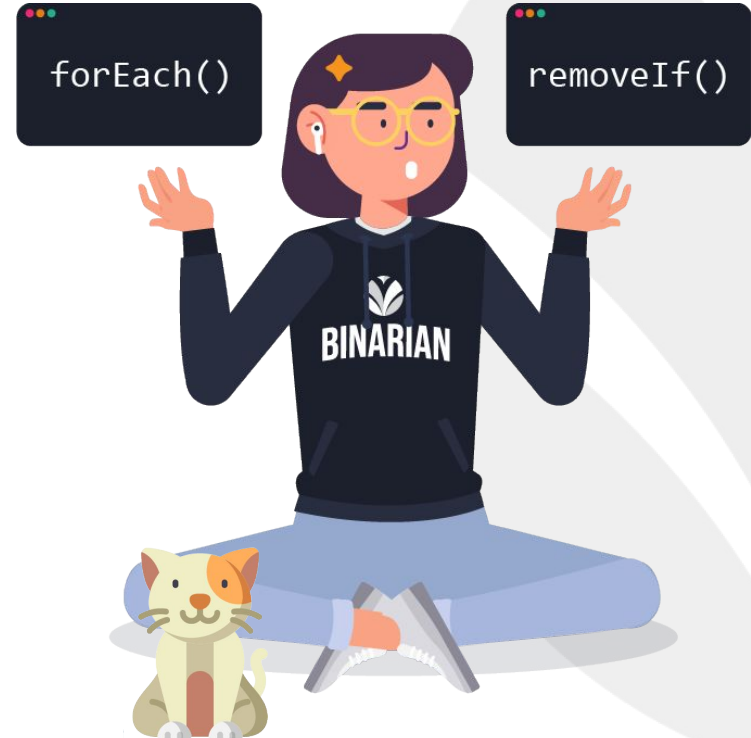


Sebelum masuk ke contohnya, kita bahas fitur yang ada di collection dulu yaaaa~

Jadi, ada beberapa fitur yang ada di collection yang dikembangkan oleh Lambda. Fiturnya tuh ada dua, yaitu:

- `forEach()`
- `removeIf()`

Sebenarnya masih banyak method lain yang dikembangkan pakai Lambda lho, tapi kita bahas dua dulu aja, ya!



forEach

Penasaran sama contoh penggunaan dari forEach? Perhatikan gambar di bawah, ya!

Sebenarnya, untuk melakukan perulangan dari sebuah list, kita bisa pakai ini tanpa pakai forEach, lho.

```
List<String> words = Arrays.asList("Abc", "def");  
for (String word : words){  
    System.out.println(word);  
}
```


- Dengan menggunakan lambda:

```
words.forEach(word → System.out.println(word);
```

- Dengan menggunakan method reference:

```
words.forEach(System.out::println);
```

Selain itu, `forEach` juga bisa dipakai di `map`.

```
Map<int, String> pair = new HashMap();  
pair.put(1, "abc");  
pair.put(2, "def");  
Apabila mau melakukan print output dengan perulangan biasa  
for(Map.Entry<int, String> entry : pair){  
    System.out.println( entry.getKey() + " : " + entry.getValue() );  
}
```

Kalau pakai Lambda `forEach`, nantinya bakal jadi kayak gini!

```
map.forEach((key, value) → System.out.println(key + " : " + value));
```

removeIf()

Berikut adalah contoh penggunaan lambda yang memakai collection!

```
List<String> words = Arrays.asList("Abc", "def", "ghjkl", "mnop");  
for (String word : words){  
    if(words.length()>3){  
        words.remove(word)  
    }  
}
```

Kalau kita eksekusi pakai cara ini, maka hasilnya nggak bakal valid karena **mnop bakal tetap ada sebagai element.**

Index di listwords bakal jadi kacau karena index terakhir bakal bergeser

Berikut adalah contoh kalau kita pakai lambda!

Jadi, kalau pakai method `removeIf`, remove collection nggak bakal bermasalah, gengs.

Mantul, kannn?!



```
words.removeIf(word → word.length>3);
```

Latihan? Siapa takut!

Gimana nih, masih semangat ngerjain latihannya? Nah, di latihan kali ini silakan **implementasikan Lambda pada project Java yang kamu buat di pertemuan sebelumnya!**

Latihan ini dilakukan di kelas dan silahkan diskusikan hasil jawaban kamu dengan teman sekelas dan fasilitator, ya.

Selamat mencoba~



Dengan adanya fitur lambda, memungkinkan programmer atau developer sanggup menulis baris code lebih ringkas dan sederhana.

Selain itu, apa lagi ya fungsi dari Lambda? Yuk kita recall lagi dari materi yang sudah kita pelajari di topic ini~



Nah, selesai sudah pembahasan kita di Chapter 3 Topic 4 ini.

Selanjutnya, kita bakal bahas tentang **Java 8 Part 2** yang secara khusus ngomongin tentang fitur Stream.

Penasaran kayak gimana? Yuk langsung ke topik selanjutnya~

