

Laporan Tugas

Higher Order Functions



Dipersiapkan oleh :

Muhammad Ariq Faridzki

2242004

SEKOLAH TINGGI MANAJEMEN INFORMATIKA & KOMPUTER
"AMIK BANDUNG"
2023

Praktikum

Bagian Pertama

1. Edit kode di atas Tambahkan Fungsi Variabel baru untuk mengembalikan Fungsi Tambah,
2. serta tambahkan fungsi untuk mengurangi bilangan dan panggil fungsi tersebut menggunakan Teknik High Order Function,
3. sebutkan variabel dan kode yang ditambahkan di kotak di bawah ini, dan tuliskan hasil analisisnya yang dipelajari

```
1 import java.util.function.Function;
2
3 public class Functions {
4     public static void main(String[] args) {
5
6         Function<Integer, Integer> tambahSatu = tambah(1);
7         Function<Integer, Integer> tambahDua = tambah(2);
8
9         Function<Integer, Integer> tambahTiga = tambah(3); // Soal nomor 1
10
11         Integer result = tambahSatu.apply(6);
12         System.out.println(result);
13
14         result = tambahDua.apply(4);
15         System.out.println(result);
16
17         Function<Integer, Integer> kurangi10 = kurangi(10); // Soal nomor 2
18
19         result = kurangi10.apply(5);
20         System.out.println(result);
21
22     }
23
24     static Function<Integer, Integer> tambah(Integer x){
25         return y -> y + x;
26     }
27
28     static Function<Integer, Integer> kurangi(Integer z){
29         return y -> y - z;
30     }
31
32 }
33 }
```

Untuk Variabel Ditambahkan sebagai berikut



```
1 Function<Integer, Integer> tambahTiga = tambah(3); // Soal nom
```



```
1 Function<Integer, Integer> kurangi10 = kurangi(10);
```

Dan ini variabel function yang terpanggil :



```
1 static Function<Integer, Integer> tambah(Integer x){  
2     return y -> y + x;  
3 }  
4  
5 static Function<Integer, Integer> kurangi(Integer z){  
6     return y -> y - z;  
7 }
```

Hasil Eksekusi :



```
1 7  
2 6  
3 -5  
4
```

Hasil analisa

Dari kode yang diberikan saya menyimpulkan sebagai berikut :

Dari hal konvensional dengan menyatukan badan function dalam 1 variable function, maka lebih baik dipisah dan memanggilnya saat dibutuhkan, ini membuat fleksibilitas saat bekerja lebih baik, karena tinggal menggunakannya kembali.

Jadi hal ini bisa membuat kode lebih mudah terbaca, kalau misalnya dalam 1 function memiliki banyak sekali kode, alangkah baiknya menggunakan metode seperti ini.

Bagian Kedua

```
1 import java.util.*;
2
3 public class compare {
4     public static void main(String[] args) {
5         List<Integer> numbers = Arrays.asList(3,6,4,9,7);
6
7         Collections.sort(numbers, (a, b) ->{
8             return a.compareTo(b);
9         });
10
11        System.out.println("Regular Collections Sort : " + numbers);
12
13        Comparator<Integer> comparator = (a,b) -> {
14            return a.compareTo(b);
15        };
16
17        Comparator<Integer> reverserComparator = comparator.reversed();
18
19
20        Collections.sort(numbers, reverserComparator);
21
```

1. Apa yang kalian pelajari dari kode diatas?

Kode diatas itu yang saya bisa pahami adalah bahwa bagian dalam Collections.Sort nomor 7 itu bisa diganti dengan custom functions yang sesuai kriteria Collections.sort

Dengan mengimplementasikan teknik Higher order functions, hal ini membuat kode kelihatan lebih rapi.

Tugas

Bagian Pertama

- Buatlah Program Menghitung Persegi Panjang dengan cara High Order Function untuk mengembalikan fungsi

Persiapan

```
1 import java.util.function.BiFunction;
2 import java.util.function.Function;
```

Menggunakan BiFunction agar bisa apply 2 argumen sekaligus.

Variable Functions

```
1
2 static BiFunction<Integer, Integer, Integer> luasLogic(){
3     return (x,y) -> x * y;
4 };
5
6 static BiFunction<Integer, Integer, Integer> kelilingLogic(Integer fixedNumber){
7     return (x,y) -> fixedNumber * (x + y);
8 };
9
10 static BiFunction<Integer, Integer, Integer> functionFactory(BiFunction<Integer, Integer, Integer> opArit){
11     return opArit;
12 }
```

1. **LuasLogic** : Digunakan untuk menghitung luas
2. **kelilingLogic** : Digunakan untuk menghitung keliling
3. **functionFactory** : Digunakan untuk menghasilkan function yang flexible dengan 2 argumen
 - a. **opArit** adalah sebagai parameter yang nantinya akan dikembalikan berbentuk functions, nanti kita hanya memberikan badan (logika) functionnya saja

Implementasi Teknik Higher Order Function

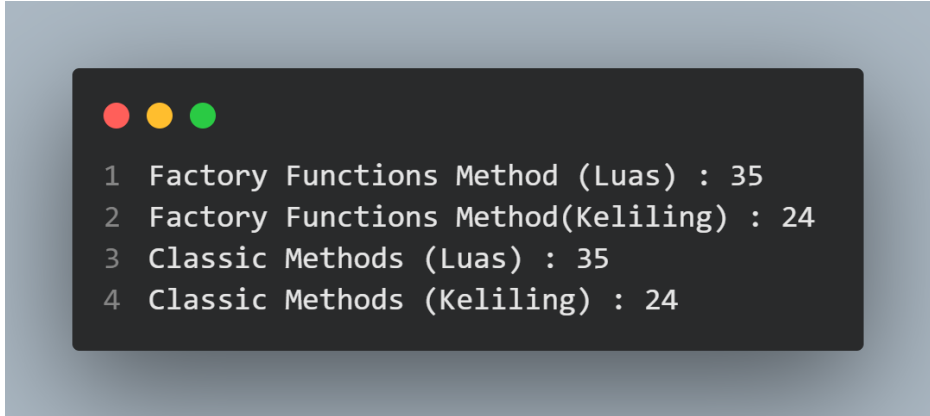
```
1 import java.util.function.BiFunction;
2 import java.util.function.Function;
3
4 public class persegi_HoF {
5     public static void main(String[] args) {
6
7         BiFunction<Integer, Integer, Integer> luasLingkaran = functionFactory((x,y) -> x * y);
8         BiFunction<Integer, Integer, Integer> kelilingLingkaran = functionFactory((x,y) -> 2 * (x + y) );
9
10        BiFunction<Integer, Integer, Integer> luasLingkaran2 = luasLogic();
11        BiFunction<Integer, Integer, Integer> kelilingLingkaran2 = kelilingLogic(2);
```

Saat sudah membuat function nya, saya hanya tinggal memberikan function yang diperlukan.

Pemanggilan Functions

```
1 Integer result = luasLingkaran.apply(5,7);
2 System.out.println("Factory Functions Method (Luas) : " +result);
3
4 result = kelilingLingkaran.apply(8,4);
5 System.out.println("Factory Functions Method (Keliling) : " + result);
6
7 Integer result2 = luasLingkaran2.apply(5,7);
8 System.out.println("Classic Methods (Luas) : " + result2);
9
10 result2 = kelilingLingkaran2.apply(8,4);
11 System.out.println("Classic Methods (Keliling) : " + result2);
```

Hasil

A screenshot of a terminal window with a dark background and light gray text. At the top left of the terminal are three colored circles: red, yellow, and green. The terminal displays four lines of output, each preceded by a number from 1 to 4.

```
1 Factory Functions Method (Luas) : 35
2 Factory Functions Method(Keliling) : 24
3 Classic Methods (Luas) : 35
4 Classic Methods (Keliling) : 24
```

Bagian Kedua

- Berikan contoh dan implementasi 2 native High Order Function di Java dan berikan penjelasannya, sertakan sumbernya

Java Stream Operation .filter()

`filter()` method adalah suatu operasi dalam operasi `.stream` yang berguna untuk memfilter isi elemen yang diberikan menggunakan kriteria, `.filter` akan membuat elemen baru dengan elemen yang sudah ditandai dengan **true** (Baeldung)

Argumen `.filter()` bisa diisi dengan ekspresi lambda, atau dengan menggunakan *Functional Interface* **Predicate**.

Predicate disini adalah suatu function yang dapat menggunakan parameter dan hanya mengembalikan tipe data boolean.

Implementasi

Persiapan :

```
1 import java.util.Arrays;
2 import java.util.List;
3 import java.util.function.Function;
4 import java.util.function.Predicate;
5 import java.util.stream.Collectors;
```

Implementasi `.filter` dalam `.stream` :

```
1 static Function<List<Integer>, List<Integer>> test = a -> a.stream()
2   .filter(isEven)
3   .collect(Collectors.toList());
```

```
1 static Predicate<Integer> isEven = x -> x % 2 == 0;
```

Jika tanpa predicate maka :

```
1 static Function<List<Integer>, List<Integer>> test = a -> a.stream()
2   .filter(x -> x % 2 == 0)
3   .collect(Collectors.toList());
```

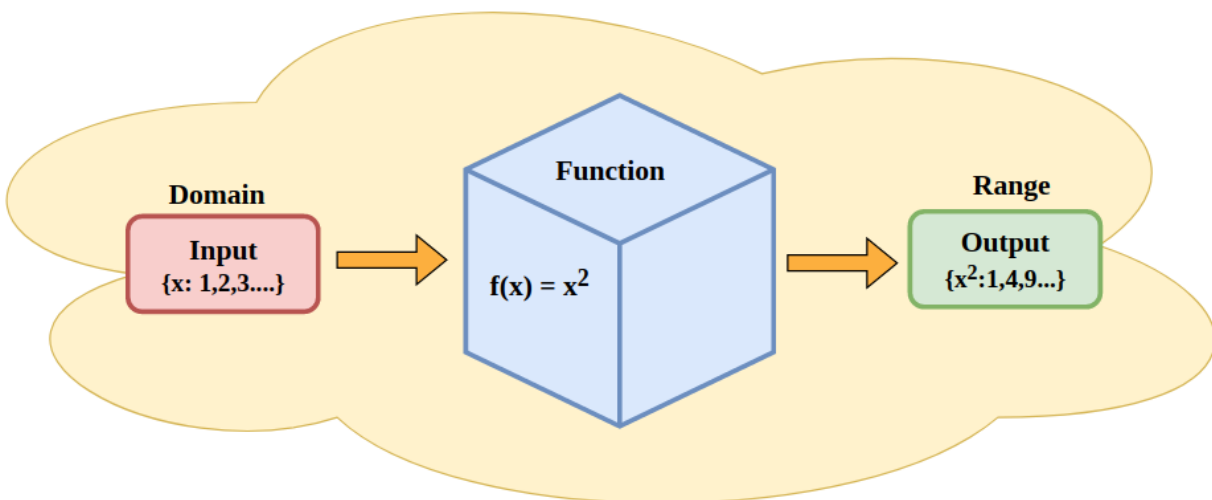
Pemanggilan :

```
1 public class HoF_functions {  
2     public static void main(String[] args) {  
3         List<Integer> numbers = Arrays.asList(3,6,4,9,7);  
4  
5         System.out.println(test.apply(numbers));  
6     }  
}
```

Maka akan menghasilkan : [6,4] dengan **List** yang baru

Java Stream Operation .map()

.map digunakan untuk mengkonversi element yang ada ke element yang baru dengan kondisi yang telah ditentukan, seperti Function dalam matematika.



Gambar 1 - Ilustrasi Function Domain dan Range : (Jain)

.map juga bersifat sama, yaitu harus dilakukan dalam .stream sebagai **intermediate operation**.

Intermediate operation sendiri yaitu suatu method yang akan mengembalikan stream baru, anggap saja seperti mengembalikan element baru dengan bertipe stream.

.map akan mengambil argumen bertipe **Functional Interface Function**, yaitu yang akan mengambil argumen dan akan mengembalikan yang telah ditentukan (fleksibel) atau bisa juga digantikan dengan ekspresi lambda

Implementasi

Implementasi hampir mirip dengan method .filter

Dan Ini menggunakan teknik Higher order function

```
1 System.out.println(numbers.stream().map(square).collect(Collectors.toList()));
```

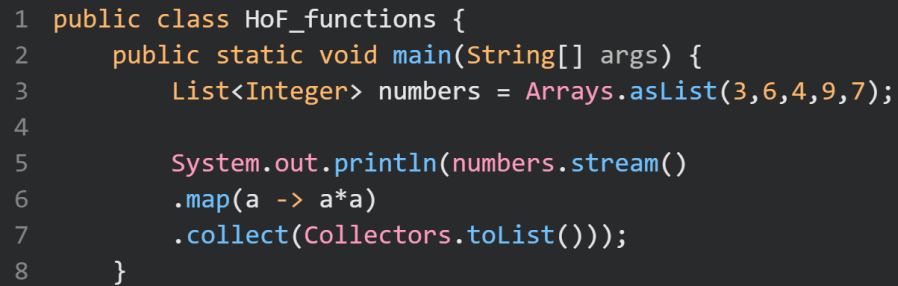
Dengan membuat variabel Functional Interface baru yang akan diberikan sebagai argumen nanti

```
1 static Function<Integer, Integer> square = a -> a * a;
```

Sedangkan ini yang biasa menggunakan ekspresi lambda sebagai penggantinya

```
1 System.out.println(numbers.stream()  
2     .map(a -> a*a)  
3     .collect(Collectors
```

Dan berikut pemanggilannya :



```
1 public class HoF_functions {  
2     public static void main(String[] args) {  
3         List<Integer> numbers = Arrays.asList(3,6,4,9,7);  
4  
5         System.out.println(numbers.stream()  
6             .map(a -> a*a)  
7             .collect(Collectors.toList()));  
8     }
```

Dan akan menghasilkan : [9, 36, 16, 81, 49]

References

- Baeldung. "Java Stream Filter with Lambda Expression." *Baeldung*, 17 October 2023, <https://www.baeldung.com/java-stream-filter-lambda>. Accessed 20 October 2023.
- Chaudhary, Namita. "Java Stream map()." *Scaler*, Namita Chaudhary, 20 12 2022, <https://www.scaler.com/topics/java-stream-map/>. Accessed 1 October 2023.
- Fadatare, Ramesh. "Java Stream Intermediate Operations Examples." *Java Guides*, <https://www.javaguides.net/2021/11/java-stream-intermediate-operations.html>. Accessed 20 October 2023.
- Jain, Sandeep. "Domain and Range of a Function: Definition, Rules, and Examples." *GeeksforGeeks*, 19 June 2023, <https://www.geeksforgeeks.org/domain-and-range/>. Accessed 20 October 2023.
- Jain, Sandeep. "Function Interface in Java with Examples." *GeeksforGeeks*, 6 March 2023, <https://www.geeksforgeeks.org/function-interface-in-java-with-examples/>. Accessed 20 October 2023.
- Marimuthu, Ganesh Kumar. "Functional Programming in Java With Examples." *Scaler*, 16 06 2023, <https://www.scaler.com/topics/java/functional-programming-in-java/>. Accessed 1 October 2023.
- Marimuthu, Ganesh Kumar. "Streams in Java- Complete Tutorial with Examples." *Scaler*, Scaler, 4 5 2023, <https://www.scaler.com/topics/java/streams-in-java/>. Accessed 1 October 2023.
- Mishra, Suraj. "Commonly Used Intermediate Stream Operations In Java | by Suraj Mishra | Javarevisited." *Medium*,

<https://medium.com/javarevisited/commonly-used-intermediate-stream-operations-in-java-5c2bdad4231e>. Accessed 20 October 2023.

Oracle. "java.util.stream (Java Platform SE 8)." *Oracle Help Center*, <https://docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html>. Accessed 20 October 2023.

Oracle. "Stream (Java Platform SE 8)." *Oracle Help Center*, Oracle, <https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html>. Accessed 1 October 2023.

Singh, Prince. "Java 8 Functional Interface - Consumer, Predicate, Supplier | Javarevisited." *Medium*, <https://medium.com/javarevisited/java-8s-consumer-predicate-supplier-and-function-bbc609a29ff9>. Accessed 20 October 2023.