
Java Lambda

Sebelum Belajar Materi Ini

- Java Dasar
- Java Object Oriented Programming
- Java Generic
- Java Collection



Agenda

- Pengenalan Lambda
- Membuat Lambda
- Java Function
- Method Reference
- Dan lain-lain

Pengenalan Lambda

Apa Itu Lambda?

- Lambda berasal dari lambda calculus, yang mengacu pada anonymous function (function tanpa nama)
- Tapi, di Java, function/method tidak bisa berdiri sendiri.
- Jadi kemungkinan pada prakteknya, lambda di Java dan di bahasa pemrograman lain akan berbeda

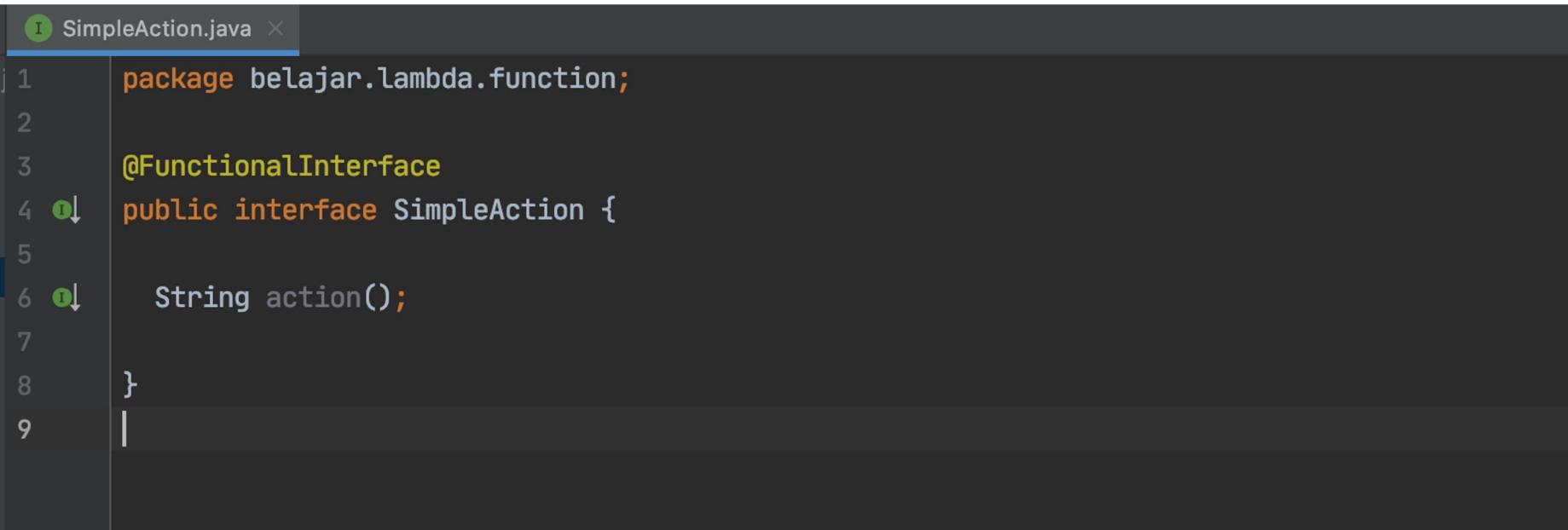
Anonymous Class

- Di Java kita hanya mengenal Anonymous Class
- Lambda di Java sebenarnya adalah versi sederhana membuat sebuah anonymous class

Syarat Lambda

- Berupa Interface
- Memiliki 1 method abstract
- Ditambahkan annotation @FunctionalInterface di Interface-nya
- Minimal menggunakan Java 8

Kode : Membuat Lambda Interface



The screenshot shows a code editor window with a dark theme. The title bar says "SimpleAction.java". The code is a Java functional interface:

```
1 package belajar.lambda.function;
2
3 @FunctionalInterface
4 public interface SimpleAction {
5
6     String action();
7
8 }
9 |
```

Line numbers 1 through 9 are visible on the left. There are two small orange circular icons with arrows pointing down on lines 4 and 6, likely indicating code completion or suggestions.

Kode : Membuat Lambda

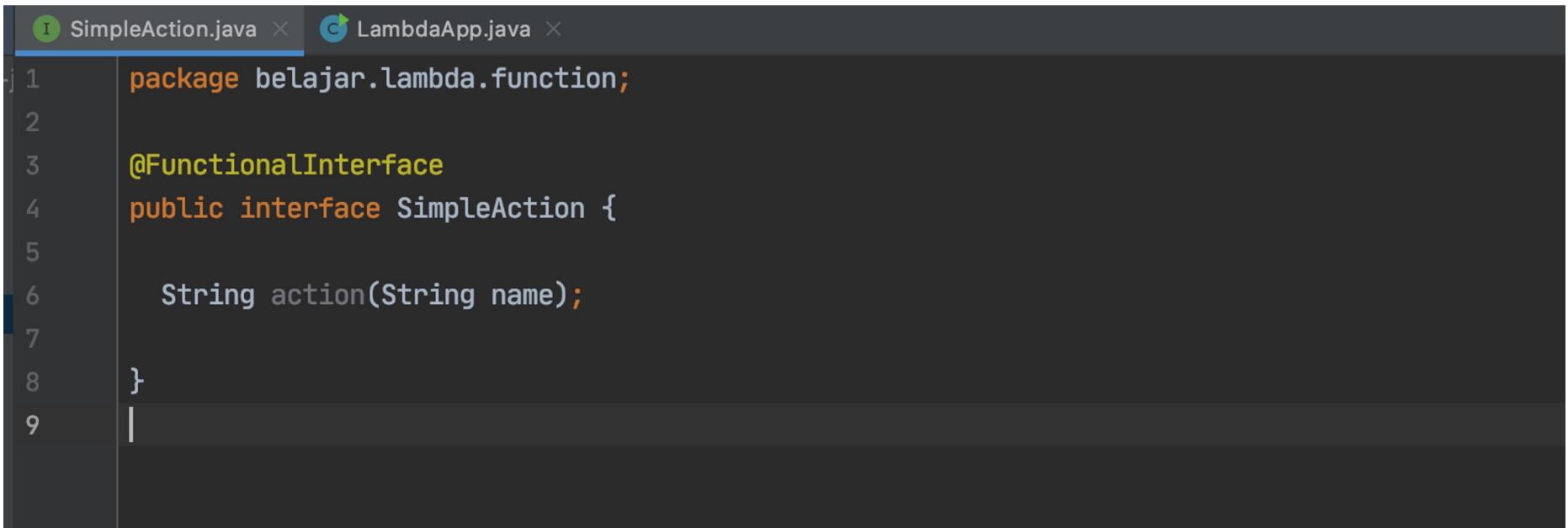
```
8
9
10    SimpleAction simpleAction = () -> {
11        return "Eko";
12    };
13
14    }
15}
16
17
```

Membuat Lambda

Membuat Lambda

- Di materi sebelumnya kita sudah mencoba membuat lambda
- Sekarang kita akan bahas beberapa cara membuat lambda lainnya

Kode : Lambda Interface Dengan Parameter



The screenshot shows a code editor interface with two tabs at the top: 'SimpleAction.java' and 'LambdaApp.java'. The 'SimpleAction.java' tab is active, indicated by a blue underline. The code in the editor is:

```
1 package belajar.lambda.function;
2
3 @FunctionalInterface
4 public interface SimpleAction {
5
6     String action(String name);
7
8 }
9 |
```



Kode : Lambda dengan Parameter

```
8
9
10    SimpleAction simpleAction1 = (String name) -> {
11        return "Hello " + name;
12    };
13
14    SimpleAction simpleAction2 = (name) -> {
15        return "Hello " + name;
16    };
17
18
19 }
```



Kode : Lambda Tanpa Blok

```
6
9
10    SimpleAction simpleAction1 = (String name) -> "Hello " + name;
11
12    SimpleAction simpleAction2 = (name) -> "Hello " + name;
13
14    SimpleAction simpleAction3 = name -> "Hello " + name;
15
16
17    }
18}
19
```

Java Util Function

Package `java.util.function`

- Saat Java 8 keluar dengan fitur Lambda nya
- Java juga menyediakan sebuah package baru bernama `java.util.function`
- Package ini berisikan banyak sekali functional interface yang bisa kita gunakan untuk kebutuhan membuat lambda
- Dengan menggunakan interface-interface yang ada di package ini, kita mungkin tidak perlu lagi membuat sendiri functional interface secara manual

Interface Consumer

```
40     */
41     @FunctionalInterface
42     public interface Consumer<T> {
43
44         /**
45          * Performs this operation on the given argument.
46          *
47          * @param t the input argument
48         */
49     void accept(T t);
50
51     /**
52      *
```

Kode : Menggunakan Consumer

```
7  
8     Consumer<String> consumer = value -> System.out.println(value);  
9  
10    consumer.accept( t: "Eko Kurniawan");  
11  
12  
13  
14  
15    |  
16  
17  
18  
19
```



Interface Function

```
39     */
40     @FunctionalInterface
41     public interface Function<T, R> {
42
43         /**
44          * Applies this function to the given argument.
45          *
46          * @param t the function argument
47          * @return the function result
48         */
49         R apply(T t);
50     }
```

Kode : Menggunakan Function

```
12  
13     Function<String, Integer> function = value -> value.length();  
14  
15     Integer length = function.apply( t: "Eko Kurniawan Khannedy");  
16  
17     System.out.println(length);  
18  
19  
20  
21  
22     }  
23 }  
24
```

Interface Predicate

```
58
59
60
39  ①  @FunctionalInterface
40  public interface Predicate<T> {
41
42      ②      /**
43          * Evaluates this predicate on the given argument.
44          *
45          * param t the input argument
46          * return {@code true} if the input argument matches the predicate,
47          * otherwise {@code false}
48          */
49      ③      boolean test(T t);
50
```

Kode : Menggunakan Predicate

```
19  
20     Predicate<String> predicate = value -> value.isBlank();  
21  
22     boolean blank = predicate.test( t: "Eko");  
23  
24     System.out.println(blank);  
25  
26  
27  
28  
29  
30 }
```

Interface Supplier

```
38  *~ @interface 1.0
39  */
40  @FunctionalInterface
41  public interface Supplier<T> {
42
43      /**
44      * Gets a result.
45      *
46      * @return a result
47      */
48      T get();
49  }
```

Kode : Menggunakan Supplier

```
26  
27     Supplier<String> supplier = () -> "Eko Kurniawan Khannedy";  
28  
29     String name = supplier.get();  
30  
31     System.out.println(name);  
32  
33  
34  
35     }  
36 }  
37 }
```

Dan Masih Banyak

- ▼ └ function
 - I BiConsumer
 - I BiFunction
 - I BinaryOperator
 - I BiPredicate
 - I BooleanSupplier
 - I Consumer
 - I DoubleBinaryOperator
 - I DoubleConsumer
 - I DoubleFunction
 - I DoublePredicate
 - I DoubleSupplier
 - I DoubleToIntFunction
 - I DoubleToLongFunction
 - I DoubleUnaryOperator
 - I Function
 - I IntBinaryOperator
 - I IntConsumer
 - I IntFunction
 - I IntPredicate
 - I IntSupplier

- I IntToDoubleFunction
- I IntToLongFunction
- I IntUnaryOperator
- I LongBinaryOperator
- I LongConsumer
- I LongFunction
- I LongPredicate
- I LongSupplier
- I LongToDoubleFunction
- I LongToIntFunction
- I LongUnaryOperator
- I ObjDoubleConsumer
- I ObjIntConsumer
- I ObjLongConsumer
- I Predicate
- I Supplier
- I ToDoubleBiFunction
- I ToDoubleFunction
- IToIntBiFunction
- I.ToIntFunction
- I ToLongBiFunction
- I>ToLongFunction
- I UnaryOperator

Method Reference

Method Reference

- Kadang saat membuat lambda, isi lambda hanya mengakses method lain atau mengakses method yang ada di parameter method lambda nya
- Kita bisa mempersingkat pembuatan lambda tersebut dengan method reference

Kode : Static Method

```
2
3     public class StringUtil {
4
5     @
6         public static boolean isLowerCase(String value) {
7             for (char c : value.toCharArray()) {
8                 if (!Character.isLowerCase(c)) return false;
9             }
10            return true;
11        }
12    }
13}
```



Kode : Method Reference Static

```
9
10
11     // Predicate<String> predicate = value -> StringUtil.isLowerCase(value);
12     Predicate<String> predicate = StringUtil::isLowerCase;
13
14     System.out.println(predicate.test( t: "Eko"));
15     System.out.println(predicate.test( t: "eko"));
16
17 }
18
19 }
20 |
```

Kode : Method Reference Non Static

```
7  public void run(){
8      // Predicate<String> predicate = value -> isLowerCase(value);
9      Predicate<String> predicate = this::isLowerCase;
10
11     System.out.println(predicate.test( t: "Eko"));
12     System.out.println(predicate.test( t: "eko"));
13 }
14
15 @ public boolean isLowerCase(String value) {
16     for (char c : value.toCharArray()) {
17         if (!Character.isLowerCase(c)) return false;
18     }
}
```

Kode : Method Reference Object

```
10
11     StringApp stringApp = new StringApp();
12
13     // Predicate<String> predicate = value -> stringApp.isLowerCase(value);
14     Predicate<String> predicate = stringApp::isLowerCase;
15
16     System.out.println(predicate.test( t: "Eko"));
17     System.out.println(predicate.test( t: "eko"));
18
19 }
20
21 }
```

Kode : Method Reference Parameter

```
19
20     // Function<String, String> function = value -> value.toUpperCase();
21     Function<String, String> function = String::toUpperCase;
22
23     System.out.println(function.apply( t: "Eko"));
24     System.out.println(function.apply( t: "Kurniawan"));
25
26 }
27
28 }
29 |
```

Lambda di Collection

Lambda di Collection

- Saat fitur Lambda keluar di Java 8, ada banyak sekali default method yang ditambahkan ke Java Collection
- Beberapa ada default method yang banyak memanfaatkan fitur Lambda
- Di materi ini kita akan bahas beberapa method yang memanfaatkan Lambda

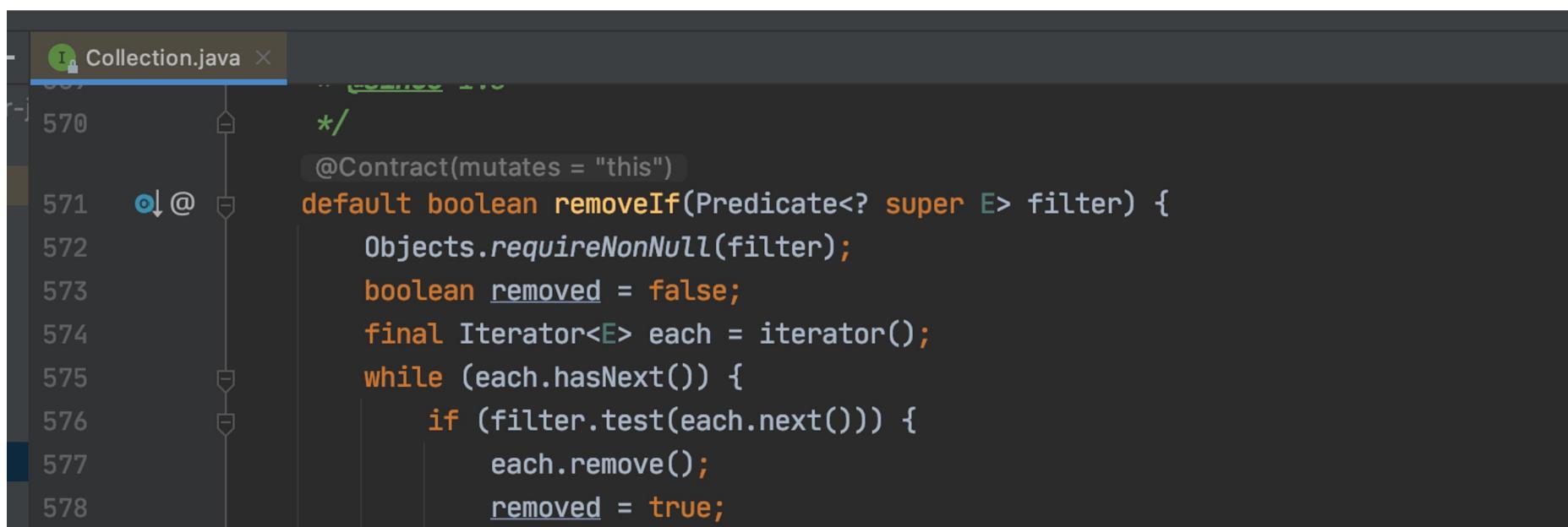
Iterable.forEach

```
I Iterable.java ×
67 *
68 * @param action The action to be performed for each element
69 * @throws NullPointerException if the specified action is null
70 * @since 1.8
71 */
72 default void forEach(Consumer<? super T> action) {
73     Objects.requireNonNull(action);
74     for (T t : this) {
75         action.accept(t);
76     }
77 }
```

Kode : Menggunakan Iterable.forEach

```
8  
9     List<String> names = List.of("Eko", "Kurniawan", "Khannedy");  
10  
11    names.forEach(name -> System.out.println(name));  
12  
13    names.forEach(System.out::println);  
14  
15 }  
16 }  
17 }
```

Collection.removeIf



The screenshot shows a Java code editor with a dark theme. The file being edited is named `Collection.java`. The code implements the `removeIf` method for a collection. The implementation uses a `Predicate` to filter elements and remove them from an `Iterator`.

```
Collection.java
570     */
571     @Contract(mutates = "this")
572     default boolean removeIf(Predicate<? super E> filter) {
573         Objects.requireNonNull(filter);
574         boolean removed = false;
575         final Iterator<E> each = iterator();
576         while (each.hasNext()) {
577             if (filter.test(each.next())) {
578                 each.remove();
579                 removed = true;
580             }
581         }
582         return removed;
583     }
```

Kode : Menggunakan Collection.removeIf

```
10  
11     List<String> names = new ArrayList<>();  
12     names.addAll(List.of("Eko", "Kurniawan", "Khannedy"));  
13  
14     names.removeIf(name -> name.length() > 5);  
15  
16     System.out.println(names);  
17  
18  
19  
20  
21
```

Map.forEach



The screenshot shows a Java code editor with a dark theme. A file named "Map.java" is open, indicated by a blue header bar. The code implements the `forEach` method for a `Map`. The implementation uses a `BiConsumer` action to process each entry. It first checks if the action is non-null, then iterates over the entry set, extracting the key and value from each entry, and finally catching an `IllegalStateException`.

```
Map.java
648     */
649     default void forEach(BiConsumer<? super K, ? super V> action) {
650         Objects.requireNonNull(action);
651         for (Map.Entry<K, V> entry : entrySet()) {
652             K k;
653             V v;
654             try {
655                 k = entry.getKey();
656                 v = entry.getValue();
657             } catch (IllegalStateException ise) {
658                 // This usually means the entry is no longer in the map
659             }
660         }
661     }
```

Kode : Menggunakan Map.forEach

```
20
21     Map<String, String> map = new HashMap<>();
22     map.put("first_name", "Eko");
23     map.put("middle_name", "Kurniawan");
24     map.put("last_name", "Khannedy");
25
26     map.forEach((key, value) -> System.out.println(key + ":" + value));
27
28
29     }
30 }
31 |
```

Dan Masih Banyak

- Cek isi source code dari Java Collection

Lambda Sebagai Lazy Parameter

Lazy Parameter

- Java tidak memiliki fitur parameter lazy seperti di bahasa pemrograman seperti Scala
- Lazy parameter artinya, parameter tersebut hanya akan dieksekusi ketika diakses
- Untungnya, dengan menggunakan Lambda, kita bisa membuat parameter layaknya lazy parameter

Kode : Bukan Lazy Parameter

```
8     public static void testScore(int value, String name) {
9         if (value > 80) {
10             System.out.println("Selamat " + name + " , Anda Lulus");
11         } else {
12             System.out.println("Coba Lagi Tahun Depan");
13         }
14     }
15
16     @
17     public static String getName() {
18         System.out.println("getName() dipanggil");
19         return "Eko";
20     }
```

Kode : Lazy Parameter

```
8      |
9      |     testScore( value: 90, () -> getName());
10     }
11
12     public static void testScore(int value, Supplier<String> name) {
13         if (value > 80) {
14             System.out.println("Selamat " + name.get() + " , Anda Lulus");
15         } else {
16             System.out.println("Coba Lagi Tahun Depan");
17         }
18     }
19 }
```

Lambda di Optional

Optional Class

- Di Java 8, java menyediakan sebuah class baru bernama Optional yang berada di package java.util
- Class ini digunakan sebagai wrapper untuk value yang bisa bernilai null
- Optional didesain agar kita lebih mudah ketika beroperasi dengan object yang bisa null
- Karena NullPointerException adalah salah satu hal yang sering sekali ditemui oleh Programmer Java

Kode : Problem NullPointerException

```
6
7         sayHello( name: "Eko");
8         sayHello( name: null);
9     }
10
11    @
12        public static void sayHello(String name) {
13            String upperName = name.toUpperCase();
14            System.out.println("HELLO " + upperName);
15        }
16    |
```

Kode : Menggunakan Optional

```
 9      sayHello( name: "Eko");
10     sayHello( name: null);
11 }
12
13 public static void sayHello(String name) {
14     Optional<String> optionalName = Optional.ofNullable(name);
15
16     Optional<String> upperName = optionalName.map(value -> value.toUpperCase());
17
18     upperName.ifPresent(value -> System.out.println("HELLO " + value));
19 }
20 }
```

Kode : Mengambil Data di Optional

```
8
9         sayHello( name: "Eko");
10        sayHello( name: null);
11    }
12
13    public static void sayHello(String name) {
14        String upperName = Optional.ofNullable(name)
15            .map(value -> value.toUpperCase())
16            .orElseGet(() -> "");
17
18        System.out.println("HELLO " + upperName);
19    }
```

Materi Selanjutnya

Materi Selanjutnya

- Apache Maven
- Java Unit Test
- Java Stream