TRAINER NOTES

Lambda Expression and Java Stream

Anonymous is very simple, that contains only one method

By Tika Yesi Kristiani

Lambda Expressions

1. Sebelumnya, jika tidak menggunakan Lambda Expression, ketika kita mempunyai sebuah interface yang memiliki parameter seperti contoh dibawah ini:

```
package com.company;

public interface StateChangeListener {
    public void onChangeState(Integer state);
}
```

2. Kemudian, kita akan coba memasukkan interface yang sudah kita buat kedalam sebuah method addStateListerner pada class StateOwner seperti code berikut ini dan kita beri nilai 5 pada parameternya:

```
public class StateOwner {
    public void addStateListener(StateChangeListener stateChangeListener){
        stateChangeListener.onChangeState(5);
    }
}
```

3. Saat kita akan memanggil method onChangeState yang sebelumnya sudah kita buat pada interface, kita harus membuat implementasi baru untuk StateChangeListener, seperti code dibawah ini

```
public static void main(String[] args) {

    StateOwner stateOwner = new StateOwner();

    stateOwner.addStateListener(new StateChangeListener() {
        @Override
        public void onChangeState(Integer state) {
            System.out.println(state);
        }
    });
```

4. Namun dengan menggunakan Lambda expression, kita dapat membuat code lebih efisien (clean code) dengan menggantinya menjadi:

```
stateOwner.addStateListener((state) -> System.out.println(state));
```

Lambda Expression ini adalah salah satu feature yang ada pada java 8 keatas, dan belum dapat diimplementasikan untuk java dibawah 8.

5. Bagaimana jika kita mempunyai lebih dari 1 parameter pada interface yang kita miliki?

```
public interface StateChangeListener {
    public void onChangeState(Integer state1, Integer state2);
}
```

6. Pada class state owner kita dapat mengisikan 2 buat parameter

```
public class StateOwner {
    public void addStateListener(StateChangeListener stateChangeListener){
        stateChangeListener.onChangeState(5, 7);
    }
}
```

7. Sehingga, kita dapat memanggil ke 2 parameter yang di butuhkan oleh interface seperti contoh code di bawah ini

```
stateOwner.addStateListener((state1, state2) -> System.out.println(state1 + state2));
```

8. Atau jika kita membutuhkan operasi yang membutuhkan lebih dari 1 line code, kita dapat melakukan hal dibawah ini

```
stateOwner.addStateListener((state1, state2) -> {
    Integer totalState = state1 + state2;
    System.out.println(totalState);
});
```

Kita bisa menyebut lambda sebagai *simple interface in one method*.

STREAM

9. Jika kita memiliki sebuah list array seperti berikut

```
List<Integer> numbers = Arrays.asList(3,5,1,26,87,2);
```

10. Ketika kita ingin mengeluarkan nilai numbers menggunakan foreach, maka kita melakukan code di bawah ini

```
for (Integer number:
     numbers ) {
    System.out.println(number);
}
```

11. Apa yang terjadi ketika kita ingin merubah nilai number kita kalikan menjadi 2? Kita akan merubah code yang ada didalam foreach

```
for (Integer number:
     numbers) {
    Integer multiplyNumber = number * 2;
    System.out.println(multiplyNumber);
}
```

12. Jika menggunakan stream, kita dapat melakukan foreach dimana sebuah stream foreach menerima sebuah interface consumer

```
numbers.stream().forEach(new Consumer<Integer>() {
    @Override
    public void accept(Integer integer) {
        System.out.println(integer);
    }
});
```

13. Namun kita juga dapat membuat stream lebih flexible karena stream dapat menerima sebuah variable sehingga kita tidak perlu merubah isi didalam stream. Jadi ketika kita mempunyai 2 requirement yang berbeda.
    - Code untuk mengeluarkan nilai array, kita dapat menampungnya kedalam sebuah variable

```
Consumer<Integer> value = new Consumer<Integer>() {
    @Override
    public void accept(Integer integer) {
        System.out.println(integer);
    }
};
```

    - Kemudian variable tersebut dapat kita masukkan kedalam stream

```
numbers.stream().forEach(value);
```

    - Jika kita mempunyai sebuah variable lain dimana fungsinya adalah nilai array dikalikan 2

```
Consumer<Integer> multiplyValue = new Consumer<Integer>() {
    @Override
    public void accept(Integer integer) {
        System.out.println(integer*2);
    }
};
```

    - Kita hanya perlu mengganti stream pada parameter menjadi multiplyValue

```
numbers.stream().forEach(multiplyValue);
```

STREAM WITH LAMBDA

14. Jika kita lihat, code diatas terlihat tidak efisien dan terlalu panjang. Kita dapat menambahkan lambda kedalam sebuah stream sehingga code tersebut lebih mudah dibaca (clean code)
15. Kita dapat melakukan simplify code nomor pada langkah nomor 12 menjadi

```
numbers.stream().forEach((x) -> System.out.println(x));
```

Dimana interface Consumer dapat digantikan oleh lambda yang mengirimkan sebuah parameter.

16. Begitu juga pada langkah nomor 13

```
Consumer<Integer> lamda = (x) -> System.out.println(x);

numbers.stream().forEach(lamda);
```

```
Consumer<Integer> lamda = (x) -> System.out.println(x);
Consumer<Integer> lamda2 = (x) -> System.out.println(x*2);
numbers.stream().forEach(lamda2);
```

STREAM FOREACH, MAP DAN FILTER

17. Jika sebelumnya kita sudah menggunakan foreach

```
numbers.stream().forEach((x) -> System.out.println(x));
```

Foreach mempunyai return void, dan berfungsi untuk melakukan perulangan

18. Map, memiliki sebuah return origin dari type datanya, misalkan integer atau double. Penggunaan map seperti contoh dibawah ini

```
numbers.stream().map((x) -> x*2).forEach((y) -> System.out.println(y));
```

Jika kita tidak dapat menampung kedalam sebuah variable untuk foreach, pada map kita dapat menampungnya kedalam sebuah variable. Disini, listOfNumber merupakan sebuah List of Integer, sehingga return dari map tersebut harus sebuah List of Integer.

```
List<Integer> listOfNumber = numbers.stream().map((number) -> number*2).collect(Collectors.toList());
System.out.println(listOfNumber);
```

19. Filter, memiliki sebuah return Boolean

```
numbers.stream().filter((x) -> x%2==0).forEach((y)-> System.out.println(y));
```

Sehingga nilai yang dikeluarkan oleh filter hanya yang bernilai true, contoh penulisan filter dan lambda lebih dari 1 line:

```
numbers.stream().filter((x) -> x%2==0).forEach((y)-> {
    Integer even = y*2;
    System.out.println(even);
});
```

Sehingga nilai yang tercetak dari array yang kita miliki hanya yang bernilai genap. Hasil true dari x % 2 == 0.

```
52
4
```