Java Advanced

Streaming API

DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

```java
public class Student {
  private String name;
  private int graduationYear;
  private int score;

  public Student(String name, int graduationYear, int score) {
    this.name = name;
    this.graduationYear = graduationYear;
    this.score = score;
  }

  public String getName() {
    return name;
  }

  public int getGraduationYear() {
    return graduationYear;
  }

  public int getScore() {
    return score;
  }
}
```

Gegeven:

```
public static void main(String[] args) {
  List<Student> students = new ArrayList<>();

   students.add(new Student("Alice", 2018, 82));
  students.add(new Student("Bob", 2017, 90));
  students.add(new Student("Carol", 2108,  67));
  students.add(new Student("David", 2018, 80));
  students.add(new Student("Eric", 2017, 55));
  students.add(new Student("Frank", 2018, 49));
  students.add(new Student("Gary", 2017,88));
  students.add(new Student("Henry", 2017, 98));
  students.add(new Student("Ivan", 2018, 66));
  students.add(new Student("John", 2017, 52));
}
```

Gevraagd:

1. De studenten van afstudeerjaar 2017 die 70 of meer hebben gescoord.

Oplossing:

```
List<Student> goodStudentsFrom2017 =
    students.stream()
     .filter(s -> s.getGraduationYear() == 2017)
     .filter(s -> s.getScore() >= 70)
     .collect(Collectors.toList());
```

Stream pipeline

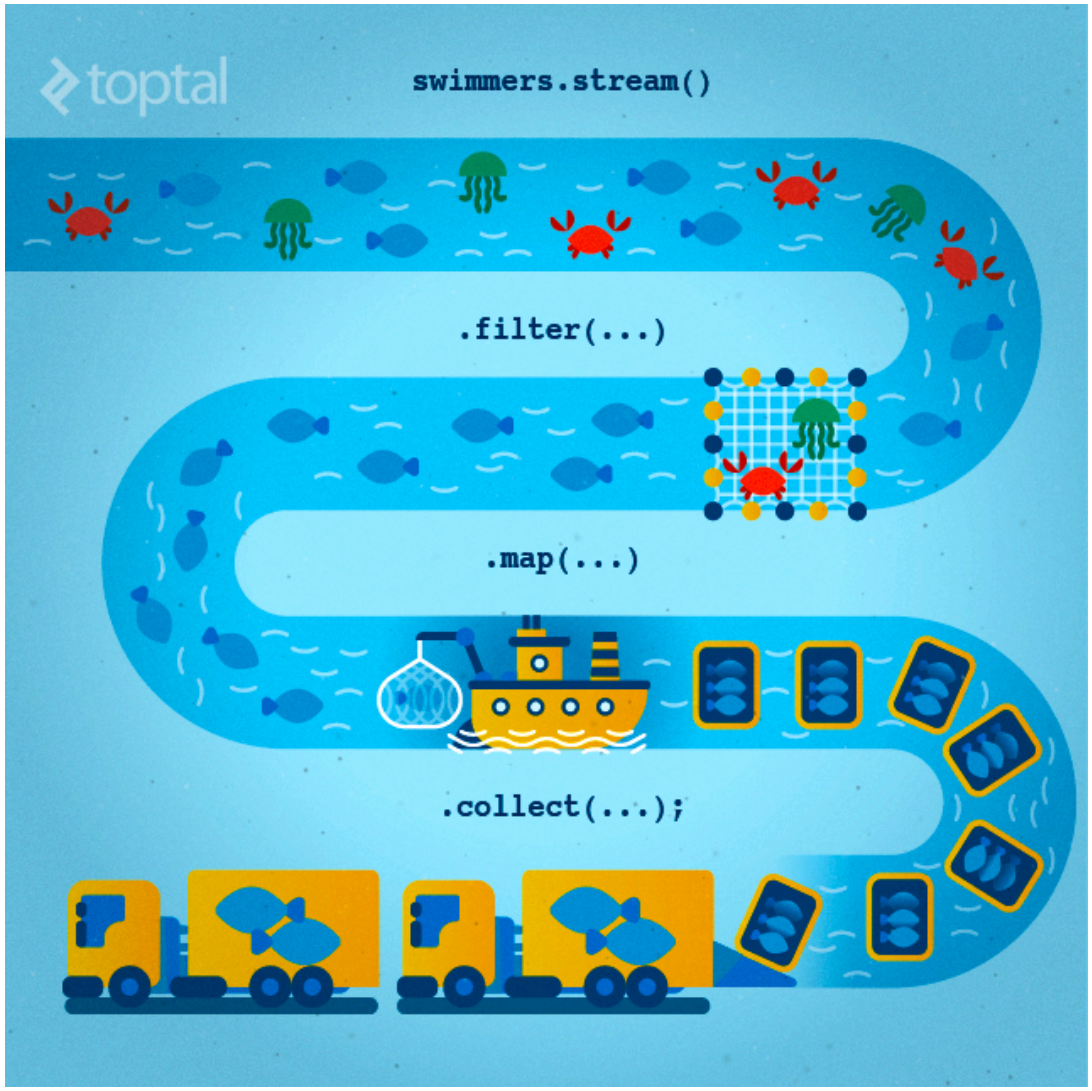
Stream pipeline bestaat uit:

- een **bron**: collection stream, genereerde stream, een array of een I/O channel

- geen of meerdere **intermediate operations** die een nieuwe stream produceren:
filter, map, sorted,…

- één **terminal operation** die een primitive value of optional, een collectie of void als resultaat geeft

Stream pipeline



Intermediate operations

- map()
- filter()
- sorted()
- limit()
- distinct()

https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html
https://docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html

Terminal operations

- collect() -> collecteren
- reduce() -> reduceren
- forEach() -> consumeren

Terminal operation: consumeren

Voorbeeld:

```
List<String> words = Arrays.asList("elephant", "zebra", "pig");
Stream<String> stream = words.stream();
Consumer<String> consumer = System.out::println;
stream.forEach(consumer);
```

Of:

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

Neem ook JavaDoc erbij om de forEach methode te bekijken.

Terminal operation: reduceren

Voorbeeld:

```
int sum = IntStream.rangeClosed(0, 10).sum();

OptionalInt max =
    IntStream.rangeClosed(0, 10).max();
```

Reducerende bewerkingen geven 1 waarde terug

die ook optioneel kan zijn.

Neem ook JavaDoc erbij om de reduceer-methodes te bekijken

Terminal operation: collecteren

Voorbeelden:

```
List<String> words = Arrays.asList("elephant", "zebra", "pig");

Set<String> result = words.stream().collect(Collectors.toSet());

int length =
    words.stream().collect(Collectors.summingInt(String::length));
```

Neem ook JavaDoc erbij om de collect methode te bekijken.

Intermediate operation: filter

Parameter: Predicate<? super T>

Voorbeelden:

```
Stream.of("elephant", "zebra", "pig")
    .filter(s -> s.contains("e"))
    .forEach(System.out::println);

Stream.of("elephant", "zebra", "pig")
    .filter(s -> s.length() <= 5)
    .filter(s -> s.contains("i"))
    .forEach(System.out::println);
```

Neem ook JavaDoc erbij om de filter methode te bekijken.

Intermediate operation: map

`Parameter: Function<? super T,? extends R>`

`Voorbeelden:`

```
OptionalInt min = Stream.of("elephant", "zebra", "pig")
    .mapToInt(s -> s.length())
    .min();
min.ifPresent(System.out::println);

Stream.of("elephant", "zebra", "pig")
    .map(String::toUpperCase)
    .forEach(System.out::println);
```

Neem ook JavaDoc erbij om de map methode te bekijken.

Intermediate operation: sorted

Voorbeeld:

```
Stream.of("elephant", "zebra", "pig")
    .sorted()
    .forEach(System.out::println);
```

Neem ook JavaDoc erbij om de sorted methode te bekijken.

Intermediate operation: limit

Voorbeeld:

```
students.stream()
    .filter(s -> s.getGraduationYear() == 2018)
    .sorted()
    .limit(3)
    .forEach(System.out::println);
```

Intermediate operation: limit

```
public class Student implements Comparable<Student>{
  private String name;
  private int graduationYear;
  private int score;

  …..

  @Override
  public String toString() {
    return name + " [" + score + "]";
  }

  @Override
  public int compareTo(Student other) {
    return Integer.compare(other.score, score);
  }
}
```

Intermediate operation: distinct

Voorbeeld:

```
Stream.of(8,3,4,8,4,5,6,4,3,8)
    .distinct()
    .sorted()
    .forEach(System.out::println);
```

Streaming API

http://www.deadcoderising.com/2015-05-19-java-8-replace-traditional-for-loops-with-intstreams/

Oefeningen

Nu kan je aan de slag met de oefeningen in jullie cursusbundel!

Veel Succes!