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

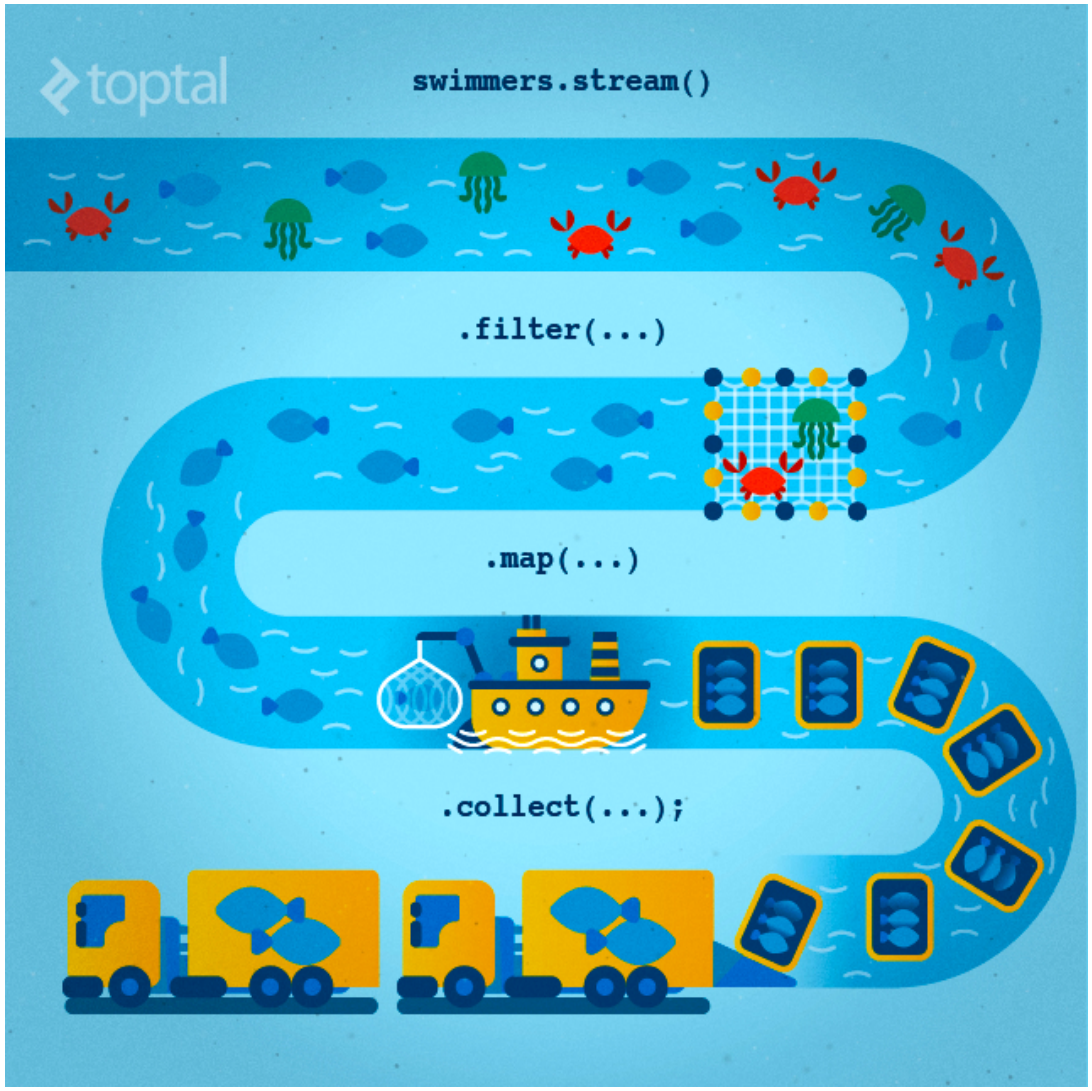
Stream pipeline bestaat uit:

- 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

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

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/stream/package-summary.html

Terminal operations

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

Terminal operation: collecteren

De elementen van een stream verzamelen in een collection.

Voorbeelden:

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

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

Neem ook JavaDoc erbij om de collect methode te bekijken.

Terminal operation: consumeren

De elementen van een stream verwerken (Consume)

Voorbeelden:

```
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

De elementen van een stream combineren tot 1 resultaat.

Dit resultaat kan optioneel (Optional) zijn.

Voorbeelden:

```
long count = Stream.of("elephant", "zebra", "pig").count();
```

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

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

Intermediate operation: filter

Het selecteren van elementen via een bepaalde conditie (Predicate)

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

De elementen van een stream transformeren van een bepaald datatype (T) naar een ander datatype (R)

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

Het sorteren van de elementen in een stream.

Voorbeeld:

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

Neem ook JavaDoc erbij om de sorted methode te bekijken.

Intermediate operation: sorted

```
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: limit

De stream ‘afkappen’ tot een maximum toegelaten elementen.

Voorbeeld:

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

Intermediate operation: distinct

Elementen uit de stream uniek maken.

Elementen die meermaals voorkomen worden verwijderd.

Voorbeeld:

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

Intermediate operation: peek

Kan gebruikt worden om je pipeline te debuggen

Voorbeeld:

```
Stream.of("one", "two", "three", "four")
        .filter(e -> e.length() > 3)
        .peek(e -> System.out.println("Filtered value: " + e))
        .map(String::toUpperCase)
        .peek(e -> System.out.println("Mapped value: " + e))
        .collect(Collectors.toList());
```

Intermediate operation: flatMap

Een Stream van verzamelingen transformeren naar een stream van objecten.

Voorbeeld:

```
List<String> animals = Arrays.asList("zebra", "dog", "dolphine");
List<String> names = Arrays.asList("Wannes", "Hans");
List<String> cities = Arrays.asList("Amsterdam", "Kopenhagen", "Oslo");

List<Integer> lengths =
      Stream.of(animals, names, cities)
      .flatMap(l -> l.stream().map(String::length))
      .collect(Collectors.toList());

assertEquals(Arrays.asList(5, 3, 8, 6, 4, 9, 10, 4), lengths);
```

Oefeningen

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

Veel Succes!