Using Java Streams

USING JAVA STREAMS TO PROCESS AND ANALYZE DATA IN MEMORY

José Paumard
PHD, JAVA CHAMPION, JAVA ROCK STAR

@JosePaumard https://github.com/JosePaumard

How to work with the Stream API

- using map / filter / reduce to analyze data

- in memory data processing

- creating Stream

- computing statistics on data

- creating collections

- working with histograms

This is a Java course

- basic knowledge of Java

- basic knowledge of the Collection API

- how to create and run a simple program

- how to write lambda expressions

Java version 8+, 11+

Agenda

First, what is map / filter / reduce

How to implement this in the right way!

How do streams differ from collections
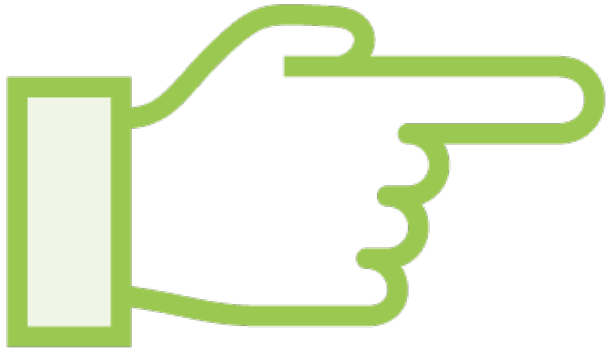
Using Map / Filter / Reduce

Given a list of people

We need to compute the average of the age of those people

For the people older than 20

This classical example is implemented

With a map / filter / reduce approach

Map / Filter / Reduce

people → **map** → age

age → **filter** → age > 20

age > 20 → **reduce** → avg(age > 20)

Implementing Map / Filter / Reduce

```java
List<Person> people = ...;

int sum = 0;
int count = 0;

for (Person person: people) {
    if (person.getAge() > 20) {
        count++;
        sum += person.getAge();
    }
}

double average = 0d;
if (count > 0) {
    average = sum / count;
}
```

Classical implementation using the Iterator pattern

Caveat:
- the code tells how to compute things
- not what to compute

Example is SQL:

```sql
SELECT AVG(age)
    FROM People
    WHERE age > 20
```

```java
List<Person> people = ...;

double average =
    people.map(person -> person.getAge())
          .filter(age -> age > 20)
          .average();
```

What about implementing map / filter / reduce on the Collection API?

Mapping Data

people → **map** → age

1M people → **map** → 1M age

Filtering Data

age → **filter** → age > 20

1M age → **filter** → 600k age > 20

Should a mapper / filter duplicate the data?

```
List<Person> people = ...;

double average =
    people.map(person -> person.getAge())
          .filter(age -> age > 20)
          .average();
```

What about implementing map / filter / reduce on the Collection API?

Is it a good idea to implement it in that way?

The Collection Framework
is not the right place to implement
map / filter / reduce

```java
List<Person> people = ...;

double average =
    people.stream()
          .map(person -> person.getAge())
          .filter(age -> age > 20)
          .average();
```

The Stream API is a concept

- used to implement the map / filter / reduce algorithm

- without duplication

List<Person>: contains the people

person.stream(): returns a Stream<Person>

By construction, a stream is empty

So building a stream is "free"

The Java Stream API implements map / filter / reduce without duplicating the data

Working with Java Streams

```
List<Person> people = ...;

people.stream()
      .map(...)
      .filter(...)
      .average();
```

◄ This is the data

◄ Creates a Stream<Person>
◄ Creates a Stream<Integer>
◄ Creates a Stream<Integer>

◄ Triggers the computation in one
   pass over the data

There are two kinds of methods on Stream:

- methods that create a Stream

- methods that produce a result

A method that creates a Stream
is an intermediate method

A method that produces a result
is terminal method

Module Wrap Up

What did you learn?

What is a Java Stream?

An in-memory implementation of the map / filter / reduce pattern

A stream does not carry any data

Intermediate operations

Terminal operations