Converting a For Loop to a Stream

José Paumard
PHD, JAVA CHAMPION, JAVA ROCK STAR
@JosePaumard https://github.com/JosePaumard

Agenda

How to go from the Iterator pattern…

… to the Stream pattern?

Two examples: a basic one

And more complex and realistic one

Converting a Simple For Loop to a Stream

```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;
}
```

Demo

Let us write some code!

And see this refactoring in action

```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;
}
```

1st step: spot the iteration

```
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;
}
```

1st step: spot the iteration

2nd step: find what is used

the age is used

so there is a mapping

```
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;
}
```

1st step: spot the iteration

2nd step: find what is used
   the age is used
   so there is a mapping

3rd step: not all the ages are used
   only the age greater than 20
   so there is a filtering

```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;
}
```

1st step: spot the iteration

2nd step: find what is used
  the age is used
  so there is a mapping

3rd step: not all the ages are used
  only the age greater than 20
  so there is a filtering

And at the end, an average age is computed
A specialized IntStream is needed

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

int average =
people.stream()
    .mapToInt(p -> p.getAge())
    .filter(age -> age > 20)
    .average
    .orElseThrow();
```

1st step: spot the iteration

2nd step: find what is used
  the age is used
  so there is a mapping

3rd step: not all the ages are used
  only the age greater than 20
  so there is a filtering

And at the end, an average age is computed
A specialized IntStream is needed

Converting a Complex For Loop

```java
double totalAmount = 0;
int frequentRenterPoints = 0;
String statement = composeHeader();
for (Rental rental: rentals) {
    totalAmount += computeRentalAmount(rental);
    frequentRenterPoints += getFrequentRenterPoints(rental);
    statement += computeStatementLine(rental);
}
statement += composeFooter(totalAmount, frequentRenterPoints);
```

A Stream does one thing at a time!

So a for loop that does 3 things…

… should be converted to 3 streams

Demo

Let us write some code!

Let us duplicate this loop

And write these loops as streams

```java
double totalAmount = 0;
for (Rental rental: rentals) {
    totalAmount += computeRentalAmount(rental);
}
```

⇩

```java
double totalAmount =
    rentals.stream()
           .mapToDouble(Statement::computeRentalAmount)
           .sum();
```

```java
int frequentRenterPoints = 0;
for (Rental rental: rentals) {
    frequentRenterPoints += getFrequentRenterPoints(rental);
}
```

```java
int frequentRenterPoints =
   rentals.stream()
          .mapToInt(Statement::computeRentalAmount)
          .sum();
```

```java
String statement = composeHeader();
for (Rental rental: rentals) {
    statement += computeStatementLine(rental);
}
statement += composeFooter(totalAmount, frequentRenterPoints);
```

```java
String header = composeHeader();
String body = rentals.stream()
                     .map(Statement:: computeStatementLine)
                     .collect(Collectors.joining());
String footer = composeFooter(...);
```

Forget about processing your data in one pass

Module Wrap Up

What did you learn?

How to refactor your code to streams

The most important idea:

To be converted to a stream

A for loop can only does one thing