Streams (fluxos de dados)

Para a Streams API, uma **Stream** pode ser definida, de forma sucinta, como **uma sequência de elementos de uma fonte de dados que suporta operações de agregação**. Ver link seguinte, para entender o que são operações de agregação:

https://www.devmedia.com.br/streams-api-trabalhando-com-colecoes-de-forma-flexivel-em-java/31980

Processamento de dados com Streams do Java SE 8 - Parte 1

http://www.oracle.com/technetwork/pt/articles/java/processamento-streams-java-se-8-2763688-ptb.html

The Java 8 Stream API Tutorial

http://www.baeldung.com/java-8-streams

Introduction to Java 8 Streams

http://www.baeldung.com/java-8-streams-introduction

Guide to the Fork/Join Framework in Java

http://www.baeldung.com/java-fork-join

Parallel Streams

See in http://www.baeldung.com/java-8-streams

Before Java 8, parallelization was complex. Emerging of the *ExecutorService* and the *ForkJoin* simplified developer's life a little bit, but they still should keep in mind how to create a specific executor, how to run it and so on.

Java 8 introduced a way of accomplishing parallelism in a functional style.

The API allows creating parallel streams, which perform operations in a parallel mode.

When the source of a stream is a *Collection* or an *Array* it can be achieved with the help of the ***parallelStream()*** method:

```
1   Stream<Product> streamOfCollection = productList.parallelStream();
2   boolean isParallel = streamOfCollection.isParallel();
3   boolean bigPrice = streamOfCollection
4     .map(product -> product.getPrice() * 12)
5     .anyMatch(price -> price > 200);
```

If the source of stream is something different than a *Collection* or an *Array*, the *parallel()* method should be used:

```
1   IntStream intStreamParallel = IntStream.range(1, 150).parallel();
2   boolean isParallel = intStreamParallel.isParallel();
```

Under the hood, *Stream API* automatically uses the *ForkJoin* framework to execute operations in parallel.

The fork/join framework was presented in Java 7. It provides tools to help speed up parallel processing by attempting to use all available processor cores.

By default, the common thread pool will be used and there is no way (at least for now) to assign some custom thread pool to it.

When using streams in parallel mode, avoid blocking operations and use parallel mode when tasks need the similar amount of time to execute (if one task lasts (dura) much longer (muito mais) than the other, it can slow down the complete app's workflow (isso pode atrasar o fluxo de trabalho do aplicativo completo)).

The stream in parallel mode can be converted back to the sequential mode by using the *sequential()* method:

```
1   IntStream intStreamSequential = intStreamParallel.sequential();
```

```
2   boolean isParallel = intStreamSequential.isParallel();
```

Antes do Java 8, a paralelização era complexa. O surgimento do `ExecutorService` e do `ForkJoin` simplificou um pouco a vida do desenvolvedor, mas eles ainda devem ter em mente como criar um **executor** específico, como executá-lo e assim por diante.

O Java 8 introduziu uma maneira de realizar o paralelismo em um estilo funcional.

A API permite criar fluxos paralelos, que executam operações em um modo paralelo.

Quando a fonte de um fluxo é uma Collection ou um Array, ela pode ser obtida com a ajuda do método `parallelStream()`:

```
1   Stream<Product> streamOfCollection = productList.parallelStream();
2   boolean isParallel = streamOfCollection.isParallel();
3   boolean bigPrice = streamOfCollection
4     .map(product -> product.getPrice() * 12)
5     .anyMatch(price -> price > 200);
```