04. Streams: trabalhando melhor com coleções

Date	@31/10/2022
Categoria	Java
Curso	Java 8: conheça as novidades dessa versão

Tópicos

- Streams: trabalhando melhor com coleções
- Filtro de cursos com Stream
- Filtrando cursos com mais de 50 alunos
- Utilizando o método map
- Tirando mais proveito do method reference
- Mais sobre a API de Stream

Streams: Trabalhando melhor com coleções

E se quisermos fazer outras tarefas com essa coleção de cursos? Por exemplo, filtrar apenas os cursos com mais de 100 alunos. Poderíamos fazer um loop que, dado o critério desejado seja atendido, adicionamos este curso em uma nova lista, a lista filtrada.

No Java 8, podemos fazer de uma forma muito mais interessante. Há como invocar um `filter`. Para sua surpresa, esse método não se encontra em `List`, nem em `Collection`, nem em nenhuma das interfaces já conhecidas. Ela está dentro de uma

nova interface, a `Stream`. Você pode pegar um `Stream` de uma coleção simplesmente invocando `cursos.stream()`:

```
Stream<Curso> streamDeCurso = cursos.stream();
```

O que fazemos com ele? O `Stream` devolvido por esse método tem uma dezena de métodos bastante úteis. O primeiro é o `filter`, que recebe um predicado (um critério), que deve devolver verdadeiro ou falso, dependendo se você deseja filtrá-lo ou não. Utilizaremos um lambda para isso:

```
Stream<Curso> streamDeCurso = cursos.stream().filter(c -> c.getAlunos() > 100);
```

Streams primitivos

Trabalhar com Streams vai ser frequente no seu dia a dia. Há um cuidado a ser tomado: com os tipos primitivos. Quando fizemos o `map(Curso::getAlunos)`, recebemos de volta um `Stream<Integer>`, que acaba fazendo o autoboxing dos `int`s. Isto é, utilizará mais recursos da JVM. Claro que, se sua coleção é pequena, o impacto será irrisório. Mas é possível trabalhar só com `int`s, invocando o método `mapToInt`:

```
IntStream stream = cursos.stream()
   .filter(c -> c.getAlunos() > 100)
   .mapToInt(Curso::getAlunos);
```