01

Qual Collection usar

Transcrição

Começando daqui? Você pode fazer o download (https://github.com/alura-cursos/java-collections/archive/aula9.zip) do projeto completo do capítulo anterior e continuar seus estudos a partir deste capítulo.

No decorrer do curso, vocês podem ter tido a seguinte dúvida: O que é uma `Collection` para o Java?

Uma coleção é todo mundo que implementa a interface `Collection`. É ela que possui os métodos que utilizamos durante o treinamento, como `add`, `contains`, `remove` e `size`. Esses quatro métodos são os principais dessa interface e foram os que mais utilizamos.

Todas as classes que forem "filhas", ou implementação de `Collection`. Quais *collections* que vimos até agora? Duas interfaces "filhas", `List` e `Set`. Mas vimos outras implementações também, em especial o `ArrayList` e o `HashSet`.

Então uma `Collection` é uma interface que define métodos e trabalha com uma coleção, com um punhado de objetos. E existem várias formas de trabalhar com um punhado de objetos, como uma forma que pode ter objetos repetidos e estarão em uma sequência de ordem que nós definirmos, que é a `List`, ou onde não pode haver objetos repetidos e que não sabemos, independentemente da ordem dos objetos, que é o `Set`, entre outras formas.

Qual vamos usar? Depende da necessidade de cada um.

Para demonstrar que varia para cada caso. Para representar as aulas da Alura, fazia sentido usarmos uma lista para termos uma ordem, saber qual viria antes ou depois. Já para os alunos, não precisaremos saber qual virá antes ou depois, e nem poderemos ter elementos repetidos, então utilizamos um conjunto, um `Set`.

Mas se você ainda não sabe qual *collection* utilizar, pode-se declarar o atributo como `Collection`. Vamos criar uma classe chamada `QualColecaoUsar`, para demonstrar isso:

```
public class QualColecaoUsar {

    public static void main(String[] args) {

        Collection<Aluno> alunos;
    }
}
```

Ainda não demos `new`, porque ainda não sabemos em quem queremos dar `new`. Em quem podemos dar `new`? Em todas as implementações que vimos! Pois todos são "filhos", "netos", e assim por diante, de `Collection`:

```
public class QualColecaoUsar {

    public static void main(String[] args) {

        Collection<Aluno> alunos = new ArrayList<>();
```

```
        }
    }
```

Com isso, nós conseguimos ainda utilizar os métodos `add`, `remove`, `size` ... Mas não conseguimos utilizar o `get`, porque ele não pertence à interface `Collection`. Entra aí a questão da necessidade. Se precisamos do método `get`, não faz muito sentido utilizarmos a interface `Collection` e sim a interface `List`.

E isso vai ficando um pouco mais claro com o tempo, no começo queremos utilizar `ArrayList` em tudo e com o tempo podemos ver que em alguns casos poderíamos ter utilizado o `HashSet`, que possui a vantagem de performance quando pesquisarmos elementos nele. E se não sabe ainda o que quer, declare simplesmente como `Collection`, conforme suas necessidades você vai definindo qual interface utilizar.

Mesmo na classe utilitária `Collections`, alguns dos seus métodos recebem `Collection` por parâmetro, para ser o mais genérico possível.

O que aprendemos neste capítulo:

- Declaração de atributos utilizando a interface `Collection`.