01

Trabalhando com ArrayList

Transcrição

Conhecer a API de Collections é algo mais que essencial para o desenvolvedor Java. Elas estão em todos os lugares, você utilizando ou não.

Dentre elas, a `ArrayList` é a que aparece com maior frequência. Antes de chegarmos em toda a hierarquia das tais "Collections", vamos praticar bastante as operações essenciais das listas, mais especificamente essa implementação. O que são exatamente as Collections? Vai ficar mais claro no decorrer do curso. Pense nelas como classes que ajudam você a manipular um punhado de objetos.

Para esse curso você estará o tempo todo importando classes e interfaces do pacote `java.util`, **fique atento!** Caso contrário você pode acabar importando classes de outros pacotes que possuem o mesmo nome.

Crie um novo projeto chamado `gerenciador-de-cursos` e vamos programar! Mesmo que você já conheça o conteúdo dessas duas primeiras aulas, que envolvem a utilização dos métodos básicos e ordenação, vale a pena recapitular, para então entrarmos em boas práticas, outras coleções e uso no dia a dia de forma real.

Adicionando elementos em uma lista

Para criar um objeto do tipo `ArrayList`, certamente fazemos como sempre: utilizando o operador `new`. Mas repare que acabamos passando um pouco mais de informações. Ao declarar a referência a uma `ArrayList`, passamos qual o tipo de objeto com o qual ela trabalhará. Se queremos uma lista de nomes de aulas, vamos declarar `ArrayList<String>`. Crie a classe `TestandoListas`, adicionando os nomes de algumas aulas que teremos nesse curso:

```
import java.util.List;
import java.util.ArrayList;

public class TestandoListas {

    public static void main(String[] args) {

        String aula1 = "Modelando a classe Aula";
        String aula2 = "Conhecendo mais de listas";
        String aula3 = "Trabalhando com Cursos e Sets";

        ArrayList<String> aulas = new ArrayList<>();
        aulas.add(aula1);
        aulas.add(aula2);
        aulas.add(aula3);

        System.out.println(aulas);
    }
}
```

Qual é o resultado desse código? Ele mostra as aulas adicionadas em sequência! Por que isso acontece? Pois a classe `ArrayList`, ou uma de suas mães, reescreveu o método `toString`, para que internamente fizesse um `for`, concatenando os seus elementos internos separados por vírgula.

Removendo elementos

Bastante simples! O que mais podemos fazer com uma lista? As operações mais básicas que podemos imaginar, como por exemplo remover um determinado elemento. Usamos o método `remove` e depois mostramos o resultado para ver que a primeira foi removida:

```
aulas.remove(0);
System.out.println(aulas);
```

Por que `0`? Pois as listas, assim como a maioria dos casos no Java, são indexadas a partir do `0`, e não do `1`.

Percorrendo uma lista

Bem, talvez não seja a melhor das ideias fazer um `System.out.println` na nossa lista, pois talvez queiramos mostrar esses itens de alguma outra forma, como por exemplo um por linha. Como fazer isso? Utilizando o `for` de uma maneira especial, chamada de `enhanced for`, ou popularmente `foreach`. Lembrando que `foreach` não existe no Java como comando, e sim como um caso especial do `for` mesmo. Olhe o código:

```
for (String aula : aulas) {
    System.out.println("Aula: " + aula);
}
```

Acessando elementos

E se eu quisesse saber apenas a primeira aula? O método aqui é o `get`. Ele retorna o primeiro elemento se passarmos o `0` como argumento:

```
String primeiraAula = aulas.get(0);
System.out.println("A primeira aula é " + primeiraAula);
```

Você pode usar esse mesmo método para percorrer a lista toda, em vez do tal do `enhanced for`. Para isso, precisamos saber quantos elementos temos nessa lista. Nesse caso, utilizamos o método `size` para limitar o nosso `for`:

```
for (int i = 0; i < aulas.size(); i++) {
    System.out.println("aula : " + aulas.get(i));
}
```

Fizemos até `i < aulas.size()` pois `size` retorna o total de elementos. Se acessássemos até `i <= aulas.size()` teríamos um problema! Uma exception do tipo `IndexOutOfBoundsException` seria lançada! Quer ver? Vamos imprimir o `size` e faça o teste com o código que temos até aqui:

```
import java.util.List;
import java.util.ArrayList;

public class TestandoListas {

    public static void main(String[] args) {
```

```
        String aula1 = "Modelando a classe Aula";
        String aula2 = "Conhecendo mais de listas";
        String aula3 = "Trabalhando com Cursos e Sets";

        ArrayList<String> aulas = new ArrayList<>();
        aulas.add(aula1);
        aulas.add(aula2);
        aulas.add(aula3);

        System.out.println(aulas);
        System.out.println(aulas.size());

        // cuidado! <= faz sentido aqui?
        for (int i = 0; i <= aulas.size(); i++) {
            System.out.println("Aula: " + aulas.get(i));
        }
    }
}
```

Mais uma forma de percorrer elementos, agora com Java 8

Percorrer com o `enhanced for` é uma forma bastante indicada. Já o `for` que fizemos utilizando o `get` possui alguns problemas que veremos em uma próxima aula e vai ficar bastante claro.

Uma outra forma de percorrer nossa lista é utilizando as sintaxes e métodos novos incluídos no Java 8. Temos um método (não um comando!) agora que se chama `forEach`. Ele recebe um objeto do tipo `Consumer`, mas o interessante é que você não precisa criá-lo, você pode utilizar uma sintaxe bem mais enxuta, mas talvez assustadora a primeira vista, chamada lambda. Repare:

```
aulas.forEach(aula -> {
    System.out.println("Percorrendo:");
    System.out.println("Aula " + aula);
});
```

Estranho não? Lambda não é o foco desse curso. Existe um curso no Alura (https://cursos.alura.com.br/course/java8-lambdas) que vai tratar apenas desse assunto e é bastante aconselhado. Aqui estamos falando que, para cada `String aula`, determinado bloco de código deve ser executado. Essa variável `aula` poderia ter o nome que você desejasse.

Ordenando a lista

Esse é bastante fácil quando temos uma `List<String>`. A classe `java.util.Collections` (repare o `s` no final, que é diferente da interface `Collection`, que será vista mais para a frente) é um conjunto de métodos estáticos auxiliares as coleções. Dentro dela há o método `sort`:

```
Collections.sort(aulas);
```

Simples, não? Mas há bastante mágica ocorrendo aqui por trás. Como é que essa classe sabe ordenar listas de `Strings`? E se fosse uma lista de, digamos, `Aula`, também funcionaria? Segure um pouco essas questões e por enquanto vamos fechar esta aula testando esse simples código final:

```
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;

class TestandoListas {

    public static void main(String[] args) {

        String aula1 = "Modelando a classe Aula";
        String aula2 = "Conhecendo mais de listas";
        String aula3 = "Trabalhando com Cursos e Sets";

        ArrayList<String> aulas = new ArrayList<>();
        aulas.add(aula1);
        aulas.add(aula2);
        aulas.add(aula3);

        System.out.println(aulas);

        Collections.sort(aulas);
        System.out.println("Depois de ordenado:");
        System.out.println(aulas);
    }
}
```

O que aprendemos neste capítulo:

- A implementação `ArrayList`.
- O pacote `java.util`.
- Métodos de manipulação do `ArrayList`.
- `ForEach` do Java 8.