# 03. Relacionamentos com coleções

| | |
|---|---|
| Date | @26/10/2022 |
| Categoria | Java |
| Curso | Java Collections: Dominando Listas Sets e Mapas |

## Tópicos

- Relacionamentos com coleções
- Criando relacionamentos com coleções
- Declarando as diferentes implementações como List
- Adicionando novas aulas em um Curso
- Encapsulando o acesso às aulas
- Diferença entre ArrayList e LinkedList

---

## Declarando as diferentes implementações como List

Qual a melhor forma de declarar a referência (tipo) de uma nova lista? Deixando mais genérico, utilizando a interface `List`.

```
List<Aula> aulas = new ArrayList<>();
```

Em vez de declararmos a referência a uma `ArrayList<Aula>` (ou `LinkedList<Aula>`), o ideal é deixarmos mais genérico, utilizando a interface `List`. Por quê? Pelo motivo que já vimos ao estudar orientação a objetos aqui no Alura: não temos motivo para ser

super específico na instância que iremos usar. Se declararmos apenas como `List`, poderemos mudar de implementação, como para uma `LinkedList`, sem problema algum de compilação, por não termos nos comprometido com uma implementação em específico.

---

## Encapsulando o acesso às aulas

Faça o método `getAulas` da classe `Curso` retornar um `Collections.unmodifiableList(aulas)`, para que não seja mais possível alterar o valor dessa lista por fora da própria classe `Curso`.

---

## Diferença entre ArrayList e LinkedList

E o mistério da `LinkedList`? E se tivéssemos usado `ArrayList` na declaração do atributo `aulas` da classe `Curso`? O resultado seria exatamente o mesmo!

Então qual é a diferença? Basicamente performance. A `ArrayList`, como diz o nome, internamente usa uma array para guardar os elementos. Ela consegue fazer umas operações de maneira muito eficiente, como invocar o método `get(indice)`. Se você precisa pegar o décimo quinto elemento, ele te devolve isso bem rápido. Onde uma `ArrayList` é lenta? Quando você for, por exemplo, inserir um novo elemento na primeira posição. Pois a implementação vai precisar mover todos os elementos que estão no começo da lista para a próxima posição. Se há muitos elementos, isso vai demorar... chamamos isso em computação de consumo de tempo linear.

Já a `LinkedList` possui uma grande vantagem aqui. Ela utiliza a estrutura de dados chamada lista ligada. Ela é muito rápida para adicionar e remover elementos na *cabeça* da lista, isso é, na primeira posição. Mas ela é lenta se você precisar acessar um determinado elemento, pois a implementação precisará percorrer todos os elementos até chegar ao décimo quinto, por exemplo.

---