07

Faça como eu fiz

Nesta aula, a gente aprendeu a manipular listas com elementos nulos, modificando a função que imprime a lista formatada:

```
fun List<Livro?>.imprimeComMarcadores() {
    val textoFormatado = this
        .filterNotNull()
        .joinToString(separator = "\n") {
            " - ${it.titulo} de ${it.autor}"
        }
    println(" #### Lista de Livros #### \n$textoFormatado")
}
```

Além de trabalhar com listas com elementos possivelmente nulos, temos que nos preocupar também com elementos válidos, mas que possuem algum atributo nulo. Para esse caso, vimos como agrupar a nossa lista de livros pelo campo editora que é do tipo `String?` utilizando a função `groupBy` e o operador `?:` (*elvis operator*):

```
listaDeLivros
    .groupBy { it.editora ?: "Editora desconhecida" }
    .forEach { (editora: String, livros: List<Livro>) ->
        println("$editora: ${livros.joinToString { it.titulo }}")
    }
```

Para compreender melhor a diferença entre listas mutáveis e listas somente leitura, nós criamos uma classe `Prateleira` compondo uma lista mutável de livros e dois métodos que utilizavam a função `sortBy` para organizar a lista de livros:

```
data class Prateleira (
    val genero: String,
    val livros: MutableList<Livro>
) {
    fun organizaLivrosPorAutor(): MutableList<Livro> {
        this.livros.sortBy { it.autor }
        return this.livros
    }
    fun organizaLivrosPorAnoPublicacao(): MutableList<Livro> {
        this.livros.sortBy { it.anoPublicacao }
        return this.livros
    }
}
```

Então, testamos nossa classe:

```
val prateleira = Prateleira(genero = "Literatura", livros = listaDeLivros)

val porAutor = prateleira.organizarPorAutor()
val porAnoPublicacao = prateleira.organizarPorAnoPublicacao()
```

```
porAutor.imprimeComMarcadores()
porAnoPublicacao.imprimeComMarcadores()
```

Ao realizar este teste, percebemos que havia uma inconsistência com o nosso código, pois as duas variáveis continham a mesma lista organizada da mesma maneira, e não é isso o que estávamos esperando.

Voltamos, então, para a nossa classe `Prateleira` e modificamos o tipo da nossa lista de livros para uma lista somente leitura `List<Livro>` . Ao fazermos isso, percebemos que a chamada da função `sortBy` passou a apresentar erro de compilação, já que ela não está presente em listas não mutáveis. Fizemos a seguinte alteração:

```
data class Prateleira (
   val genero: String,
   val livros: List<Livro>
) {
   fun organizarPorAutor(): List<Livro> {
       return livros.sortedBy { it.autor }
   }
   fun organizarPorAnoPublicacao(): List<Livro> {
       return livros.sortedBy { it.anoPublicacao }
   }
}
```

Com a função `sortedBy` obtivemos o resultado esperado já que, a partir de agora, cada chamada de um dos métodos de organização da lista gera uma nova lista organizada e não reorganiza a lista de livros do objeto `Prateleira` .