**Imutabilidade**

A imutabilidade, citada anteriormente ao falarmos sobre record, é um conceito importante em Java, que se refere à capacidade de um objeto não poder ser alterado depois de criado. Existem algumas classes que são imutáveis por padrão, como por exemplo, as classes String, Integer, Boolean, entre outras. Isso significa que, uma vez criado um objeto dessas classes, não é possível modificar o seu estado.

Vamos exemplificar. Dado o record abaixo:

public record Estudante(String nome, int idade) {}COPIAR CÓDIGO

Uma vez criado um objeto Estudante, seus valores não podem ser modificados:

Estudante estudante1 = new Estudante(“Alice”, 19);COPIAR CÓDIGO

Observe que após essa criação, eu não consigo setar outro nome ou idade para o objeto estudante1.

estudante1.setNome(“Maria”); //Essa possibilidade não existe

estudante1.nome = “Maria”; //Essa possibilidade não existeCOPIAR CÓDIGO

Qualquer uma das tentativas acima, vai apresentar erro de compilação, pois não possível atribuir nenhum outro nome a variável estudante1.

Com relação ao record, fica bem claro, certo? Mas e a String, por exemplo? Eu consigo fazer os passos abaixo no código:

String nome = “Maria”;

nome = “Alice”;COPIAR CÓDIGO

Se a String é imutável, o certo era eu não conseguir atribuir o conteúdo “Alice” à variável nome, correto?

No caso da String e de outras classes imutáveis que citei acima, a variável nome contém uma referência ao objeto da classe String que contém o valor "Maria".

No entanto, quando você tenta alterar o valor da string, o que realmente acontece é que um novo objeto da classe String é criado com o novo valor e a variável é atualizada para armazenar uma referência ao novo objeto.

Por isso, podemos dizer que a classe String é imutável, porque uma vez que um objeto da classe String é criado, ele não pode ser alterado. No entanto, as variáveis que armazenam referências a objetos da classe String podem ser atualizadas para referenciar novos objetos, que são criados a partir do conteúdo do objeto original.

A imutabilidade é importante por várias razões, entre elas:

* Concorrência: objetos imutáveis são seguros para uso em ambientes concorrentes, já que não há necessidade de sincronização.
* Segurança: objetos imutáveis são seguros contra alterações acidentais ou mal-intencionadas.
* Desempenho: objetos imutáveis podem ser armazenados em cache e reutilizados, o que pode melhorar o desempenho.