Java Records

Profa. Karen Selbach Borges

Atualizado em março/2025

Introdução

O Java Records são um recurso introduzido no Java 14 (como uma funcionalidade preview) e se tornou oficial no Java 16.

Oferecem uma maneira concisa e segura de criar classes imutáveis.

Classes imutáveis são aquelas cujas instâncias não podem ser modificadas durante toda a vida útil do objeto.

Ter um objeto imutável garante que seu valor não será modificado em nenhum lugar do sistema.

O que é um Java Record

É a mesma ideia de construção de um JavaBean, possui:

- construtor,
- atributos,
- métodos acessores
- toString
- equals
- hashCode

Exemplo de JavaBean

```
public class User {

    private String name;

    private String password;

    public User(String name, String password) {
        this.name = name;
        this.password = password;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        User user = (User) o;
        return name.equals(user.name) &&
password.equals(user.password);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, password);
    }

    @Override
    public String toString() {
        return "User{" +
                "name='" + name + '\'' +
                ", password='" + password + '\'' +
                '}';
    }
}
```

A mesma classe usando JavaRecord

```
public record User(String name, String password){}
```

OBS: todo record é uma especialização da classe java.lang.Record.

Logo, é possível trabalhar com os métodos equals, toString e hashCode, sem necessidade de especializá-los.

Construtores

Por padrão um java record possui um construtor que inicializa todos os atributos da classe.

Entretanto, é possível customizar o construtor.

```
public record Person(String name, int age) {

    public Person {
        if (name == null) {
            throw new IllegalArgumentException("Name cannot be null");
        }
        if (age < 0) {
            throw new IllegalArgumentException("Age cannot be negative");
        }
    }

}
```

Construtores

É possível, também, sobrecarregar o construtor.

```
public record Person(String name, int age) {
    public Person(String name, int age) {
        this(name, age);
    }

    public Person(String name) {
        this(name, 0);
    }
}
```

Importante

- Dado a contexto de imutabilidade: .
  - não existem métodos setter.
  - não é possível acessar diretamente um atributo. EX: o código a seguir não compila:

    ```
    Pessoa p = new Pessoa("João", 30);
    p.idade = 31;
    ```

  - não é possível estender um record diretamente de outro record ou classe. Ou seja, um record não pode ser uma subclasse de outro record, mas pode implementar interfaces.

- Os métodos acessores não usam a terminologia com “get” apenas o próprio nome do atributo, ou seja, ao invés de getName(), apenas name(). Ex: String name = user.name();

Quando usar

Os Records são uma ótima opção para representar dados, como, por exemplo, DTOs (Data Transfer Objects).

São, também, uma ótima opção para representar dados que não precisam ser alterados após a criação do objeto, como, por exemplo, configurações de aplicação.

Records não são uma boa opção para representar entidades de banco de dados, pois eles não suportam herança, o que significa que não é possível estender um Record em outra classe.

Pelo fato de os Records serem imutáveis por padrão, eles não podem ser utilizados para representar entidades da JPA (Java Persistence API), pois a JPA necessita de entidades mutáveis.

Referências

LIMA, Cleison. Conheça o recurso de Records no Java. Treinaweb. Disponível em https://www.treinaweb.com.br/blog/conheca-o-recurso-de-records-no-java