09

A interface CharSequence

Nos vídeos talvez você tenha percebido que alguns métodos da classe `String` recebem uma variável do tipo `CharSequence`. O tipo `CharSequence` é uma interface que a própria classe `String` implementa (pois uma `String` é uma sequência de caracteres!):

```
public class String implements CharSequence {
```

Quando usamos a classe `String` até poderìamos declarar a variável com o tipo da inteaface, mas isso é raro de se ver:

```
CharSequence seq = "é uma sequencia de caracteres";
```

O interessante é que existem outras classes que também implementam a interface `CharSequence`. Em outras palavras, existem outras classes que são sequências de caracteres além da classe `String`. Por quê?

A classe `StringBuilder`

Vimos que a classe `String` é especial pois gera objetos imutáveis. Isso é considerado benéfico pensando no design mas é ruim pensando no desempenho (e por isso devemos usar aspas duplas na criação, pois a JVM quer contornar os problemas no desempenho com otimizações).

Agora vem um problema: imagina que você precisa criar um texto enorme e precisa concatenar muitas `String`, por exemplo:

```
String texto = "Socorram";
texto = texto.concat("-");
texto = texto.concat("me");
texto = texto.concat(", ");
texto = texto.concat("subi ");
texto = texto.concat("no ");
texto = texto.concat("ônibus ");
texto = texto.concat("em ");
texto = texto.concat("Marrocos");
System.out.println(texto);
```

Nesse pequeno exemplo já criamos vários objetos, só porque estamos concatenas algumas `String`s. Isso é nada bom pensando no desempenho e para resolver isso existe a classe `StringBuilder` que ajuda na concatenação de `String`s de forma mais eficiente.

Veja o mesmo código usando o `StringBuilder`:

```
StringBuilder builder = new StringBuilder("Socorram");
builder.append("-");
builder.append("me");
builder.append(", ");
builder.append("subi ");
```

```
builder.append("no ");
builder.append("ônibus ");
builder.append("em ");
builder.append("Marrocos");
String texto = builder.toString();
System.out.println(texto);
```

O `StringBuilder` é uma classe comum. Repare que usamos o `new` para a criação do objeto. Além disso, como o objeto é mutável, utilizamos a mesma referência, sem novas atribuições.

A interface CharSequence

Agora o legal é que a classe `StringBuilder` também implementa a interface `CharSequence` :

```
public class StringBuilder implements CharSequence {
```

```
CharSequence cs = new StringBuilder("também é uma sequencia de caracteres");
```

Isso faz que alguns métodos da classe `String` sabem trabalhar com o `StringBuilder` , por exemplo:

```
String nome = "ALURA";
CharSequence cs = new StringBuilder("al");

nome = nome.replace("AL", cs);

System.out.println(nome);
```

Vice-versa a classe `StringBuilder` tem métodos que recebem o tipo `CharSequence` . Dessa forma podemos trabalhar de maneira compatível com as duas classes, baseado numa interface comum.