02

Manipulando dados usando a classe StringBuilder

Para suportar Strings mutáveis, o Java possui as classes `StringBuffer` e `StringBuilder`. A operação mais básica é o `append` que permite concatenar ao mesmo objeto:

```
StringBuffer sb = new StringBuffer();
sb.append("Caelum");
sb.append(" - ");
sb.append("Ensino e Inovação");

System.out.println(sb);  // Caelum - Ensino e Inovação
```

COPIAR CÓDIGO

Repara que o `append` não devolve novos objetos como em `String`, mas altera o próprio `StringBuffer`, que é mutável.

Podemos criar um objeto desse tipo de diversas maneiras diferentes:

```
// vazio
StringBuilder sb1 = new StringBuilder();
// conteudo inicial
StringBuilder sb2 = new StringBuilder("java");
// tamanho inicial do array para colocar a string
StringBuilder sb3 = new StringBuilder(50);
// baseado em outro objeto do mesmo tipo
StringBuilder sb4 = new StringBuilder(sb2);
```

COPIAR CÓDIGO

Tenha cuidado: ao definir o tamanho do array, não estamos criando uma `String` de tamanho definido, somente um array desse tamanho que será utilizado pelo `StringBuilder`, portanto:

```
StringBuilder sb3 = new StringBuilder(50);
System.out.println(sb3); // linha em branco
System.out.println(sb3.length()); // 0
```

COPIAR CÓDIGO

As classes `StringBuffer` e `StringBuilder` têm exatamente a mesma interface (mesmos métodos), sendo que a primeira é ::thread-safe:: e a última não (e foi adicionada no Java 5). Quando não há compartilhamento entre threads, use sempre que possível a `StringBuilder`, que é mais rápida por não precisar se preocupar com ::locks::.

Inclusive, em Java, quando fazemos concatenação de Strings usando o `+`, por baixo dos panos, é usado um `StringBuilder`. Não existe a operação `+` na classe `String`. O compilador troca todas as chamadas de concatenação por `StringBuilder`s (podemos ver isso no bytecode compilado).

Principais métodos de StringBuffer e StringBuilder

Há a família de métodos `append` com overloads para receber cada um dos primitivos, Strings, arrays de chars, outros `StringBuffer` etc. Todos eles devolvem o próprio `StringBuffer` / `Builder` o que permite chamadas encadeadas:

```
StringBuffer sb = new StringBuffer();
sb.append("Caelum").append(" - ").append("Ensino e Inovação");
System.out.println(sb);  // Caelum - Ensino e Inovação
```

COPIAR CÓDIGO

O método `append` possui uma versão que recebe `Object` e chama o método `toString` de seu objeto.

Há ainda os métodos `insert` para inserir coisas no meio. Há versões que recebem primitivos, Strings, arrays de char etc. Mas todos têm o primeiro argumento recebendo o índice onde queremos inserir:

```
StringBuffer sb = new StringBuffer();
sb.append("Caelum - Inovação");
sb.insert(9, "Ensino e ");

System.out.println(sb); // Caelum - Ensino e Inovação
```

COPIAR CÓDIGO

Outro método que modifica é o `delete`, que recebe os índices inicial e final:

```
StringBuffer sb = new StringBuffer();
sb.append("Caelum - Ensino e Inovação");
sb.delete(6, 15);

System.out.println(sb); // Caelum e Inovação
```

COPIAR CÓDIGO

Para converter um `StringBuffer` / `Builder` em `String`, basta chamar o `toString` mesmo. O método `reverse` inverte seu conteúdo:

```
System.out.println(new StringBuffer("guilherme").reverse());
                                                // emrehliug
```

COPIAR CÓDIGO

Fora esses, também há o `charAt`, `length()`, `equals`, `indexOf`, `lastIndexOf`, `substring`.

Cuidado, pois o método `substring` não altera o valor do seu `StringBuilder` ou `StringBuffer`, mas retorna a `String` que você deseja. Existe também o método `subSequence` que recebe o início e o fim e funciona da mesma maneira que o `substring` com dois argumentos.